内容に関する質問は
katagiri@cc.nagoya-u.ac.jp
まで

2017年9月11日(月)9:30～12:30
情報基盤センター4F　演習室

第5回 FX100システム利用型 MPI講習会(初級)(試行)

名古屋大学情報基盤センター　片桐孝洋

プログラム

- 9:30 - 10:00　受付
- 10:00 - 10:30　端末設定など（演習）
  - FX100システムへのログイン
  - FX100システムへのジョブの投入方法
- 10:30 - 12:00（演習）
  - 名古屋大学情報基盤センターの計算機および利用形態
  - FX100システムの計算機構成、利用方法
  - サンプルプログラムの実行
- 13:30　- 15:00　並列プログラミングの基本（座学）
  - 並列計算の基礎、性能評価指標、アムダールの法則
  - MPIインターフェース説明、集団通信関数（コレクティブ通信）
  - データ分散方式
  - 先進的並列化技法：
    - ピュアMPI実行、ハイブリッドMPI実行、NUMA最適化、など

プログラム

- 15:15 - 17:00 MPIプログラム並列化実習（演習）
  - 行列-行列積の並列アルゴリズム
  - 行列-行列積の並列化実習１（簡易並列化方式での演習）
  - 行列-行列積の並列化実習２（完全並列化方式での演習）
- 17:00 - 17:30 自由演習、および、スパコン利用相談会

名大情報基盤センターの
スパコンのご紹介

名大情報基盤センターのスパコン変遷

96 | | | | 00 | | | | | 05 | | | | 09 | | | | 13 | | 15 | |

Fujitsu M-1800 → Fujitsu GP7000F/90

メインフレーム系


Fujitsu VPP500 → Fujitsu VPP5000

ベクトル型スパコン


Fujitsu HPC2500

スカラー型スパコン

Fujitsu HX600, M9000 → Fujitsu CX400 → CX400

アプリケーションサーバ


Fujitsu FX1 → Fujitsu FX10 → FX100

スーパーコンピュータ



現システム

- 4～5年間隔でリプレース
- 2005年にベクトル型からスカラー型に転換
- 超並列型、クラスタ型、大規模共有メモリ型などで様々な計算需要に応える

中間レベルアップ方式の活用

- 導入計画

- 中間レベルアップの活用
  - プロセッサ開発ロードマップやエクサスパコン計画を考慮
  - 「利用したくなる」環境を提供し続ける
  - その時代に適した対消費電力性能により、エネルギーを有効活用する（エコな計算機システム）

システムの概要 (Phase 1)

複合現実大規模可視化システム
(2014年3月, HPCI補正予算)

ピーク性能 0.58 PF

ストレージ 7 PB

高性能コンピュータシステム
Phase 1 (2013年10月～)

ピーク性能 0.56 PF

ストレージ 4 PB

大規模SMP型
演算サーバ

ストレージ装置

Fujitsu PRIMEHPC
FX10

Fujitsu PRIMERGY
CX400

可視化装置

ストレージ装置

バーチャルリアリティ装置

システムの概要
(Phase 2 (最終フェーズ))

高性能コンピュータシステムPhase 2
(2015年9月～2019年3月（予定）)

ピーク性能 4.0 PF

ストレージ 11 PB

Fujitsu PRIMEHPC FX100へ更新

Fujitsu PRIMERGY CX400を更新

ストレージ装置を増設

複合現実大規模可視化システム
(2014年3月, HPCI補正予算)

ピーク性能 0.58 PF

ストレージ 7 PB

高性能コンピュータシステム
Phase 1 (2013年10月～)

ピーク性能 0.56 PF

ストレージ 4 PB

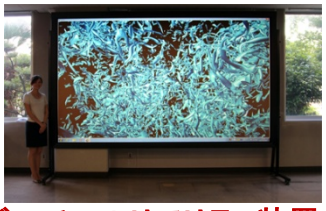
大規模SMP型
演算サーバ

ストレージ装置

Fujitsu PRIMEHPC
FX10

Fujitsu PRIMERGY
CX400

ストレージ装置



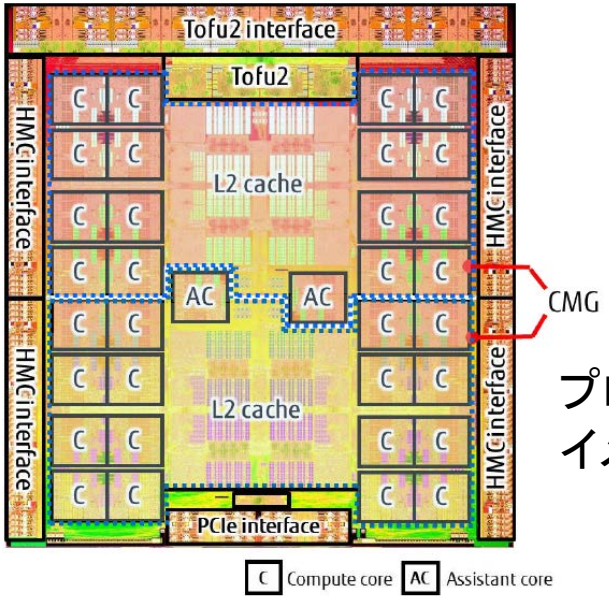
バーチャルリアリティ装置

可視化装置

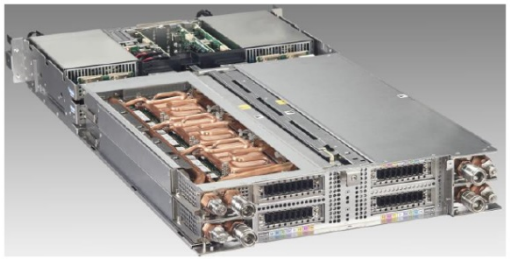
FX100システムのハードウェア概要

全体構成	計算ノード数	2,880
	理論演算性能	**3.2 PFLOPS**
	メモリ容量	90 TiB
CPU	プロセッサ	SPARC64 XIfx
	アーキテクチャ	SPARC9 + HPC-ACE
	コア数	32 + 2アシスタントコア
	共有L2キャッシュ	24 MB
	動作周波数	2.2 GHz
	理論演算性能	1,126 GFLOPS
ノード	アーキテクチャ	1 CPU / ノード
	メモリ容量	32 GB (HMC)
	メモリ理論帯域	240 GB/s (read) + 240 GB/s (write)
	インターコネクト	Tofu Interconnect 2
	インターコネクト 理論帯域	リンクあたり 12.5 GB/s x 2 (双方向)

ラックイメージ



プロセッサ
イメージ



メインユニット
イメージ

CX400システムのハードウェア概要

CX2 (CX400/270)

全体構成	計算ノード数	184
	理論演算性能	**279.9 TFLOPS**
	メモリ容量	23 TiB
プロセッサ	モデル	Xeon E5-2697v2 (2.7 GHz)
	マイクロアーキテクチャ	IvyBridge
	コア数	12
コプロセッサ	モデル	Xeon Phi 3120P (1.1GHz), 57コア
	アーキテクチャ	Intel MIC
ノード	構成	2 CPU + 1 コプロセッサ
	理論演算性能	1,521.6 GFLOPS
	メモリ容量	128 GiB
	メモリ理論帯域	119 GB/s

CX1 (CX400/2550) 中間レベルアップ

全体構成	計算ノード数	384
	理論演算性能	**447.2 TFLOPS**
	メモリ容量	48 TiB
プロセッサ	モデル	Xeon E5-2697v3 (2.6 GHz)
	マイクロアーキテクチャ	Haswell
	コア数	14
ノード	構成	2 CPU
	理論演算性能	1,164.8 GFLOPS
	メモリ容量	128 GiB
	メモリ理論帯域	136 GB/s

SGI UV2000のハードウェア構成

ハードウェア構成

機種名	SGI UV2000
プロセッサ	Intel Xeon E5-4650 (2.4 GHz, 8 コア) (Ivy Bridge)
総CPU数(総コア数)	160 CPU（160ソケット）(1,280 コア)
総演算性能	24 TFlop/s
総メモリ容量	20 TiB

ccNUMA (Cache Coherent Non-Uniform Memory Access)方式により、
大規模共有メモリを実現
【128GB／ソケット】

ITC

名古屋大学
NAGOYA UNIVERSITY

主な利用可能ソフトウェア(1/3)

- **FX100システム**
  - 富士通コンパイラ
  - MPI(富士通), BLAS, LAPACK, ScaLAPACK, SSL II（富士通）
  - 構造解析 LS-DYNA、計算化学 Gaussian、可視化AVS、OpenFOAM

「京」後継の超並列型

- 利用者開発ソフトウェア利用（超並列実行）
- ISVアプリ、フリーソフトの移植も推進中

ITC

名古屋大学
NAGOYA UNIVERSITY

主な利用可能ソフトウェア(2/3)

- **CX400システム**
  - 富士通Fortran/C/C++/ XPFortran，Intel Fortran /C/C++, **Python (2.7, 3.3)**
  - MPI(富士通,Intel), BLAS, LAPACK, ScaLAPACK, SSLII(富士通), C-SSL II, SSL II/MPI, MKL(Intel), NUMPAC, FFTW, HDF5, IPP(Intel)
  - LS-DYNA, ABAQUS, ADF, AMBER, GAMESS, Gaussian, Gromacs, LAMMPS, NAMD, HyperWorks, OpenFOAM, STAR-CCM+, Poynting, 富士通Technical Computing Suite 4 (HPC Portal).

使いやすいIntelクラスタ型

- 利用者開発ソフトウェア利用
- ISVアプリ、フリーソフトも充実
- 最新Xeon (Haswell)
- メニーコア(MIC, Xeon Phi (KNC))

ITC

名古屋大学 NAGOYA UNIVERSITY

主な利用可能ソフトウェア(3/3)

大規模共有メモリ型
- 計算結果を可視化するISVアプリが充実
- リモートデスクトップ利用も可能

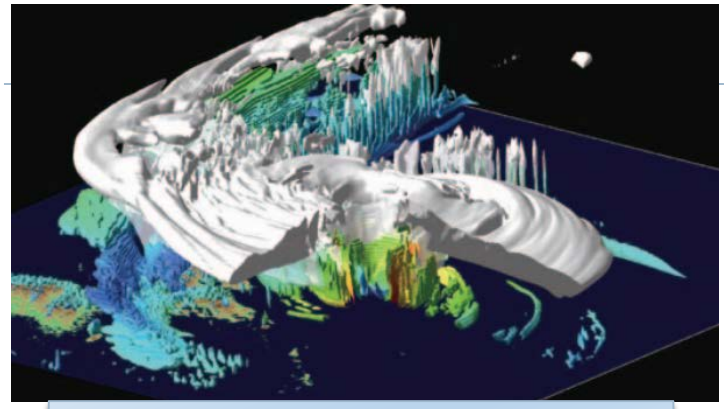
UV2000システム

- Intelコンパイラ, Intel Fortran/C/C++, Python (2.6, 3.5)
- MPT(SGI), MKL(Intel), PBS Professional, FFTW, HDF5, NetCDF, scikit-learn, **TensorFlow**
- AVS/Express Dev/PCE, EnSightHPC, FieldviewParallel, IDL, ENVI (SARscape), ParaView, 3DAVSplayer, POV-Ray, NICE DCV(SGI), ffmpeg, ffplay, LS-prepost, osgviewer, vmd (Visual Molecular Dynamics)

可視化システム

- AVS/Express Dev/PCE, EnshightHPC, 3DAVSPlayer, IDL, ENVI(SARscape), ParaView, osgviewer, EasyVR, POV-Ray, 3dsMAX, 3-matic, VMD, ffmpeg, ffplay, LS-prepost

大規模ストレージと高精細可視化装置



気象系シミュレーション可視化例

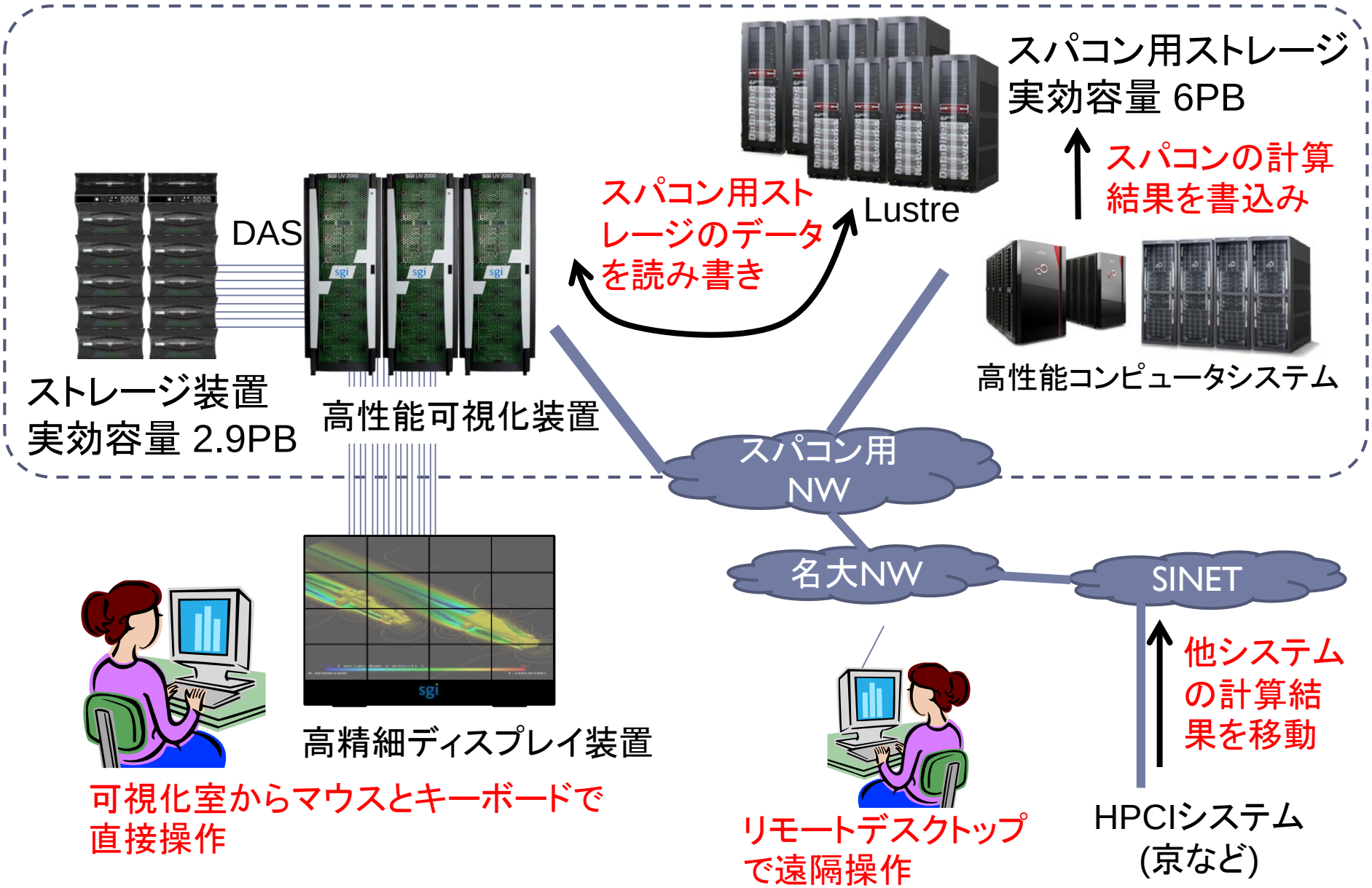
医用画像データ
可視化例

- **本センターの強み**
  - プラズマ流体、地球流動現象、生体分子、医療分野などに対する高効率な可視化技術の開発、可視化装置の構築
- **システム導入の目的**
  - 数値シミュレーション等による超大規模データから高精細な3次元可視化映像の生成を実現
  - シミュレーションデータと実世界の映像を融合することで、新しい知見を深めることを支援
- **特徴**
  - スパコンからデータを移動させることなく高度な後処理が可能
  - 可視化装置に大容量ストレージと高精細ディスプレイを直結させ、HPC利用に特化
- **新サービス開拓**
  - データサイエンスを支援するサービス
  - ビッグデータ、機械学習など、新しい計算需要に対応
  - 計算サービスと、大規模データ蓄積、機械学習用ツール、高性能ファイルシステム、および可視化の連結
  - 遠隔大規模データの可視化

ITC

名古屋大学
NAGOYA UNIVERSITY

名大ITCの計算機システム概要

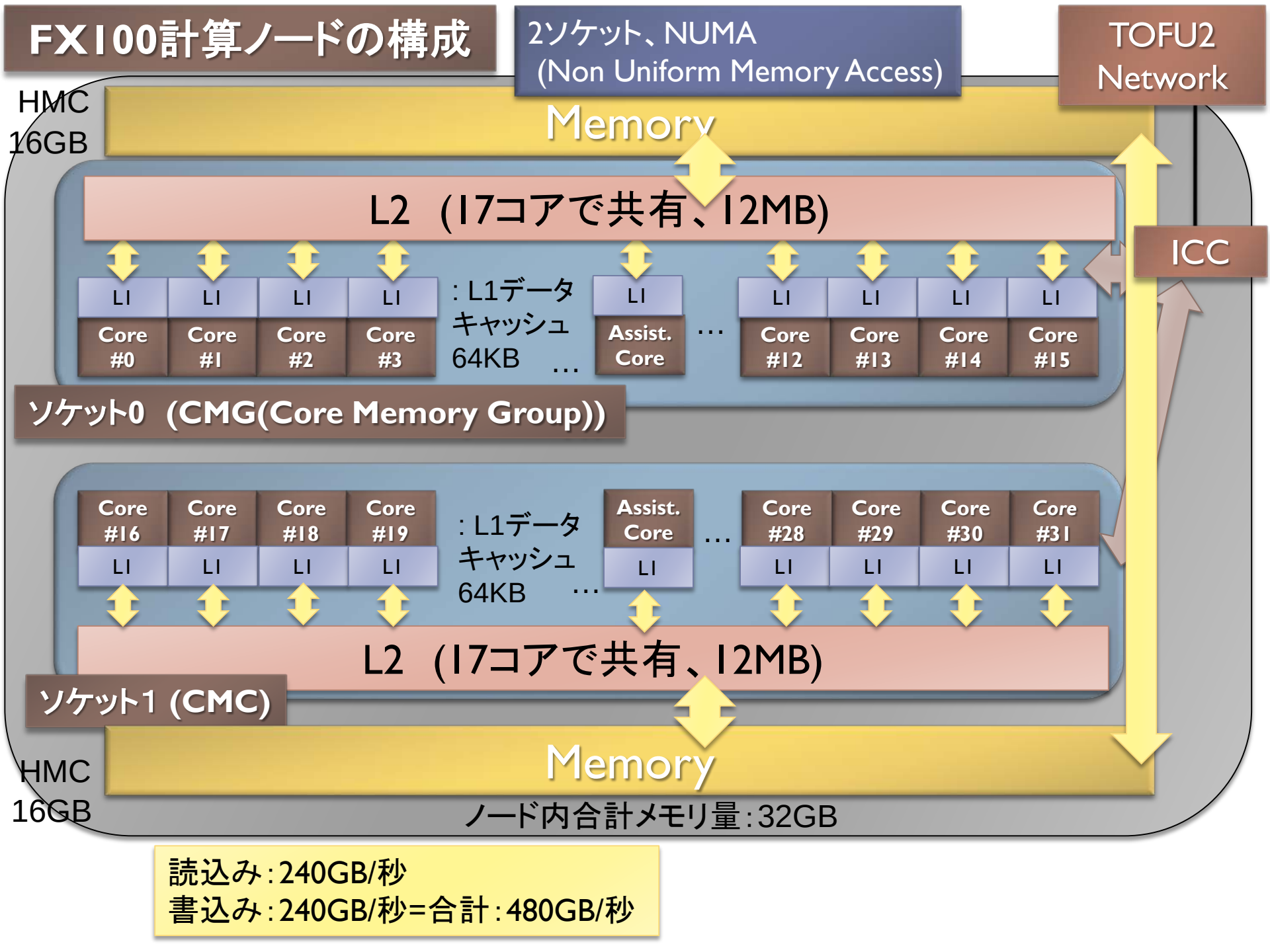


Fujitsu PRIMEHPC FX100 の計算機アーキテクチャ

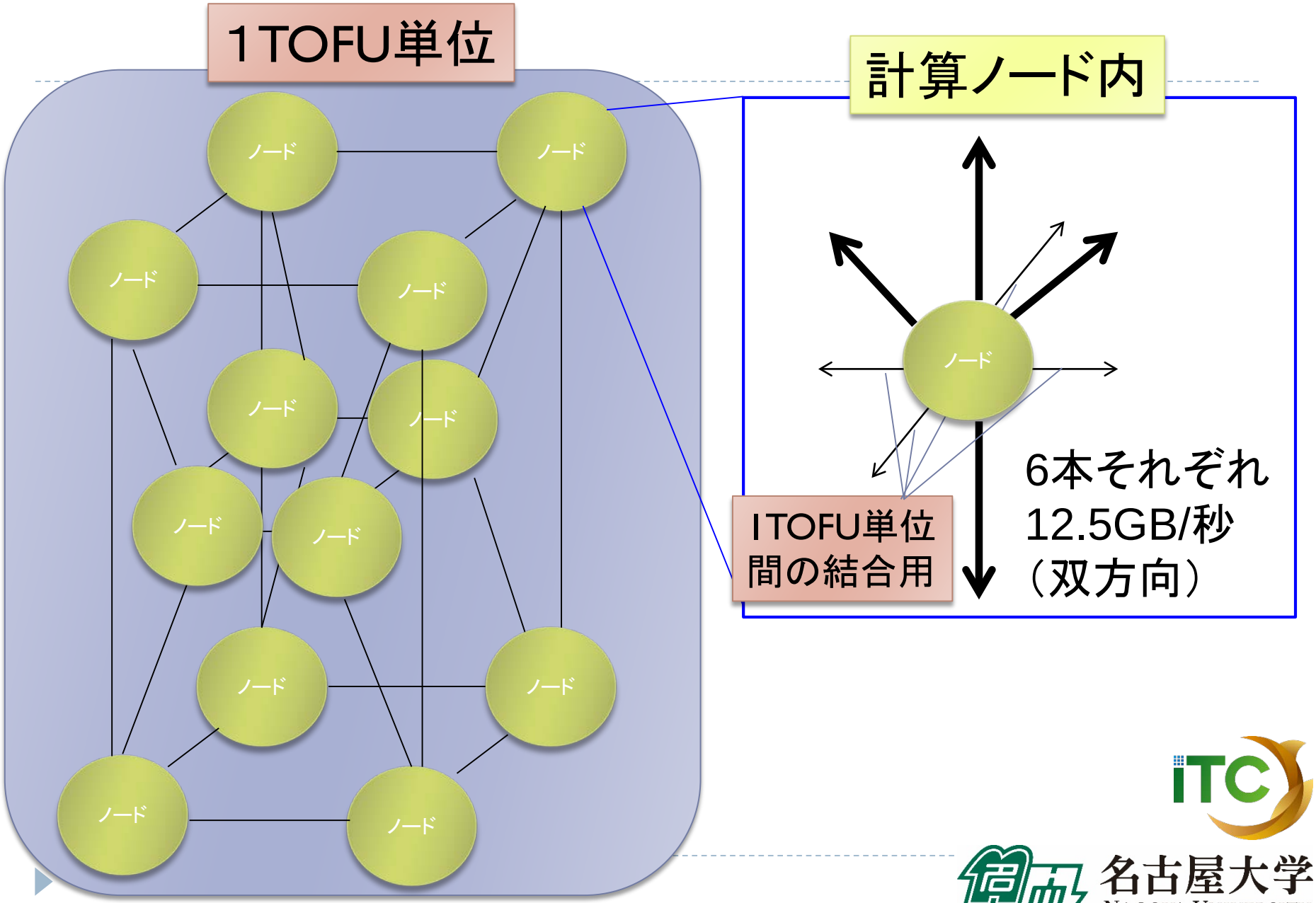
FX10とFX100のアーキテクチャ比較

	FX10	FX100
演算能力／ノード	倍精度／単精度： 236 GFLOPS	倍精度：1.011 TFLOPS 単精度：2.022 TFLOPS
演算コア数	16	32
アシスタントコア	なし	2
SIMD幅	128	256
SIMD命令	浮動小数点演算、連続ロード／ストア	右に加え、整数演算、ストライド＆間接ロード／ストア
L1Dキャッシュ／コア	32KB、2ウェイ	64KB、4ウェイ
L2キャッシュ／ノード	12MB	24MB
メモリバンド幅	85GB/秒	480GB/秒

出典：https://www.ssken.gr.jp/MAINSITE/event/2015/20151028-sci/lecture-04/SSKEN_sci2015_miyoshi_presentation.pdf


FX100計算ノードの構成
2ソケット、NUMA
(Non Uniform Memory Access)
TOFU2
Network
HMC
16GB
Memory
L2 (17コアで共有、12MB)
ICC
L1
Core
#0
Core
#1
Core
#2
Core
#3
: L1データ
キャッシュ
64KB
Assist.
Core
Core
#12
Core
#13
Core
#14
Core
#15
ソケット0 (CMG(Core Memory Group))
Core
#16
Core
#17
Core
#18
Core
#19
Core
#28
Core
#29
Core
#30
Core
#31
L2 (17コアで共有、12MB)
ソケット1 (CMC)
HMC
16GB
Memory
ノード内合計メモリ量:32GB
読込み:240GB/秒
書込み:240GB/秒=合計:480GB/秒

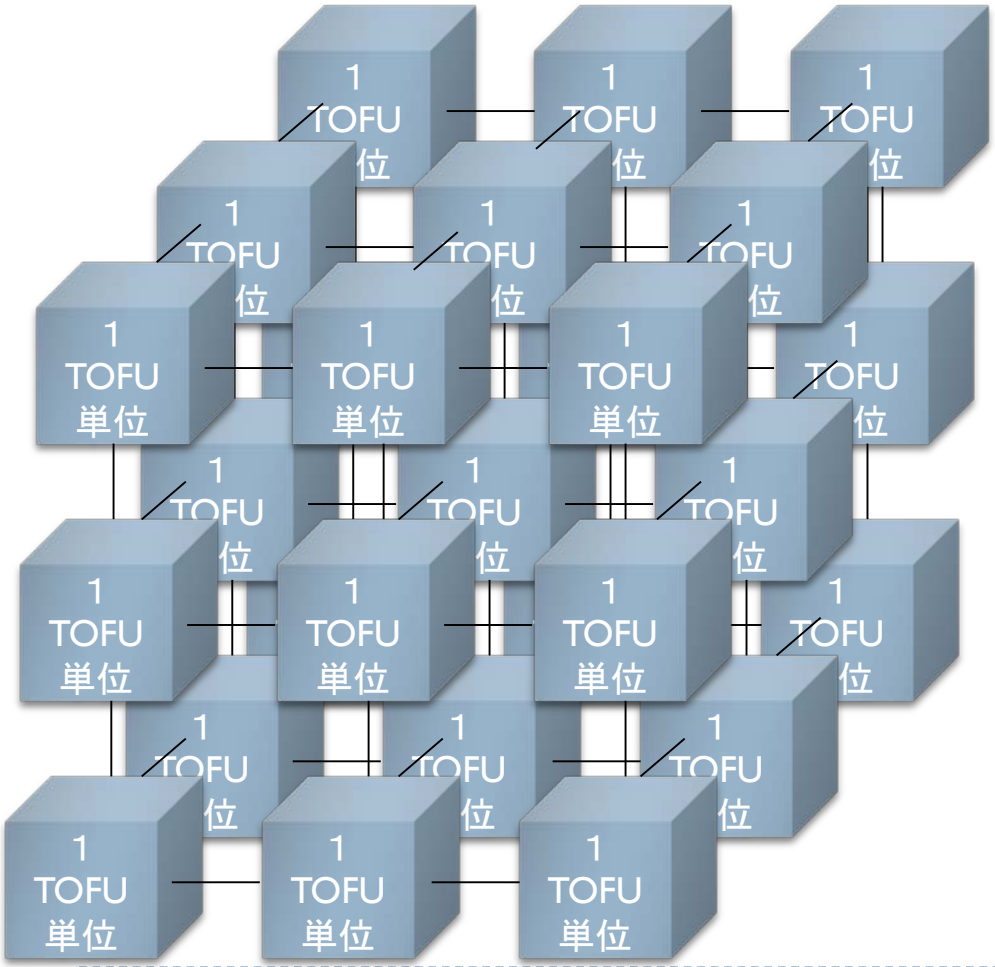
FX100の通信網（１TOFU単位）



名古屋大学
NAGOYA UNIVERSITY

FX100の通信網（１TOFU単位間の結合）

３次元接続



- ユーザから見ると、
  X軸、Y軸、Z軸について、
  奥の1TOFUと、手前の
  １TOFUは、繋がってみえます
（３次元トーラス接続）
- ただし物理結線では
  - X軸はトーラス
  - Y軸はメッシュ
  - Z軸はメッシュまたは、
    トーラス

になっています

課金体系

名大スパコンの課金体系 (1/5)

- 前払い定額制(プリペイド形式)
  - 利用すべき資源の料金を前払いして利用
  - 利用ポイントに変換して利用
- 単年度会計(本年4月1日～翌年3月31日)
  - 年度途中で申込み可能だが、利用終了は年度末
  - 年度末に余った利用ポイントは没収
- 一度の申込みで、全てのスパコンと可視化システムを利用可能
  - FX100、CX400、UV2000(可視化サーバ)

名大スパコンの課金体系 (2/5)

- 基本負担金（アカデミックユーザのみ）
  - 登録料1名につき10,000円、10,000ポイントの付加
- 追加負担金（アカデミックユーザのみ）
  - 100,000 円未満： 1円＝1ポイント
  - 100,000 円以上500,000 円未満：
    1 円当たり1.2 ポイント
  - 500,000 円以上1,000,000 円未満：
    1 円当たり1.5 ポイント
  - 1,000,000 円以上：
    1 円当たり2 ポイント

名大スパコンの課金体系 (3/5)

- **消費ポイント**
  - 計算課金

    利用ノード数（ソケット数）×経過時間[s] * 0.002
    - 基本負担金1万円(1万ポイント付加)
      - 1ノードを約1,388時間(約2ヶ月)利用可能
    - 追加負担金10万円(12万ポイント付加)
      - 2ノードを1年間利用可能、または、8ノードを3か月分利用可能
    - 追加負担金50万円(75万ポイント付加)
      - 12ノードを1年間利用可能、または、48ノードを3か月分利用可能
    - 追加負担金100万円(200万ポイント付加)
      - 32ノードを1年間利用可能、または、128ノードを3か月分利用可能
    - 参考
      - FX100の8ノード： 256コア (理論演算性能9.0 TFLOPS)
      - CX400/2550の8ノード： 224コア (理論演算性能9.3 TFLOPS)

名大スパコンの課金体系 (4/5)

- **消費ポイント**
  - ファイル課金（非データ蓄積ユーザ）
    - 300GB 以下の場合：徴収しない
    - ファイルの使用容量が300GB を超えた場合：
      超えた容量について、1GB につき
      1日当たり 0.01 ポイント
    - 例) ITB 利用:　700GB課金 ⇒ 7ポイント/日
      ⇒ 217円/月、2604円/年
    - **※60TBを超える場合は、全体容量を考慮して、削除依頼をさせていただくことがあります。**
    - **※60TBを超える容量が必要な場合は、事前にご相談ください。**

名大スパコンの課金体系 (5/5)
（平成29年度改訂）

- **消費ポイント**
  - ファイル課金（データ蓄積ユーザ）
    - 300GB 以下の場合：徴収しない
    - ファイルの使用容量が300GB を超えた場合：
      超えた容量について、1GB につき
      1日当たり 0.024 ポイント
    - 例) ITB 利用:　700GB課金 ⇒ 16.8ポイント/日
      ⇒ 521円/月、6250円/年
    - **※60TBを超える場合は、全体容量を考慮して、削除依頼をさせていただくことがあります。**
    - **※60TBを超える容量が必要な場合は、事前にご相談ください。**

ITC

名古屋大学
NAGOYA UNIVERSITY

大規模ファイルシステム課金制度（平成29年度改訂）

- アカデミックユーザのみ
- 1口：
  1パーティション(64TB)につき
  年額　540,000 円
- 2口以上で、
  支払金額＊10%
  のポイント付加の特典付き

