# 京速コンピュータ「京」における CGPOP Miniappの性能評価

中尾昌広，佐藤三久

Center for Computational Sciences, University of Tsukuba
RIKEN Advanced Institute for Computational Science



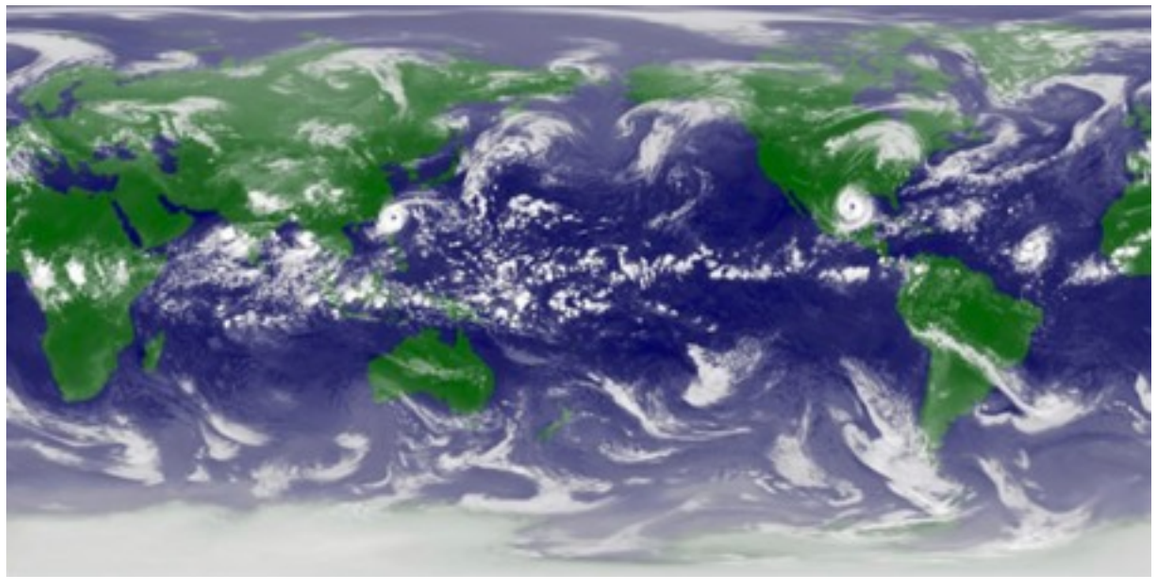

第139回HPC研究会@東京大学 柏の葉キャンパス, 2013年5月29日

# 研究背景

- エクサスケール規模の計算環境では，超高精度な気象予測が可能になると考えられている

- 多国間国際研究協力事業（G8）「エクサスケール・コンピューティングによる精緻な気候シミュレーションの実現」では，エクサスケールに向けた研究が各国共同で行われている

- 日本，アメリカ，フランス，ドイツ，スペイン，カナダ



http://www.ccs.tsukuba.ac.jp/CCS/research/depart_intro/depart_environ

# 研究目的

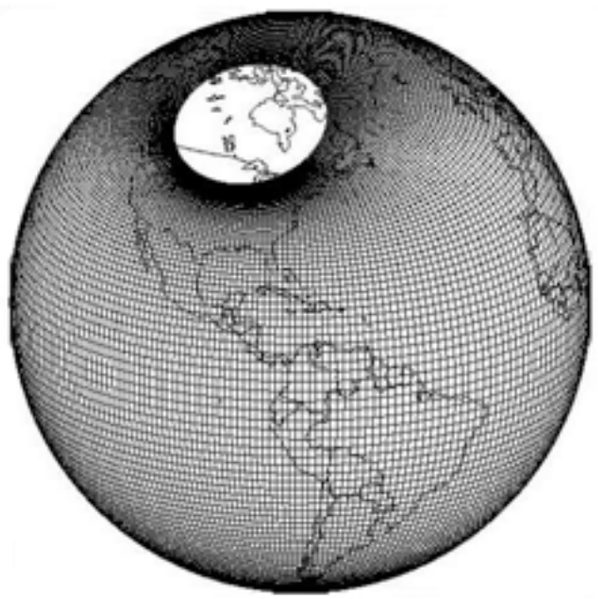
- 海洋シミュレーションコードPOP（Parallel Ocean Program）のミニアプリである**CGPOP Miniapp**を京にポーティングし，大規模環境下における性能チューニングのノウハウの蓄積
- CGPOPの性能チューニングを通して，今後のエクサスケールに向けたプログラミングモデルについて考察

# この後の発表の流れ

- CGPOP Miniappの説明
- 京への性能チューニング
  - 集合通信最適化
  - 1対1通信最適化
  - スレッド化
- 今後のプログラミング環境についての考察
- まとめ

# CGPOP Miniapp（1/2）

- コロラド州立大学が開発
- ロスアラモス国立研究所が開発した海洋シミュレーションPOPのミニアプリ
  - CGPOPは，POP中の性能への影響が大きい共役勾配法（CG法）を抽出したもの
- 全球を2次元グリッド（3600x2400）と定義
- Fortran90 + MPI / Coarray Fortran + MPI
  - POPのコード行数は約71,000行に対し，CGPOPのコード行数は約3,000行



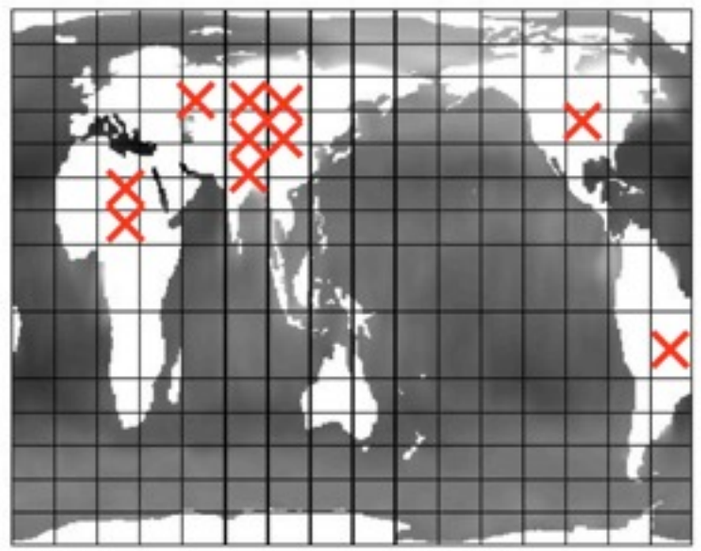
http://www.cs.colostate.edu/hpc/cgpop/

# CGPOP Miniapp（2/2）

- CGPOPで用いられる並列数について
  - 全球を3600x2400の領域に定義し，あるブロックサイズで分割する
  - ブロックサイズが36x24の場合，計算されるブロックは，陸地のみのブロックを除いた**7545**（= 100x100 - 2455）である
  - 計算対象のブロック毎にプロセスを割り当てる

| Block Size | Num. of BLK |
|---|---|
| 225x150 | 228 |
| 180x120 | 358 |
| 144x96 | 541 |
| 120x80 | 764 |
| 90x60 | 1312 |
| 72x48 | 2009 |

| Block Size | Num. of BLK |
|---|---|
| 60x40 | 2822 |
| 48x32 | 4324 |
| 45x30 | 4884 |
| 36x24 | 7545 |
| 30x20 | 10705 |
| 24x16 | 16528 |