お試し利用、リテラシー利用

- トライアルユース
  - ソフトウェアの動作確認などを、無料で行える制度です。
    お1人様1回限りで申請できます。
  - 企業においては、同一の課で1回のみです。
  - アカデミックユーザ（無審査）、企業ユーザ（書類審査）
  - 10,000ポイント付加
  - 有効期限1ヶ月
- リテラシー利用（アカデミックユーザのみ）
  - 学部・大学院の講義や演習で利用いただける制度です。
  - 利用登録25 件につき10,000 円、50,000ポイント付与
  - 有効期限：上限6ヶ月（講義・演習実施期間に依存）

ITC

名古屋大学
NAGOYA UNIVERSITY

産業利用（平成29年度改訂）

- 公開型
  - 10アカウントまで15万円
  - アカデミック利用の料金の2.5倍（15万円当たり60,000 ポイント）
  - 4月～6月の利用のみ、1口当たり15,000ポイントを付与の特典付
  - 利用報告の義務有り（最大2年間延長可能）
- 非公開型
  - 10アカウントまで25万円
  - アカデミック利用の料金の5倍（25万円当たり50,000 ポイント）
- 申込み金額に応じたポイント優遇はございません。
- 詳しくは、産業利用のパンフレットをご参照ください。

講習会

- 並列化など、各種講習会を実施しております
  - http://www.icts.nagoya-u.ac.jp/ja/center/service/course.html
- 平成28年11月から、FX100システムを利用した演習付きの「MPI講習会（初級）」が試行実施されています。
  - 日程が変更されることがあります。詳しくは、以下のスーパーコンピュータシステムのページをご参照ください。
  - 参加費は無料です
  - 企業の方の参加も可能です
  - http://www.icts.nagoya-u.ac.jp/ja/sc/
- 平成29年度関連講習会（予定）

- ~~第4回 6月15日（木）MPI講習会（初級）~~
- 第5回 9月11日（月）MPI講習会（初級）
- 第6回 9月25日（月）ライブラリ利用講習会（新設）
- 第7回 12月7日（木）MPI講習会（初級）
- 第8回 3月12日（月）ライブラリ利用講習会

名古屋大学 NAGOYA UNIVERSITY

コンサルティング

- 並列化、利用高度化、ISVアプリの利用方法などに関するコンサルティングを行っています。
- 本センター教職員や学内外の専門家で構成される専門分野相談員によるコンサルティング（面談）ができます。
  - Web受付 Q&A SYSTEM
    - 各種ご質問、ご相談等は下記Webサイトからお問合せください。
    - https://qa.icts.nagoya-u.ac.jp/
  - 面談相談
    - 実際に画面を見ながらなど、電話やメールでは伝えづらいご質問やご相談には面談でも受け付けています。
    - 事前にお約束の上、本センター3階図書室内のIT相談コーナーにお越しください。または、相談員が訪問させていただくことも可能です。
      - 連絡先：052-789-4366（IT相談コーナー直通）、または、上記のQ&A SYSTEM

スパコン利用の方法

スパコン利用の流れ

1. 申請書の提出
2. アカウント発行
3. 専用ポータル
   (https://portal.cc.nagoya-u.ac.jp/)
   にログインし、公開鍵を登録
4. ssh で、センタースパコンへログイン
5. プログラミング、コンパイル、実行
6. 年度末に利用報告書の提出

ログイン先のアドレス

- FX100システム

  fx.cc.nagoya-u.ac.jp

- CX400システム

  cx.cc.nagoya-u.ac.jp

- UV2000システム

  uvf.cc.nagoya-u.ac.jp

利用ファイルシステム

- 共有ファイルシステム
- /home ：約0.5PB、ホーム領域
  （個人は500GBまで）
- /center ：約1.0PB、ISV、OSS (ソフトウェア) 領域
- /large ：約1.5PB、データ領域
- /large2 ：約3.0PB、データ領域
- /large は10TB 以内、/large2 は50TB以内でのご利用を想定しております。これ以上を利用される方は、事前にご連絡ください。
- 料金改定後、データ蓄積ユーザと判定されると、値上げ料金となります
- ファイル・ステージング無し

名大情報基盤センタースパコンのバッチキュー構成 (1/3)

- FX100システム

リソースグループ名	最大ノード数	最大CPUコア数	最大経過時間（標準値）	最大経過時間（制限値）	最大メモリ容量 (*)	割当方法 (Tofu)	割当方法 (離散)	備考
fx-interactive	4	128	1時間	24時間	28GiB x 4	不可	可	会話型バッチ
fx-debug	32	1,024	1時間	1時間	28GiB x 32	不可	可	デバック用
fx-small	16	512	24時間	168 時間	28GiB x 16	不可	可	
fx-middle	96	3,072	24時間	72 時間	28GiB x 96	可	可	
fx-large	192	6,144	24時間	72 時間	28GiB x 192	可	可	
fx-xlarge	864	27,648	24時間	24時間	28GiB x 864	可	可	
fx-special	2,592	82,944	unlimited	unlimited	28GiB x 2,592	可	可	事前予約制

* ユーザープログラムが使用可能な最大メモリ容量はノードあたり28GiBです．スレッドのスタック領域も含みます．

名古屋大学
NAGOYA UNIVERSITY

名大情報基盤センタースパコンのバッチキュー構成 (2/3)

- CX400/2550システム

リソースグループ名	最大ノード数	最大CPUコア数	最大経過時間（標準値）	最大経過時間（制限値）	最大メモリ容量 (*)	備考
cx-debug	4	112	1時間	1時間	112GiB x 4	デバック用
cx-share	1(共有)	14	24時間	168 時間	56GiB × 1	ノード共有
cx-small	8	224	24時間	168 時間	112GiB x 8	
cx-middle	32	896	24時間	72 時間	112GiB x 32	
cx-large	128	3,584	24時間	72 時間	112GiB x 128	
cx-special	384	10,752	unlimited	unlimited	112GiB x 384	事前予約制

* ユーザープログラムが使用可能な最大メモリ容量はノードあたり112GiBです.

名大情報基盤センタースパコンのバッチキュー構成 (3/3)

- CX400/270システム（Xeon Phiノード）

リソースグループ名	最大ノード数	最大CPUコア数	最大Phi数	最大経過時間（標準値）	最大経過時間（制限値）	最大メモリ容量 (*)	備考
cx2-debug	4	96	1 x 4	1時間	1時間	112GiB x 4	デバック用
cx2-single	1	24	1 x 1	24時間	336時間	112GiB x 1	
cx2-small	8	192	1 x 8	24時間	72時間	112GiB x 8	
cx2-middle	32	768	1 x 32	24時間	72時間	112GiB x 32	
cx2-special	150	3,600	1 x 150	unlimited	unlimited	112GiB x 150	事前予約制

* ユーザープログラムが使用可能な最大メモリ容量はノードあたり112GiBです.

名大情報基盤センタースパコンのバッチキュー構成 (3/3)

- UV2000システム

リソースグループ名	並列数(標準値)	並列数(制限値)	メモリ容量(標準値)	メモリ容量(制限値)	最大経過時間（標準値）	最大経過時間（制限値）	
uv-middle	64	128	0.9 TiB	1.8 TiB	24 時間	168 時間	
uv-large	256	512	3.6 TiB	7.2 TiB	12 時間	168 時間	

ITC
名古屋大学
NAGOYA UNIVERSITY

テストプログラム起動

ＵＮＩＸ備忘録

- Emacsの起動： emacs <編集ファイル名>
  - ^x ^s （^はcontrol）：テキストの保存
  - ^x ^c ： 終了
    （ ^z で終了すると、スパコンの負荷が上がる。絶対にしないこと。）
  - ^g ： 訳がわからなくなったとき。
  - ^k ：カーソルより行末まで消す。
    消した行は、一時的に記憶される。
  - ^y ：^kで消した行を、現在のカーソルの場所にコピーする。
  - ^s 文字列： 文字列の箇所まで移動する。
  - ^M x goto-line： 指定した行まで移動する。
    (^MはESCキーを押す)

ＵＮＩＸ備忘録

- rm ファイル名 :　ファイル名のファイルを消す。
  - rm　*~ : test.c~ などの、~がついたバックアップファイルを消す。
- ls : 現在いるフォルダの中身を見る。
- cd　フォルダ名 :　フォルダに移動する。
  - cd　.. : 一つ上のフォルダに移動。
  - cd　~ : ホームディレクトリに行く。訳がわからなくなったとき。
- cat　ファイル名 :　ファイル名の中身を見る
- make : 実行ファイルを作る
  （Makefile があるところでしか実行できない）
  - make　clean : 実行ファイルを消す。
    （clean がMakefileで定義されていないと実行できない）

サンプルプログラムの実行

初めての並列プログラムの実行

サンプルプログラム名

- C言語版・Fortran90版共通ファイル：

  Samples-fx100.tar

- tarで展開後、C言語とFortran90言語のディレクトリが作られる
  - C/ ：C言語用
  - F/ ：Fortran90言語用
- 上記のファイルが置いてある場所

  /center/a49904a

並列版Helloプログラムをコンパイルしよう (1/2)

1. /center/a49904a にある Samples-fx100.tar を
   自分のディレクトリにコピーする
   $ cp /center/a49904a/Samples-fx100.tar ./
2. Samples-fx.tar を展開する
   $ tar xvf Samples-fx100.tar
3. Samples フォルダに入る
   $ cd Samples
4. C言語 : $ cd C
   Fortran90言語 : $ cd F
5. Hello フォルダに入る
   $ cd Hello

並列版Helloプログラムをコンパイルしよう(2/2)

6. ピュアMPI用のMakefileをコピーする
   $ cp Makefile_pure Makefile
7. make する
   $ make
8. 実行ファイル(hello)ができていることを確認する
   $ ls

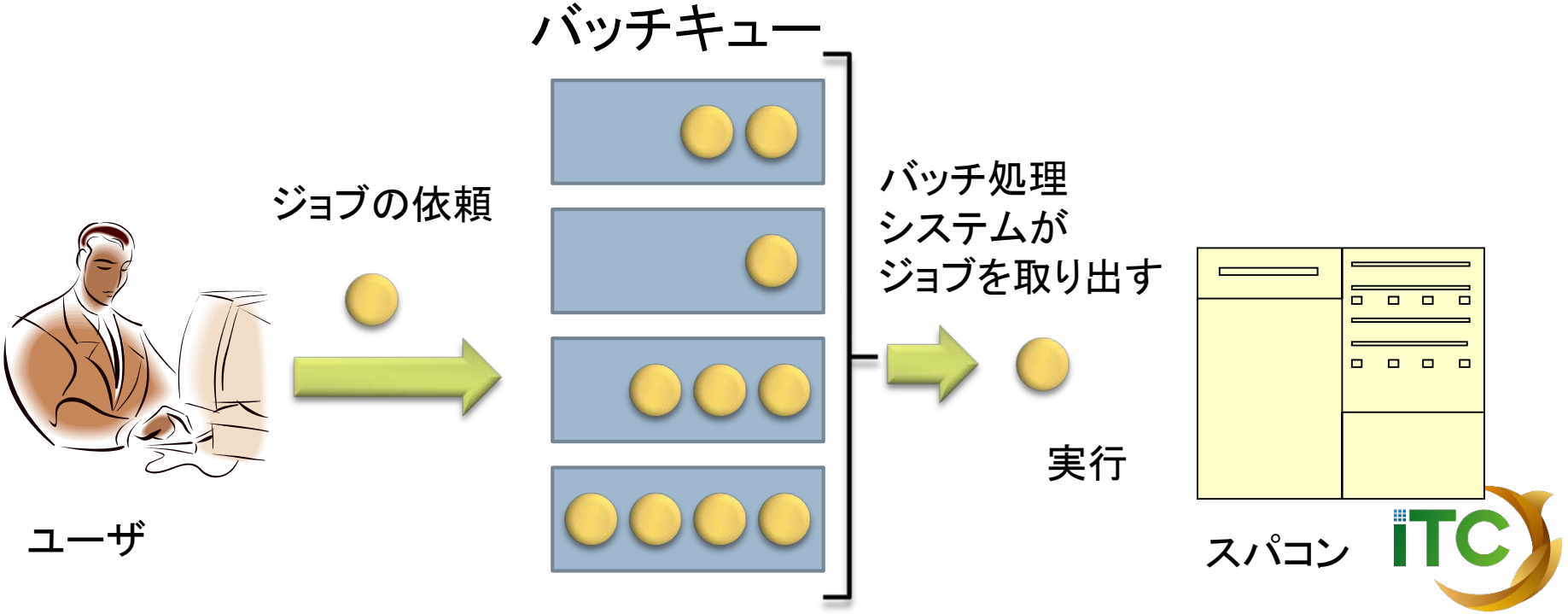
バッチ処理とジョブの投入

FX100のジョブ実行形態の例

- 以下の2通りがあります
- インタラクティブジョブ実行
  - PCでの実行のように、コマンドを入力して実行する方法
  - スパコン環境では、あまり一般的でない
  - デバック用、大規模実行はできない
  - 名大FX100では、以下に限定（名大基盤センターの運用方針）
    - **最大4ノード（128コア）（標準1時間まで、最大で24時間）**
- バッチジョブ実行
  - バッチジョブシステムに処理を依頼して実行する方法
  - スパコン環境で一般的
  - 大規模実行用
  - 名大FX100では：
    - **通常サービス： 最大864ノード（27,648コア）（24時間まで）**
    - **申込み制： 2,592ノード（82,944コア）（実行時間制限無し）**

バッチ処理とは

- スパコン環境では、インタラクティブ実行（コマンドラインで実行すること）は提供されていないことがあります。特に、大規模並列実行ができないようになっています。
- ジョブはバッチ処理で実行します。



コンパイラの種類とインタラクティブ実行およびバッチ実行の例（FX100）

- インタラクティブ実行、およびバッチ実行で、利用するコンパイラ（C言語、C＋＋言語、Fortran90言語）の種類が違います
- インタラクティブ実行では
  - オウンコンパイラ（そのノードで実行する実行ファイルを生成するコンパイラ）を使います
- バッチ実行では
  - クロスコンパイラ（そのノードでは実行できないが、バッチ実行する時のノードで実行できる実行ファイルを生成するコンパイラ）を使います
- それぞれの形式（富士通社の例）
  - オウンコンパイラ：＜コンパイラの種類名＞
  - クロスコンパイラ：＜コンパイラの種類名＞px
  - 例）富士通Fortran90コンパイラ
    - オウンコンパイラ：frt
    - クロスコンパイラ：frtpx

バッチキューの設定のしかた

- バッチ処理は、富士通社のバッチシステム(PJM)で管理されています。
- 以下、主要コマンドを説明します。
  - ジョブの投入：
    pjsub <ジョブスクリプトファイル名>
  - 自分が投入したジョブの状況確認： pjstat
  - 投入ジョブの削除： pjdel <ジョブID>
  - バッチキューの状態を見る： pjstat --rsc
  - 過去の投入履歴を見る： pjstat --history
  - システム制限値を見る： pjstat --limit

インタラクティブ実行のやり方の例

- コマンドラインで以下を入力
  - 1ノード実行用

    $ pjsub --interact

  - 4ノード実行用

    $ pjsub --interact -L “node=4”

pjstat --rsc の実行画面例

```
$ pjstat --rsc

RSCUNIT                 RSCUNIT_SIZE   RSCGRP                          RSCGRP_SIZE
fx[ENABLE,START]             4x7x9     fx-interactive[ENABLE,START]       8x3x2
fx[ENABLE,START]             4x7x9     fx-debug[ENABLE,START]             8x3x2
fx[ENABLE,START]             4x7x9     fx-small[ENABLE,START]             8x3x16
fx[ENABLE,START]             4x7x9     fx-middle[ENABLE,START]            2592
fx[ENABLE,START]             4x7x9     fx-large[ENABLE,START]             2592
fx[ENABLE,START]             4x7x9     fx-xlarge[ENABLE,START]            2592
fx[ENABLE,START]             4x7x9     fx-special[ENABLE,START]           2592
```

使える
キュー名
(リソース
グループ)

現在使えるか
ENABLE: キューに
ジョブ投入可能
START: ジョブが流れて
いる

ノードの
物理構成情報

pjstat の実行画面例

```
$ pjstat
 ACCEPT QUEUED  STGIN  READY RUNING RUNOUT STGOUT   HOLD  ERROR   TOTAL
      0      0      0      0      0      2      0      0      0      0
s     0      0      0      0      0      2      0      0      0      0
JOB_ID JOB_NAME   MD   ST    USER   START_DATE     ELAPSE_LIM  NODE_REQUIRE CORE V_MEM
696    run_4.sh   NM   RUN   user01 07/20 15:26:35 0000:10:00          4            -    -
697    run_4.sh   NM   RUN   user01 07/20 15:26:36 0000:10:00          4            -    -
```

表 3-14 FX ジョブ情報の表示項目

項目	説明
JOB_ID	ジョブ ID
JOB_NAME	ジョブ名
MD	ジョブモード(normal,step)
ST	ジョブの現在の状態
USER	ユーザー名
RSCGRP	リソースグループ名 (-v --pattern=1 指定時のみ)
START_DATE	ジョブが実行前の場合は開始予測時刻("()"で表示)、実行中および実行後の場合は実際に実行を開始した時刻。 実行開始時刻を指定して投入したジョブが実行を開始するまでの間、時刻の後ろに「@」が出力される。 バックフィルが適用されたジョブは、時刻の後ろに「<」が出力される。
ELAPSE_LIM	ジョブの経過時間(実行中でないジョブは"--:--:--)
NODE_REQUIRE	ジョブのノード数とノード形状(nnnn:XXxYxZ)

JOBスクリプトサンプルの説明（ピュアＭＰＩ）
(hello-pure.bash, C言語、Fortran言語共通)

```
#!/bin/bash
#PJM -L “rscgrp=fx-debug"
#PJM -L "node=12"
#PJM --mpi "proc=384"
#PJM -L "elapse=1:00"
mpirun ./hello
```

リソースグループ名
：fx-debug

利用ノード数

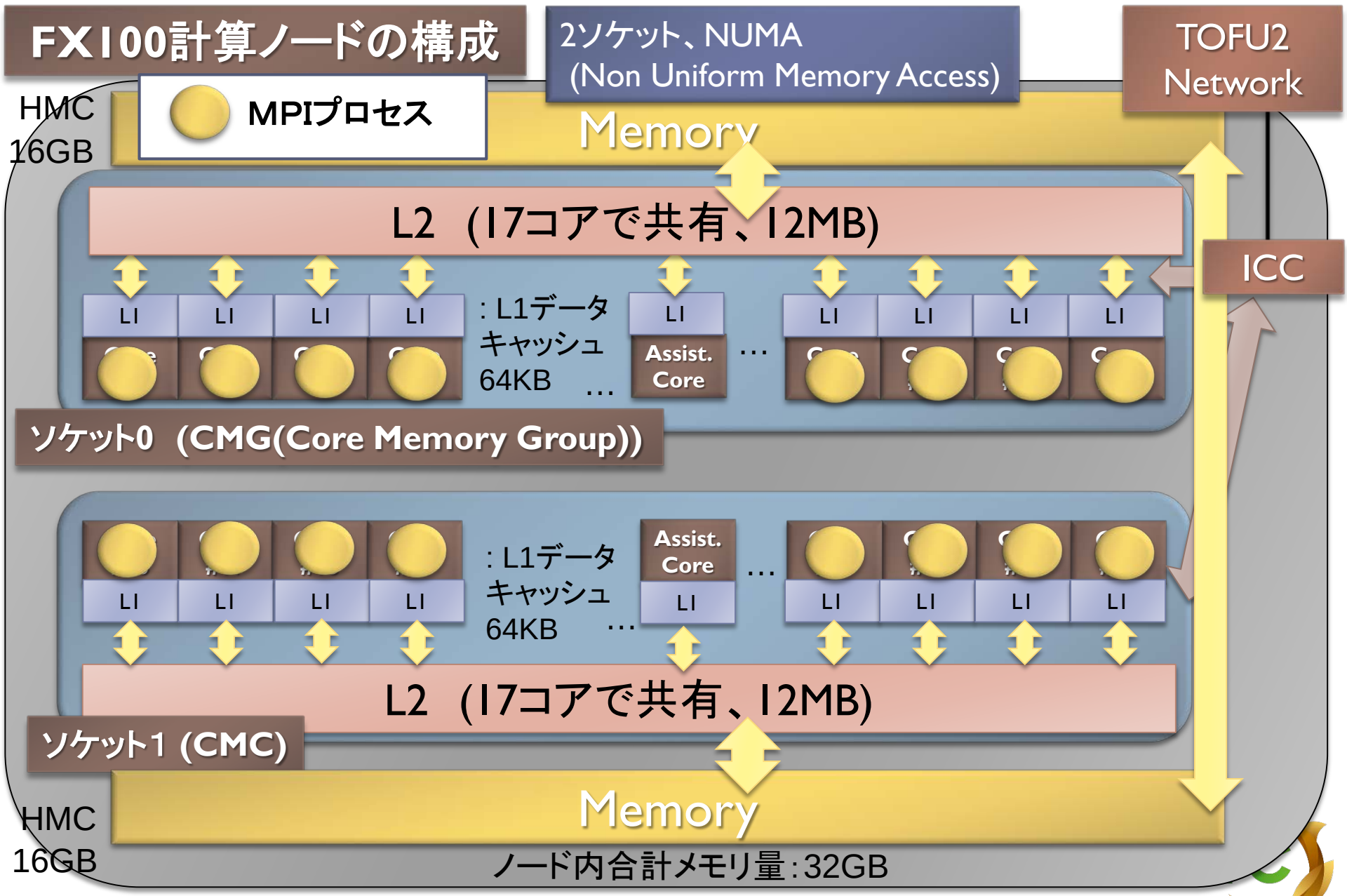
利用コア数
（MPIプロセス数）

実行時間制限
：1分

MPIジョブを32＊12 = 384プロセスで実行する。

名古屋大学
NAGOYA UNIVERSITY

FX100計算ノードの構成



読込み:240GB/秒
書込み:240GB/秒=合計:480GB/秒

名古屋大学
NAGOYA UNIVERSITY

並列版Helloプログラムを実行しよう（ピュアMPI）

1. Helloフォルダ中で以下を実行する
   $ pjsub hello-pure.bash
2. 自分の導入されたジョブを確認する
   $ pjstat
3. 実行が終了すると、以下のファイルが生成される
   hello-pure.bash.eXXXXXX
   hello-pure.bash.oXXXXXX （XXXXXXは数字）
4. 上記の標準出力ファイルの中身を見てみる
   $ cat hello-pure.bash.oXXXXXX
5. “Hello parallel world!”が、
   32プロセス*12ノード=384個表示されていたら成功。

バッチジョブ実行による標準出力、標準エラー出力

- バッチジョブの実行が終了すると、標準出力ファイルと標準エラー出力ファイルが、ジョブ投入時のディレクトリに作成されます。
- 標準出力ファイルにはジョブ実行中の標準出力、標準エラー出力ファイルにはジョブ実行中のエラーメッセージが出力されます。

ジョブ名.oXXXXX --- 標準出力ファイル
ジョブ名.eXXXXX --- 標準エラー出力ファイル
(XXXXX はジョブ投入時に表示されるジョブのジョブID)

並列版Helloプログラムを実行しよう（ハイブリッドMPI）

1. Helloフォルダ中で以下を実行する
   $ pjsub hello-hy16.bash
2. 自分の導入されたジョブを確認する
   $ pjstat
3. 実行が終了すると、以下のファイルが生成される
   hello-hy16.bash.eXXXXXX
   hello-hy16.bash.oXXXXXX （XXXXXXは数字）
4. 上記標準出力ファイルの中身を見てみる
   $ cat hello-hy16.bash.oXXXXXX
5. “Hello parallel world!”が、
   1プロセス*12ノード=12 個表示されていたら成功。

JOBスクリプトサンプルの説明（ハイブリッドMPI）
(hello-hy16.bash, C言語、Fortran言語共通)

```
#!/bin/bash
#PJM -L “rscgrp=fx-debug"
#PJM -L "node=12"
#PJM --mpi "proc=12"
#PJM -L "elapse=1:00"
export OMP_NUM_THREADS=32
mpirun ./hello
```

リソースグループ名
：fx-debug

利用ノード数

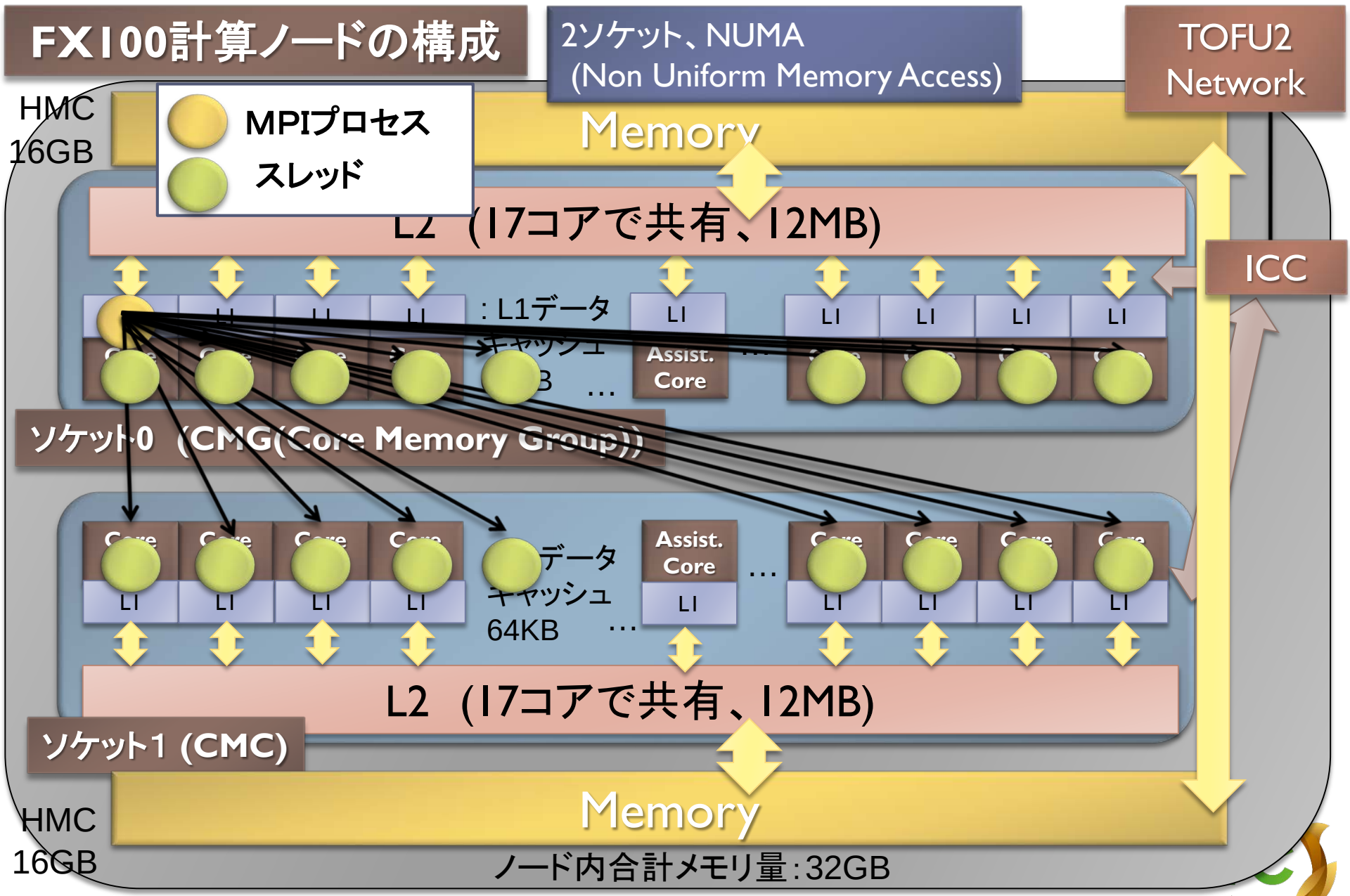
利用コア数
（MPIプロセス数）

実行時間制限：1分

1MPIプロセス当たり
32スレッド生成

MPIジョブを1＊12 = 12 プロセスで
実行する。

FX100計算ノードの構成



読込み:240GB/秒
書込み:240GB/秒=合計:480GB/秒

名古屋大学
NAGOYA UNIVERSITY

MPI実行時のリダイレクトについて

- 一般に、スーパーコンピュータでは、
  MPI実行時の入出力のリダイレクトができません
  - ×例）mpirun ./a.out < in.txt > out.txt

- 専用のリダイレクト命令が用意されています。
- FX10でリダイレクトを行う場合、以下のオプションを指定します。
  - ○例) mpirun --stdin ./in.txt --stdout out.txt ./a.out

MPIプロセスのノード割り当て

- 名大FX100システムでは、何もしないと(デフォルトでは)、確保ノードが物理的に連続に確保されない
  ⇒通信性能が劣化する場合がある
- 物理的に連続したノード割り当てをしたい場合は、ジョブスクリプトにその形状を記載する
  - ただしノード割り当て形状を指定すると、待ち時間が増加する
- 記載法: #PJM -L "node= <形状>:<機能>"
  - <形状> := { 1次元 | 2次元 | 3次元 }
    - 1次元 := { a }, 2次元 := {a x b}, 3次元 := {a x b x c}
  - <機能> := { 離散 | メッシュ | トーラス }
    - 離散 := { noncont }, メッシュ := {mesh}, トーラス := {torus} : 12ノード以上
- 例:24ノード、3次元(2x4x3)、トーラス
  - #PJM -L "node= 2x4x3 : torus"

NUMA指定について

- NUMA計算機では、MPIプロセスのソケットへの割り当てが性能面で重要となる（NUMA affinityとよぶ）
- MPIプロセスのソケット（富士通用語でCMC）の割り当ては、FX100では富士通社のNUMA affinityで設定する
- 環境変数で設定する

NUMAメモリポリシー指定

- 環境変数名 : plm_ple_memory_allocation_policy
- 代入する値
  - localalloc: プロセスが動作中のCPU(コア)の属するNUMAノードからメモリを割り当てる。
  - interleave_local: プロセスの「ローカルノード集合」内の各NUMAノードから交互にメモリ割り当てる。
  - interleave_nonlocal: プロセスの「非ローカルノード集合」内の各NUMAノードから交互にメモリ割り当てる。
  - interleave_all: プロセスの「全ノード集合」内の各NUMAノードから交互にメモリを取得する。
  - bind_local: プロセスの「ローカルノード集合」に属する各NUMAノードで、ノードIDの若い順にメモリ割り当てを行う。
  - bind_nonlocal: プロセスの「非ローカルノード集合」に属する各NUMAノードで、ノードIDの若い順にメモリ割り当てを行う。
  - bind_all: プロセスの「全ノード集合」のNUMAノードにバインドする。
  - prefer_local: プロセスの「ローカルノード集合」のうち、NUMAノードIDが最も若いものを「優先ノード」とし、「優先ノード」からメモリ割り当てを行う。
  - prefer_nonlocal: プロセスの「非ローカルノード集合」のうち、NUMAノードIDが最も若いも
  - のを「優先ノード」とし、「優先ノード」からメモリ割り当てを行う。
- 通常は、localallocでよい。
- export plm_ple_memory_allocation_policy=localalloc

CPU(コア)割り当てポリシー指定

- 環境変数名 : plm_ple_numanode_assign_policy
- 代入する値
  - simplex: NUMAノードを占有するように割り当てる。
  - share_cyclic: NUMAノードを他のプロセスと共有するように割り当てる。異なるNUMAノードに順番にプロセスを割り当てる。
  - share_band: NUMAノードを他のプロセスと共有するように割り当てる。同一NUMAノードに連続してプロセスを割り当てる。
- 例） export plm_ple_numanode_assign_policy=simplex
- 各ソケットを各MPIプロセスで独占したいときはsimplexを指定
  - 各ノードへ割り当てるMPIプロセス数が2個で、それぞれのMPIプロセスは16個のスレッド実行するとき
- MPIプロセスをプロセス順に各ソケットに詰め込みたいときは、share_bandを指定
  - ノード当たり32個のMPIプロセスを、ランク番号が近い順に割り当てたい場合

サンプルプログラムの説明

- Hello/
  - 並列版Helloプログラム
  - hello-pure.bash, hello-hy16.bash : ジョブスクリプトファイル
- Cpi/
  - 円周率計算プログラム
  - cpi-pure.bash ジョブスクリプトファイル
- Wa1/
  - 逐次転送方式による総和演算
  - wa1-pure.bash ジョブスクリプトファイル
- Wa2/
  - 二分木通信方式による総和演算
  - wa2-pure.bash ジョブスクリプトファイル
- Cpi_m/
  - 円周率計算プログラムに時間計測ルーチンを追加したもの
  - cpi_m-pure.bash ジョブスクリプトファイル

MPIプログラム実例

MPIの起動

- MPIを起動するには
  1. MPIをコンパイルできるコンパイラでコンパイル
     - 実行ファイルは a.out とする（任意の名前を付けられます）
  2. 以下のコマンドを実行
     - インタラクティブ実行では、以下のコマンドを直接入力
     - バッチジョブ実行では、ジョブスクリプトファイル中に記載

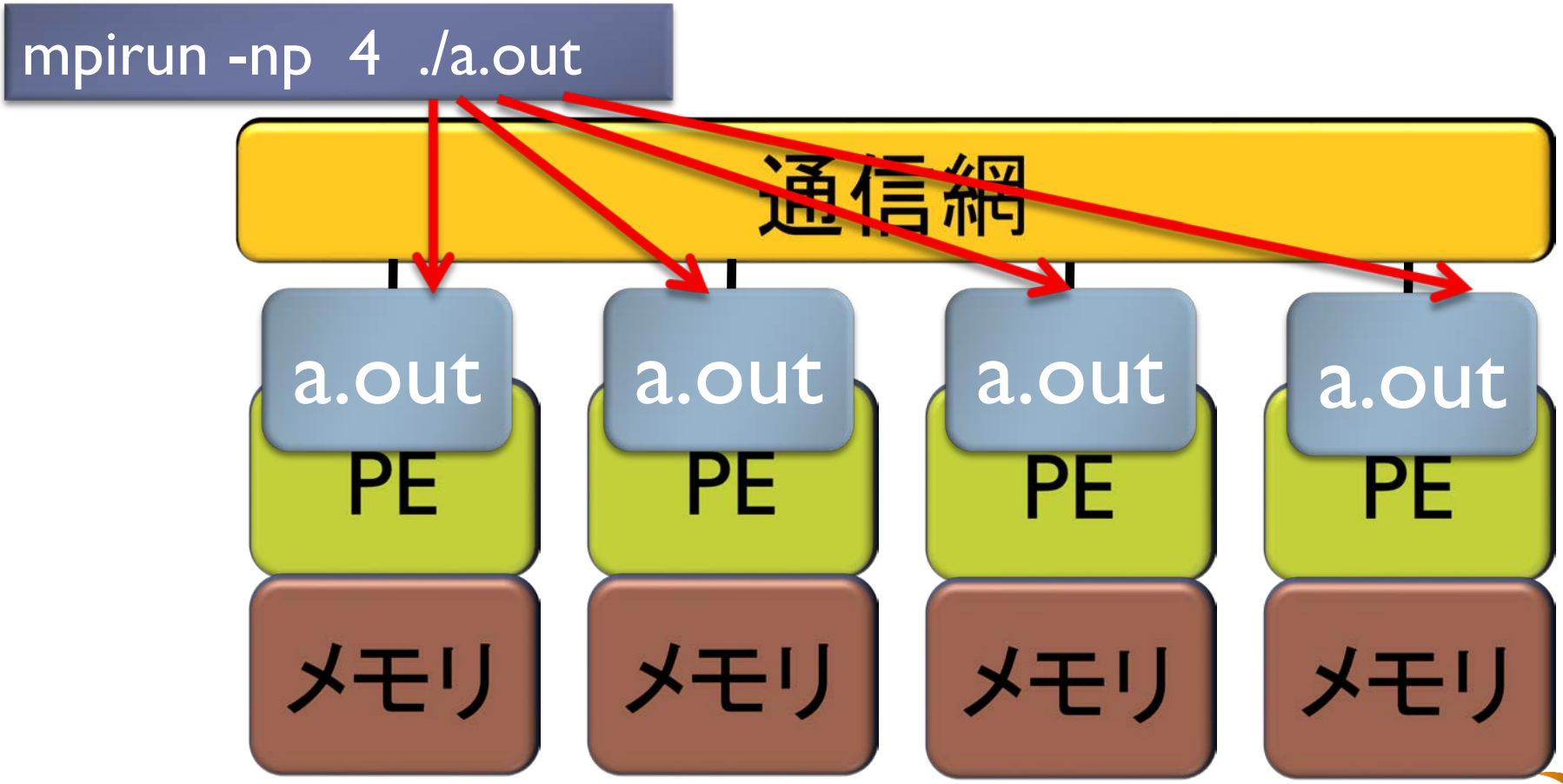
```
$ mpirun  –np  8  ./a.out
```

- mpirun: MPI起動コマンド
- –np 8: MPIプロセス数
- ./a.out: MPIの実行ファイル名

※スパコンのバッチジョブ実行では、MPIプロセス数は専用の指示文で指定する場合があります。
その場合は以下になることがあります。

```
$mpirun  ./a.out
```

MPIの起動


mpirun -np 4 ./a.out
通信網
a.out
PE
メモリ
a.out
PE
メモリ
a.out
PE
メモリ
a.out
PE
メモリ

並列版Helloプログラムの説明（C言語）

このプログラムは、全PEで起動される

```
#include <stdio.h>
#include <mpi.h>

void main(int argc, char* argv[]) {

    int    myid, numprocs;
    int    ierr, rc;

    ierr = MPI_Init(&argc, &argv);
    ierr = MPI_Comm_rank(MPI_COMM_WORLD, &myid);
    ierr = MPI_Comm_size(MPI_COMM_WORLD, &numprocs);

    printf("Hello parallel world!  Myid:%d ¥n", myid);

    rc = MPI_Finalize();

    exit(0);
}
```

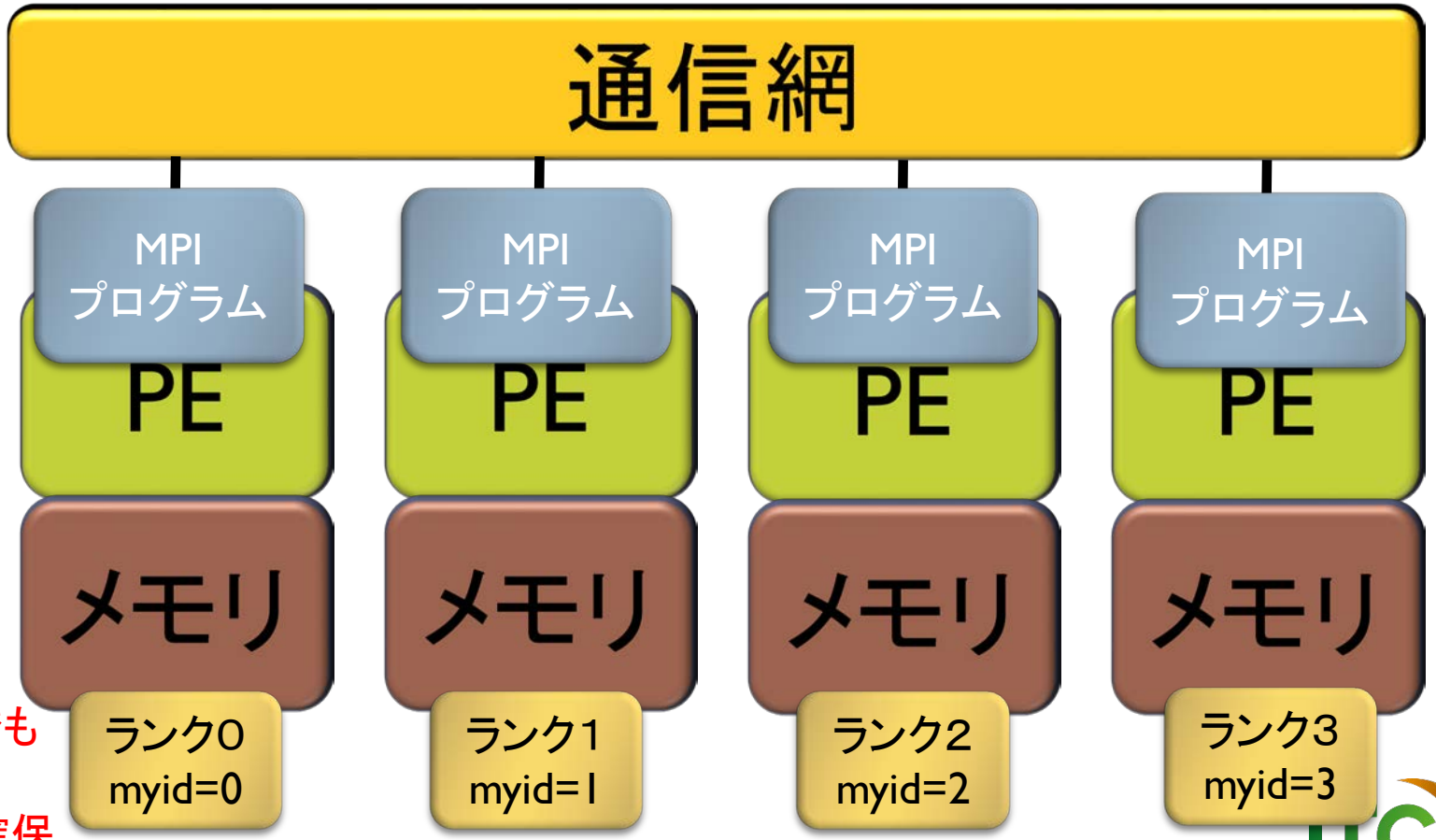
MPIの初期化

自分のID番号を取得
：各PEで値は異なる

全体のプロセッサ台数を取得
：各PEで値は同じ

MPIの終了

変数myidの説明図



同じ変数名でも
別メモリ上
に別変数で確保

並列版Helloプログラムの説明（Fortran言語）

このプログラムは、全PEで起動される

```
program main
include 'mpif.h'
common /mpienv/myid,numprocs

integer  myid, numprocs
integer  ierr

call MPI_INIT(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD, myid, ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD, numprocs, ierr)

print *, "Hello parallel world!  Myid:", myid

call MPI_FINALIZE(ierr)

stop
end
```

MPIの初期化

自分のID番号を取得
：各PEで値は異なる

全体のプロセッサ台数を取得
：各PEで値は同じ

MPIの終了

プログラム出力例

- 4プロセス実行の出力例

Hello parallel world! Myid:0

Hello parallel world! Myid:3

Hello parallel world! Myid:1

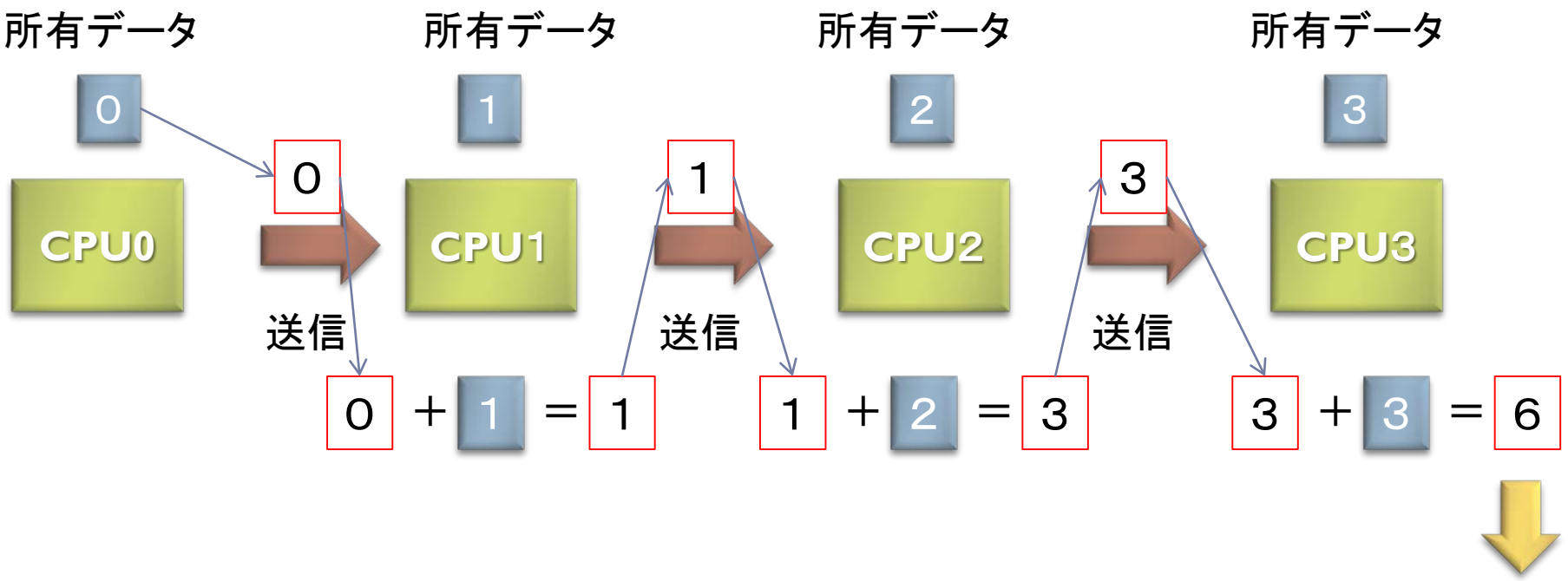
Hello parallel world! Myid:2

●4プロセスなので、表示が4個でる
（1000プロセスなら1000個出力がでる）
●myid番号が表示される。全体で重複した番号は無い。
●必ずしも、myidが0から3まで、連続して出ない
→各行は同期して実行されていない
→実行ごとに結果は異なる

総和演算プログラム（逐次転送方式）

- 各プロセスが所有するデータを、全プロセスで加算し、
  あるプロセス1つが結果を所有する演算を考える。
- 素朴な方法（逐次転送方式）
  1. （0番でなければ）左隣のプロセスからデータを受信する；
  2. 左隣のプロセスからデータが来ていたら；
     1. 受信する;
     2. ＜自分のデータ＞と＜受信データ＞を加算する;
     3. （最終ランクでなければ）右隣のプロセスに＜2の加算した結果を＞送信する;
     4. 処理を終了する;
- 実装上の注意
  - 左隣りとは、（myid-1）のIDをもつプロセス
  - 右隣りとは、（myid+1）のIDをもつプロセス
    - myid=0のプロセスは、左隣りはないので、受信しない
    - myid=p-1のプロセスは、右隣りはないので、送信しない

バケツリレー方式による加算



１対１通信利用例（逐次転送方式、C言語）

```
void main(int argc, char* argv[]) {
  MPI_Status istatus;
….
  dsendbuf = myid;
  drecvbuf = 0.0;
  if (myid != 0) {
     ierr = MPI_Recv(&drecvbuf, 1, MPI_DOUBLE, myid-1, 0,
                MPI_COMM_WORLD, &istatus);
  }
  dsendbuf = dsendbuf + drecvbuf;
  if (myid != nprocs-1) {
    ierr = MPI_Send(&dsendbuf, 1, MPI_DOUBLE, myid+1, 0,
                MPI_COMM_WORLD);
  }
  if (myid == nprocs-1) printf ("Total = %4.2lf ¥n", dsendbuf);
….
}
```

受信用システム配列の確保

自分より一つ少ないID番号（myid-1）から、double型データ1つを受信しdrecvbuf変数に代入

自分より一つ多いID番号（myid+1）に、dsendbuf変数に入っているdouble型データ1つを送信

1対1通信利用例（逐次転送方式、Fortran言語）

```
      program main
      integer  istatus(MPI_STATUS_SIZE)
      ....
      dsendbuf = myid
      drecvbuf = 0.0
      if (myid .ne. 0) then
        call MPI_RECV(drecvbuf, 1, MPI_DOUBLE_PRECISION,
     &              myid-1, 0, MPI_COMM_WORLD, istatus, ierr)
      endif
      dsendbuf = dsendbuf + drecvbuf
      if (myid .ne. numprocs-1) then
        call MPI_SEND(dsendbuf, 1, MPI_DOUBLE_PRECISION,
     &          myid+1, 0, MPI_COMM_WORLD, ierr)
      endif
     if (myid .eq. numprocs-1) then
        print *, "Total = ", dsendbuf
      endif
      ....
      stop
      end
```

受信用システム配列の確保

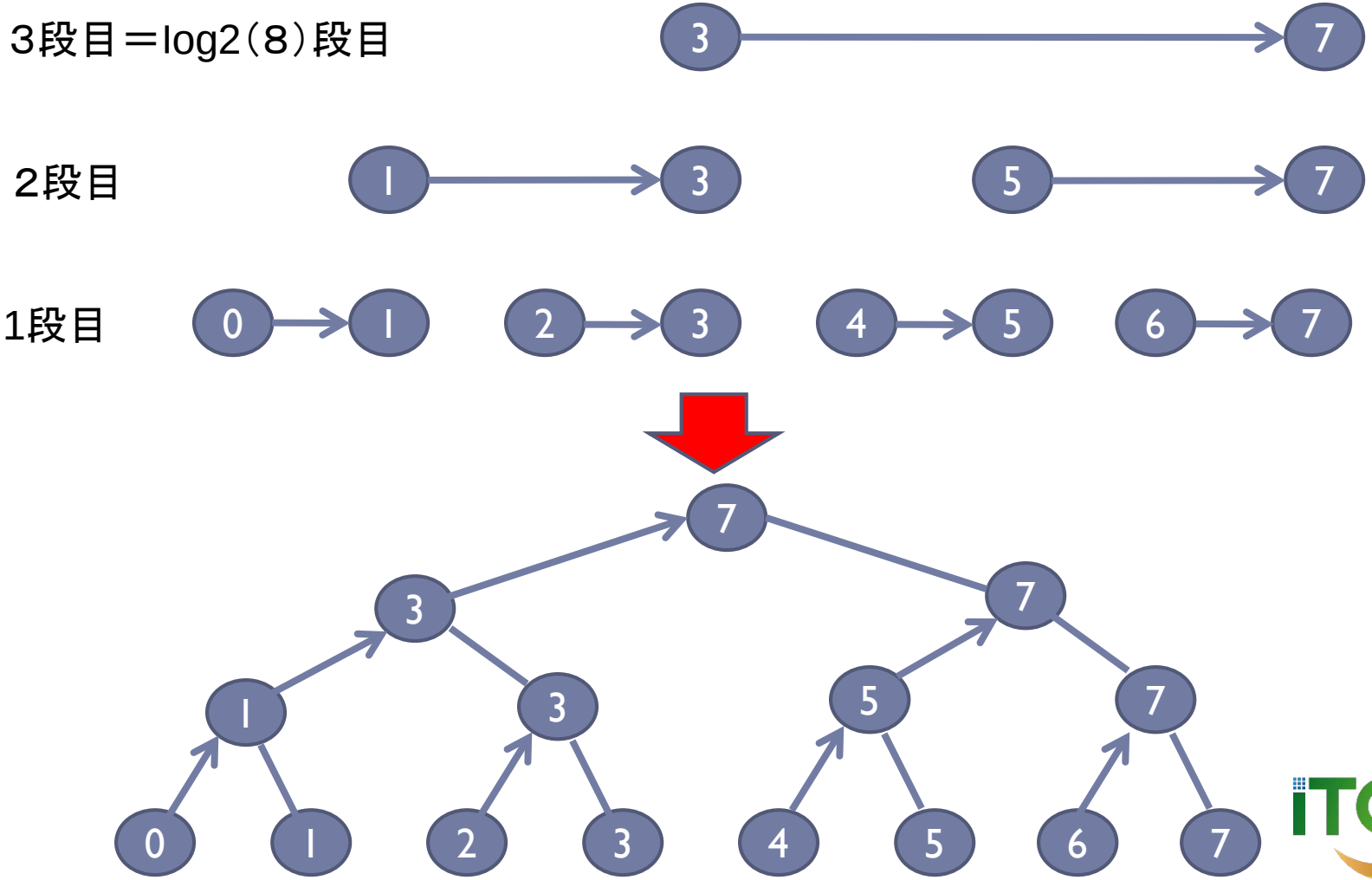
自分より一つ少ないID番号（myid-1）から、double型データ1つを受信しdrecvbuf変数に代入

自分より一つ多いID番号（myid+1）に、dsendbuf変数に入っているdouble型データ1つを送信

総和演算プログラム（二分木通信方式）

- 二分木通信方式
  1. k = 1;
  2. for (i=0; i < log2(nprocs); i++)
  3. if ((myid & k) == k)
     - (myid – k)番 プロセス からデータを受信;
     - 自分のデータと、受信データを加算する;
     - k = k * 2;
  4. else
     - (myid + k)番 プロセス に、データを転送する;
     - 処理を終了する;

総和演算プログラム（二分木通信方式）



総和演算プログラム（二分木通信方式）

- 実装上の工夫
  - 要点：プロセス番号の2進数表記の情報を利用する
  - 第i段において、受信するプロセスの条件は、以下で書ける：

    myid & k が k と一致
    - ここで、k = 2^(i-1) 。
    - つまり、プロセス番号の2進数表記で右からi番目のビットが立っているプロセスが、送信することにする
  - また、送信元のプロセス番号は、以下で書ける：

    myid + k
    - つまり 、通信が成立するPE番号の間隔は2^(i-1) ←二分木なので
  - 送信プロセスについては、上記の逆が成り立つ。

総和演算プログラム（二分木通信方式）

- 逐次転送方式の通信回数
  - 明らかに、nprocs－１ 回
- 二分木通信方式の通信回数
  - 見積もりの前提
    - 各段で行われる通信は、完全に並列で行われる（通信の衝突は発生しない）
  - 段数の分の通信回数となる
  - つまり、log2(nprocs) 回
- 両者の通信回数の比較
  - プロセッサ台数が増すと、通信回数の差（＝実行時間）がとても大きくなる
  - １０２４構成では、１０２３回　対　１０回！
  - でも、必ずしも二分木通信方式がよいとは限らない（通信衝突の多発）

性能プロファイラ

- 富士通コンパイラには、性能プロファイラ機能がある
- 富士通コンパイラでコンパイル後、実行コマンドで指定し利用する
- 以下の2種類があります
- 基本プロファイラ
  - 主な用途：プログラム全体で、最も時間のかかっている関数を同定する
- 詳細プロファイラ
  - 主な用途：最も時間のかかっている関数内の特定部分において、メモリアクセス効率、キャッシュヒット率、スレッド実行効率、MPI通信頻度解析、を行う

性能プロファイラの種類の詳細

- 基本プロファイラ
  - コマンド例：fipp –C
  - 表示コマンド：fipppx、GUI（WEB経由）
  - ユーザプログラムに対し一定間隔(デフォルト時100 ミリ秒間隔)毎に割り込みをかけ情報を収集する。
  - 収集した情報を基に、コスト情報等の分析結果を表示。
- 詳細プロファイラ
  - コマンド例：fapp –C
  - 表示コマンド：GUI(WEB経由)
  - ユーザプログラムの中に測定範囲を設定し、測定範囲のハードウェアカウンタの値を収集。
  - 収集した情報を基に、MFLOPS、MIPS、各種命令比率、キャッシュミス等の詳細な分析結果を表示。

基本プロファイラ利用例

- プロファイラデータ用の空のディレクトリがないとダメ
- /Wa2 に Profディレクトリを作成
  $ mkdir Prof
- Wa2 の wa2-pure.bash 中に以下を記載
  fipp -C -d Prof mpirun ./wa2
- 実行する
  $ pjsub wa2-pure.bash
- テキストプロファイラを起動
  $ fipppx –A -d Prof

基本プロファイラ出力例（1/2）

```
--------------------------------------------------------------------------------
Fujitsu Instant Profiler Version 1.2.0
  Measured time             : Thu Apr 19 09:32:18 2012
  CPU frequency             : Process     0 -    127  1848 (MHz)
  Type of program           : MPI
  Average at sampling interval  : 100.0 (ms)
  Measured range            : All ranges
  Virtual coordinate        : (12, 0, 0)
--------------------------------------------------------------------------------

Time statistics
      Elapsed(s)      User(s)     System(s)
  ------------------------------------------
        2.1684       53.9800       87.0800   Application
  ------------------------------------------
        2.1684        0.5100        0.6400   Process    11
        2.1588        0.4600        0.6800   Process    88
        2.1580        0.5000        0.6400   Process    99
        2.1568        0.6600        1.4200   Process   111
...
```

基本プロファイラ出力例（2/2）

```
Procedures profile
  ********************************************************************************************
    Application - procedures
  ********************************************************************************************
         Cost       %         Mpi       %   Start     End
  ----------------------------------------------------------------------
          475   100.0000        312   65.6842     --      --  Application
  ----------------------------------------------------------------------
          312    65.6842        312   100.0000     1      45  MAIN__
           82    17.2632          0     0.0000     --     --  __GI___sched_yield
           80    16.8421          0     0.0000     --     --  __libc_poll
            1     0.2105          0     0.0000     --     --  __pthread_mutex_unlock_usercnt
  ********************************************************************************************
    Process    11 - procedures
  ********************************************************************************************
         Cost       %         Mpi       %   Start     End
  ----------------------------------------------------------------------
            5   100.0000          4    80.0000     --     --  Process    11
  ----------------------------------------------------------------------
            4    80.0000          4   100.0000     1      45  MAIN__
            1    20.0000          0     0.0000     --     --  __GI___sched_yeld
....
```

詳細プロファイラ利用例

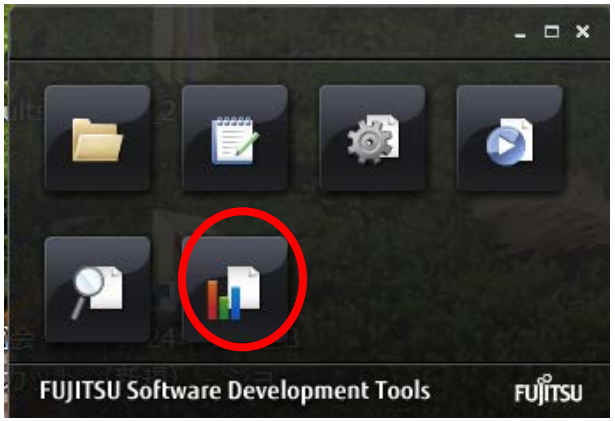
- 測定したい対象に、以下のコマンドを挿入
- Fortran言語の場合
  - ヘッダファイル：なし
  - 測定開始　手続き名： call fapp_start(name, number, level)
  - 測定終了　手続き名： call fapp_stop(name, number, level)
  - 利用例： call fapp_start(“region1”,1,1)
- C/C++言語の場合
  - ヘッダファイル： fj_tool/fapp.h
  - 測定開始　関数名： void fapp_start(const char *name, int number, int level)
  - 測定終了　関数名： void fapp_stop(const char *name, int number, int level)
  - 利用例： fapp_start(“region1”,1,1);

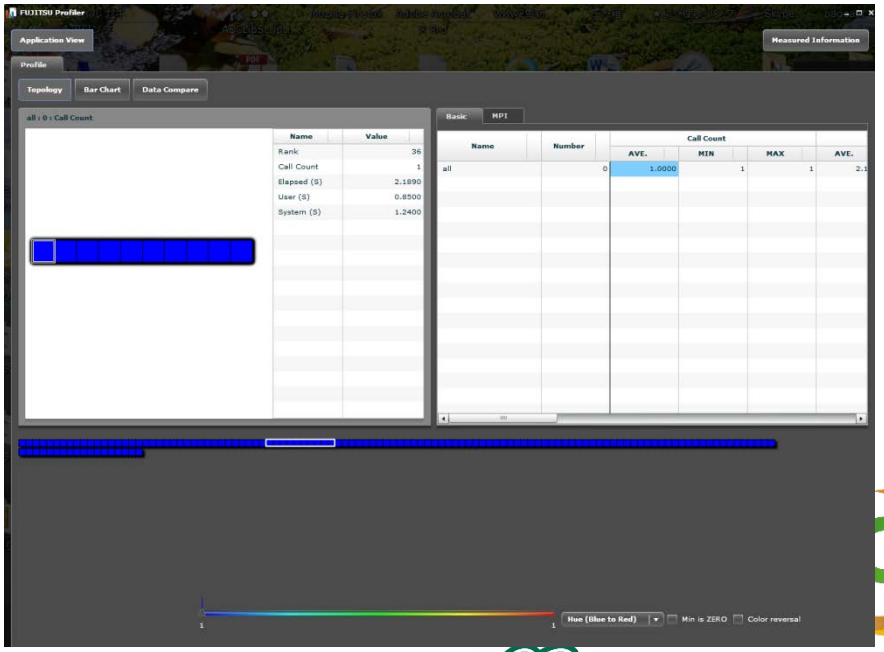
詳細プロファイラ利用例

- 空のディレクトリがないとダメなので、/Wa2 に Profディレクトリを作成
  $ mkdir Prof
- Wa2のwa2-pure.bash中に以下を記載
  (キャッシュ情報取得時)
  fapp -C -d Prof –L 1 –lhwm –Hevent=Cache mpirun ./wa2
- 実行する
  $ pjsub wa2-pure.bash

詳細プロファイラGUIによる表示例

- 右のボックスで、プロファイラ部分をクリック
- プロファイルデータがあるフォルダを指定する
- うまくいくと、右のような解析データが見える





名古屋大学
NAGOYA UNIVERSITY

詳細プロファイラで取れるデータ

- プロセス間の通信頻度情報
  （GUI上で色で表示）
- 各MPIプロセスにおける以下の情報
  - Cache: キャッシュミス率
  - Instructions: 実行命令詳細
  - Mem_access: メモリアクセス状況
  - Performance: 命令実行効率
  - Statistics: CPU core 動作状況

精密ＰＡ可視化機能（エクセル形式）

- 性能プロファイルは見にくい
- １１回程度実行しないといけないが、
  性能プロファイルデータ（マシン語命令の種類や、実行時間に占める割合など）を、Excelで可視化してくれるツール（精密PA可視化ツール）が、名古屋大学情報基盤センターのFX100では、提供されている
- 手順
  1. 対象箇所（ループ）を、専用のAPIで指定する
  2. プロファイルを入れるフォルダ11か所をつくる
  3. プロファイルのためのコマンドで11回実行する
  4. エクセル形式に変換する
  5. 4のエクセル形式を手元のパソコンに持ってくる
  6. 5のファイルを、指定のエクセルと同一のフォルダに入れてから、指定のエクセルを開く

精密ＰＡ可視化のための指示ＡＰＩ

- 以下のAPIで、対象となるループを挟む（Fortranの場合）

```
call start_collection(“region”)
    <対象となるループ>
call stop_collection(“region”)
```

- “region”は、対象となる場所の名前なので、任意の名前を付けることが可能
（後で、専用エクセルを開くときに使う）

実行のさせ方

- a.out という実行ファイルの場合、以下のように11回実行する
- 実行するディレクトリに、pa1、pa2、…、pa11という、ディレクトリを作っておく必要がある
- 以下のように実行する

```
fapp -C –d pa1 -Hpa=1 mpiexec a.out
fapp -C –d pa2 -Hpa=2 mpiexec a.out
fapp -C –d pa3 -Hpa=3 mpiexec a.out
fapp -C –d pa4 -Hpa=4 mpiexec a.out
fapp -C –d pa5 -Hpa=5 mpiexec a.out
fapp -C –d pa6 -Hpa=6 mpiexec a.out
   ….
fapp -C –d pa11 -Hpa=11 mpiexec a.out
```

ITC

名古屋大学
NAGOYA UNIVERSITY

エクセルデータへの変換

- pa1、pa2、…、pa11の中に、プロファイルデータがあることを確認する
- 以下のコマンドを実行する

```
fapppx -A -d pa1 -o output_prof_1.csv -tcsv -Hpa
fapppx -A -d pa2 -o output_prof_2.csv -tcsv -Hpa
fapppx -A -d pa3 -o output_prof_3.csv -tcsv -Hpa
fapppx -A -d pa4 -o output_prof_4.csv -tcsv -Hpa
fapppx -A -d pa5 -o output_prof_5.csv -tcsv -Hpa
fapppx -A -d pa6 -o output_prof_6.csv -tcsv -Hpa
  ….
fapppx -A -d pa11 -o output_prof_11.csv -tcsv -Hpa
```