http://www.cs.colostate.edu/~stonea/works/pgas2011_talk.pdf

# CGPOPの性能チューニング on 京

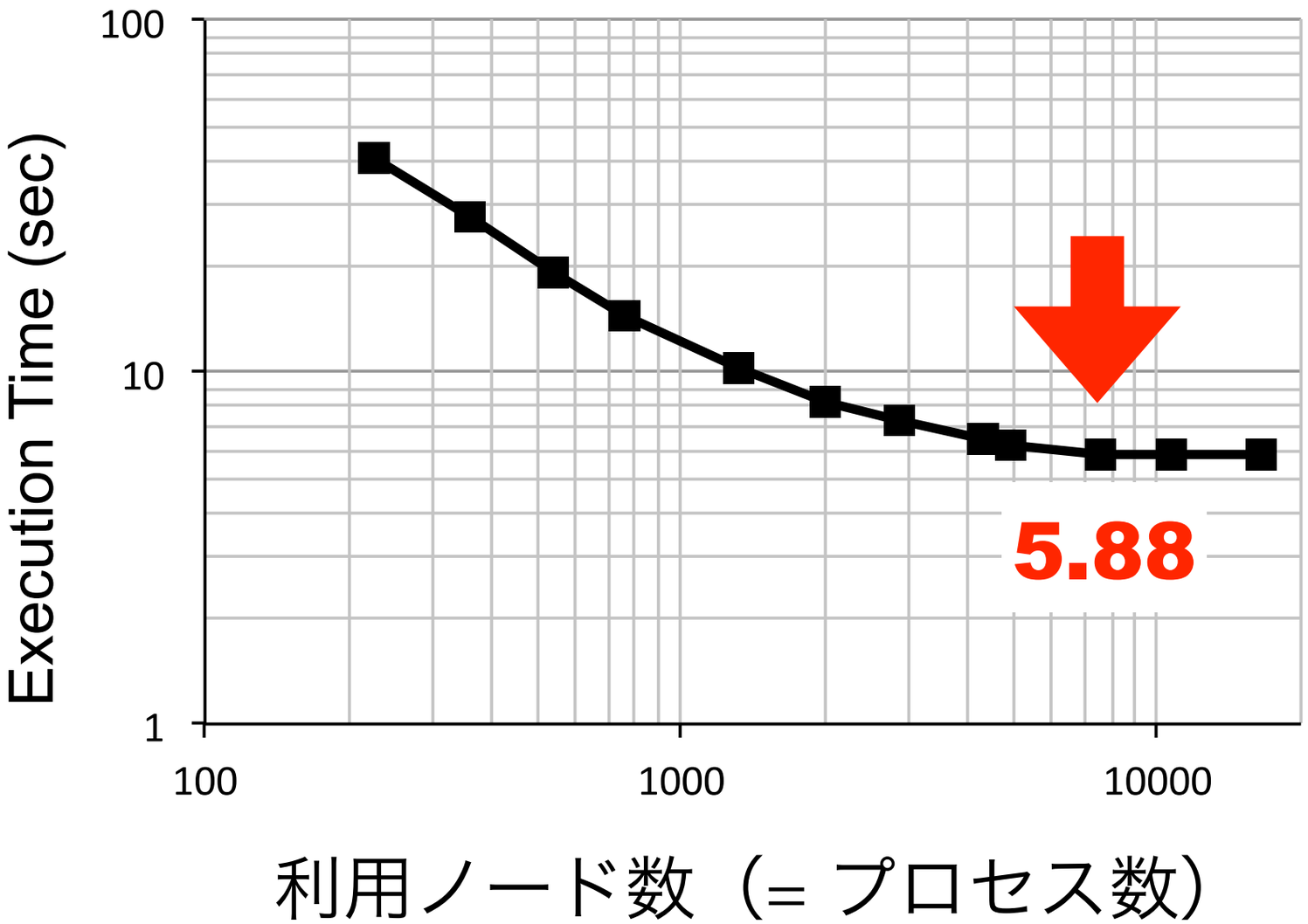
- 方針
  - アルゴリズムは変えない
  - 通信については，京のハードウェアサポート機構を利用

- 手順

1. 性能のボトルネックとなる箇所を探す
2. 性能チューニング
3. 再計測

} 繰り返す

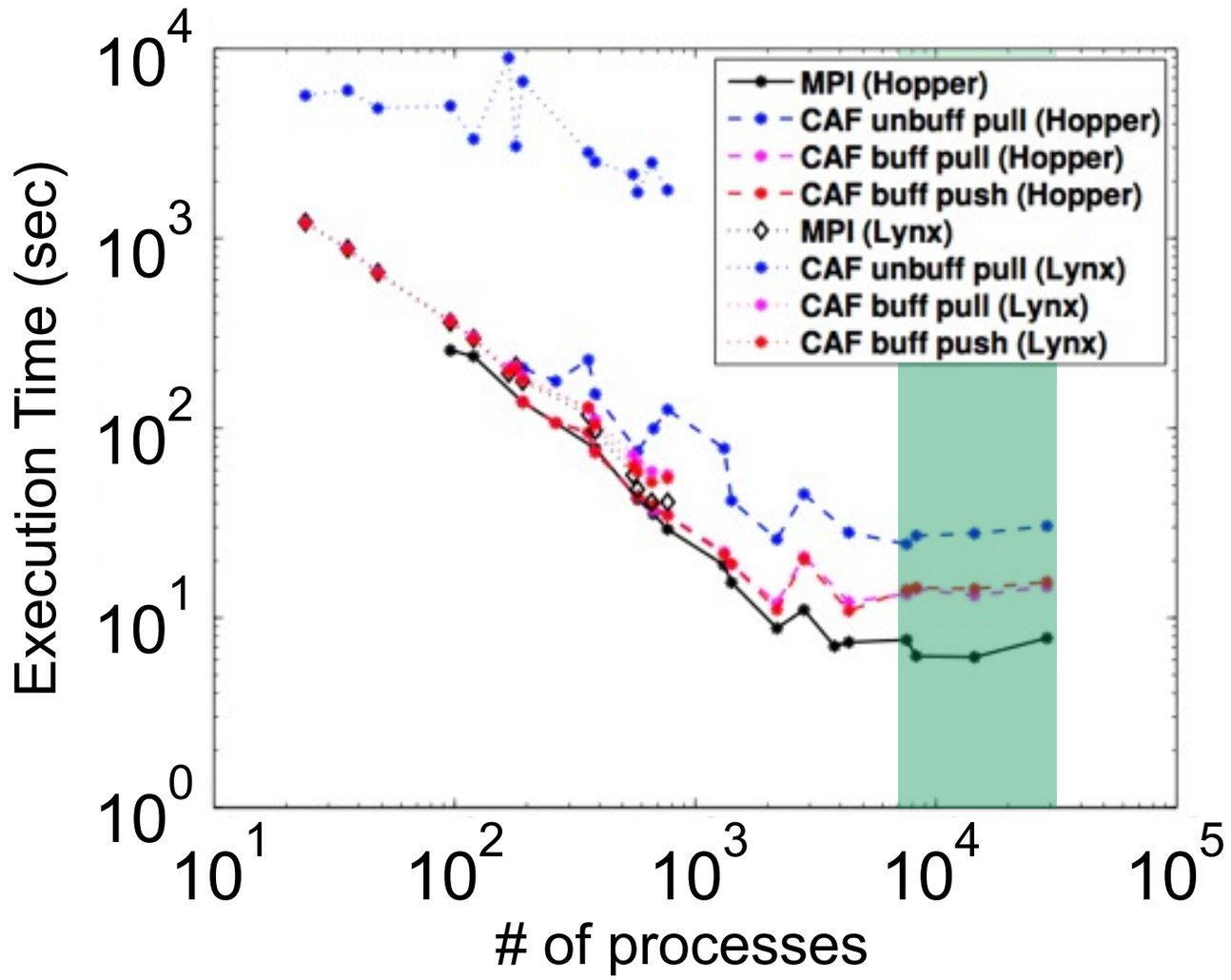
# 初期評価




- 後で述べるスレッド化と比較するため，1ノード1プロセス
- 実はflat-MPIでも，最良値はほぼ同じ
- 7545ノードから性能はほとんど変化しない

# 他のクラスタでの評価

- “Evaluating Coarray Fortran with the CGPOP Miniapp”, Andrew I. Stone, et al, PGAS11, Oct. 2011
- Crayマシンを使った評価，flat MPI
  - Hopper (XE6)
  - Lynx（XT5）
- 約7000プロセス（約300ノード）で性能が横ばいになる
  - 最良点は6秒程度