エクセルデータを手元のＰＣに転送

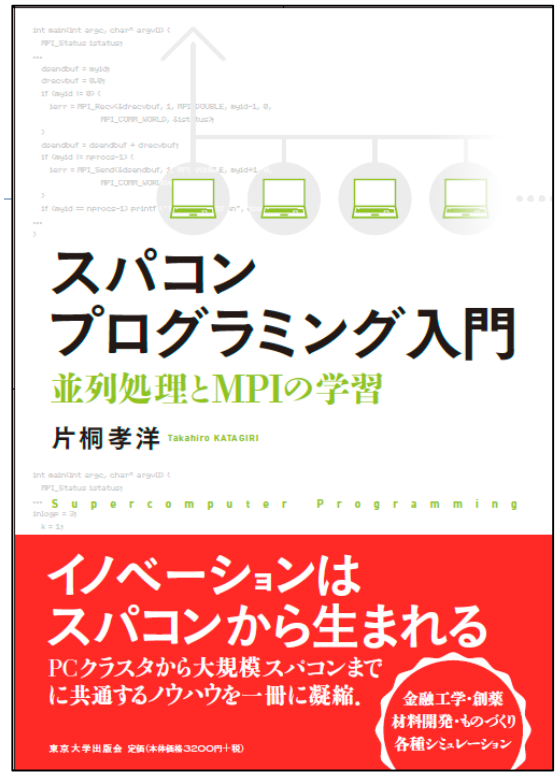
- 以下のFX100上のエクセルデータを、手元のPCに転送
  - output_prof_1.csv、output_prof_2.csv、…、output_prof_11.csv
- 上記のエクセルデータが入ったフォルダで、ポータル上で公開されている専用エクセルを開く
- 詳細なエクセルデータの利用法、分析されたデータの見方は、マニュアル参照

演習課題

1. 逐次転送方式のプログラムを実行
   - Wa1 のプログラム
2. 二分木通信方式のプログラムを実行
   - Wa2のプログラム
3. 時間計測プログラムを実行
   - Cpi_mのプログラム
4. プロセス数を変化させて、サンプルプログラムを実行
5. Helloプログラムを、以下のように改良
   - MPI_Sendを用いて、プロセス0からChar型のデータ“Hello World!!”を、その他のプロセスに送信する
   - その他のプロセスでは、MPI_Recvで受信して表示する

並列プログラミングの基本（座学）

教科書（演習書）



- 「スパコンプログラミング入門
  ー並列処理とMPIの学習ー」
  - 片桐 孝洋　著、
  - 東大出版会、ISBN978-4-13-062453-4、
    発売日：2013年3月12日、判型:A5, 200頁
  - 【本書の特徴】
    - C言語で解説
    - C言語、Fortran90言語のサンプルプログラムが付属
    - 数値アルゴリズムは、図でわかりやすく説明
    - 本講義の内容を全てカバー
    - 内容は初級。初めて並列数値計算を学ぶ人向けの
      入門書

教科書（演習書）

- 「並列プログラミング入門:
  サンプルプログラムで学ぶOpenMPとOpenACC」（仮題）
  - 片桐 孝洋　著
  - 東大出版会、ISBN-10: 4130624563、
    ISBN-13: 978-4130624565、発売日： 2015年5月25日
  - 【本書の特徴】
    - C言語、Fortran90言語で解説
    - C言語、Fortran90言語の複数のサンプルプログラムが入手可能
      （ダウンロード形式）
    - 本講義の内容を全てカバー
    - Windows PC演習可能(Cygwin利用)。スパコンでも演習可能。
    - 内容は初級。初めて並列プログラミングを学ぶ人向けの
      入門書

参考書

- 「スパコンを知る:
  その基礎から最新の動向まで」
  - 岩下武史、片桐孝洋、高橋大介 著
  - 東大出版会、ISBN-10: 4130634550、
    ISBN-13: 978-4130634557、
    発売日 : 2015年2月20日、176頁
  - 【本書の特徴】
    - スパコンの解説書です。以下を
      分かりやすく解説します。
      - スパコンは何に使えるか
      - スパコンはどんな仕組みで、なぜ速く計算できるのか
      - 最新技術、今後の課題と将来展望、など

参考書

- 「並列数値処理　－高速化と性能向上のために－」
  - 金田康正 東大教授 理博 編著、
    片桐孝洋 東大特任准教授 博士（理学）著、黒田久泰 愛媛大准教授 博士（理学）著、山本有作 神戸大教授 博士（工学）著、五百木伸洋 ㈱日立製作所 著、
  - コロナ社、発行年月日：2010/04/30，判　型：A5，ページ数：272頁、ISBN：978-4-339-02589-7，定価：3,990円（本体3,800円＋税5%）
  - 【本書の特徴】
    - Fortran言語で解説
    - 数値アルゴリズムは、数式などで厳密に説明
    - 本講義の内容に加えて、固有値問題の解法、疎行列反復解法、FFT、ソート、など、主要な数値計算アルゴリズムをカバー
    - 内容は中級～上級。専門として並列数値計算を学びたい人向き

スパコンの基本：並列処理
並列化とは何か？

- 逐次実行のプログラム（実行時間T）を、p台の計算機を使って、T / pにすること。

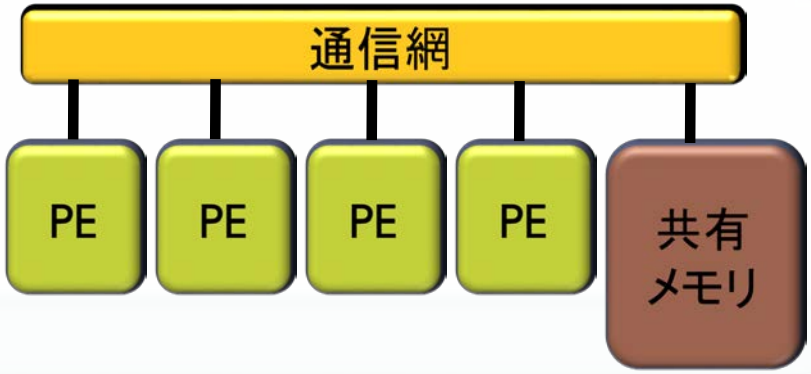
- 素人考えでは自明。
- 実際は、できるかどうかは、対象処理の内容（アルゴリズム）で　**大きく**　難しさが違う
  - アルゴリズム上、絶対に並列化できない部分の存在
  - 通信のためのオーバヘッドの存在
    - 通信立ち上がり時間
    - データ転送時間

並列計算機の分類

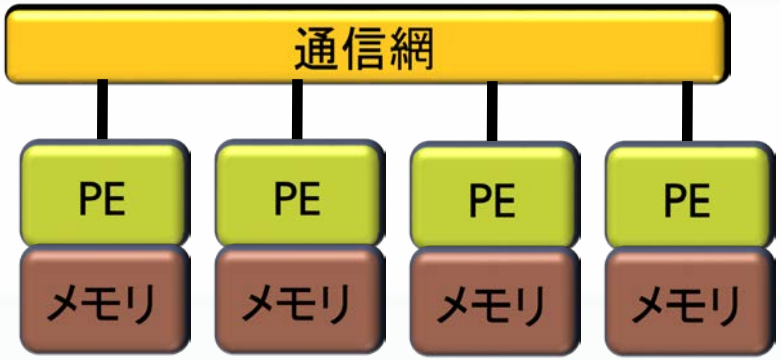
- Michael J. Flynn教授（スタンフォード大）の分類（1966）
- 単一命令・単一データ流
  （SISD, Single Instruction Single Data Stream）
- 単一命令・複数データ流
  （SIMD, Single Instruction Multiple Data Stream）
- 複数命令・単一データ流
  （MISD, Multiple Instruction Single Data Stream）
- 複数命令・複数データ流
  （MIMD, Multiple Instruction Multiple Data Stream）

並列計算機のメモリ型による分類

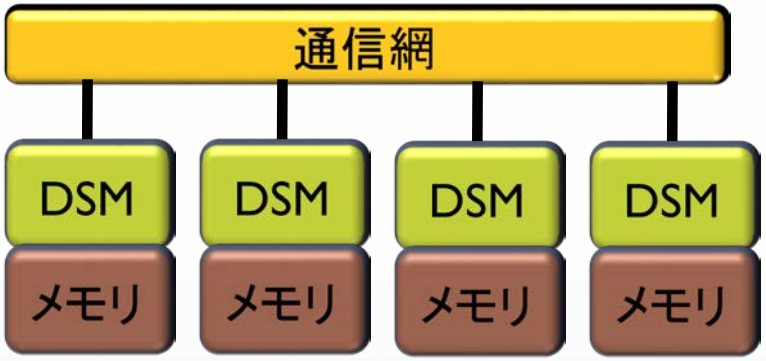
1. 共有メモリ型
（SMP、
Symmetric Multiprocessor）



2. 分散メモリ型
（メッセージパッシング）

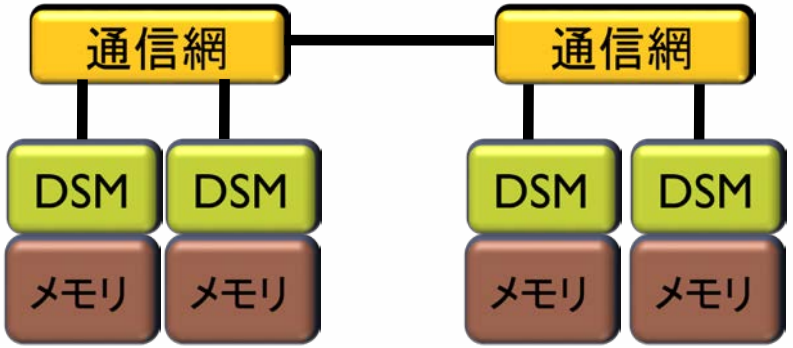


3. 分散共有メモリ型
（DSM、
Distributed Shared Memory）



並列計算機のメモリ型による分類

4. （分散メモリ型並列計算機）
   共有・非対称メモリ型
（ccNUMA、Cache Coherent Non-Uniform Memory Access）

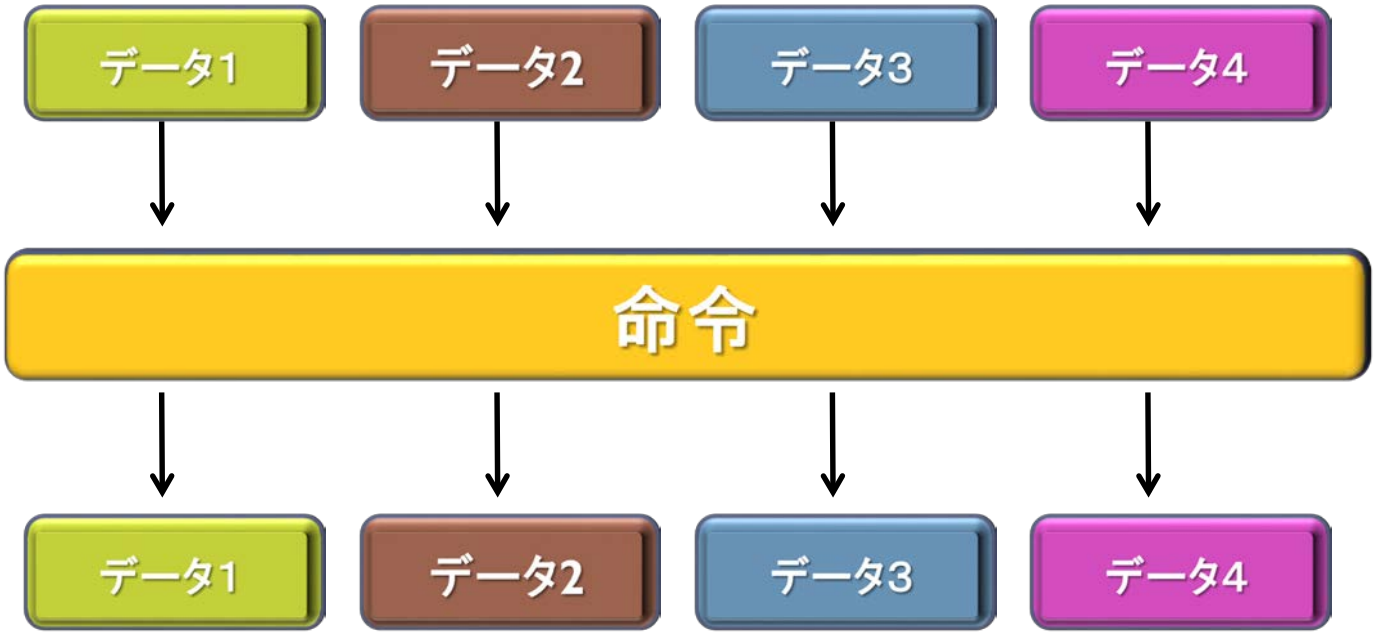


並列計算機の分類とMPIとの関係

- MPIは分散メモリ型計算機を想定
  - MPIは、分散メモリ間の通信を定めているため
- MPIは共有メモリ型計算機でも動く
  - MPIは、共有メモリ内でもプロセス間通信ができるため
- MPIを用いたプログラミングモデルは、（基本的に）SIMD
  - MPIは、（基本的には）プログラムが1つ（＝命令と等価）しかないが、データ（配列など）は複数あるため

並列プログラミングのモデル

- 実際の並列プログラムの挙動はMIMD
- アルゴリズムを考えるときは＜SIMDが基本＞
  - 複雑な挙動は理解できないので



並列プログラミングのモデル

- MIMD上での並列プログラミングのモデル
  1. SPMD（Single Program Multiple Data）
     - 1つの共通のプログラムが、並列処理開始時に、全プロセッサ上で起動する
     - MPI（バージョン1）のモデル

  2. Master / Worker（Master / Slave）
     - 1つのプロセス（Master）が、複数のプロセス（Worker）を管理（生成、消去）する。

並列プログラムの種類

- マルチプロセス
  - MPI（Message Passing Interface）
  - HPF（High Performance Fortran）
    - 自動並列化Fortranコンパイラ
    - ユーザがデータ分割方法を明示的に記述
- マルチスレッド
  - Pthread (POSIX スレッド)
  - Solaris Thread (Sun Solaris OS用)
  - NT thread (Windows NT系、Windows95以降)
    - スレッドの Fork（分離）と Join（融合）を明示的に記述
  - Java
    - 言語仕様としてスレッドを規定
  - OpenMP
    - ユーザが並列化指示行を記述

プロセスとスレッドの違い
•メモリを意識するかどうかの違い
　•別メモリは「プロセス」
　•同一メモリは「スレッド」

マルチプロセスとマルチスレッドは
共存可能
→ハイブリッドMPI/OpenMP実行

並列処理の実行形態（1）

- データ並列
  - データを分割することで並列化する。
  - データの操作（＝演算）は同一となる。
  - データ並列の例：行列ー行列積

SIMDの
考え方と同じ

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \begin{pmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{pmatrix} = \begin{pmatrix} 1*9+2*6+3*3 & 1*8+2*5+3*2 & 1*7+2*4+3*1 \\ 4*9+5*6+6*3 & 4*8+5*5+6*2 & 4*7+5*4+6*1 \\ 7*9+8*6+9*3 & 7*8+8*5+9*2 & 7*7+8*4+9*1 \end{pmatrix}$$

●並列化

全CPUで共有

CPU0 $\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$

CPU1 $\begin{pmatrix} 4 & 5 & 6 \end{pmatrix}$

CPU2 $\begin{pmatrix} 7 & 8 & 9 \end{pmatrix}$

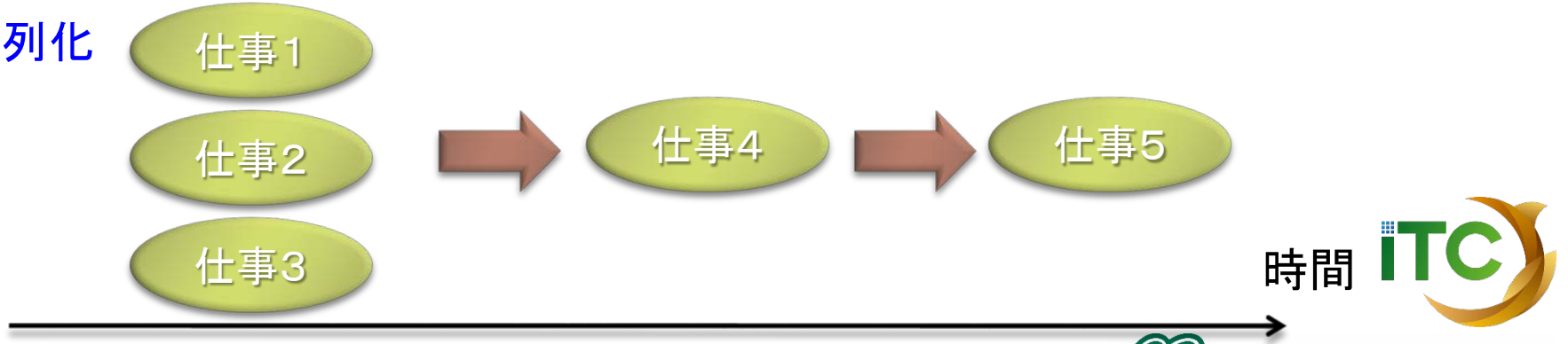
$\begin{pmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{pmatrix} = \begin{pmatrix} 1*9+2*6+3*3 & 1*8+2*5+3*2 & 1*7+2*4+3*1 \\ 4*9+5*6+6*3 & 4*8+5*5+6*2 & 4*7+5*4+6*1 \\ 7*9+8*6+9*3 & 7*8+8*5+9*2 & 7*7+8*4+9*1 \end{pmatrix}$

並列に計算：初期データは異なるが演算は同一

並列処理の実行形態（２）

- タスク並列
  - タスク（ジョブ）を分割することで並列化する。
  - データの操作（＝演算）は異なるかもしれない。
  - タスク並列の例：カレーを作る
    - 仕事１：野菜を切る
    - 仕事２：肉を切る
    - 仕事３：水を沸騰させる
    - 仕事４：野菜・肉を入れて煮込む
    - 仕事５：カレールゥを入れる

●並列化

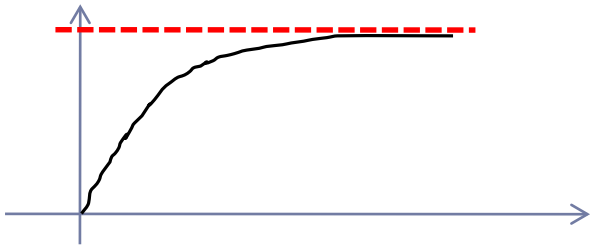


性能評価指標

並列化の尺度

性能評価指標－台数効果

- 台数効果
  - 式：　$S_P = T_S / T_P \quad (0 \le S_p)$
  - T_S ：逐次の実行時間、　T_P ：P台での実行時間
  - P台用いて　$S_P = P$ のとき、理想的な（ideal）速度向上
  - P台用いて　$S_P > P$ のとき、スーパリニア・スピードアップ
    - 主な原因は、並列化により、データアクセスが局所化されて、キャッシュヒット率が向上することによる高速化
- 並列化効率
  - 式：　$E_P = S_P / P \times 100 \quad (0 \le E_p)\ [\%]$
- 飽和性能
  - 速度向上の限界
  - Saturation、「さちる」



アムダールの法則

- 逐次実行時間を　K　とする。
  そのうち、並列化ができる割合を　α　とする。
- このとき、台数効果は以下のようになる。

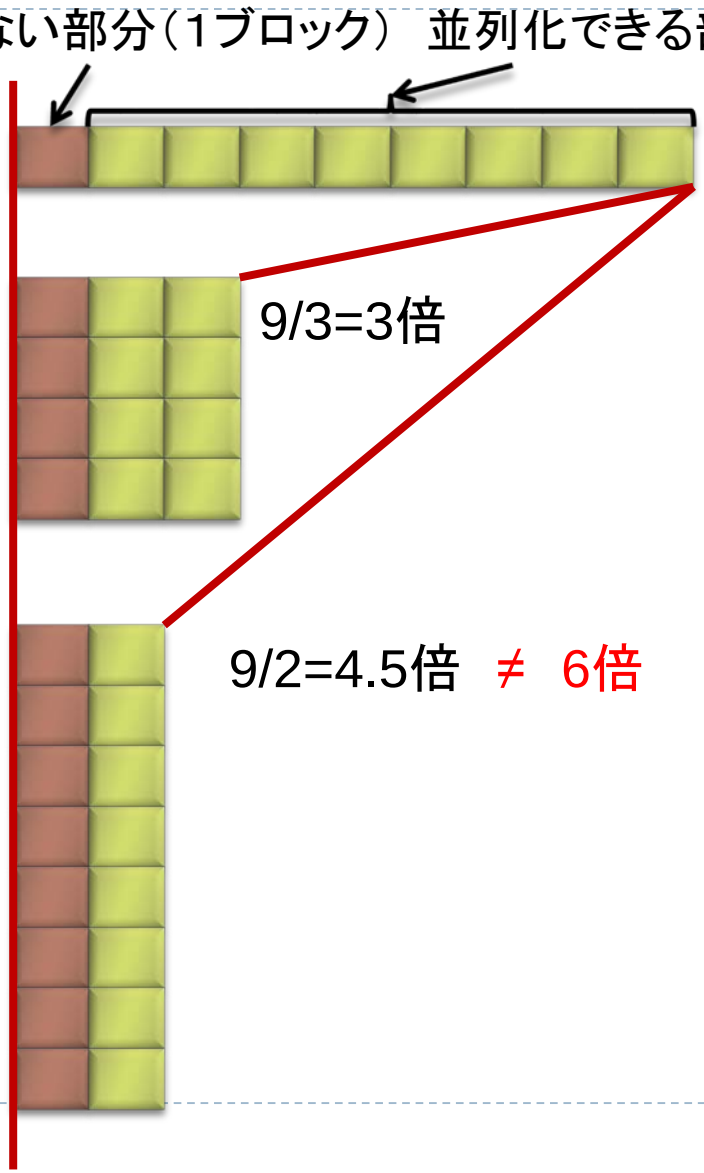
$$S_P = K/(K\alpha/P + K(1-\alpha))$$

$$= 1/(\alpha/P + (1-\alpha)) \;=\; 1/(\alpha(1/P-1)+1)$$

- 上記の式から、たとえ無限大の数のプロセッサを使っても（P→∞）、台数効果は、高々　1／（1－α）　である。
  （アムダールの法則）
  - 全体の90%が並列化できたとしても、無限大の数のプロセッサをつかっても、1/(1-0.9) = 10 倍　にしかならない！

  →高性能を達成するためには、少しでも並列化効率を上げる実装をすることがとても重要である

アムダールの法則の直観例



●逐次実行

＝88.8%が並列化可能

●並列実行（4並列）

9/3=3倍

●並列実行（8並列）

9/2=4.5倍 ≠ 6倍

名古屋大学
NAGOYA UNIVERSITY

内容に関する質問は
katagiri@cc.nagoya-u.ac.jp
まで

MPIの基礎

名古屋大学情報基盤センター　片桐孝洋

名古屋大学
NAGOYA UNIVERSITY

MPIの特徴

- メッセージパッシング用のライブラリ規格の1つ
  - メッセージパッシングのモデルである
  - コンパイラの規格、特定のソフトウエアやライブラリを指すものではない！
- 分散メモリ型並列計算機で並列実行に向く
- 大規模計算が可能
  - 1プロセッサにおけるメモリサイズやファイルサイズの制約を打破可能
  - プロセッサ台数の多い並列システム（MPPシステム、Massively Parallel Processingシステム）を用いる実行に向く
    - 1プロセッサ換算で膨大な実行時間の計算を、短時間で処理可能
  - 移植が容易
    - API（Application Programming Interface）の標準化
- スケーラビリティ、性能が高い
  - 通信処理をユーザが記述することによるアルゴリズムの最適化が可能
  - プログラミングが難しい（敷居が高い）

MPIの経緯（1/2）

- MPIフォーラム（http://www.mpi-forum.org/）が仕様策定
  - 1994年5月1.0版（MPI-1）
  - 1995年6月1.1版
  - 1997年7月1.2版、および 2.0版（MPI-2）
- **米国アルゴンヌ国立研究所、およびミシシッピ州立大学で開発**
- MPI-2 では、以下を強化：
  - 並列I/O
  - C++、Fortran 90用インターフェース
  - 動的プロセス生成/消滅
    - 主に、並列探索処理などの用途

MPIの経緯 MPI3.1策定

- 以下のページで経緯・ドキュメントを公開中
  - http://mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf
    (Implementation Status, as of June 4, 2015)
- 注目すべき機能
  - ノン・ブロッキングの集団通信機能
    （MPI_IALLREDUCE、など）
  - 片方向通信（RMA、Remote Memory Access)
  - Fortran2008 対応、など

MPIの経緯 MPI4.0策定

- 以下のページで経緯・ドキュメントを公開中
  - http://mpi-forum.org/mpi-40/
- 検討されている機能
  - ハイブリッドプログラミングへの対応
  - MPIアプリケーションの耐故障性（Fault Tolerance, FT）
  - いくつかのアイデアを検討中
    - Active Messages (メッセージ通信のプロトコル)
      - 計算と通信のオーバラップ
      - 最低限の同期を用いた非同期通信
      - 低いオーバーヘッド、パイプライン転送
      - バッファリングなしで、インタラプトハンドラで動く
    - Stream Messaging
    - 新プロファイル・インターフェース

MPIの実装

- MPICH（エム・ピッチ）
  - 米国アルゴンヌ国立研究所が開発
- LAM（Local Area Multicomputer）
  - ノートルダム大学が開発
- その他
  - OpenMPI （FT-MPI、LA-MPI、LAM/MPI、PACX-MPIの統合プロジェクト）
  - YAMPII（（旧）東大・石川研究室）（SCore通信機構をサポート）
- 注意点:メーカ独自機能拡張がなされていることがある

MPIによる通信

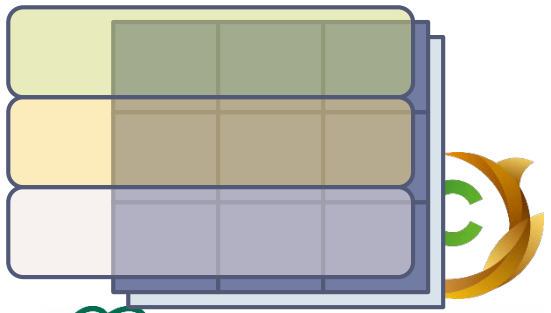
- 郵便物の郵送に同じ
- 郵送に必要な情報：
  1. 自分の住所、送り先の住所
  2. 中に入っているものはどこにあるか
  3. 中に入っているものの分類
  4. 中に入っているものの量
  5. （荷物を複数同時に送る場合の）認識方法（タグ）
- MPIでは：
  1. 自分の認識ID、および、送り先の認識ID
  2. データ格納先のアドレス
  3. データ型
  4. データ量
  5. タグ番号

MPI関数

- システム関数
  - MPI_Init；MPI_Comm_rank；MPI_Comm_size；MPI_Finalize；
- １対１通信関数
  - ブロッキング型
    - MPI_Send； MPI_Recv；
  - ノンブロッキング型
    - MPI_Isend； MPI_Irecv；
- １対全通信関数
  - MPI_Bcast
- 集団通信関数
  - MPI_Reduce； MPI_Allreduce； MPI_Barrier；
- 時間計測関数
  - MPI_Wtime

コミュニケータ

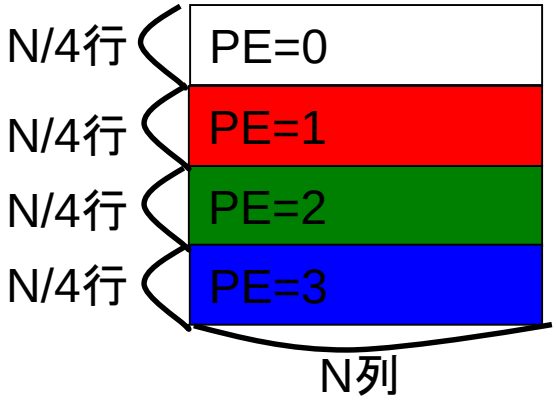
- MPI_COMM_WORLDは、コミュニケータとよばれる概念を保存する変数
- コミュニケータは、操作を行う対象のプロセッサ群を定める
- 初期状態では、0番～numprocs –1番までのプロセッサが、1つのコミュニケータに割り当てられる
  - この名前が、“MPI_COMM_WORLD”
- プロセッサ群を分割したい場合、MPI_Comm_split 関数を利用
  - メッセージを、一部のプロセッサ群に放送するときに利用
  - “マルチキャスト”で利用



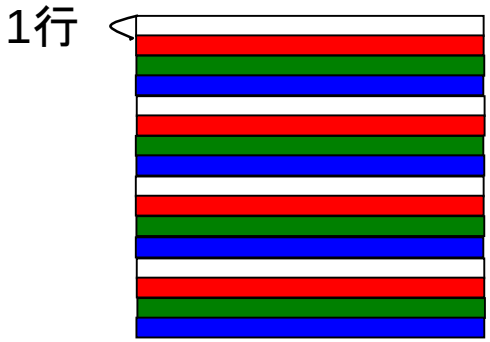
基本演算

- 逐次処理では、「データ構造」が重要
- 並列処理においては、「データ分散方法」が重要になる！
  1. 各PEの「演算負荷」を均等にする
     - ロード・バランシング： 並列処理の基本操作の一つ
     - 粒度調整
  2. 各PEの「利用メモリ量」を均等にする
  3. 演算に伴う通信時間を短縮する
  4. 各PEの「データ・アクセスパターン」を高速な方式にする
     （＝逐次処理におけるデータ構造と同じ）
- 行列データの分散方法
  - ＜次元レベル＞： １次元分散方式、２次元分散方式
  - ＜分割レベル＞： ブロック分割方式、サイクリック（循環）分割方式

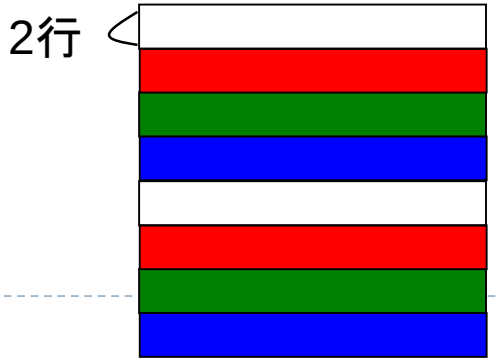
1次元分散



- (行方向) ブロック分割方式
- (Block, *) 分散方式



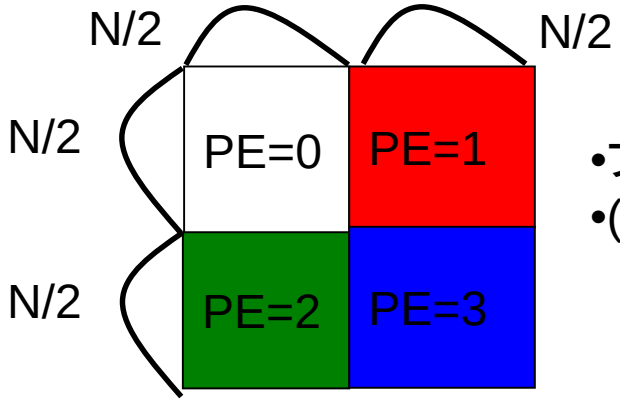
- (行方向) サイクリック分割方式
- (Cyclic, *) 分散方式



- (行方向)ブロック・サイクリック分割方式
- (Cyclic(2), *) 分散方式

この例の「2」: ＜ブロック幅＞とよぶ

2次元分散



•ブロック・ブロック分割方式
•(Block, Block)分散方式

0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3

•サイクリック・サイクリック分割方式
•(Cyclic, Cyclic)分散方式

0	0	1	1	0	0	1	1
0	0	1	1	0	0	1	1
2	2	3	3	2	2	3	3
2	2	3	3	2	2	3	3
0	0	1	1	0	0	1	1
0	0	1	1	0	0	1	1
2	2	3	3	2	2	3	3
2	2	3	3	2	2	3	3

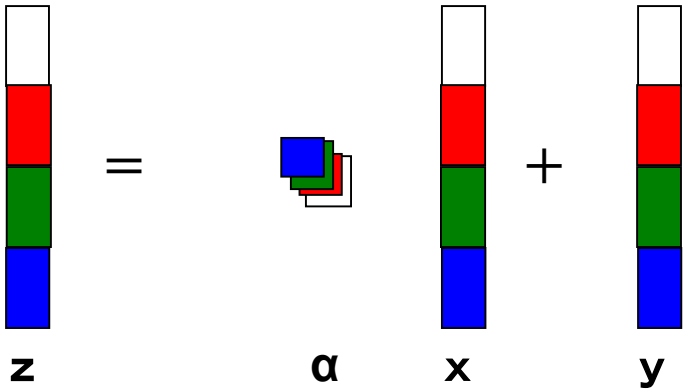
•二次元ブロック・サイクリック分割方式
•(Cyclic(2), Cyclic(2))分散方式