# CGPOPのアルゴリズムと各処理時間

- CGPOPのアルゴリズム（繰り返しの箇所のみ）

1. GlobalSum（集合通信）

   MPI_Allreduce + 待ち時間

2. UpdateHalo（袖通信）

   MPI_Isend/Irecv

3. 計算処理

※ 2と3は負荷不均等なので,
　1には待ち時間が発生

# CGPOPのアルゴリズムと各処理時間

1. GlobalSum（集合通信）

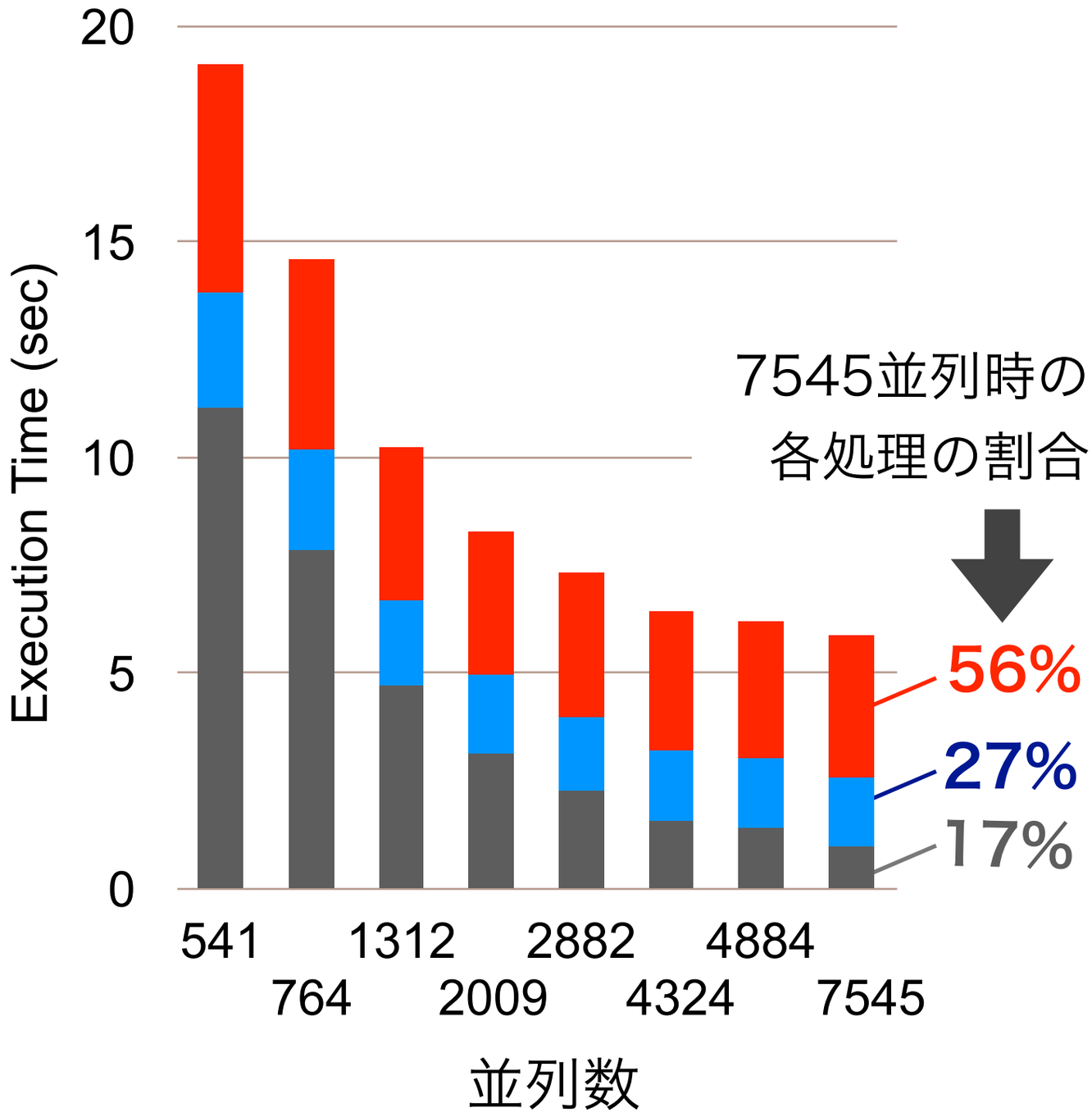
   MPI_Allreduce + 待ち時間

2. UpdateHalo（袖通信）

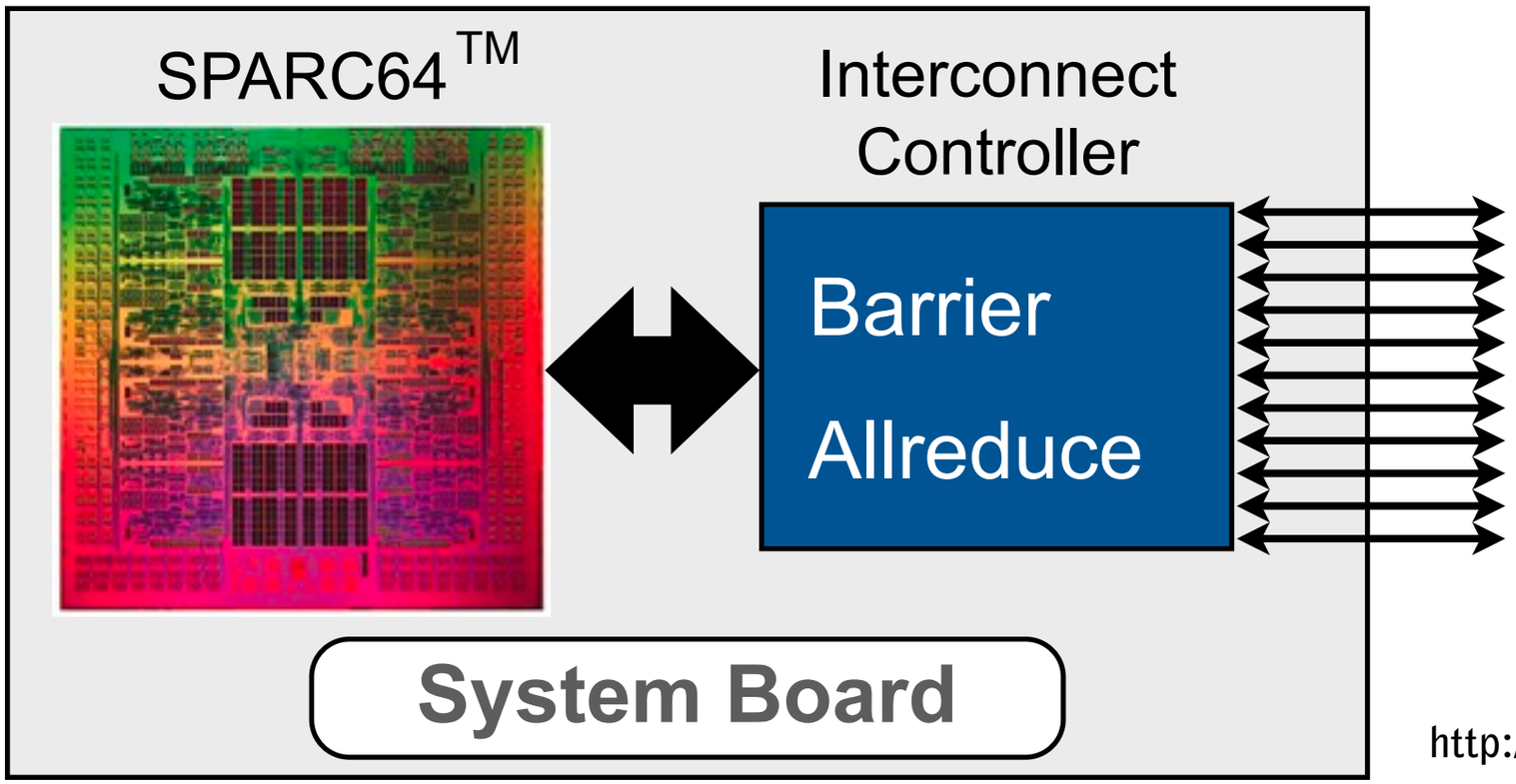
   MPI_Isend/Irecv

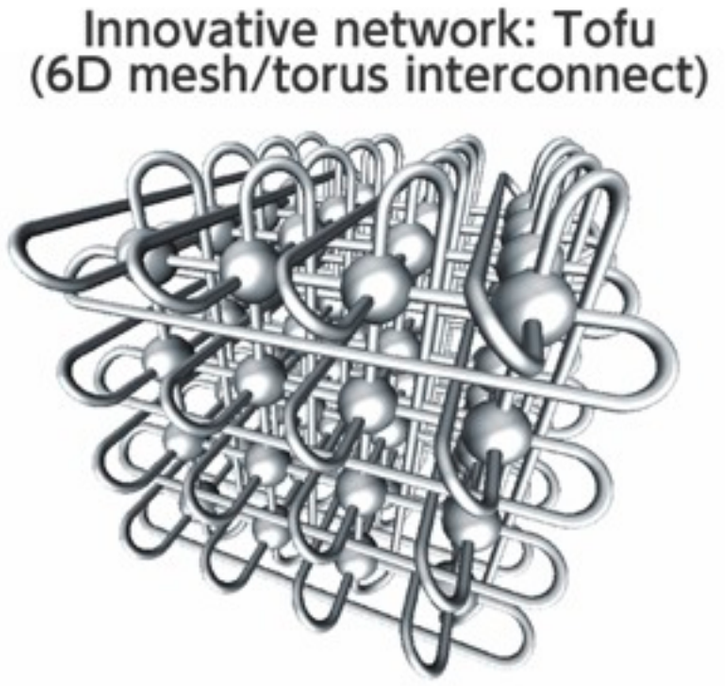
3. 計算処理

上の順で最適化を行う



# 京のネットワークについて



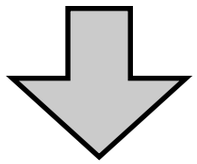

http://www.aics.riken.jp/en/kcomputer/system-2.html

- Interconnect ControllerはMPI_BarrierやAllreduceなどの集合通信に対するハードウェアサポートを行う機能がある
- いくつかの条件を満たした場合，自動的にサポートされる

# MPI_Allreduceのハードウェアサポート

- 条件（今回の実験に関係あるものだけ抜粋）
  - バリアゲートを必要数確保すること（1ノード1プロセスだと問題なし）
  - 転送データは1要素であること
- コード変更

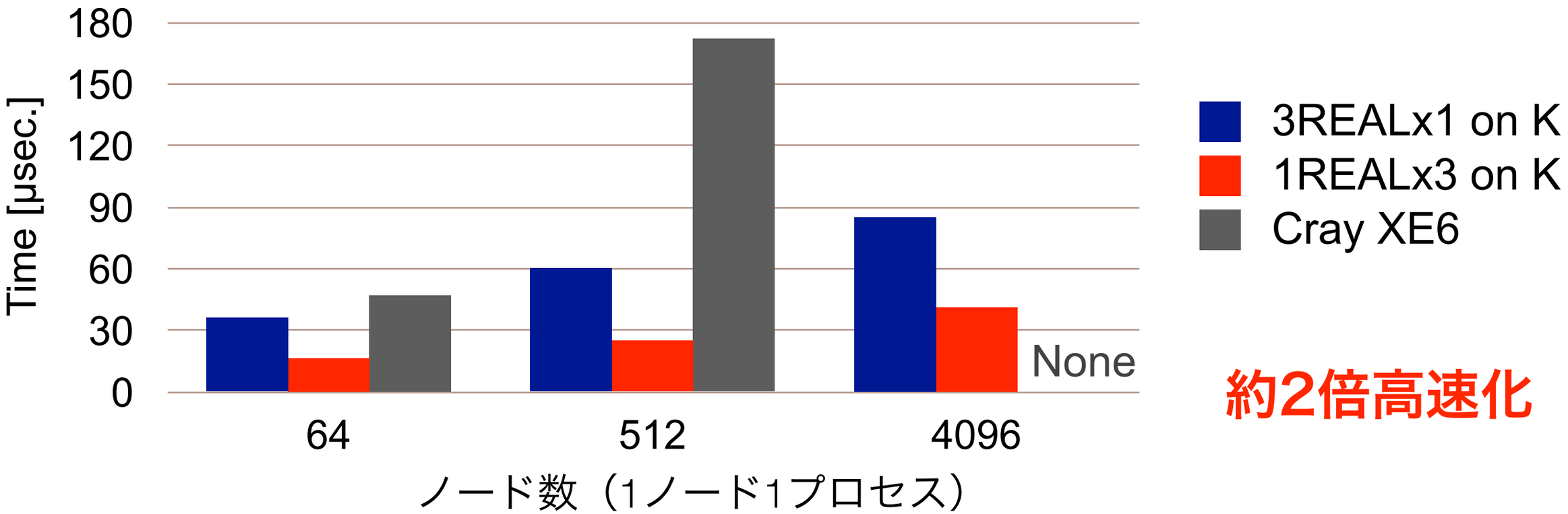
```
length = 3
call MPI_Allreduce(send, recv_sum, length, MPI_REAL, ...)
```



```
do i=1, 3
  call MPI_Allreduce(send(i), recv_sum(i), 1, MPI_REAL, ...)
end do
```

# 予備実験1

- どの程度高速化するか？
  - 京でREAL型3要素を1回で転送する場合【3REALx1 on K】
  - 京でREAL型1要素を3回で転送する場合【1REALx3 on K】
  - Cray XE6（Gemini Network）でREAL型3要素を1回で転送する場合 【Cray XE6】

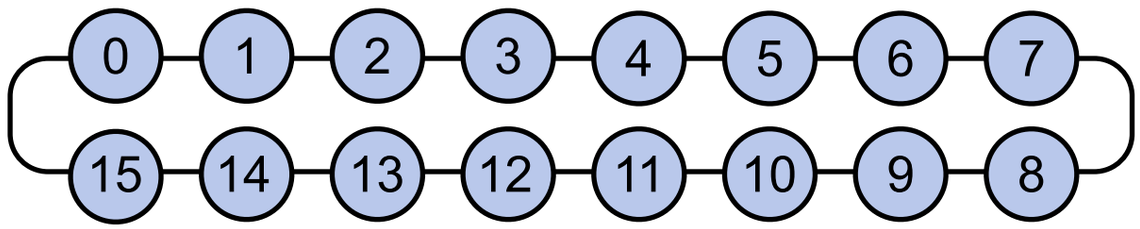



**約2倍高速化**

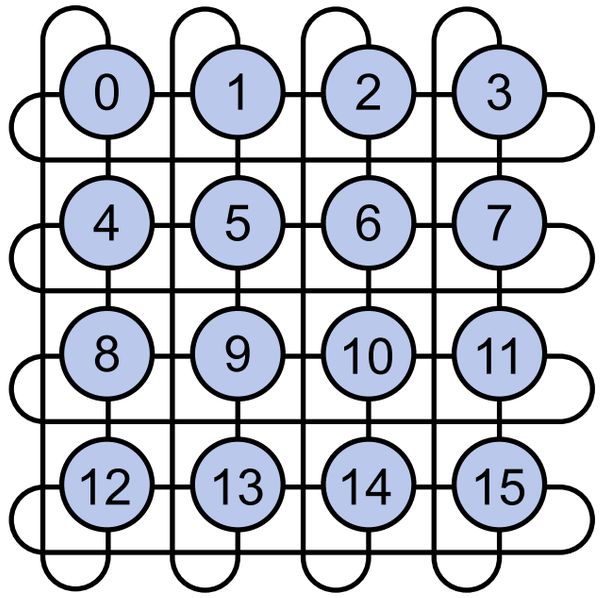
# ジョブ形状

- Tofuインターコネクトは6次元メッシュ/トーラス
- ユーザは，ジョブ形状（プロセス配置）を1, 2, 3次元の中から選択することができる

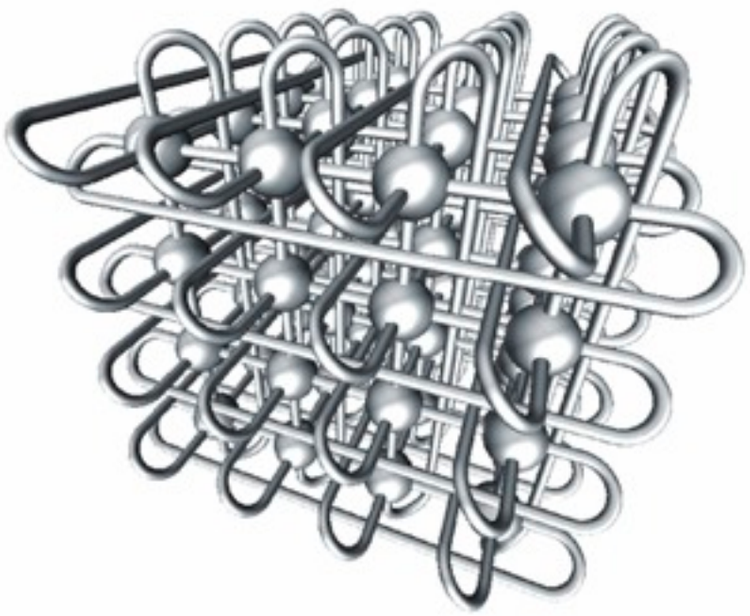
1D.




2D.




3D.