ベクトルどうしの演算

- 以下の演算

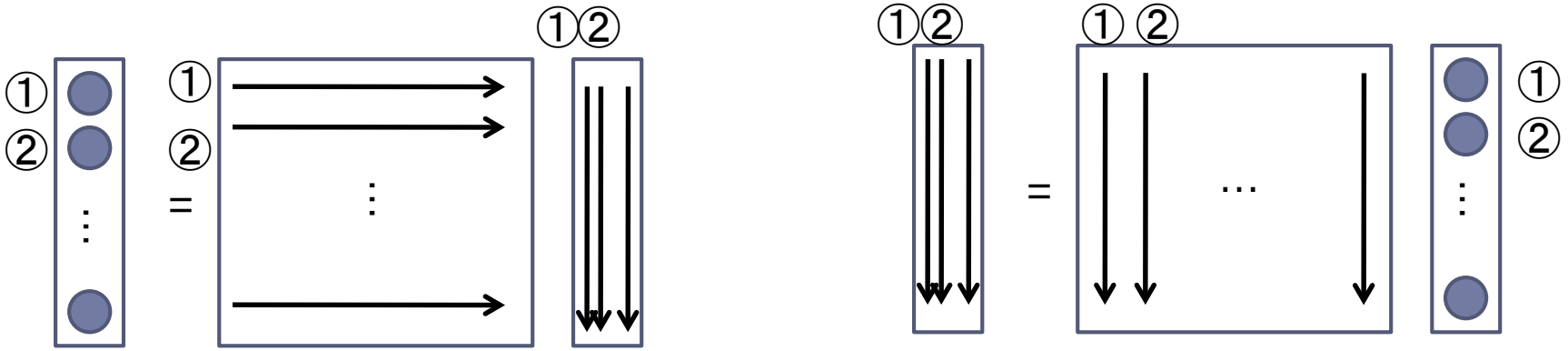
$$z = a\,x + y$$

  - ここで、αはスカラ、z、x、y　はベクトル
- どのようなデータ分散方式でも並列処理が可能
  - ただし、スカラ α は全PEで所有する。
  - ベクトルはO(n)のメモリ領域が必要なのに対し、スカラはO(1)のメモリ領域で大丈夫。
    →スカラメモリ領域は無視可能
  - 計算量：O(N/P)
  - あまり面白くない



行列とベクトルの積

- ＜行方式＞と＜列方式＞がある。
  - ＜データ分散方式＞と＜方式＞組のみ合わせがあり、少し面白い




```
for(i=0;i<n;i++){
    y[i]=0.0;
    for(j=0;j<n;j++){
        y[i] += a[i][j]*x[j];
    }
}
```

＜行方式＞： 自然な実装
C言語向き

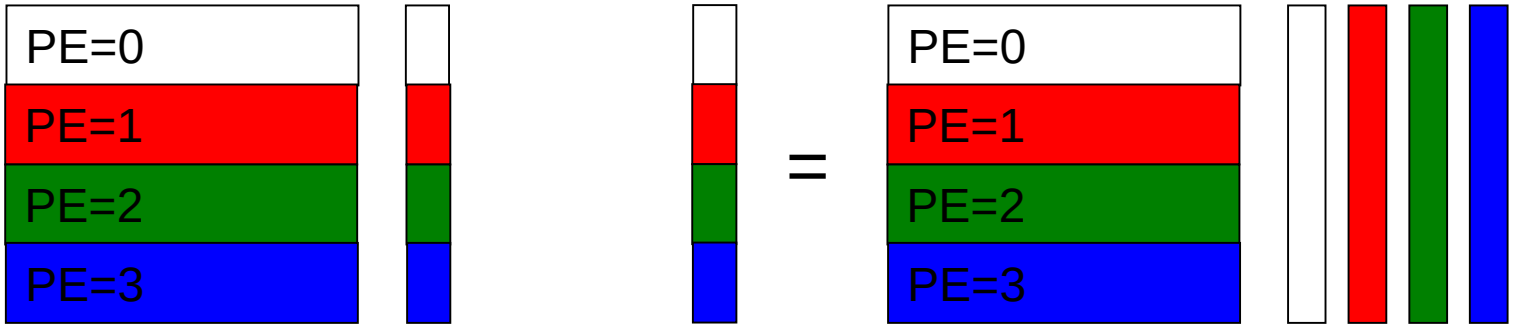
```
for(j=0; j<n; j++) y[j]=0.0;
for(j=0; j<n; j++) {
   for (i=0; i<n; i++) {
      y[i] += a[i][j]*x[j];
   }
}
```

＜列方式＞： Fortran言語向き

行列とベクトルの積

＜行方式の場合＞

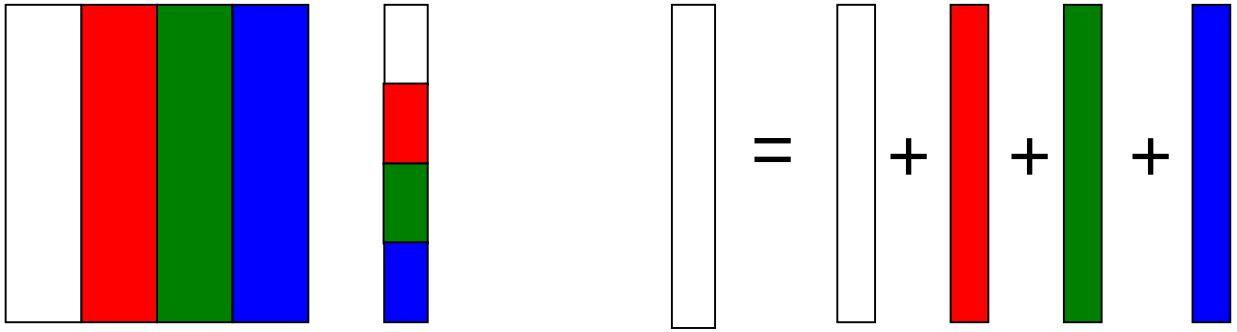
＜行方向分散方式＞　：行方式に向く分散方式



右辺ベクトルを MPI_Allgather関数を利用し、全PEで所有する

各PE内で行列ベクトル積を行う

＜列方向分散方式＞　：ベクトルの要素すべてがほしいときに向く



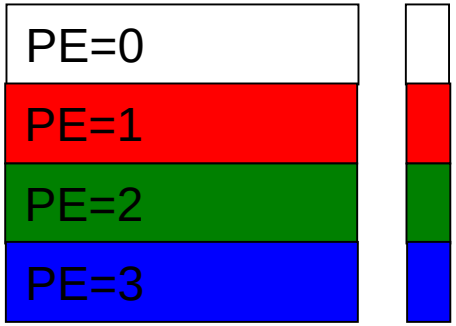
各PE内で行列-ベクトル積を行う

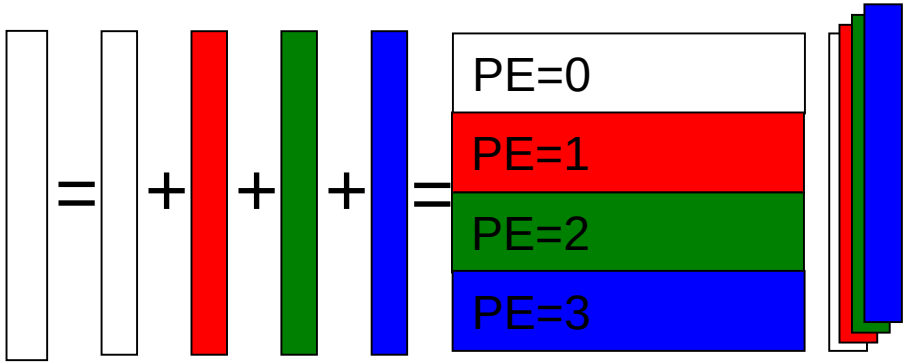
MPI_Reduce関数で総和を求める
（※ある1PEにベクトルすべてが集まる）

行列とベクトルの積

＜列方式の場合＞

＜行方向分散方式＞　：無駄が多く使われない

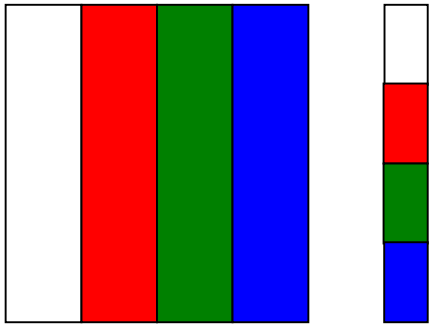


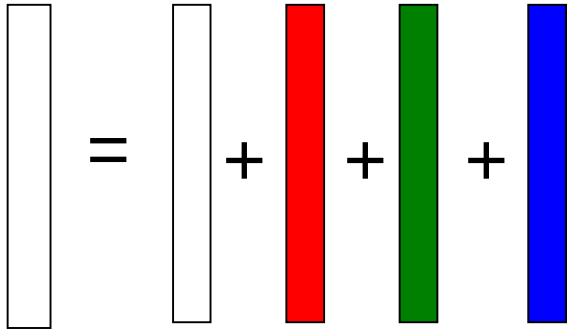


右辺ベクトルを MPI_Allgather関数を利用して、全PEで所有する

結果をMPI_Reduce関数により総和を求める

＜列方向分散方式＞　：列方式に向く分散方式





各PE内で行列-ベクトル積を行う

MPI_Reduce関数で総和を求める
（※ある1PEにベクトルすべてが集まる）

基本的なMPI関数

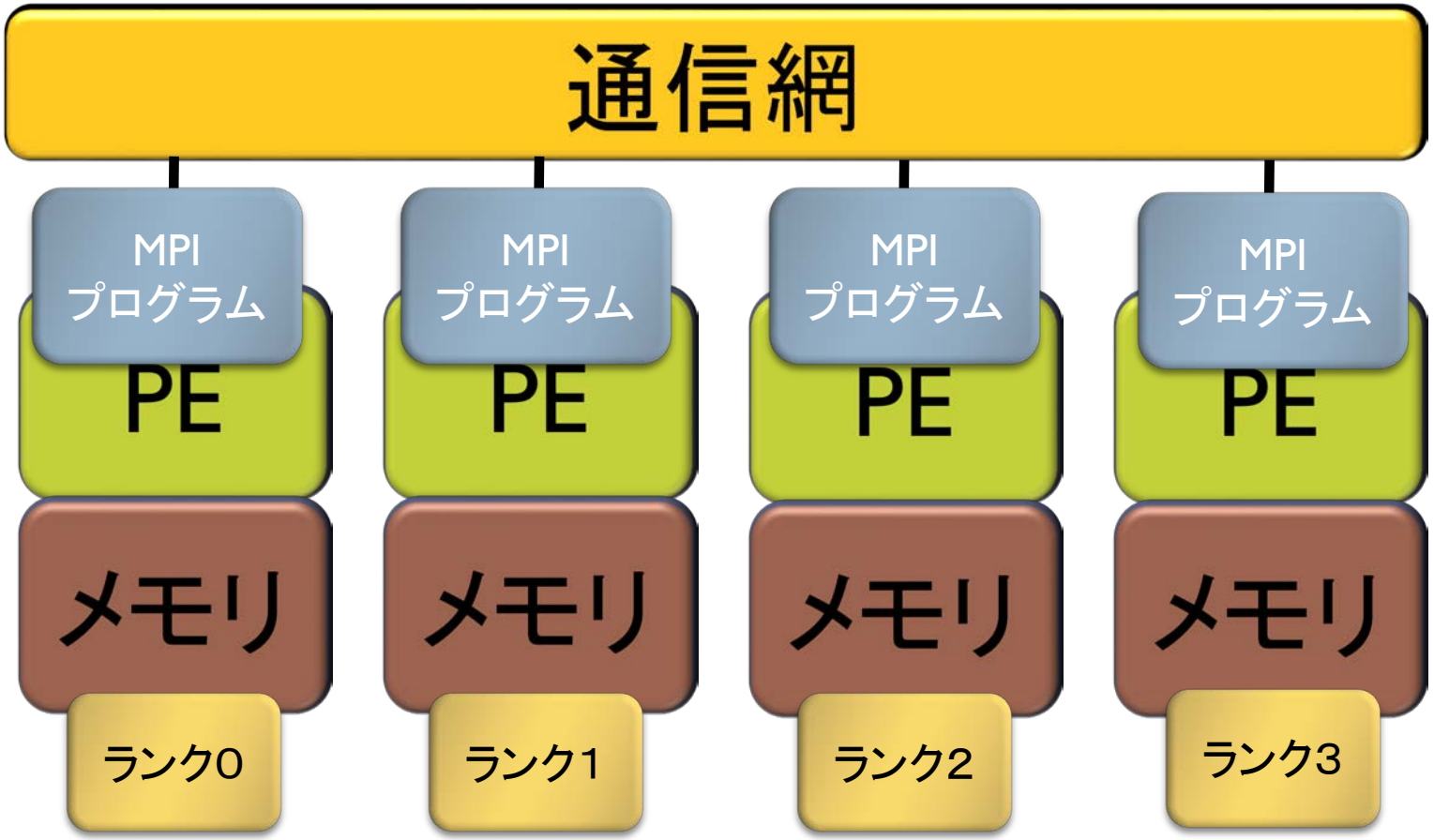
送信、受信のためのインタフェース

略語とMPI用語

- MPIは「プロセス」間の通信を行います。
- プロセスは、HT（ハイパースレッド）などを使わなければ、「プロセッサ」（もしくは、コア）に1対1で割り当てられます。
- 今後、「MPIプロセス」と書くのは長いので、ここではPE（Processer Elementsの略）と書きます。
  - ただし用語として「PE」は、現在あまり使われていません。
- ランク（Rank）
  - 各「MPIプロセス」の「識別番号」のこと。
  - 通常MPIでは、MPI_Comm_rank関数で設定される変数（サンプルプログラムではmyid）に、0～全PE数－1　の数値が入る
  - 世の中の全MPIプロセス数を知るには、MPI_Comm_size関数を使う。

  （サンプルプログラムでは、numprocs に、この数値が入る）

ランクの説明図


通信網
MPI
プログラム
MPI
プログラム
MPI
プログラム
MPI
プログラム
PE
PE
PE
PE
メモリ
メモリ
メモリ
メモリ
ランク0
ランク1
ランク2
ランク3

C言語インターフェースと Fortranインターフェースの違い

- C版は、 整数変数ierr が戻り値

  ierr = MPI_Xxxx(….);

- Fortran版は、最後に整数変数ierrが引数

  call MPI_XXXX(…., ierr)

- システム用配列の確保の仕方
  - C言語

    MPI_Status istatus;
  - Fortran言語

    integer istatus(MPI_STATUS_SIZE)

C言語インターフェースと Fortranインターフェースの違い

- MPIにおける、データ型の指定
  - C言語

MPI_CHAR (文字型) 、 MPI_INT (整数型)、
MPI_FLOAT (実数型)、MPI_DOUBLE(倍精度実数型)

  - Fortran言語

MPI_CHARACTER (文字型) 、MPI_INTEGER (整数型)、
MPI_REAL (実数型)、MPI_DOUBLE_PRECISION(倍精度実数型) 、MPI_COMPLEX(複素数型)

- 以降は、C言語インタフェースで説明する

基礎的なMPI関数—MPI_Recv（１／２）

- ierr = MPI_Recv(recvbuf, icount, idatatype, isource, itag, icomm, **istatus**);

  - recvbuf : 受信領域の先頭番地を指定する。
  - icount : 整数型。受信領域のデータ要素数を指定する。
  - idatatype : 整数型。受信領域のデータの型を指定する。
    - MPI_CHAR (文字型) 、MPI_INT (整数型)、MPI_FLOAT (実数型)、 MPI_DOUBLE(倍精度実数型)
  - isource : 整数型。受信したいメッセージを送信するPEのランクを指定する。
    - 任意のPEから受信したいときは、MPI_ANY_SOURCE を指定する。

基礎的なMPI関数—MPI_Recv（２／２）

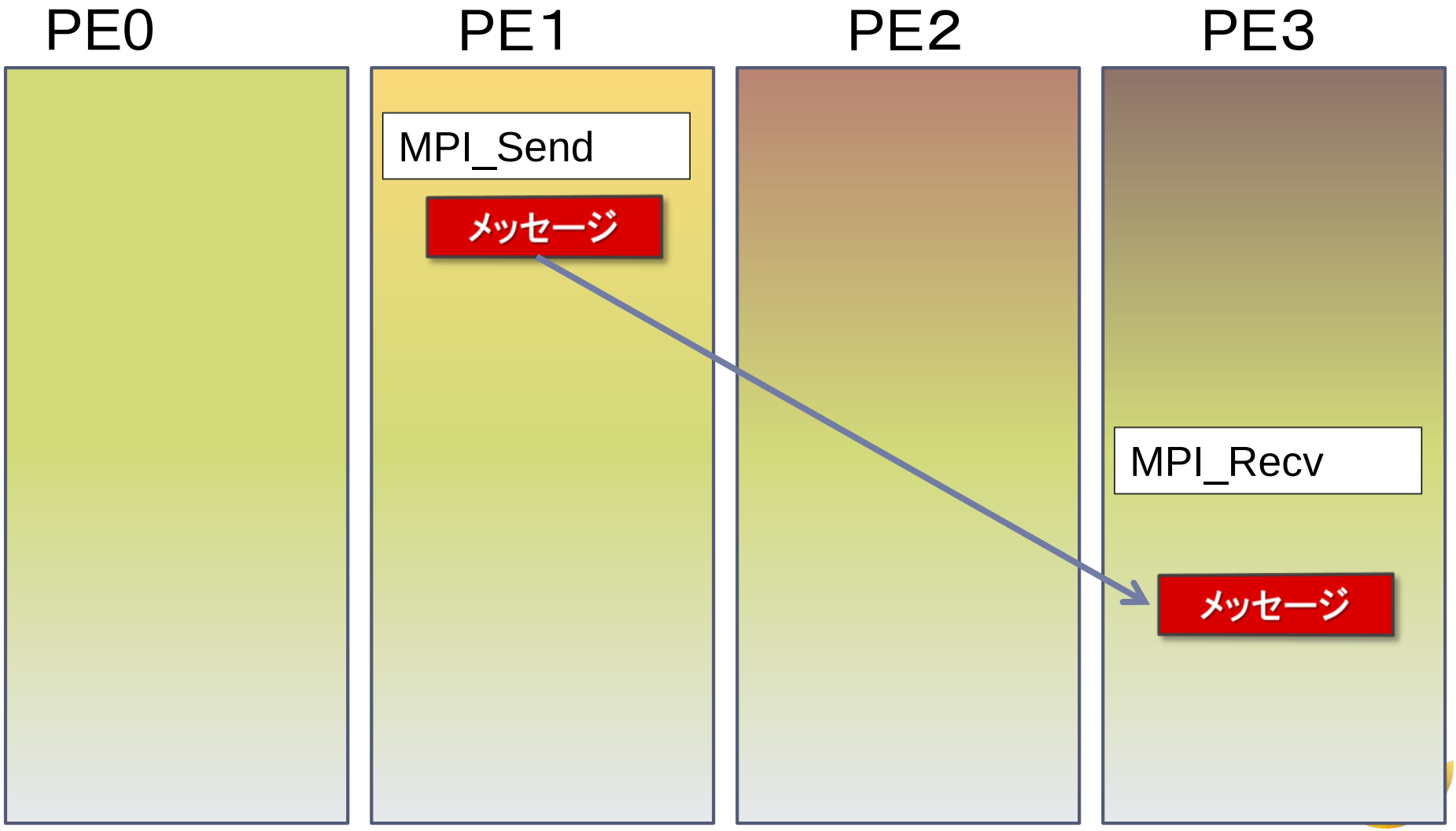
- itag： 整数型。受信したいメッセージに付いているタグの値を指定。
  - 任意のタグ値のメッセージを受信したいときは、MPI_ANY_TAG を指定。
- icomm： 整数型。PE集団を認識する番号であるコミュニケータを指定。
  - 通常ではMPI_COMM_WORLD を指定すればよい。
- istatus： MPI_Status型（整数型の配列）。受信状況に関する情報が入る。かならず専用の型宣言をした配列を確保すること。
  - 要素数がMPI_STATUS_SIZEの整数配列が宣言される。
  - 受信したメッセージの送信元のランクが istatus[MPI_SOURCE]、タグが istatus[MPI_TAG] に代入される。
  - **C言語: MPI_Status istatus;**
  - **Fortran言語: integer istatus(MPI_STATUS_SIZE)**
- ierr(戻り値)： 整数型。エラーコードが入る。

基礎的なMPI関数—MPI_Send

- ierr = MPI_Send(sendbuf, icount, idatatype, idest, itag, icomm);

  - sendbuf : 送信領域の先頭番地を指定
  - icount : 整数型。送信領域のデータ要素数を指定
  - idatatype : 整数型。送信領域のデータの型を指定
  - idest : 整数型。送信したいPEのicomm内でのランクを指定
  - itag : 整数型。受信したいメッセージに付けられたタグの値を指定
  - icomm : 整数型。プロセッサー集団を認識する番号であるコミュニケータを指定
  - ierr (戻り値) : 整数型。エラーコードが入る。

Send－Recvの概念（１対１通信）

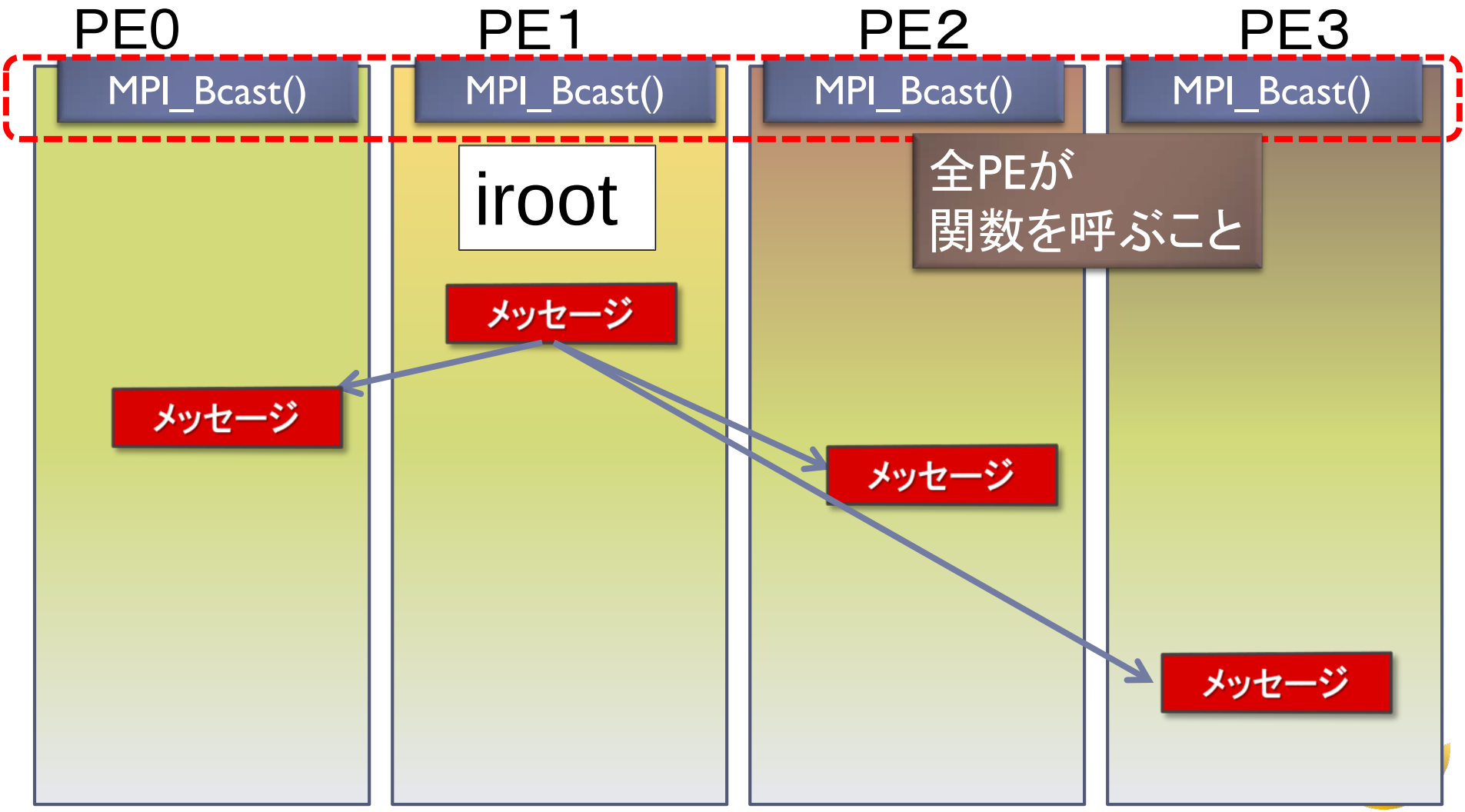


基礎的なMPI関数—MPI_Bcast

- ierr = MPI_Bcast(sendbuf, icount, idatatype,
  iroot, icomm);

  - sendbuf : 送信および受信領域の先頭番地を指定する。
  - icount : 整数型。送信領域のデータ要素数を指定する。
  - idatatype : 整数型。送信領域のデータの型を指定する。
  - iroot : 整数型。送信したいメッセージがあるPEの番号を指定する。全PEで同じ値を指定する必要がある。
  - icomm : 整数型。PE集団を認識する番号であるコミュニケータを指定する。
  - ierr (戻り値) : 整数型。エラーコードが入る。

MPI_Bcastの概念（集団通信）

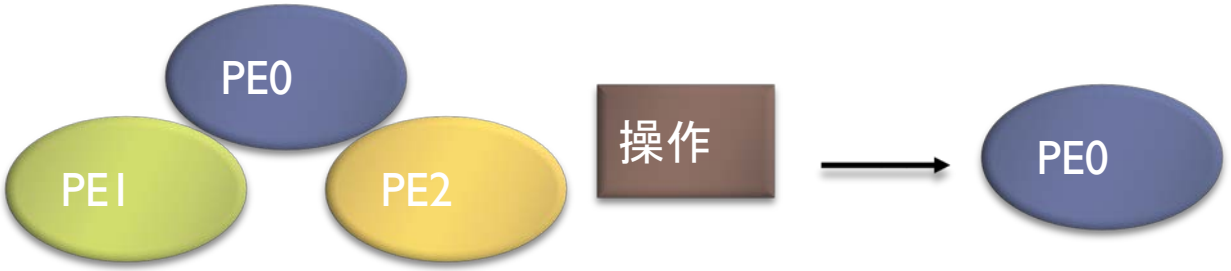


リダクション演算

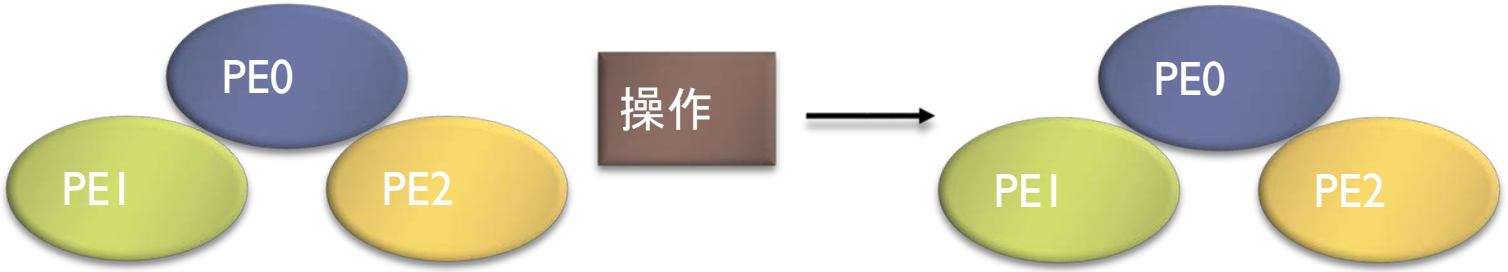
- ＜操作＞によって＜次元＞を減少（リダクション）させる処理
  - 例：　内積演算
    ベクトル（n次元空間）　→　スカラ（1次元空間）
- リダクション演算は、通信と計算を必要とする
  - 集団通信演算（collective communication operation）と呼ばれる
- 演算結果の持ち方の違いで、2種のインタフェースが存在する

リダクション演算

- 演算結果に対する所有PEの違い
  - MPI_Reduce関数
    - リダクション演算の結果を、ある一つのPEに所有させる



  - MPI_Allreduce関数
    - リダクション演算の結果を、全てのPEに所有させる



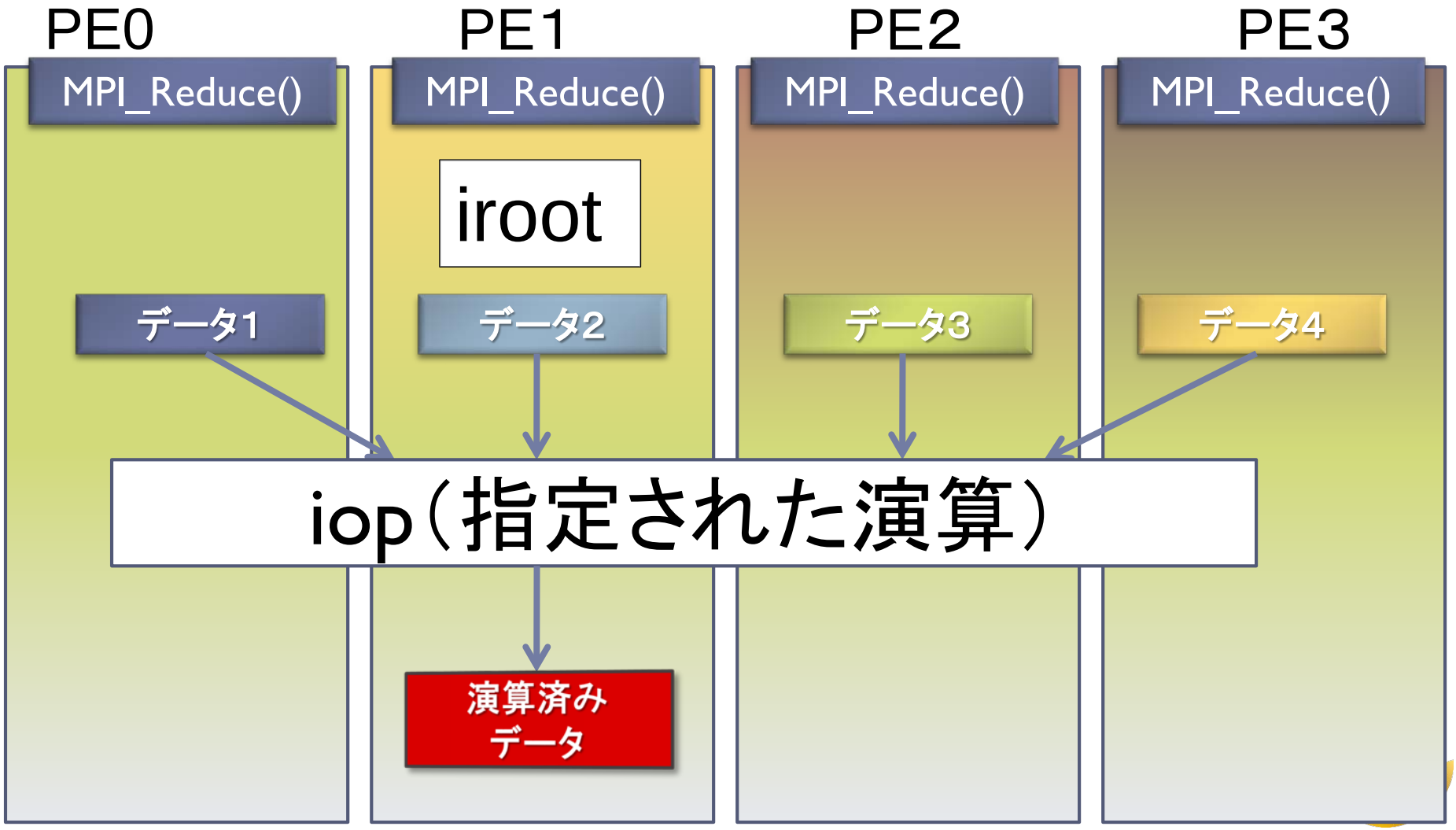
基礎的なMPI関数—MPI_Reduce

- ierr = MPI_Reduce(sendbuf, recvbuf, icount,
  idatatype, iop, iroot, icomm);
  - sendbuf : 送信領域の先頭番地を指定する。
  - recvbuf : 受信領域の先頭番地を指定する。iroot で指定したPEのみで書き込みがなされる。
    送信領域と受信領域は、同一であってはならない。
    すなわち、異なる配列を確保しなくてはならない。
  - icount : 整数型。送信領域のデータ要素数を指定する。
  - idatatype : 整数型。送信領域のデータの型を指定する。
    - (Fortran)＜最小／最大値と位置＞を返す演算を指定する場合は、MPI_2INTEGER(整数型)、 MPI_2REAL (単精度型)、MPI_2DOUBLE_PRECISION(倍精度型) 、を指定する。