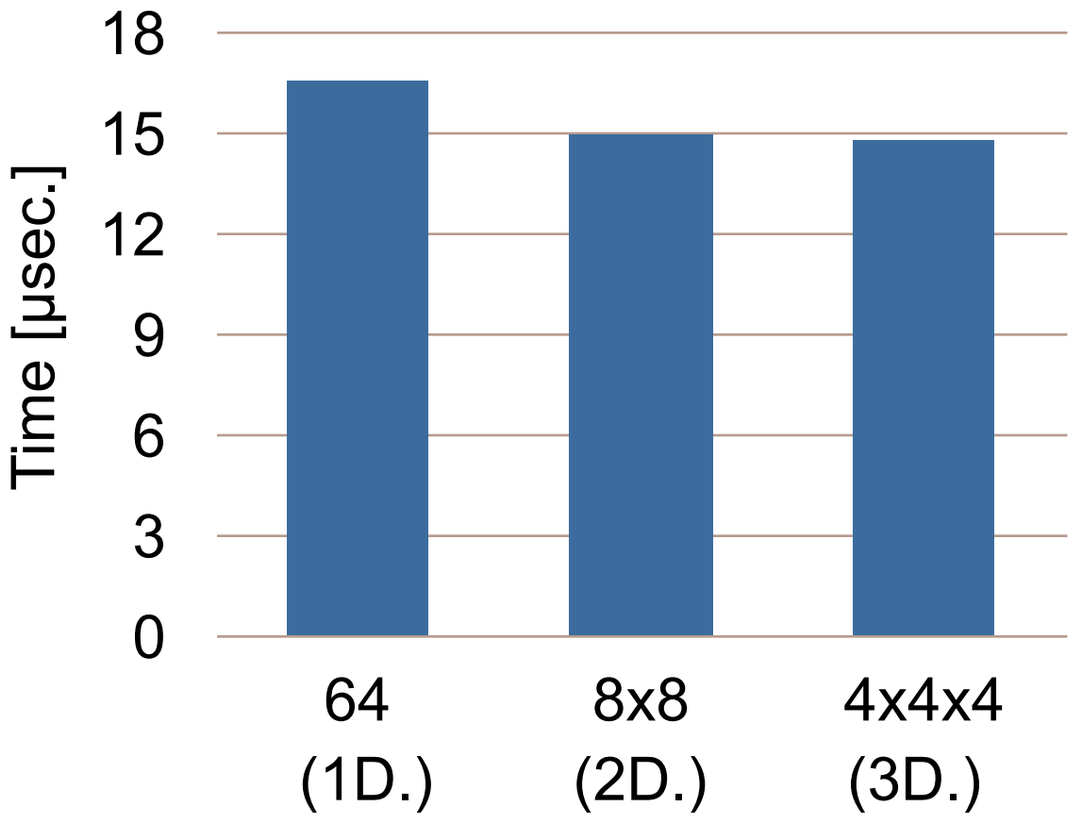


高次元の方が，ホップ数が少なくなるため
性能は高くなると予測できる

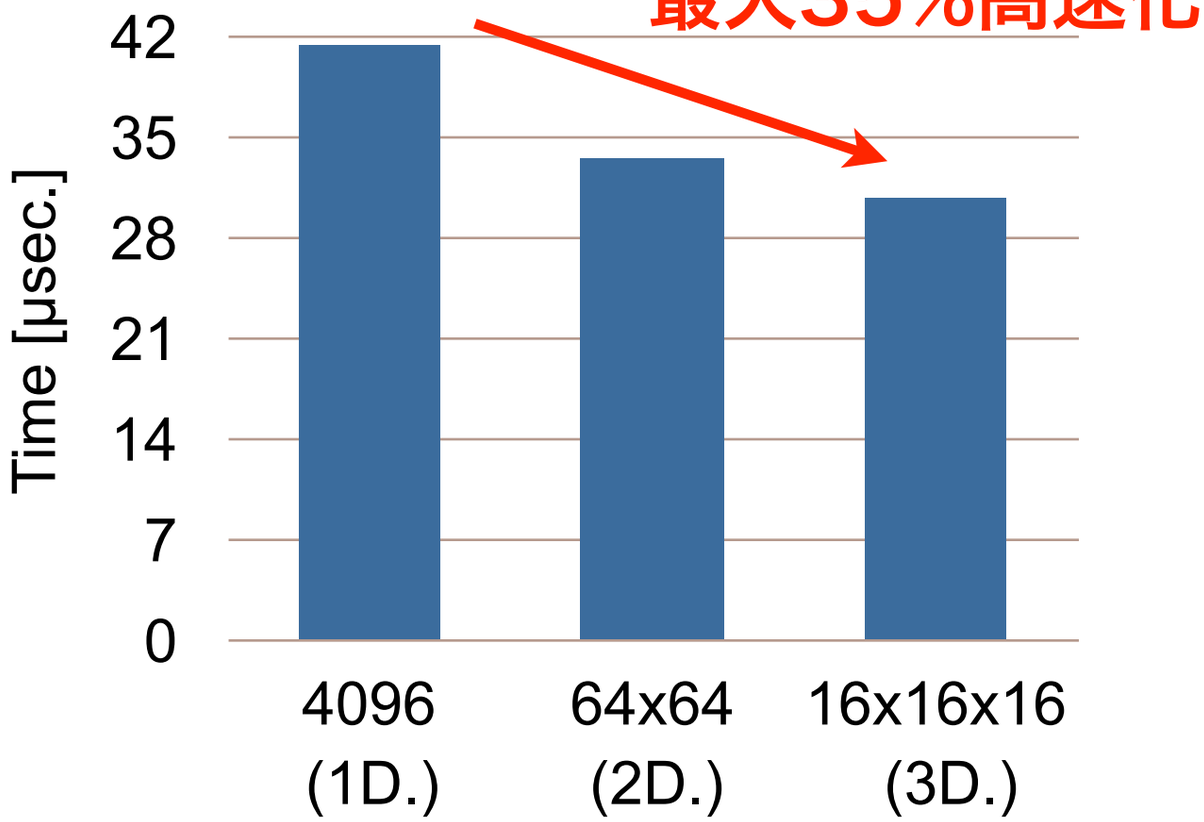
# 予備実験2

- 1, 2, 3次元ジョブでMPI_Allreduceの速度評価
- 1要素 x 3のみ

- 64ノード

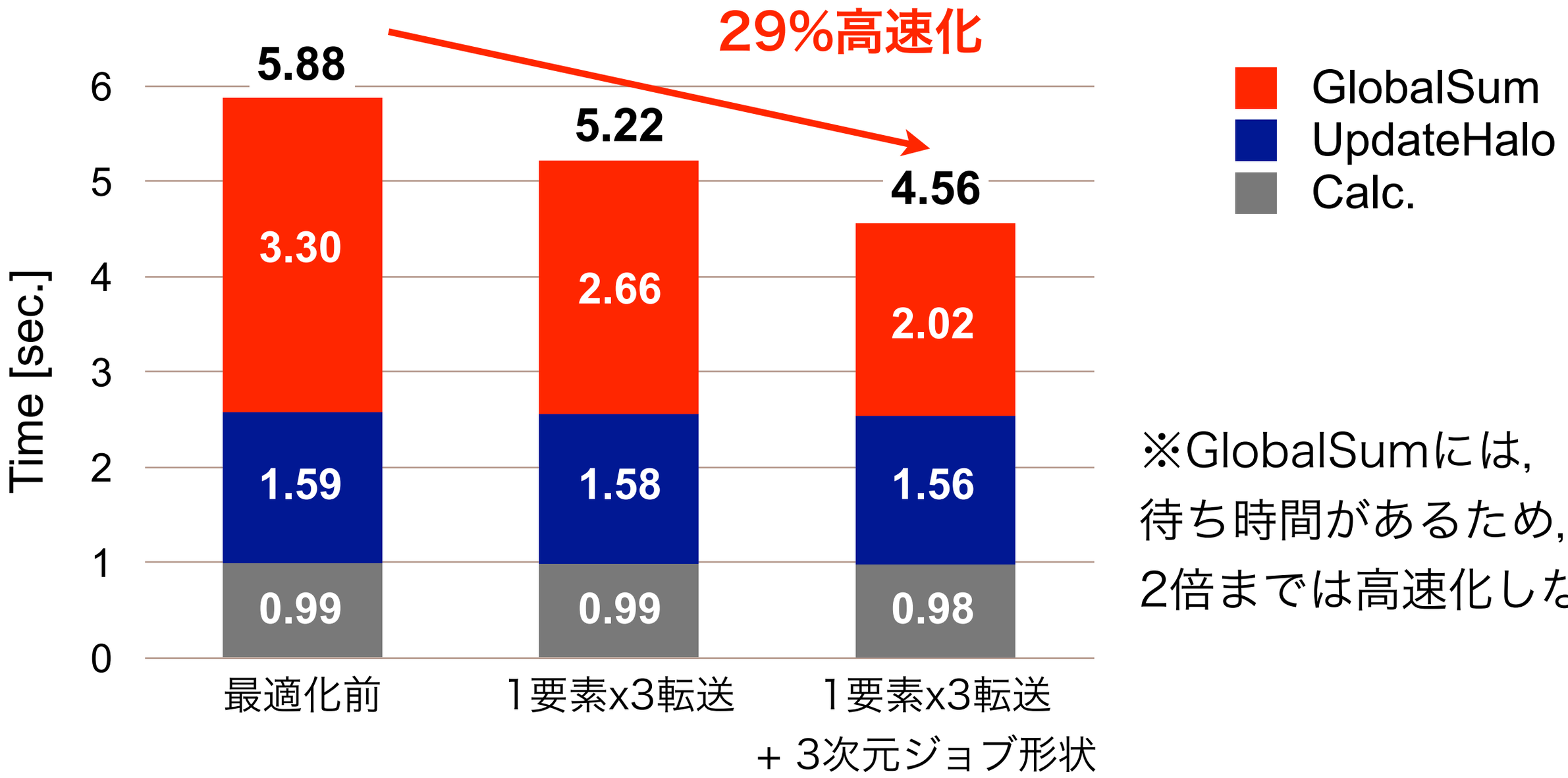



- 4096ノード



# GlobalSum最適化の結果

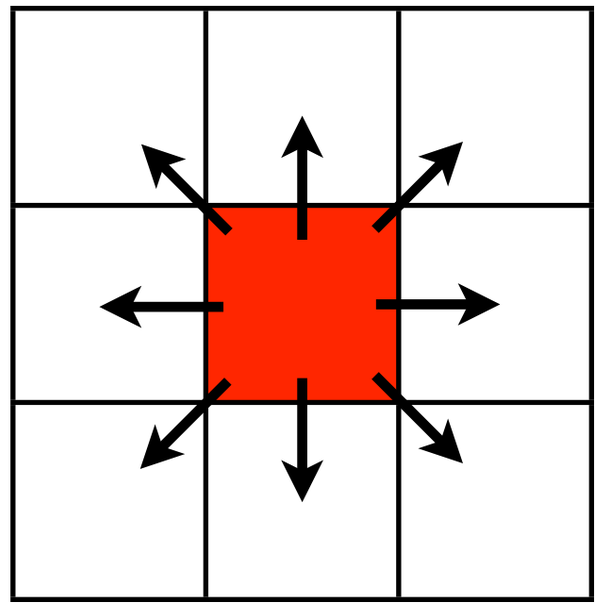
- 左の2つは7545ノード，右は19x20x20 = 7600ノードの結果

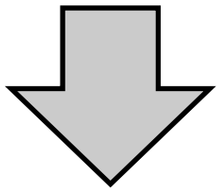



※GlobalSumには，
待ち時間があるため，
2倍までは高速化しない

# UpdateHalo（袖通信）の最適化

- 袖通信
  - 隣接ノード（2次元ステンシルなので、
    最大8ノード）との1対1通信 + pack/unpack
  - MPI_Isend/Irecv + Waitallで実装