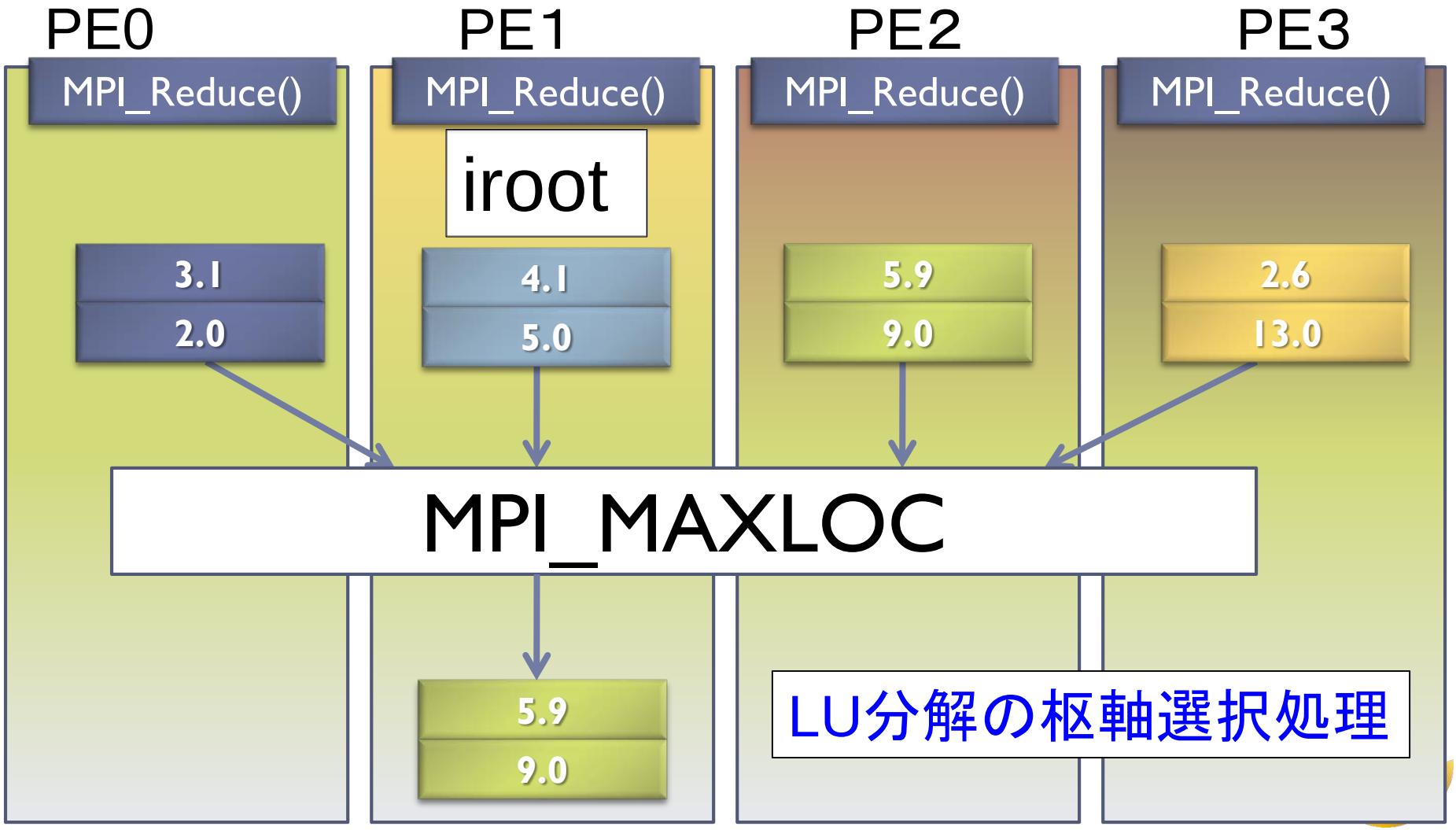
基礎的なMPI関数—MPI_Reduce

- iop： 整数型。演算の種類を指定する。
  - MPI_SUM (総和)、MPI_PROD (積)、MPI_MAX (最大)、MPI_MIN (最小)、 MPI_MAXLOC (最大と位置)、MPI_MINLOC (最小と位置) など。
- iroot： 整数型。結果を受け取るPEのicomm 内でのランクを指定する。全てのicomm 内のPEで同じ値を指定する必要がある。
- icomm： 整数型。PE集団を認識する番号であるコミュニケータを指定する。
- ierr : 整数型。 エラーコードが入る。

MPI_Reduceの概念（集団通信）



MPI_Reduceによる２リスト処理例 (MPI_2DOUBLE_PRECISION と MPI_MAXLOC)



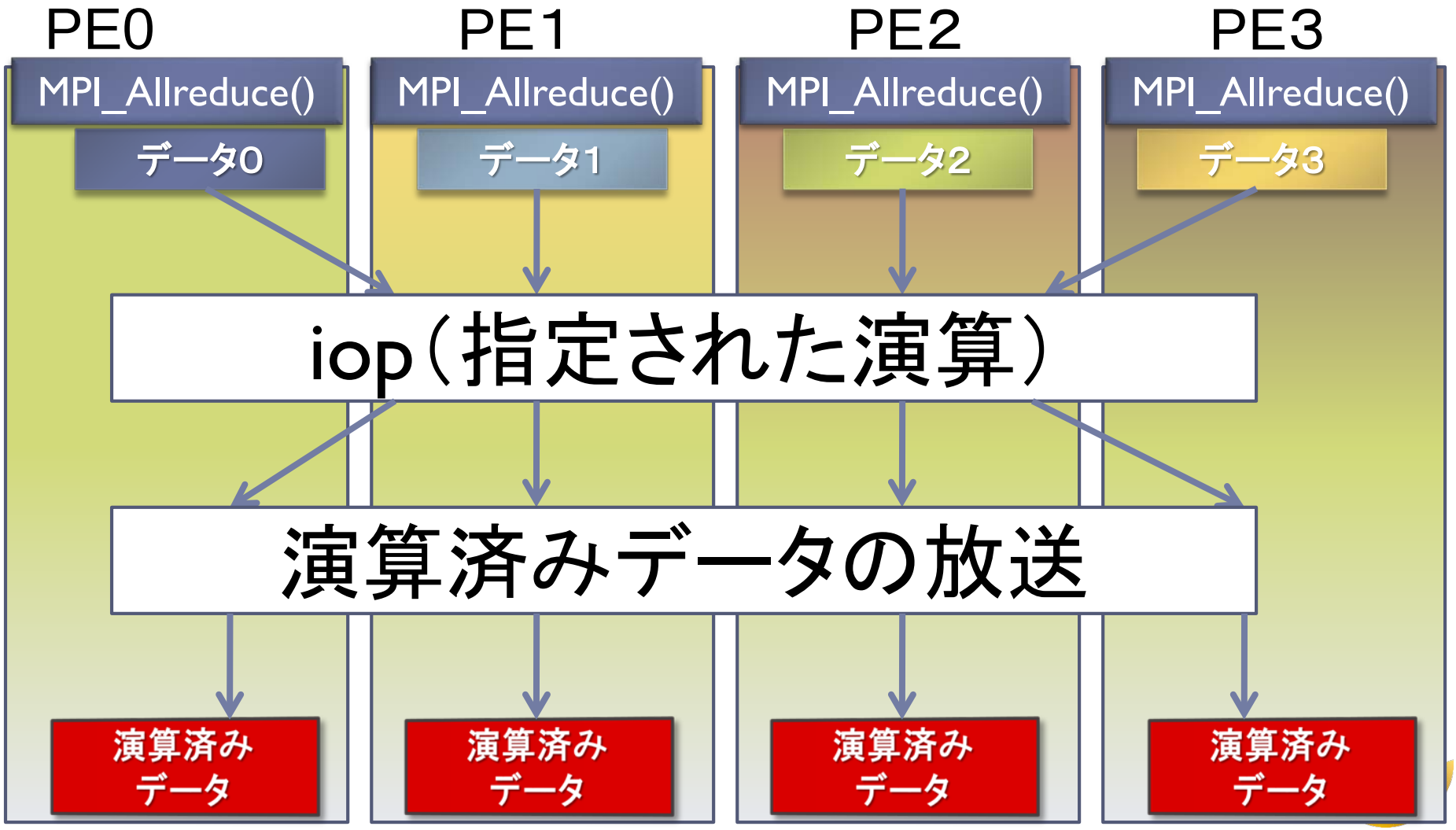
基礎的なMPI関数—MPI_Allreduce

- ierr = MPI_Allreduce(sendbuf, recvbuf, icount, idatatype, iop, icomm);
  - sendbuf：送信領域の先頭番地を指定する。
  - recvbuf：受信領域の先頭番地を指定する。iroot で指定したPEのみで書き込みがなされる。
    送信領域と受信領域は、同一であってはならない。
    すなわち、異なる配列を確保しなくてはならない。
  - icount：整数型。送信領域のデータ要素数を指定する。
  - idatatype：整数型。送信領域のデータの型を指定する。
    - 最小値や最大値と位置を返す演算を指定する場合は、MPI_2INT(整数型)、MPI_2FLOAT (単精度型)、MPI_2DOUBLE(倍精度型) を指定する。

基礎的なMPI関数—MPI_Allreduce

- iop： 整数型。演算の種類を指定する。
  - MPI_SUM (総和)、MPI_PROD (積)、MPI_MAX (最大)、MPI_MIN (最小)、 MPI_MAXLOC (最大と位置)、MPI_MINLOC (最小と位置) など。
- icomm： 整数型。PE集団を認識する番号であるコミュニケータを指定する。
- ierr : 整数型。 エラーコードが入る。

MPI_Allreduceの概念（集団通信）

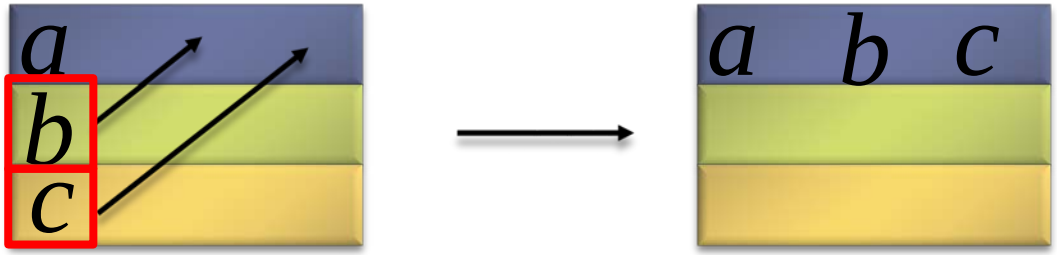


リダクション演算

- 性能について
  - リダクション演算は、1対1通信に比べ遅い
    - プログラム中で多用すべきでない！
  - MPI_Allreduce は MPI_Reduce に比べ遅い
    - MPI_Allreduce は、放送処理が入る。
    - なるべく、MPI_Reduce を使う。

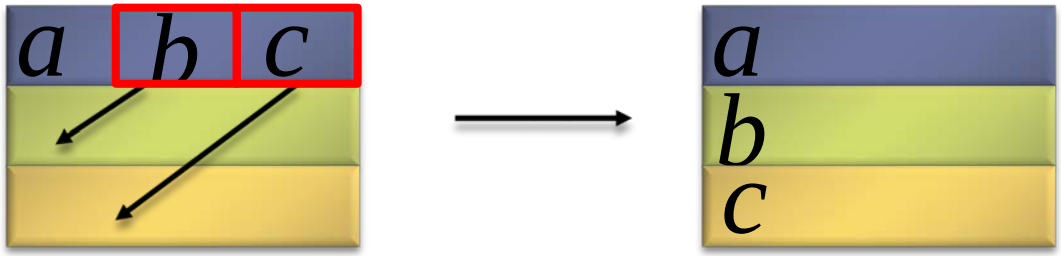
行列の転置

- 行列 A が（Block, * ）分散されているとする。
- 行列 A の転置行列 A^T を作るには、MPIでは次の2通りの関数を用いる
  - MPI_Gather関数



集めるメッセージサイズが各PEで均一のとき使う

  - MPI_Scatter関数



集めるサイズが各PEで均一でないときは：
MPI_GatherV関数
MPI_ScatterV関数

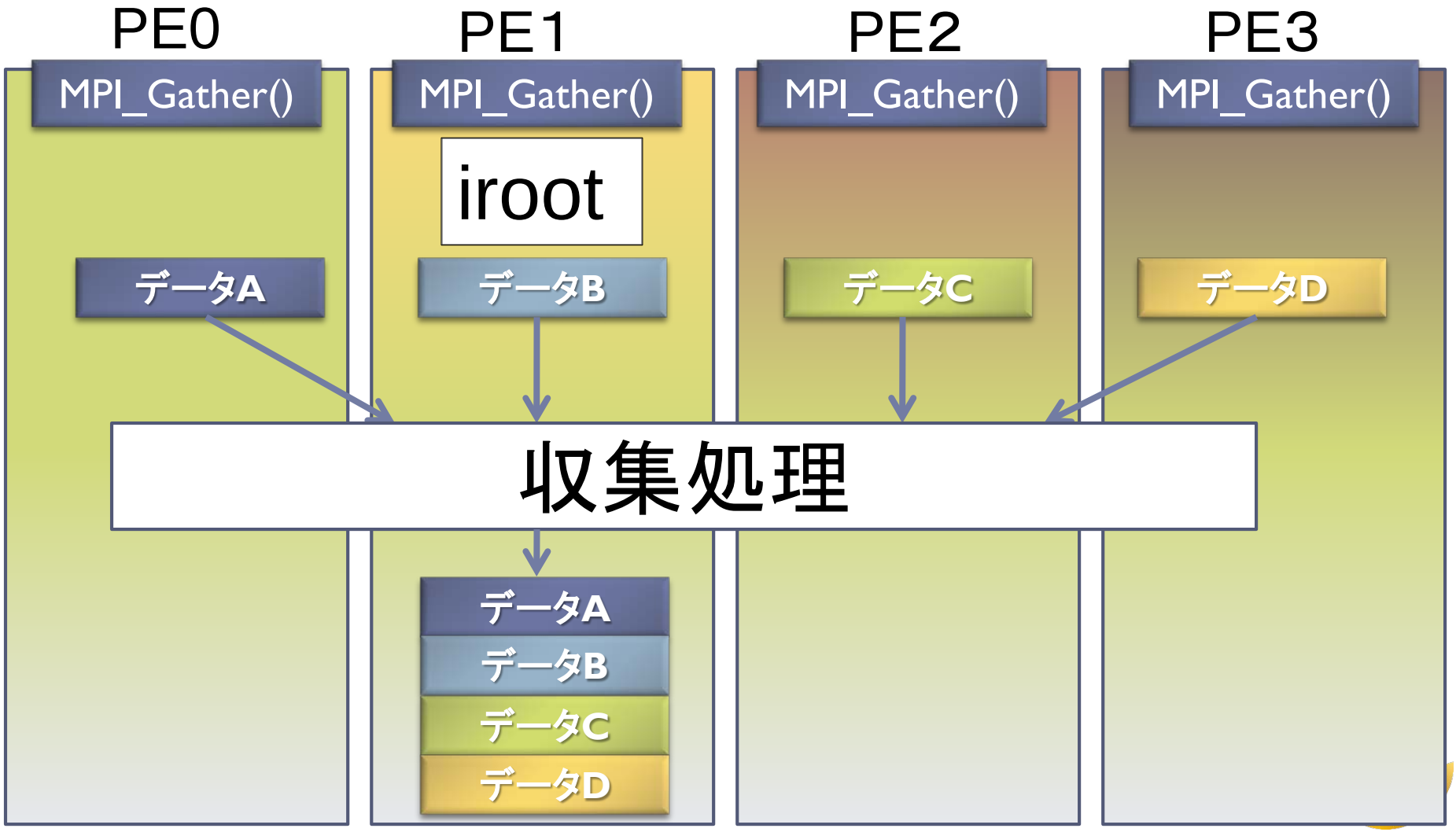
基礎的なMPI関数—MPI_Gather

- ierr = MPI_Gather (sendbuf, isendcount, isendtype,
  recvbuf, irecvcount, irecvtype, iroot, icomm);
  - sendbuf : 送信領域の先頭番地を指定する。
  - isendcount: 整数型。送信領域のデータ要素数を指定する。
  - isendtype : 整数型。送信領域のデータの型を指定する。
  - recvbuf : 受信領域の先頭番地を指定する。iroot で指定したPEのみで書き込みがなされる。
    - なお原則として、送信領域と受信領域は、同一であってはならない。すなわち、異なる配列を確保しなくてはならない。
  - irecvcount: 整数型。受信領域のデータ要素数を指定する。
    - この要素数は、1PE当たりの送信データ数を指定すること。
    - MPI_Gather 関数では各PEで異なる数のデータを収集することはできないので、同じ値を指定すること。

基礎的なMPI関数—MPI_Gather

- irecvtype： 整数型。受信領域のデータ型を指定する。
- iroot： 整数型。収集データを受け取るPEのicomm 内でのランクを指定する。
  - 全てのicomm 内のPEで同じ値を指定する必要がある。
- icomm： 整数型。PE集団を認識する番号であるコミュニケータを指定する。
- ierr： 整数型。エラーコードが入る。

MPI_Gatherの概念（集団通信）



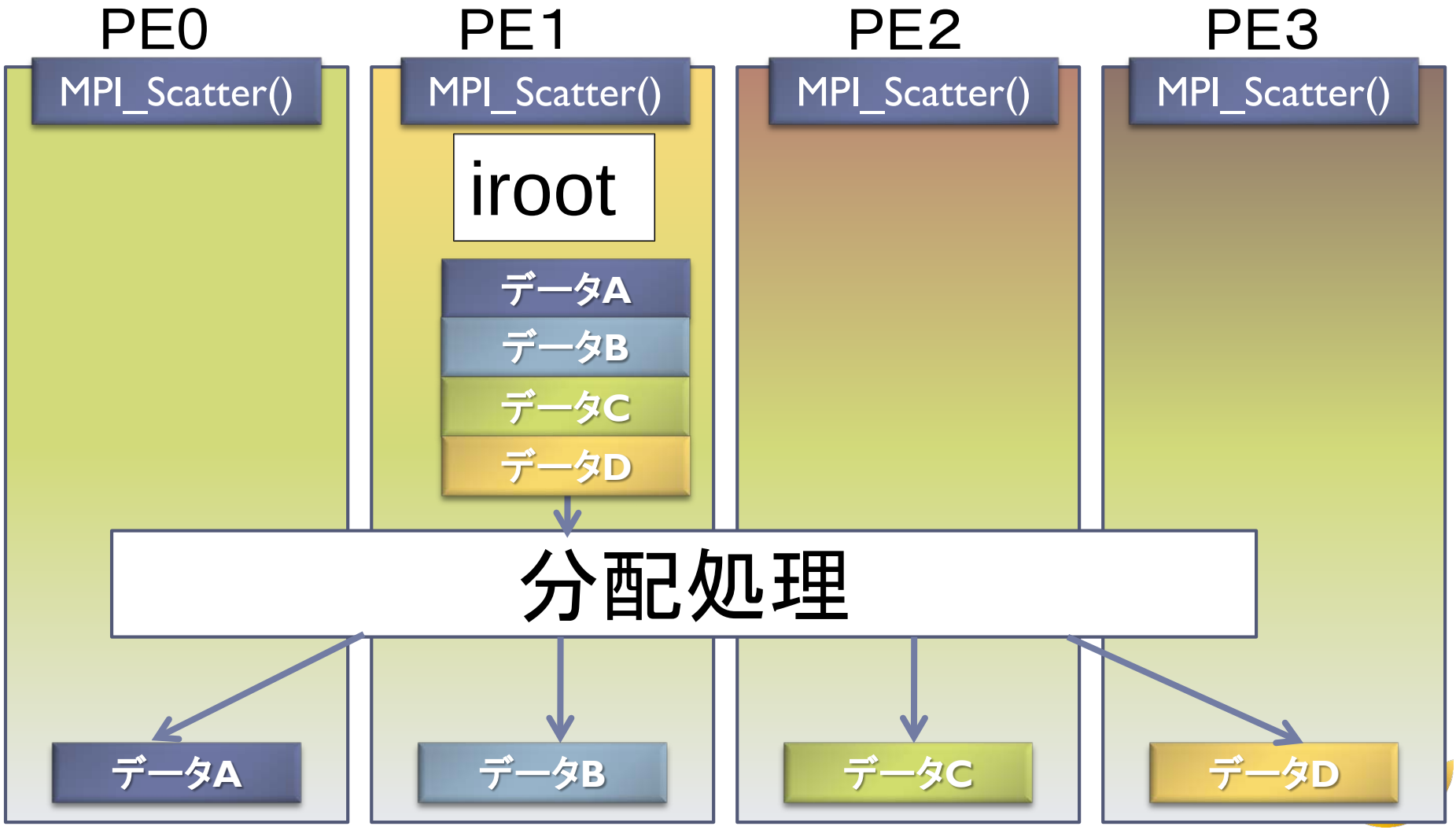
基礎的なMPI関数—MPI_Scatter

- ierr = MPI_Scatter (sendbuf, isendcount, isendtype, recvbuf, irecvcount, irecvtype, iroot, icomm);
  - sendbuf : 送信領域の先頭番地を指定する。
  - isendcount: 整数型。送信領域のデータ要素数を指定する。
    - この要素数は、1PE当たりに送られる送信データ数を指定すること。
    - MPI_Scatter 関数では各PEで異なる数のデータを分散することはできないので、同じ値を指定すること 。
  - isendtype : 整数型。送信領域のデータの型を指定する。 iroot で指定したPEのみ有効となる。
  - recvbuf : 受信領域の先頭番地を指定する。
    - なお原則として、送信領域と受信領域は、同一であってはならない。すなわち、異なる配列を確保しなくてはならない。
  - irecvcount: 整数型。受信領域のデータ要素数を指定する。

基礎的なMPI関数—MPI_Scatter

- irecvtype： 整数型。受信領域のデータ型を指定する。
- iroot： 整数型。収集データを受け取るPEのicomm 内でのランクを指定する。
  - 全てのicomm 内のPEで同じ値を指定する必要がある。
- icomm： 整数型。PE集団を認識する番号であるコミュニケータを指定する。
- ierr： 整数型。エラーコードが入る。

MPI_Scatterの概念（集団通信）



参考文献

1. MPI並列プログラミング、P.パチェコ 著 ／ 秋葉 博 訳
2. 並列プログラミング虎の巻MPI版、青山幸也 著、
   高度情報科学技術研究機構（RIST） 神戸センター
   （ http://www.hpci-office.jp/pages/seminar_text ）
3. Message Passing Interface Forum
   （ http://www.mpi-forum.org/ ）
4. 並列コンピュータ工学、富田眞治著、昭晃堂（1996）

MPIプログラミング実習（演習）

講義の流れ

1. 行列-行列とは（30分）
2. 行列-行列積のサンプルプログラムの実行
3. サンプルプログラムの説明
4. 演習課題（1）：簡単なもの
5. 演習課題（2）：ちょっと難しいもの（中級）

行列 - 行列積の演習の流れ

- 演習課題（Ⅱ）
  - 簡単なもの（30分程度で並列化）
  - 通信関数が一切不要
- 演習課題（Ⅲ）
  - ちょっと難しい（1時間以上で並列化）
  - 1対1通信関数が必要
  - 演習課題（Ⅰ）が早く終わってしまった方は、やってみてください。

行列-行列積とは

実装により性能向上が見込める基本演算

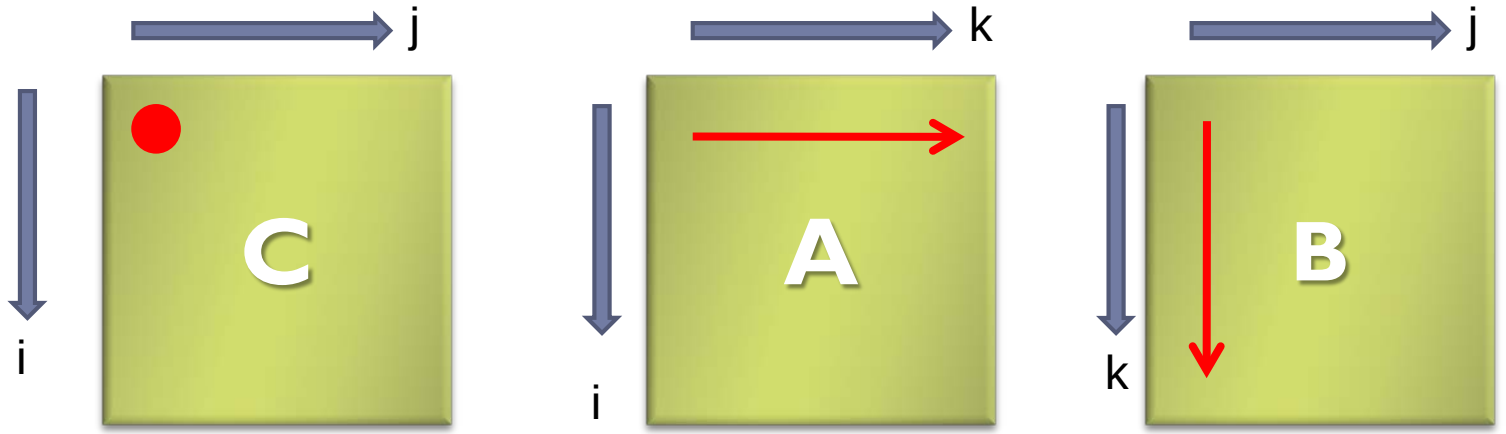
1.5　行列の積

- 行列積　C＝A・B は、コンパイラや計算機のベンチマークに使われることが多い
  - 理由１：実装方式の違いで性能に大きな差がでる
  - 理由２：手ごろな問題である（プログラムし易い）
  - 理由３：科学技術計算の特徴がよく出ている
    1. 非常に長い＜連続アクセス＞がある
    2. キャッシュに乗り切らない＜大規模なデータ＞に対する演算である
    3. メモリバンド幅を食う演算（メモリ・インテンシブ）な処理である

行列積コード例（C言語）

- コード例

```
for (i=0; i<n;  i++)
  for (j=0; j<n; j++)
    for (k=0; k<n; k++)
      C[i][j] += A[i][k] *B[k][j];
```



1.5　行列の積

- 行列積　$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \ (i, j = 1, 2, \ldots, n)$

  の実装法は、次の二通りが知られている:

1. ループ交換法
   - 連続アクセスの方向を変える目的で、行列-行列積を実現する3重ループの順番を交換する
2. ブロック化（タイリング）法
   - キャッシュにあるデータを再利用する目的で、あるまとまった行列の部分データを、何度もアクセスするように実装する

1.5　行列の積

- ループ交換法
  - 行列積のコードは、以下のような3重ループになる（C言語）

```
for(i=0; i<n; i++) {
  for(j=0; j<n; j++) {
    for(k=0; k<n; k++) {
      c[ i ][ j ] = c[ i ][ j ] + a[ i ][ k ] * b[ k][ j ];
    }
  }
}
```

  - 最内部の演算は、外側の3ループを交換しても、計算結果が変わらない
    → 6通りの実現の方法がある

1.5　行列の積

- ループ交換法
  - 行列積のコードは、以下のような3重ループになる（Fortran言語）

```
do i=1, n
  do j=1, n
    do k=1, n
      c( i , j ) = c( i, j) + a( i , k ) * b( k , j )
    enddo
  enddo
enddo
```

  - 最内部の演算は、外側の3ループを交換しても、計算結果が変わらない
    → 6通りの実現の方法がある

1.5　行列の積

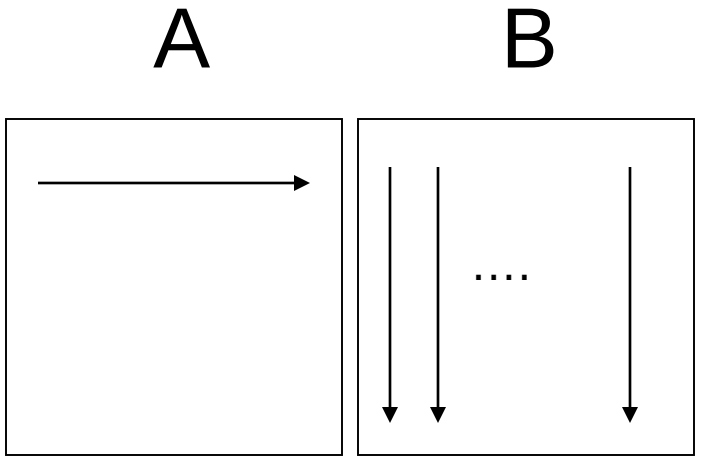
- 行列データへのアクセスパターンから、以下の3種類に分類できる
  1. 内積形式 (inner-product form)
     最内ループのアクセスパタンが
     ＜ベクトルの内積＞と同等
  2. 外積形式 (outer-product form)
     最内ループのアクセスパタンが
     ＜ベクトルの外積＞と同等
  3. 中間積形式 (middle-product form)
     内積と外積の中間

1.5　行列の積

- 内積形式 (inner-product form)
  - ijk, jikループによる実現(C言語)

```
for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
        dc = 0.0;
        for (k=0; k<n; k++){
            dc = dc + A[ i ][ k ] * B[ k ][ j ];
        }
        C[ i ][ j ]= dc;
    }
}
```

※以降、最外のループからの変数の順番で実装法を呼ぶ。たとえば上記のコードは＜ijkループ＞。

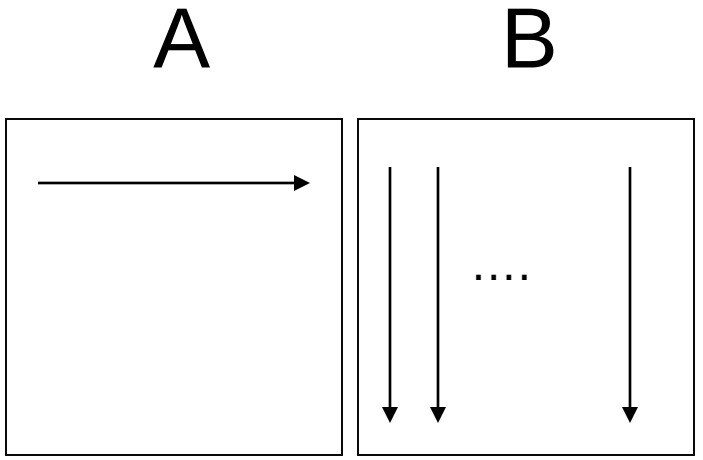


●行方向と列方向のアクセスあり
→行方向・列方向格納言語の
両方で性能低下要因
解決法:
A, Bどちらか一方を転置しておく

1.5　行列の積

- 内積形式 (inner-product form)
  - ijk, jikループによる実現(Fortran言語)

```
do i=1, n
  do j=1, n
    dc = 0.0d0
    do k=1, n
      dc = dc + A( i , k ) * B( k , j )
    enddo
    C( i , j ) = dc
  enddo
enddo
```



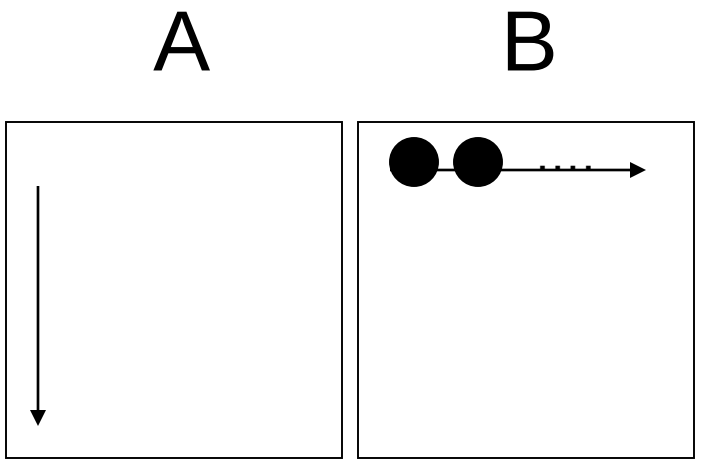
●行方向と列方向のアクセスあり
→行方向・列方向格納言語の
両方で性能低下要因
解決法:
A, Bどちらか一方を転置しておく

※以降、最外のループからの変数の順番で実装法を呼ぶ。たとえば上記のコードは＜ijkループ＞。

1.5　行列の積

- 外積形式 (outer-product form)
  - kij, kjiループによる実現（C言語）

```
for (i=0; i<n; i++) {
   for (j=0; j<n; j++) {
      C[ i ][ j ] = 0.0;
   }
}
for (k=0; k<n; k++) {
  for (j=0; j<n; j++) {
     db = B[ k ][ j ];
     for (i=0; i<n; i++) {
      C[ i ][ j ]= C[ i ][ j ]+ A[ i ][ k ]* db;
     }
  }
}
```

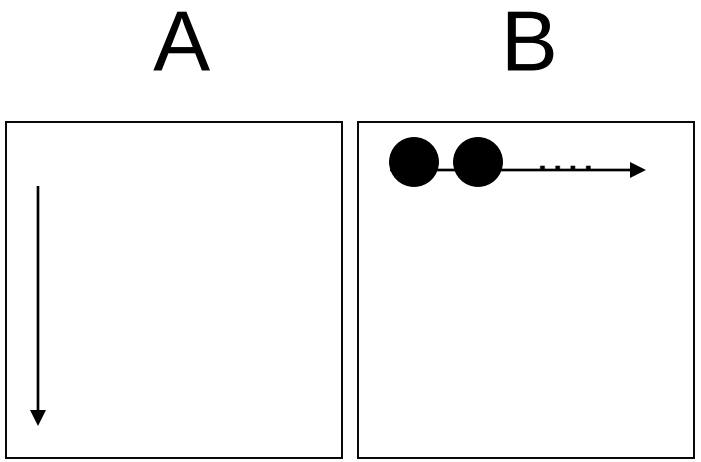


●kjiループでは
列方向アクセスがメイン
→列方向格納言語向き
（Fortran言語）

1.5　行列の積

- 外積形式 (outer-product form)
  - kij, kjiループによる実現（Fortran言語）

```
do i=1, n
  do j=1, n
    C( i , j ) = 0.0d0
  enddo
enddo
do k=1, n
  do j=1, n
    db = B( k , j )
    do i=1, n
      C( i , j ) = C( i , j )+ A( i , k ) * db
    enddo
  enddo
enddo
```

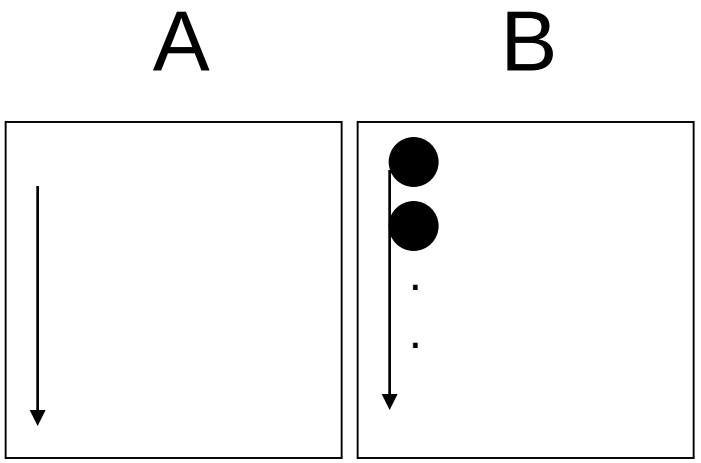


●kjiループでは
列方向アクセスがメイン
→列方向格納言語向き
（Fortran言語）

1.5　行列の積

- 中間積形式 (middle-product form)
  - ikj, jkiループによる実現(C言語)

```
for (j=0; j<n; j++) {
    for (i=0; i<n; i++) {
        C[ i ][ j ] = 0.0;
    }
    for (k=0; k<n; k++) {
      db = B[ k ][ j ];
      for (i=0; i<n; i++) {
        C[ i ][ j ] = C[ i ][ j ] + A[ i ][ k ] * db;
      }
    }
}
```

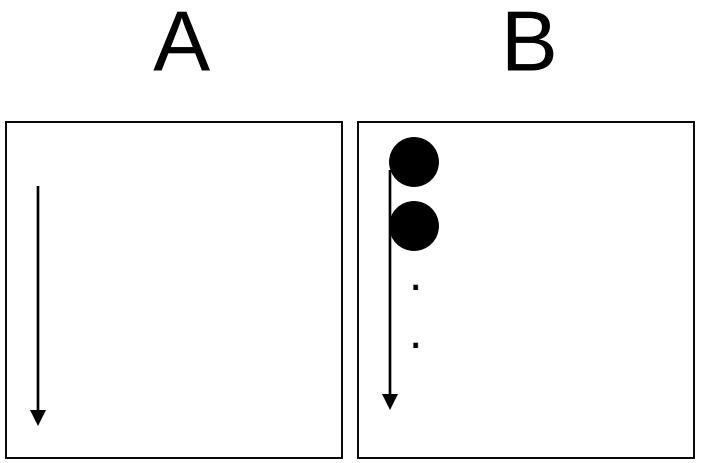


●jkiループでは
全て列方向アクセス
→列方向格納言語に
最も向いている
(Fortran言語)

1.5　行列の積

- 中間積形式 (middle-product form)
  - ikj, jkiループによる実現（Fortran言語）

```
do j=1, n
  do i=1, n
    C( i , j ) = 0.0d0
  enddo
  do k=1, n
    db = B( k , j )
    do i=1, n
      C( i , j ) = C( i , j ) + A( i , k ) * db
    enddo
  enddo
enddo
```

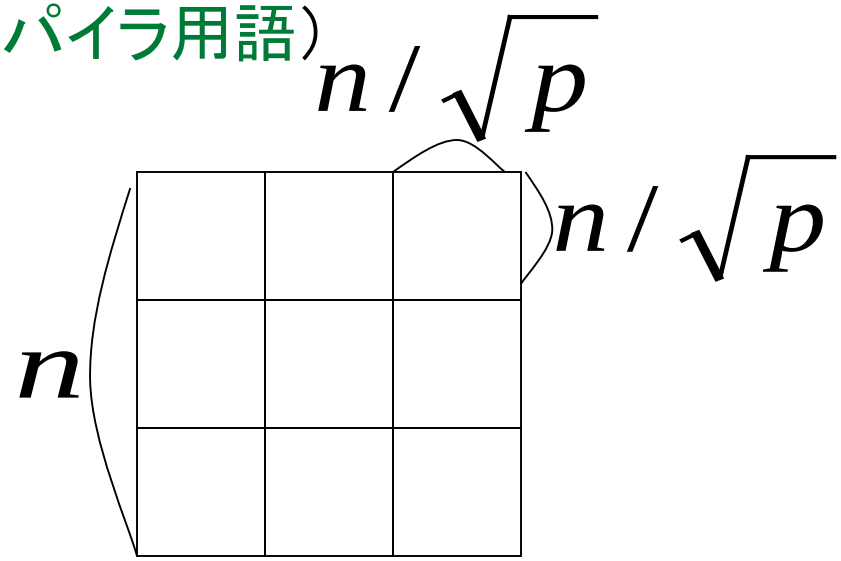


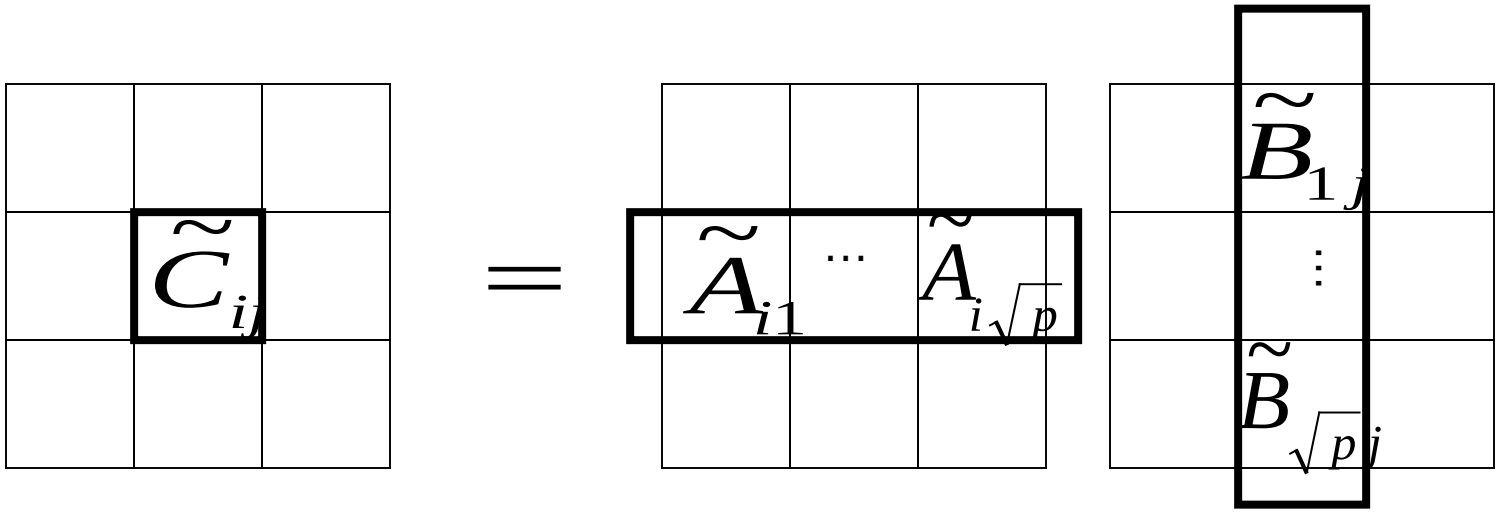
●jkiループでは
全て列方向アクセス
→列方向格納言語に
最も向いている
（Fortran言語）

1.5　行列の積

- 小行列ごとの計算に分けて（配列を用意し）計算
（＜ブロック化＞, ＜タイリング＞:コンパイラ用語）
- 以下の計算

$$\tilde{C}_{ij} = \sum_{k=1}^{\sqrt{n}} \tilde{A}_{ik} \tilde{B}_{kj}$$





1.5　行列の積

- 各小行列をキャッシュに収まるサイズにする。
  1. ブロック単位で高速な演算が行える
  2. 並列アルゴリズムの変種が構築できる
- 並列行列積アルゴリズムは、データ通信の形態から、以下の2種に分類可能:
  1. セミ・シストリック方式
     - 行列A、Bの小行列の一部をデータ移動（Cannonのアルゴリズム）
  2. フル・シストリック方式
     - 行列A、Bの小行列のすべてをデータ移動（Foxのアルゴリズム）

サンプルプログラムの実行
（行列-行列積）

行列-行列積のサンプルプログラムの注意点

- C言語版/Fortran言語版の共通ファイル名

  Mat-Mat-fx100.tar

行列-行列積のサンプルプログラムの実行

- 以下のコマンドを実行する

  $ cp /center/a49904a/Mat-Mat-fx100.tar ./

  $ tar xvf Mat-Mat-fx100.tar

  $ cd Mat-Mat

- 以下のどちらかを実行

  $ cd C :C言語を使う人

  $ cd F :Fortran言語を使う人

- 以下共通

  $ make

  $ pjsub mat-mat.bash

- 実行が終了したら、以下を実行する

  $ cat mat-mat.bash.oXXXXXX

名古屋大学 NAGOYA UNIVERSITY

行列-行列積のサンプルプログラムの実行（C言語）

- 以下のような結果が見えれば成功

N = 1000

Mat-Mat time = 0.114355 [sec.]

17489.383704 [MFLOPS]

OK!

1コアのみで、17.5GFLOPSの性能

行列-行列積のサンプルプログラムの実行（Fortran言語）

- 以下のような結果が見えれば成功

NN = 1000

Mat-Mat time[sec.] = 0.11861014366614990

MFLOPS = 16861.96418965883

OK!

1コアのみで、16.8GFLOPSの性能

サンプルプログラムの説明

- #define N 1000
  の、数字を変更すると、行列サイズが変更できます
- #define DEBUG 0
  の「0」を「1」にすると、行列-行列積の演算結果が検証できます。
- MyMatMat関数の仕様
  - Double型N×N行列AとBの行列積をおこない、Double型N×N行列Cにその結果が入ります

Fortran言語のサンプルプログラムの注意

- 行列サイズNの宣言は、以下のファイルにあります。

  mat-mat.inc

- 行列サイズ変数が、NNとなっています。

  integer NN

  parameter (NN=1000)

演習課題（1）

- MyMatMat関数を並列化してください。
  - #define N 192
  - #define DEBUG 1

  として、デバッグをしてください。
- 行列A、B、Cは、各PEで重複して、かつ全部（N×N）所有してよいです。

演習課題（１）

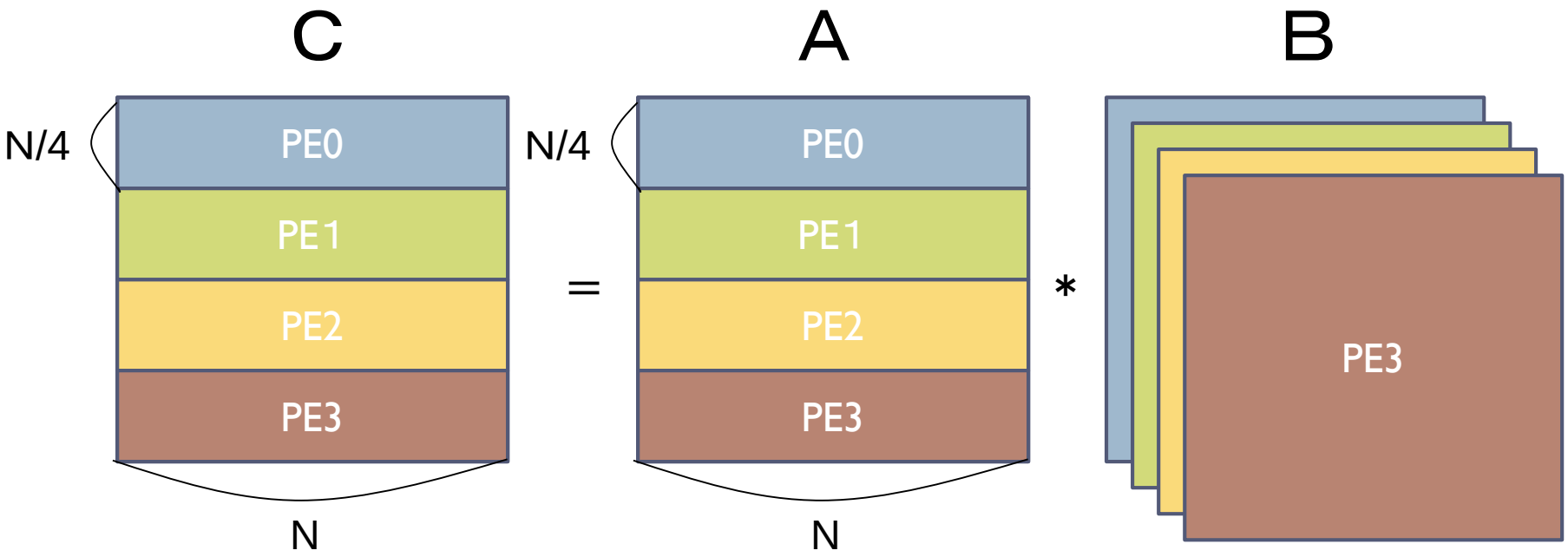
- サンプルプログラムでは、行列A、Bの要素を全部1として、行列-行列積の結果をもつ行列Cの全要素がNであるか調べ、結果検証しています。デバックに活用してください。
  - 行列Cの分散方式により、

演算結果チェックルーチンの並列化が必要

になります。注意してください。

並列化のヒント

- 以下のようなデータ分割にすると、とても簡単です。



- 通信関数は一切不要です。

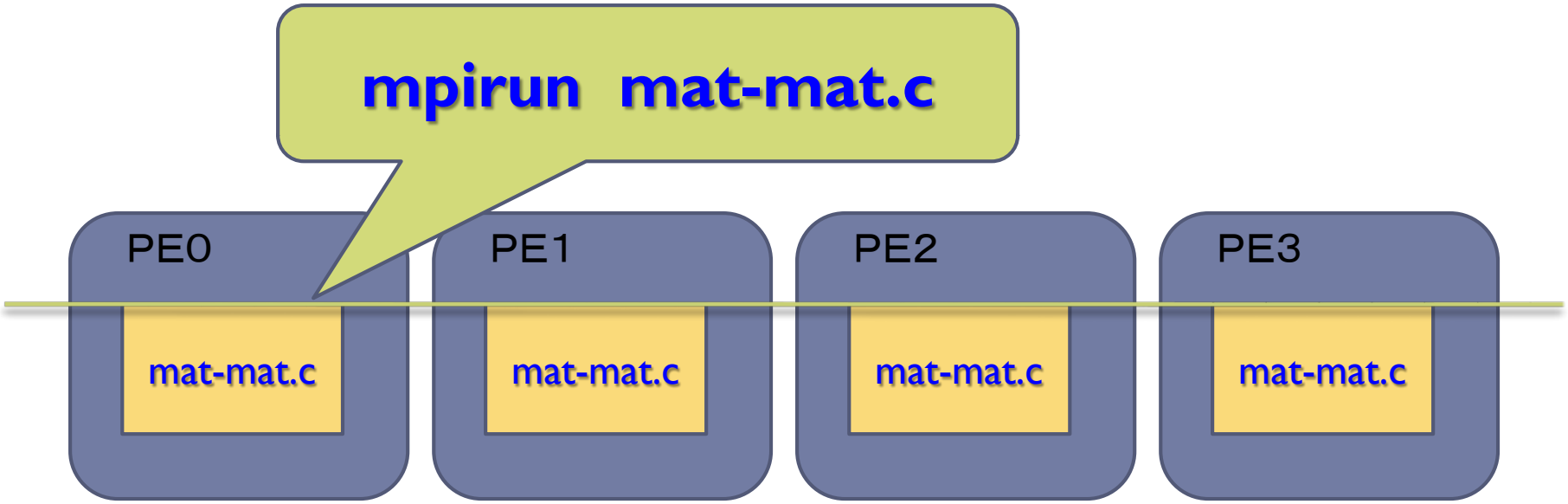
MPI並列化の大前提（再確認）

- SPMD
  - 対象のプログラム（mat-mat.c, mat-mat.f）は、
    - すべてのPEで、かつ、
    - 同時に起動された状態

    から処理が始まる。
- 分散メモリ型並列計算機
  - 各PEは、完全に独立したメモリを持っている。他のPEからは、通信なしには参照できない。（共有メモリではない）

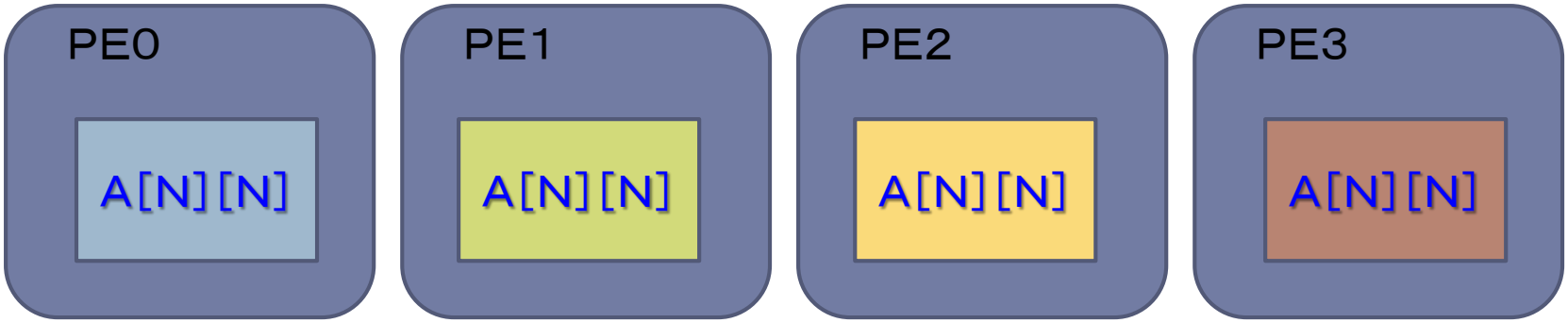
MPI並列化の大前提（再確認）

- 各PEでは、＜同じプログラムが同時に起動＞されて開始されます。

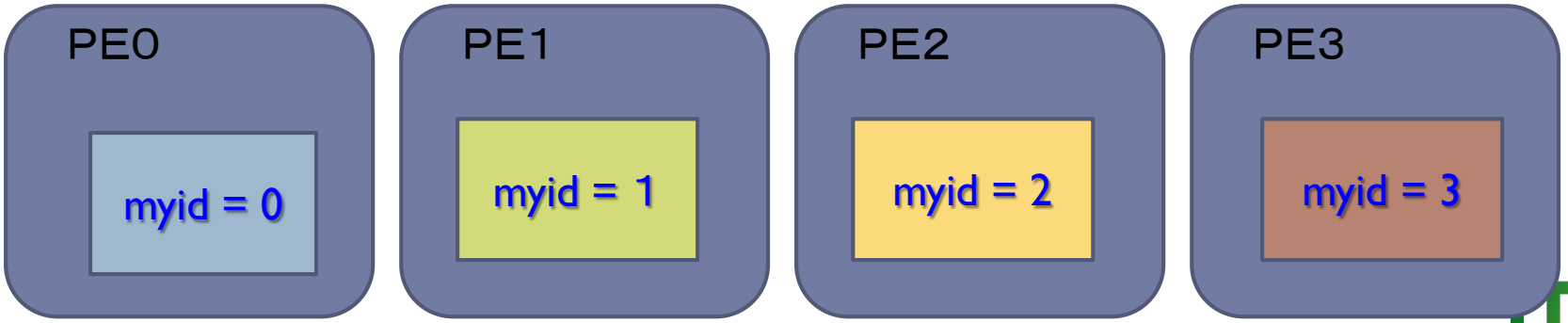


MPI並列化の大前提（再確認）

- 各PEでは、＜別配列が個別に確保＞されます。

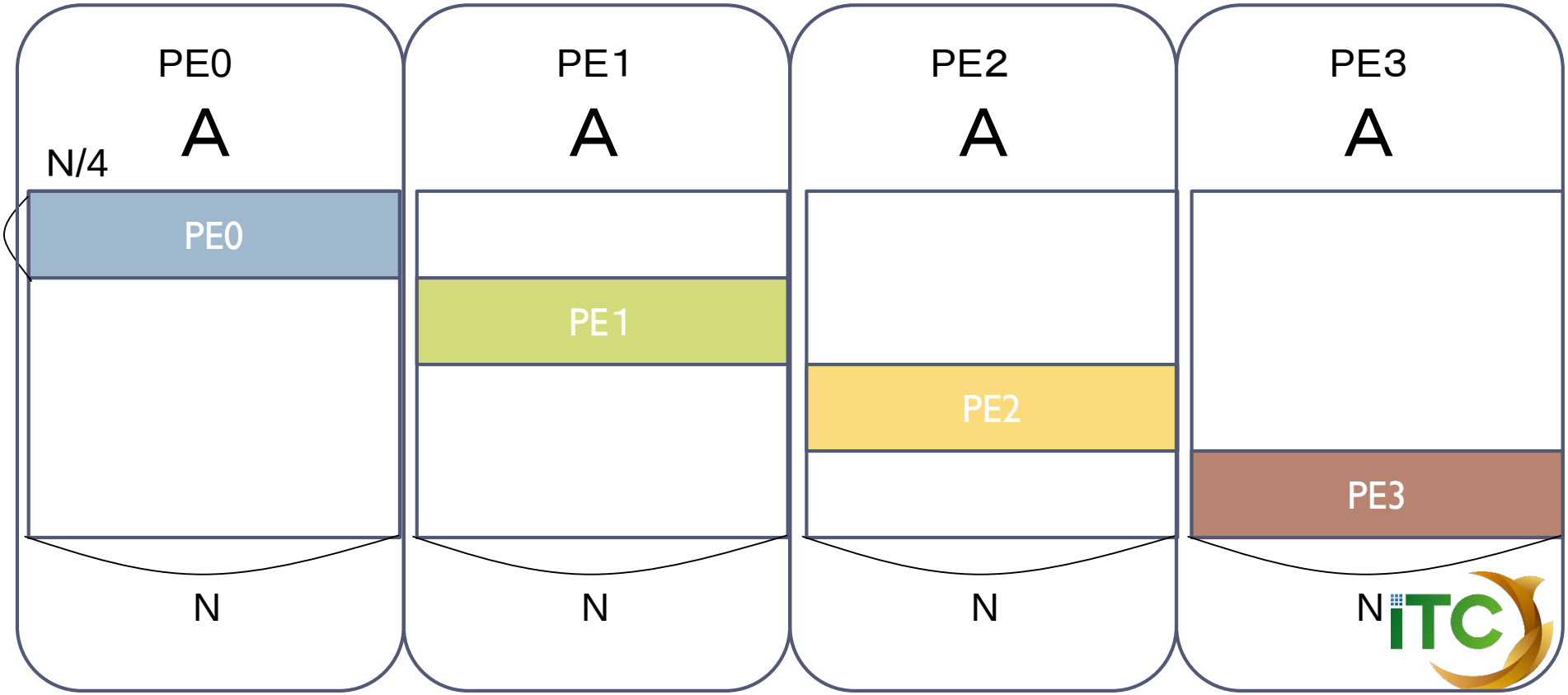


- myid変数は、MPI_Init()関数（もしくは、サブルーチン）が呼ばれた段階で、＜各PE固有の値＞になっています。



各PEでの配列の確保状況

- 実際は、以下のように配列が確保されていて、
  部分的に使うだけになります

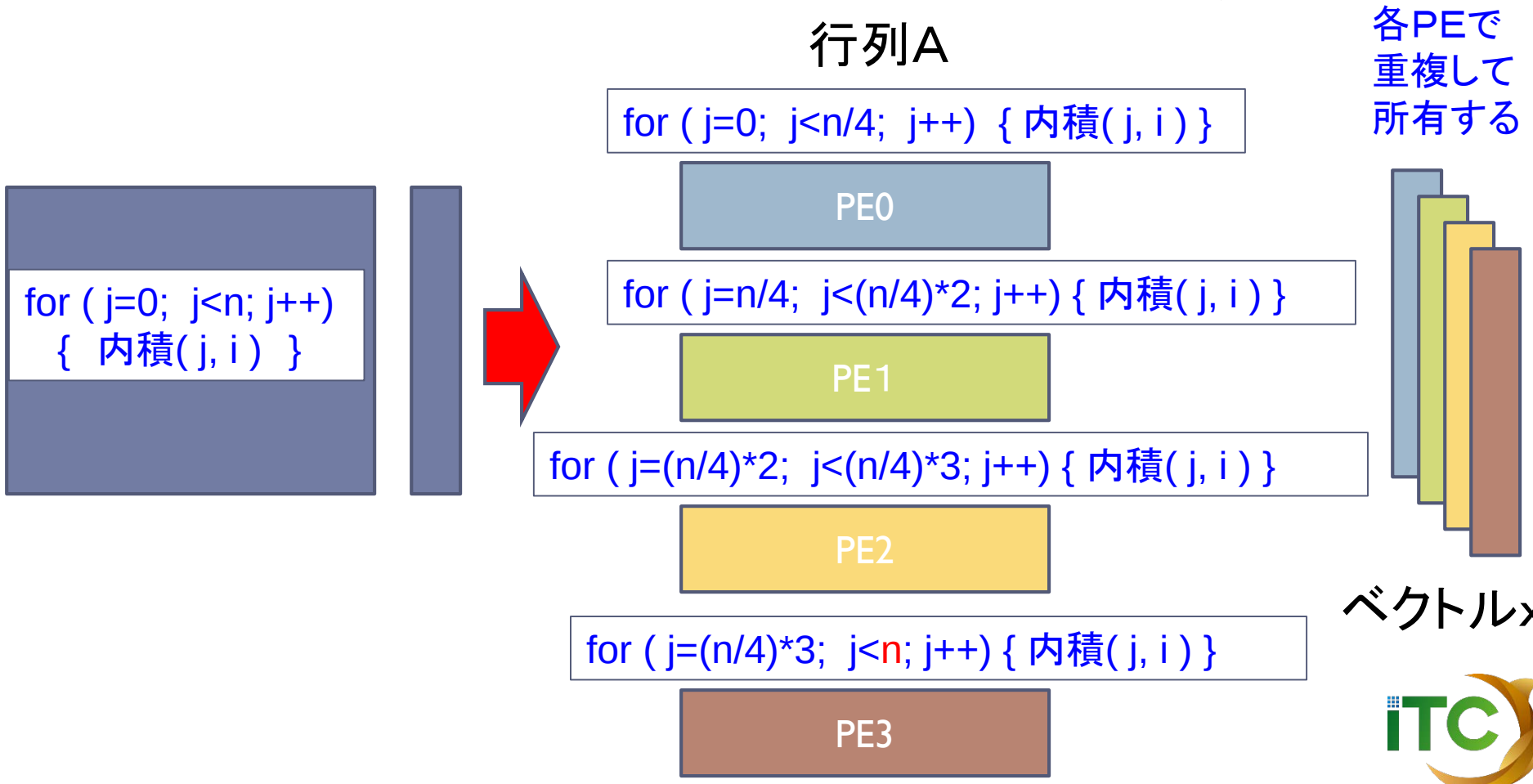


本実習プログラムのTIPS

- myid, numprocs は大域変数です
  - myid (=自分のID)、および、numprocs(=世の中のPE台数)の変数は大域変数です。MyMatVec関数内で、引数設定や宣言なしに、参照できます。
- myid, numprocs の変数を使う必要があります
  - MyMatMat関数を並列化するには、myid 、および、numprocs変数を利用しないと、並列化ができません。

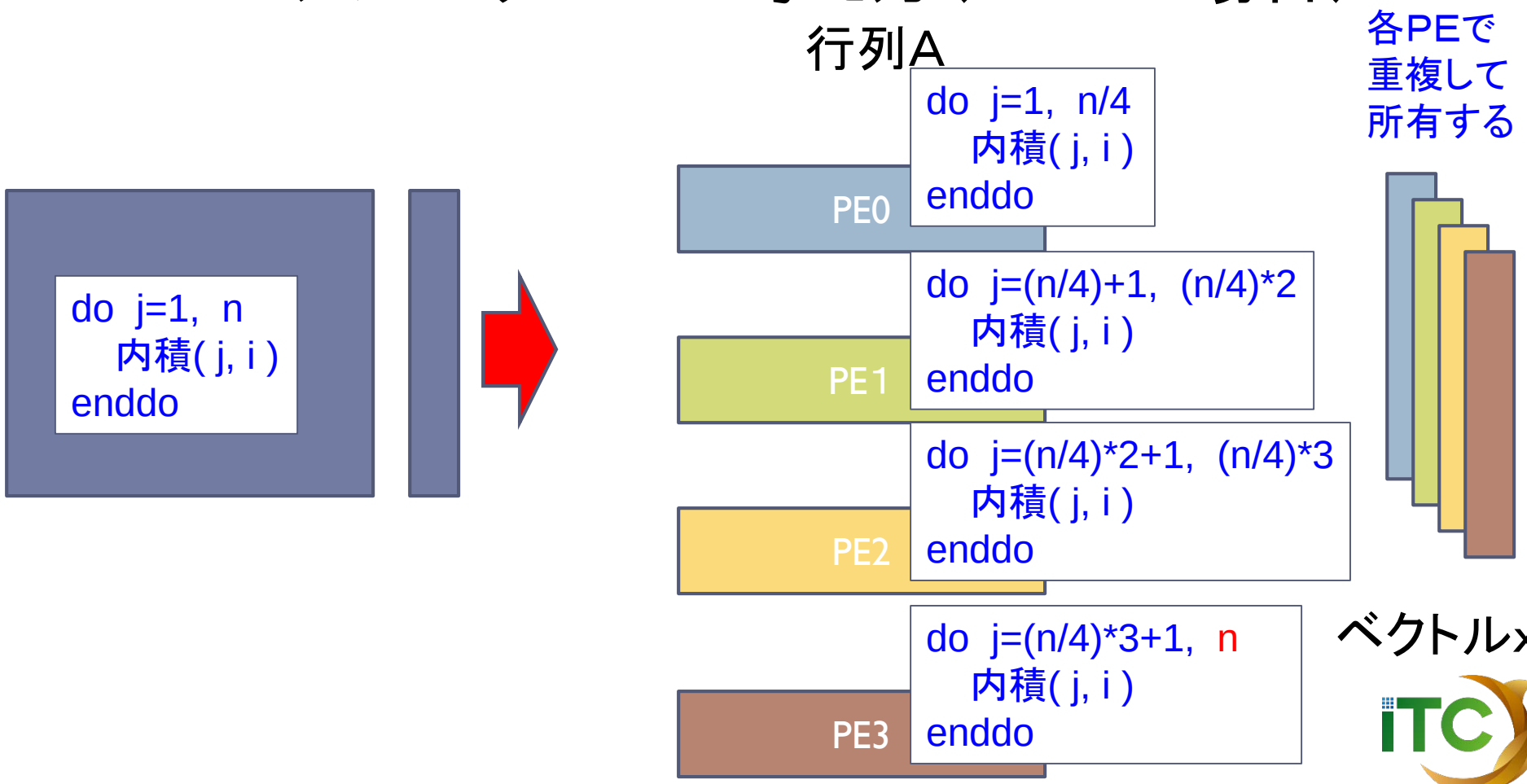
並列化の考え方
（行列-ベクトル積の場合、Ｃ言語）

- SIMDアルゴリズムの考え方（4PEの場合）



並列化の考え方
（行列-ベクトル積の場合、Fortran言語）

- SIMDアルゴリズムの考え方（4PEの場合）



名古屋大学
NAGOYA UNIVERSITY

並列化の方針（C言語）

1. 全PEで行列AをN×Nの大きさ、ベクトルx、yをNの大きさ、確保してよいとする。
2. 各PEは、担当の範囲のみ計算するように、ループの開始値と終了値を変更する。
   - ブロック分散方式では、以下になる
     （n が numprocs で割り切れる場合）

```
ib = n / numprocs;
for ( j=myid*ib;  j<(myid+1)*ib;  j++)    { … }
```