  - 7545プロセスの場合は，300Byte程度の
    細粒度な通信が多発する
    - レイテンシを低く抑えることが重要





Tofuインターコネクト用の低レベルライブラリである
拡張RDMAインタフェースを用いる

# UpdateHalo（袖通信）の最適化

- 拡張RDMAインタフェース

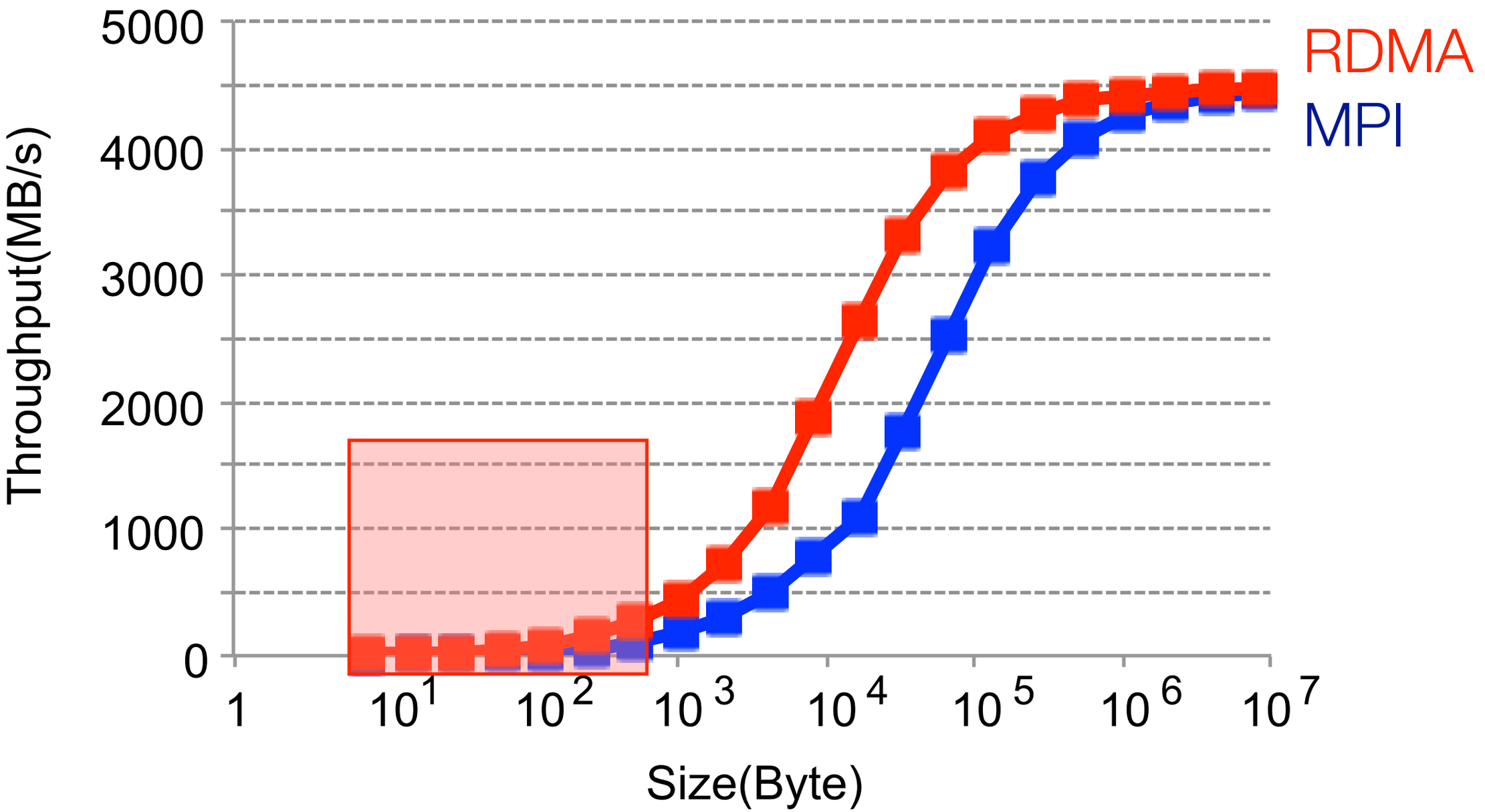
（ParallelNavi for MP10 V1.0より抜粋）

| 関数名 | 機能概要 |
|---|---|
| FJMPI_Rdma_init | 拡張RDMAインターフェースの初期化 |
| FJMPI_Rdma_finalize | 拡張RDMAインターフェースの終了処理 |
| FJMPI_Rdma_reg_mem | メモリの登録 |
| FJMPI_Rdma_get_remote_addr | リモートDMAアドレスの獲得 |
| FJMPI_Rdma_put | RDMA WRITE通信 |
| FJMPI_Rdma_get | RDMA READ通信 |
| FJMPI_Rdma_poll_cq | RDMA完了確認 |

C言語用のAPIしか用意されていないため，Fortran用のインタフェースを作成した

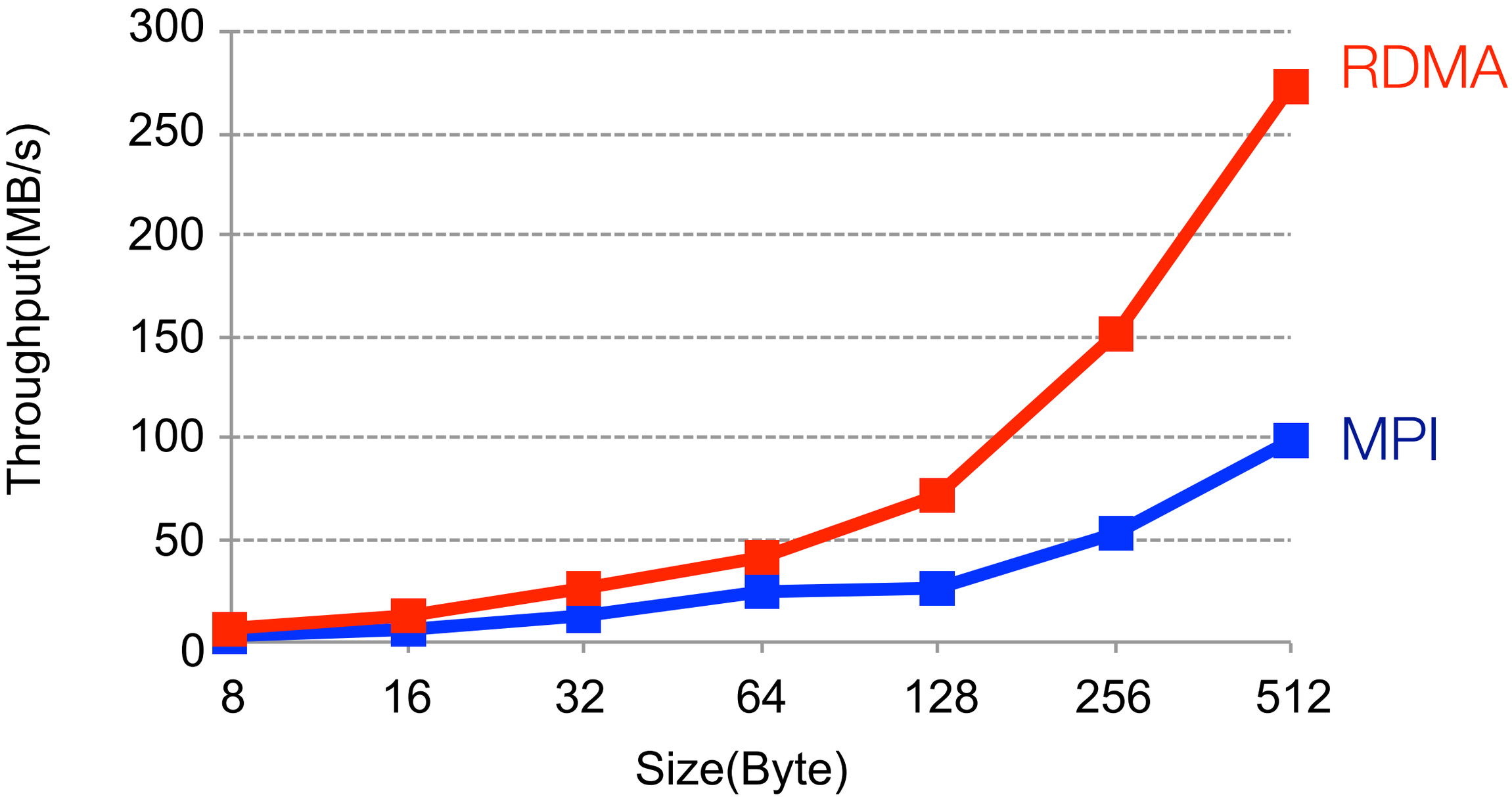
# 予備実験3（pingpong）

- RDMAとMPI_Isend/Irecvとの通信性能比較



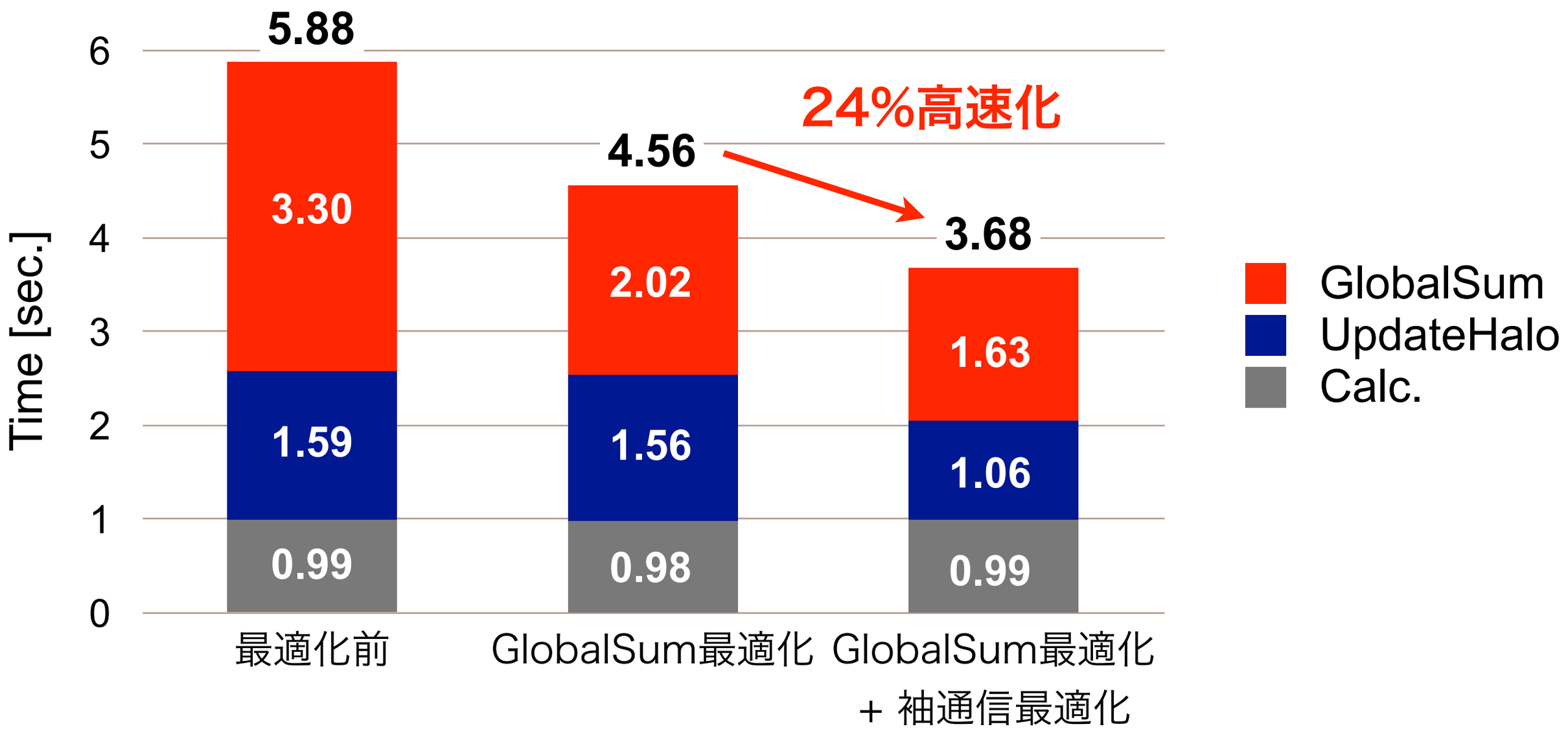
# 予備実験3（pingpong）

- RDMAとMPI_Isend/Irecvとの比較



# 袖通信最適化の結果

- 7600ノードの結果



# GlobalSumが早くなる理由

1. GlobalSum（集合通信）**+ Idle時間**

   `MPI_Allreduce`

2. UpdateHalo（袖通信）

   `MPI_Isend/Irecv`

3. 計算処理

UpdateHaloが短くなると，
Idle時間も短くなる

# 計算処理高速化

- **スレッド化**
- SIMD化
- セクタキャッシュの利用
- ループ最適化
  - ループアンローリング
  - ループ交換
  - ループ分割
- 最適化制御行の利用　など

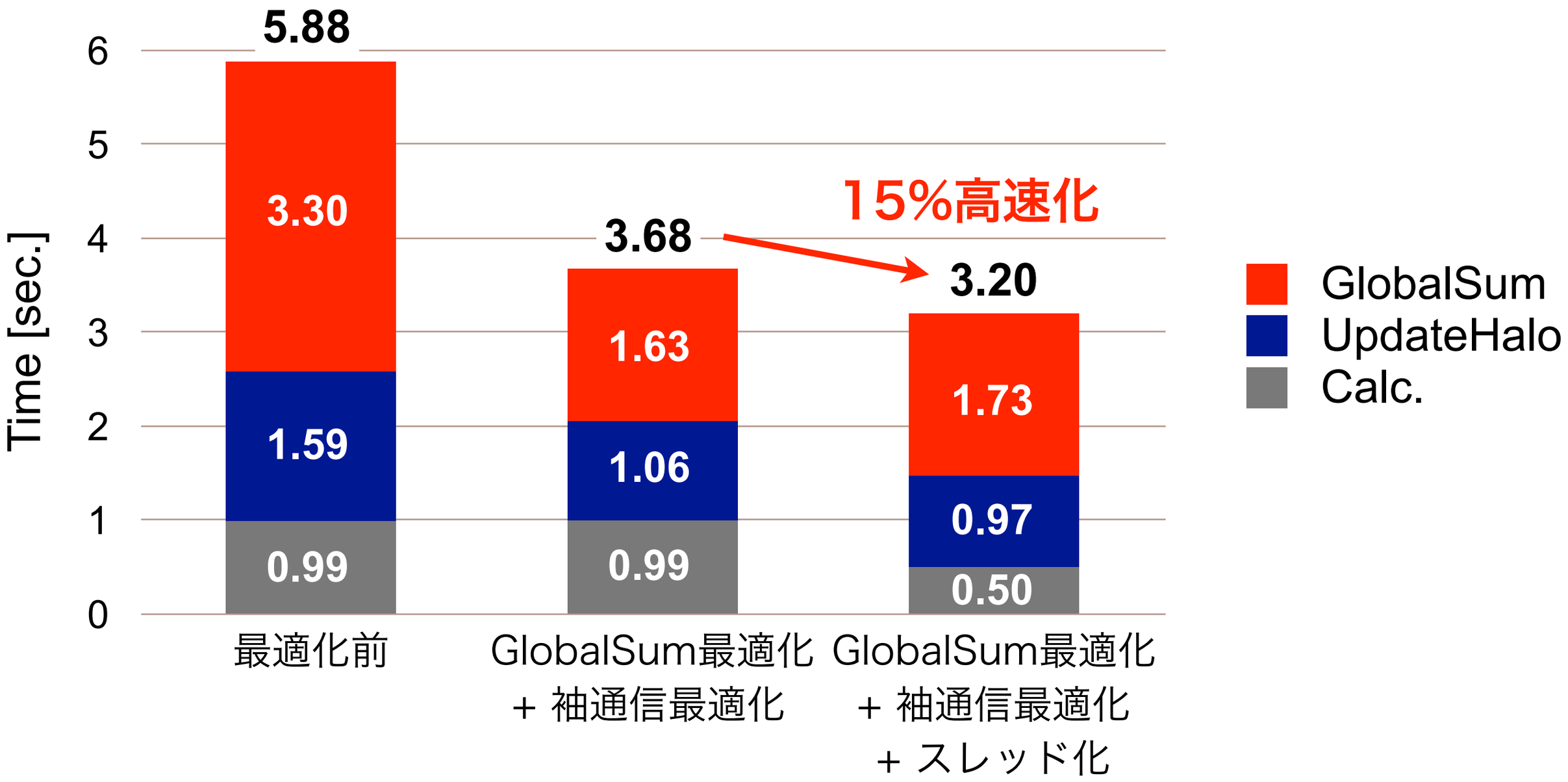
# OpenMPを利用したスレッド化

- 計算の約65%を占めている2重ループ文をスレッド化
- 残りの箇所はループ長が短いため，スレッド化の効果なし

```
!$OMP PARALLEL DO DEFAULT(SHARED) PRIVATE(tmp, j)
do i=1,n2
    tmp = 0.0
    do j=Mat%Ia(i),Mat%Ia(i+1)-1
        tmp = tmp + Mat%Mat(j)*X(Mat%Ja(j))
    enddo
    Y(i) = tmp
enddo
!$OMP END PARALLEL DO
```

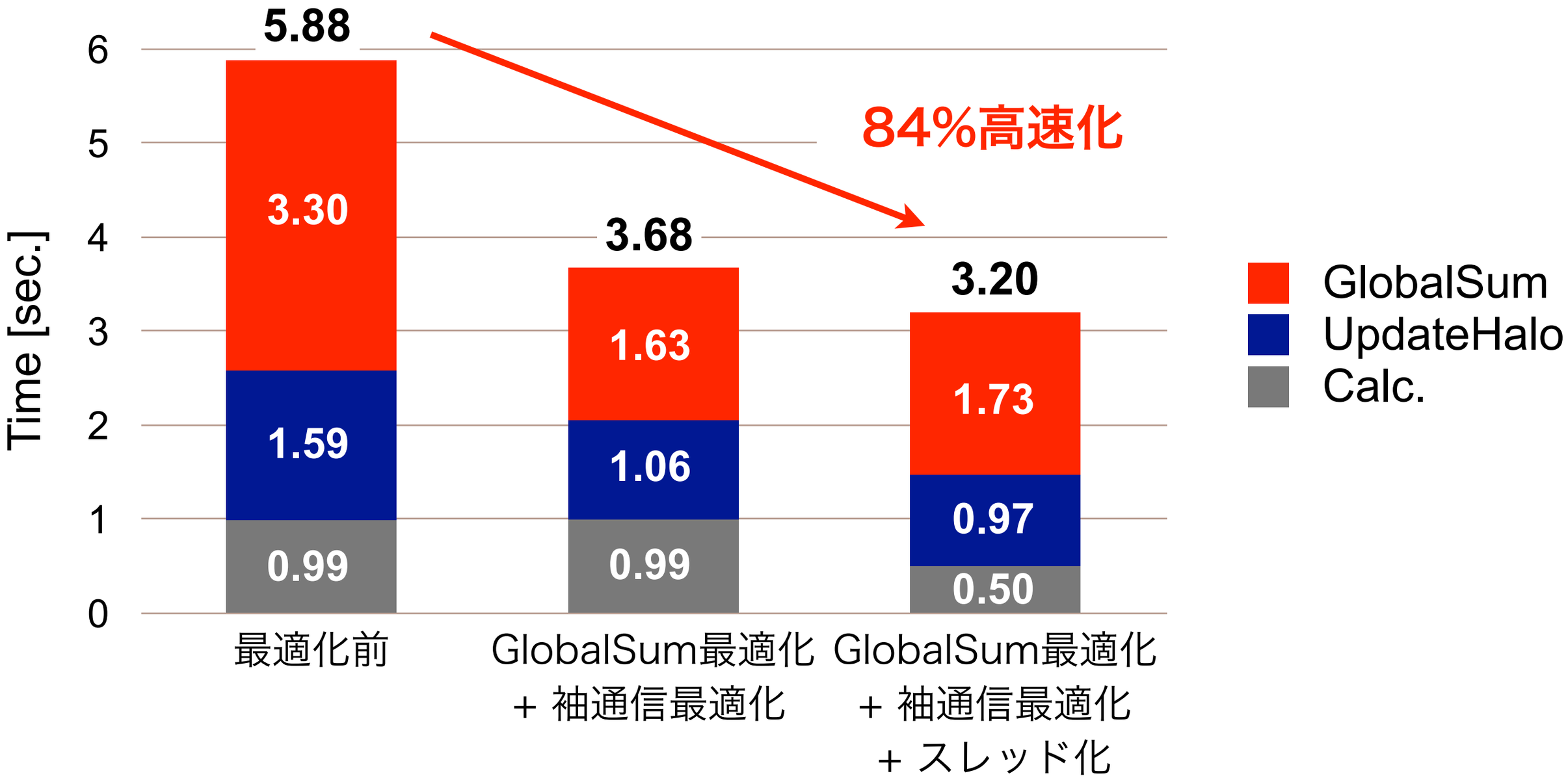
# スレッド化の結果

- 7600ノードの結果（8スレッド）



# スレッド化の結果

- 7600ノードの結果（8スレッド）

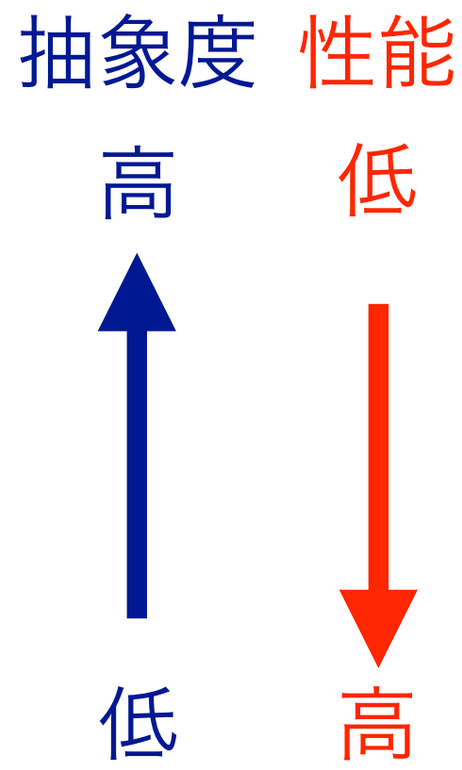


# エクサに向けたプログラミングモデル

- GlobalSum最適化（集団通信最適化）
- 袖通信最適化（1対1通信最適化）
- 計算最適化（スレッド化）

# エクサに向けたプログラミングモデル

- 袖通信最適化（1対1通信最適化）
  - レイテンシが小さいRDMAの利用が重要
  - 今回は京が提供するRDMA APIを利用した
  - 性能を保ちつつ，ポータビリティを上げるには？
- 案
  - XcalableMPのグローバルビュー機能（袖定義と反映）
  - XcalableMPやCoarray Fortranなどの片側通信機能
  - 片側通信ライブラリ（GASNet, ARMCI, ..）の利用
  - 【各社共通のRDMAインタフェースを定義する】
  - ハードウェア特有のAPIを用いる（今回の方法）



# まとめ

- 海洋シミュレーションのミニアプリであるCGPOP Miniappを京にポーティング
- 性能チューニング
  - GlobalSum最適化（集団通信最適化）
  - 袖通信最適化（1対1通信最適化）
  - 計算最適化（スレッド化）
- 7600ノード利用時に，84%の高速化を達成
- 今後のプログラミングモデルについて
  - RDMAに対するポータビリティが重要