3. （2の並列化が完全に終了したら）各PEで担当のデータ部分しか行列を確保しないように変更する。
   - 上記のループは、以下のようになる。

```
for ( j=0;  j<ib;  j++)  { … }
```

並列化の方針（Fortran言語）

1. 全PEで行列AをN×Nの大きさ、ベクトルx、yをNの大きさ、確保してよいとする。
2. 各PEは、担当の範囲のみ計算するように、ループの開始値と終了値を変更する。
   - ブロック分散方式では、以下になる
     （n が numprocs で割り切れる場合）

```
ib = n / numprocs
do j = myid*ib+1, (myid+1)*ib
    …..
enddo
```

3. （2の並列化が完全に終了したら）各PEで担当のデータ部分しか行列を確保しないように変更する。
   - 上記のループは、以下のようになる。

```
do j=1, ib
    …
enddo
```

実装上の注意

- ループ変数をグローバル変数にすると、性能が出ません。
  必ずローカル変数か、定数（ 2 など）にしてください。

ローカル変数にすること

```
for (i=i_start; i<i_end; i++) {
    …
    …
}
```

参考資料

名古屋大学
NAGOYA UNIVERSITY

サンプルプログラムの実行（行列-行列積（その２））

行列-行列積のサンプルプログラムの注意点

- C言語版/Fortran言語版のファイル名

  Mat-Mat-d-fx100.tar

行列-行列積(2)のサンプルプログラムの実行

- 以下のコマンドを実行する
  $ cp /center/a49904a/Mat-Mat-d-fx100.tar ./
  $ tar xvf Mat-Mat-d-fx100.tar
  $ cd Mat-Mat-d
- 以下のどちらかを実行
  $ cd C ：C言語を使う人
  $ cd F ：Fortran言語を使う人
- 以下共通
  $ make
  $ pjsub mat-mat-d.bash
- 実行が終了したら、以下を実行する
  $ cat mat-mat-d.bash.oXXXXXX

行列-行列積のサンプルプログラムの実行（C言語版）

- 以下のような結果が見えれば成功

```
N  = 384
Mat-Mat time  = 0.000135 [sec.]
 841973.194818 [MFLOPS]
 Error! in ( 0 , 2 )-th argument in PE 0
 Error! in ( 0 , 2 )-th argument in PE 61
 Error! in ( 0 , 2 )-th argument in PE 51
 Error! in ( 0 , 2 )-th argument in PE 59
 Error! in ( 0 , 2 )-th argument in PE 50
 Error! in ( 0 , 2 )-th argument in PE 58
....
```

並列化が完成していないのでエラーが出ます。ですが、これは正しい動作です

行列-行列積のサンプルプログラムの実行（Fortran言語）

- 以下のような結果が見えれば成功

```
NN  =  384
Mat-Mat time  =  1.29550899146124 7E-03
MFLOPS =  87414.45135502046
 Error! in ( 1 , 3 )-th argument in PE  0
 Error! in ( 1 , 3 )-th argument in PE  61
 Error! in ( 1 , 3 )-th argument in PE  51
 Error! in ( 1 , 3 )-th argument in PE  58
 Error! in ( 1 , 3 )-th argument in PE  55
 Error! in ( 1 , 3 )-th argument in PE  63
 Error! in ( 1 , 3 )-th argument in PE  60
```

並列化が
完成して
いないので
エラーが出ます。
ですが、
これは正しい
動作です。

サンプルプログラムの説明

- #define N 384
  - 数字を変更すると、行列サイズが変更できます
- #define DEBUG 1
  - 「0」を「1」にすると、行列-行列積の演算結果が検証できます。
- MyMatMat関数の仕様
  - Double型の行列A((N/NPROCS)×N行列)と B((N×(N/NPROCS)行列))の行列積をおこない、Double型の(N/NPROCS)×N行列Cに、その結果が入ります。

Fortran言語のサンプルプログラムの注意

- 行列サイズNの宣言は、以下のファイルにあります。
  mat-mat-d.inc
- 行列サイズ変数が、NNとなっています。
  integer NN
  parameter (NN=384)

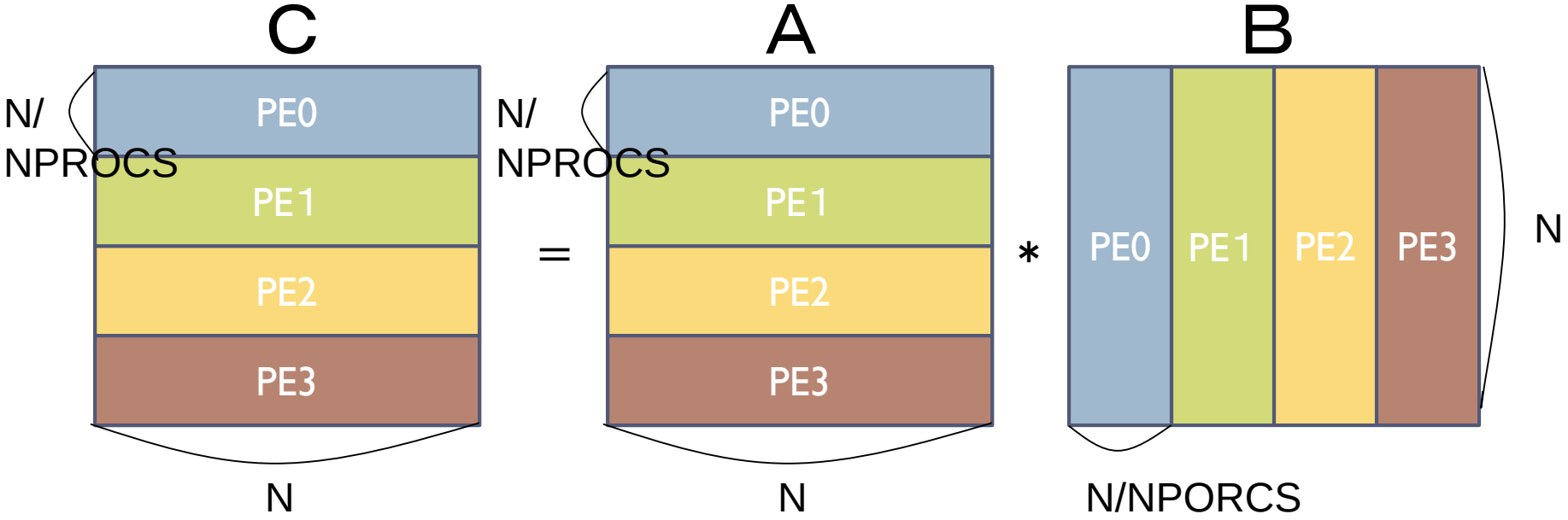
演習課題（１）

- MyMatMat関数（手続き）を並列化してください。
  - デバック時は
    #define N 384
    としてください。

- 行列A、B、Cの初期配置（データ分散）を、十分に考慮してください。

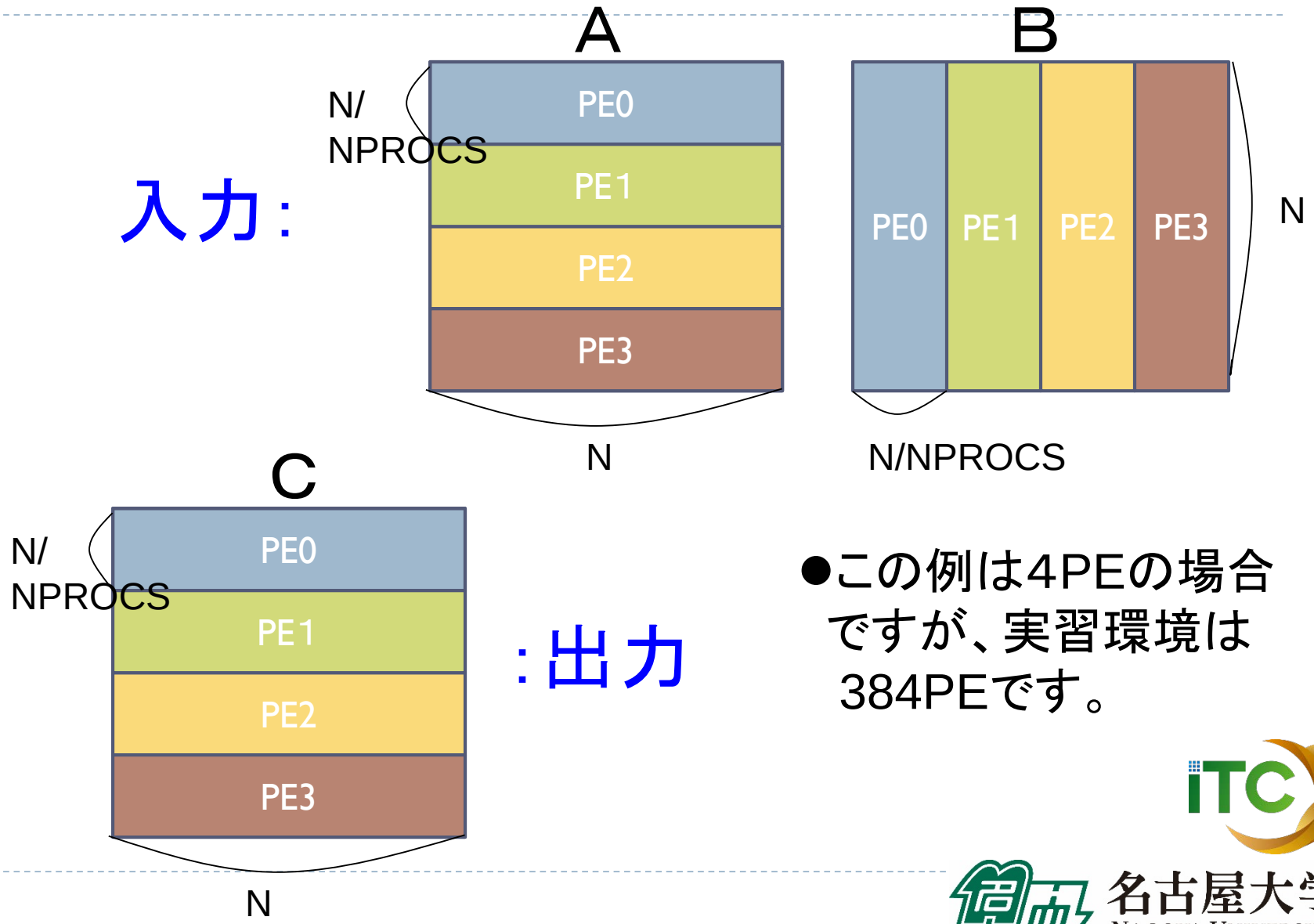
行列A、B、Cの初期配置

- 行列A、B、Cの配置は以下のようになっています。
  （ただし以下は4PEの場合で、実習環境は384PEです。）



- １対１通信関数が必要です。
- 行列A、B、Cの配列のほかに、受信用バッファの配列が必要です。

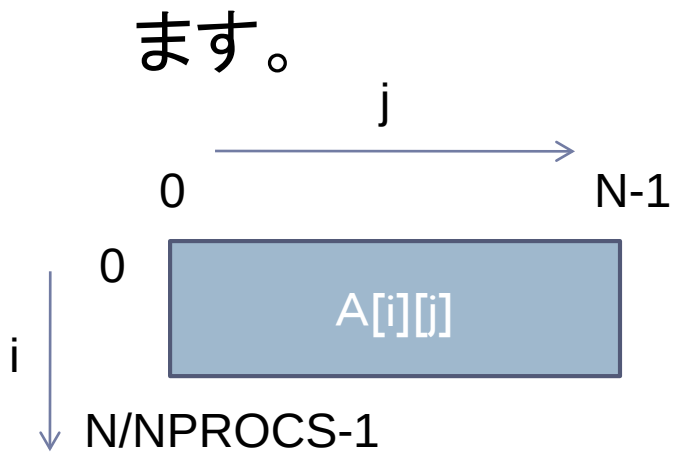
入力と出力仕様

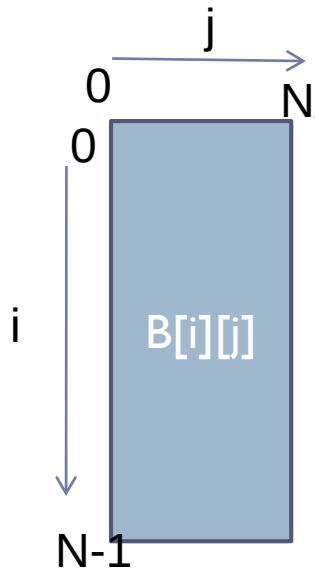


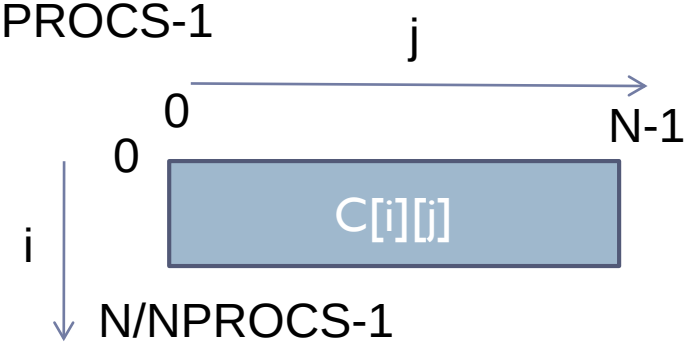
●この例は4PEの場合ですが、実習環境は384PEです。

並列化の注意（C言語）

- 各配列は、完全に分散されています。
- 各PEでは、以下のようなインデックスの配列となっています。



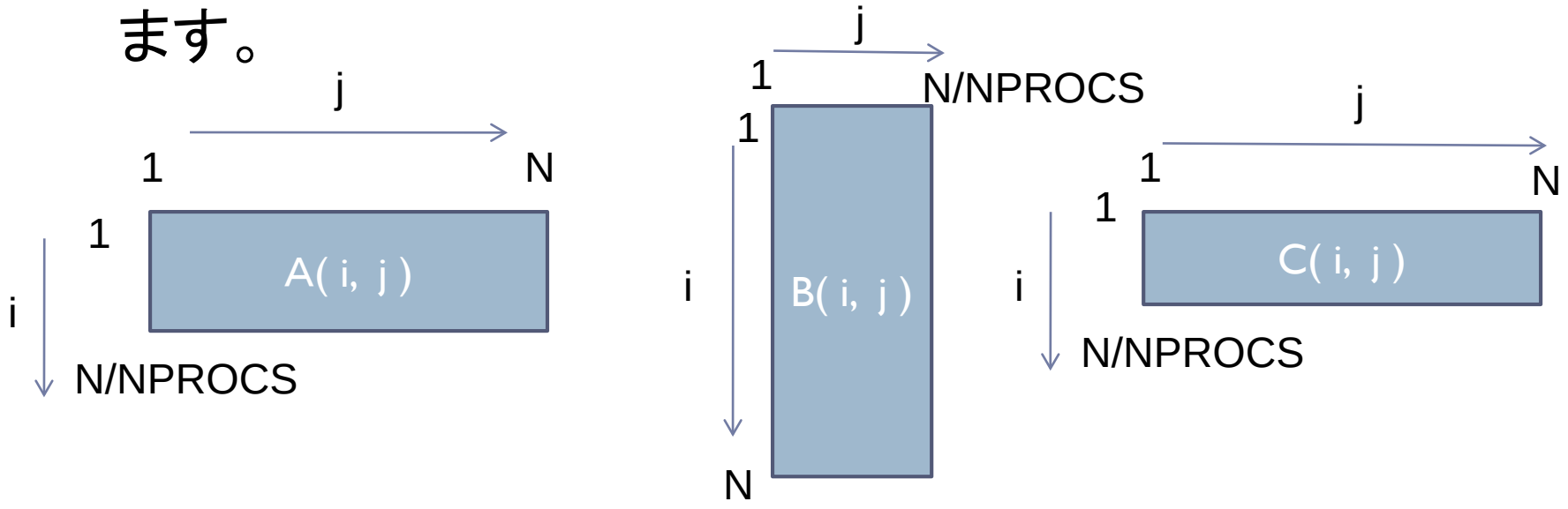




- 各PEで行う、ローカルな行列-行列積演算時のインデックス指定に注意してください。

並列化の注意（Fortran言語）

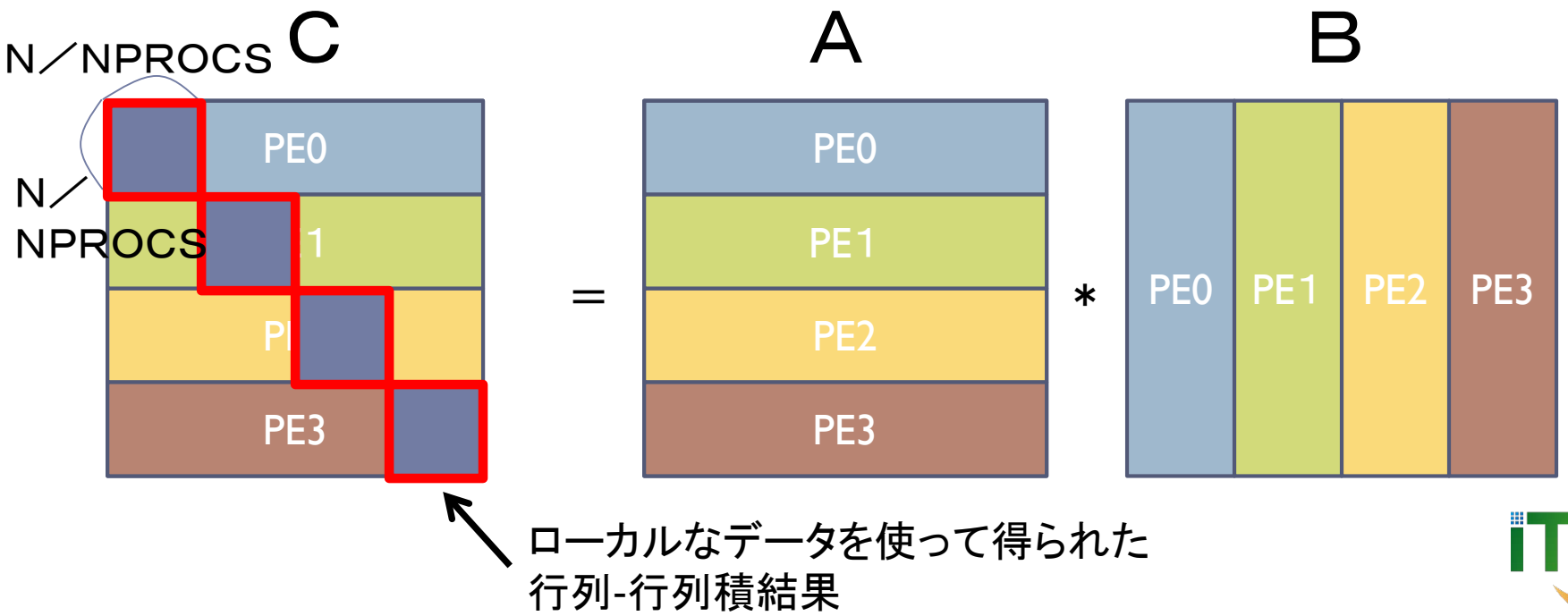
- 各配列は、完全に分散されています。
- 各PEでは、以下のようなインデックスの配列となっています。



- 各PEで行う、ローカルな行列-行列積演算時のインデックス指定に注意してください。

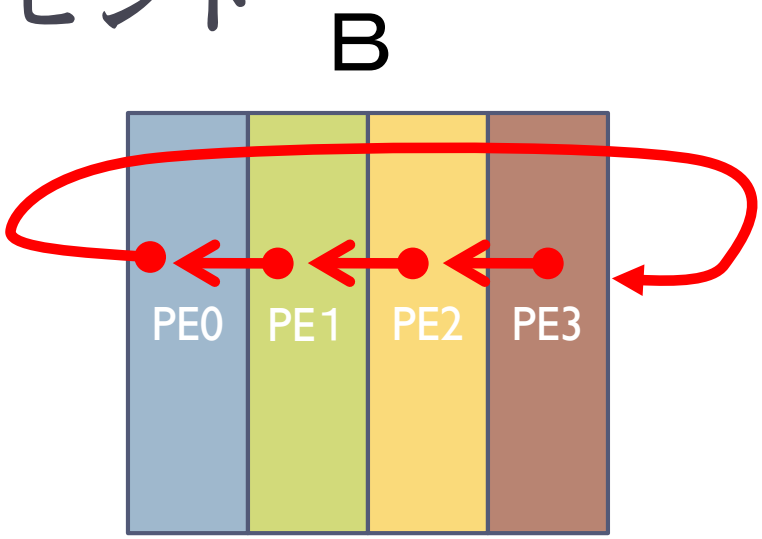
並列化のヒント

- 行列積を計算するには、各PEで完全な行列Bのデータがないので、行列Bのデータについて通信が必要です。
- たとえば、以下のように計算する方法があります。
- ステップ1

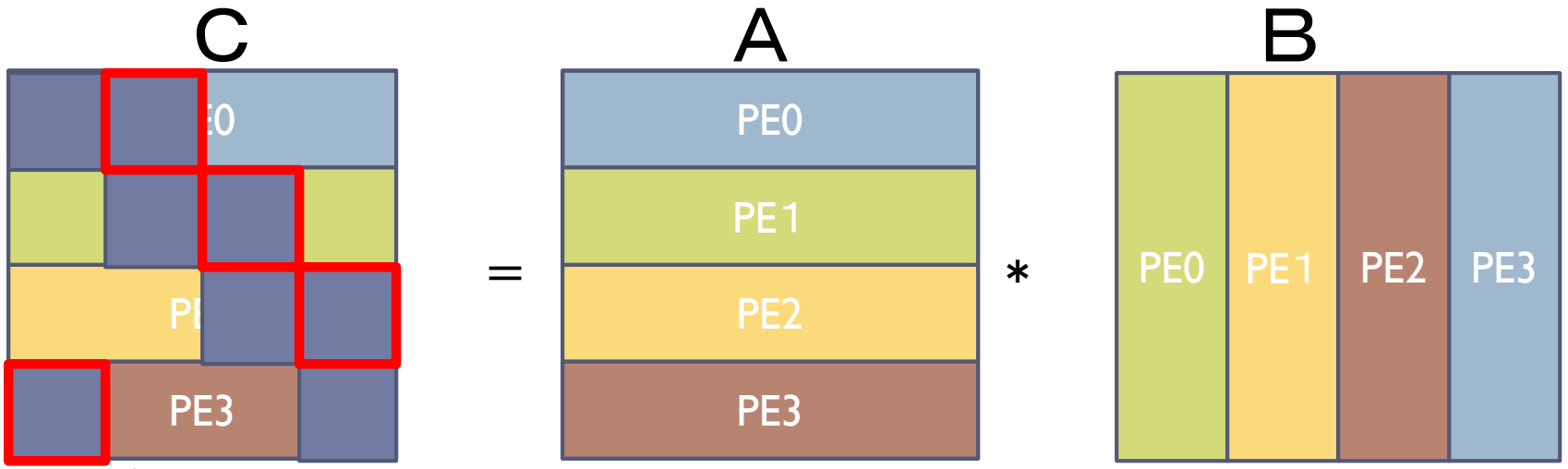


並列化のヒント

- ステップ2



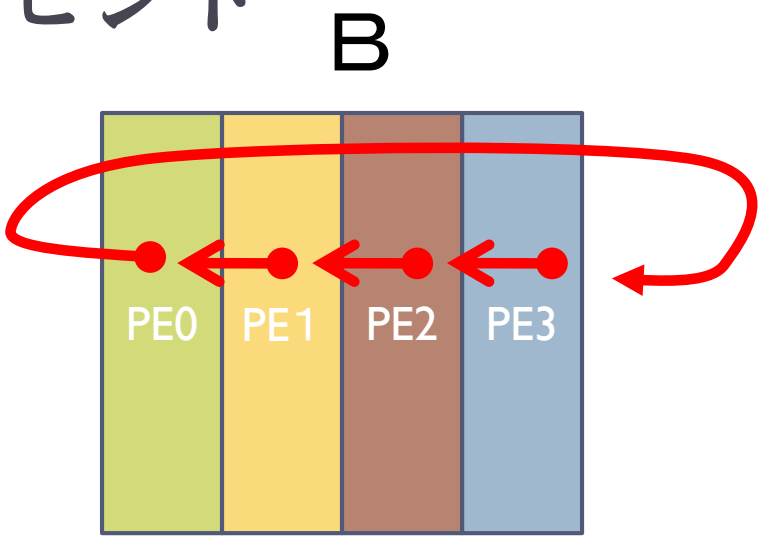
自分の持っているデータを
ひとつ左隣りに転送する
（PE0は、PE3に送る）
【循環左シフト転送】



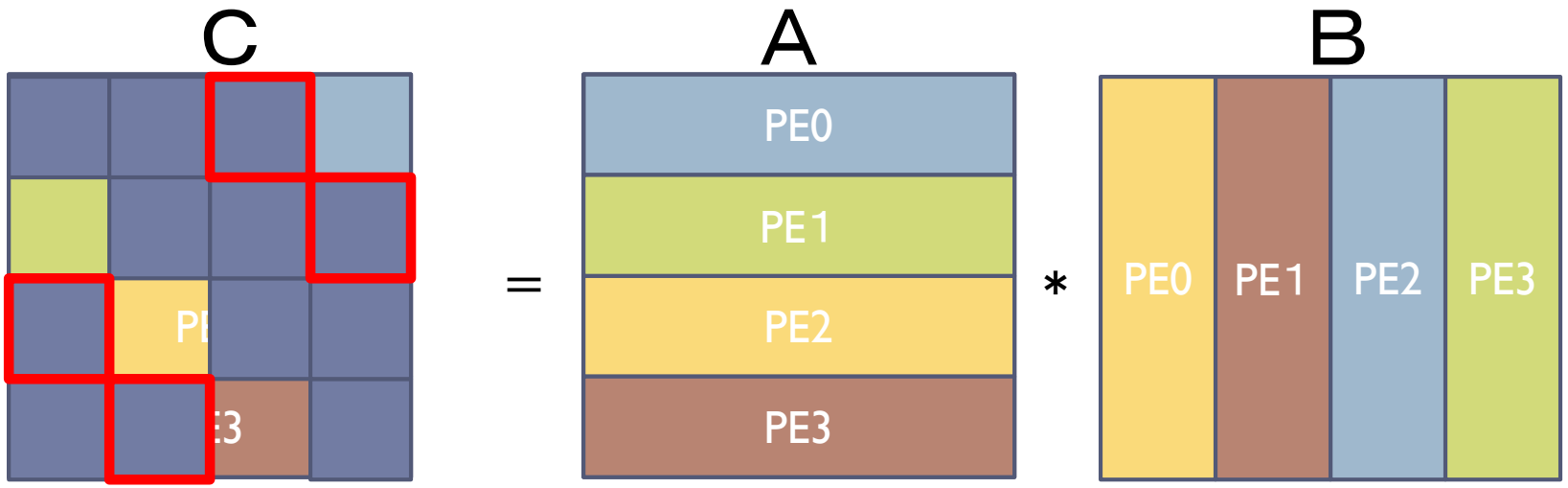
ローカルなデータを使って得られた
行列-行列積結果

並列化のヒント

- ステップ3



自分の持っているデータを
ひとつ左隣りに転送する
（PE0は、PE3に送る）
【循環左シフト転送】

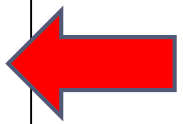


ローカルなデータを使って得られた
行列-行列積結果

並列化の注意（１／３）

- 循環左シフト転送を実装する際、全員がMPI_Sendを先に発行すると、その場所で処理が止まる。
（正確には、動いたり、動かなかったり、する）

```
…
  MPI_Send(…);
  MPI_Recv(…);
…
```



このMPI_Sendで止まる

並列化の注意（２／３）

- MPI_Send で止まる理由
  1. MPI_Sendの処理中で、大きいメッセージを送るとき、システムのバッファ領域がなくなる。
  2. バッファ領域が空くまで待つ（スピンウェイトする）。
  3. バッファ領域が空くには、相手の受信が呼ばれる必要がある。
  4. しかし、世の中に受信（MPI_Recv）をコールする人はいない。
  5. バッファ領域が、永遠に空かない。
     →一生、スピンウェイトから脱出できない。
     （MPI_Sendの箇所でずっと止まる）

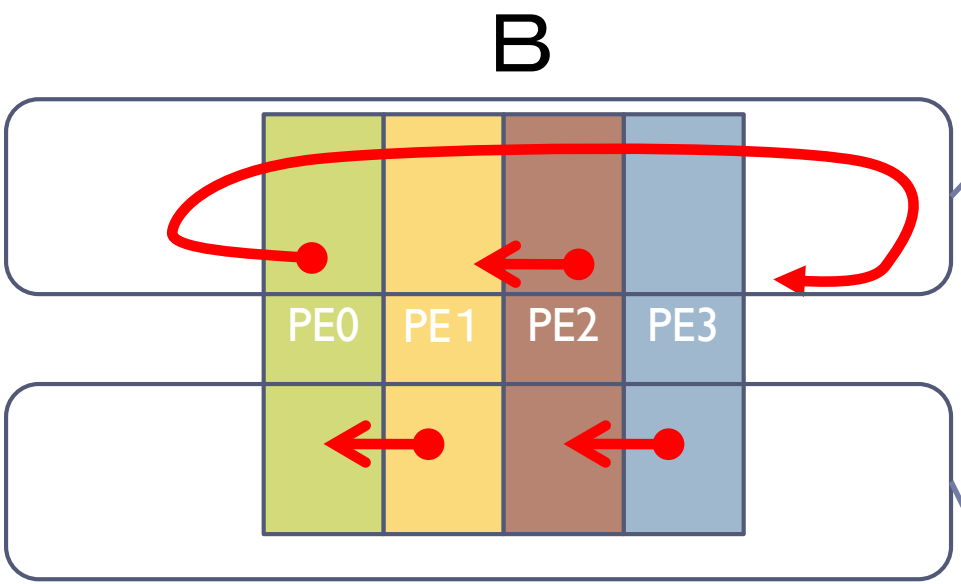
並列化の注意（３／３）

- これ（デットロック）を回避するため、以下の実装を行う。
  - PE番号が2で割り切れるPE：
    - MPI_Send();
    - MPI_Recv();
  - それ以外のPE：
    - MPI_Recv();
    - MPI_Send();

それぞれに対応

デットロック回避の通信パターン

- 以下の2ステップで、循環左シフト通信をする



ステップ1：
2で割り切れるPEがデータを送る

ステップ2：
2で割り切れないPEがデータを送る

基礎的なMPI関数—MPI_Recv（１／２）

- ierr = MPI_Recv(recvbuf, icount, idatatype, isource, itag, icomm, istatus);

  - recvbuf : 受信領域の先頭番地を指定する。
  - icount : 整数型。受信領域のデータ要素数を指定する。
  - idatatype : 整数型。受信領域のデータの型を指定する。
    - MPI_CHAR (文字型) 、MPI_INT (整数型)、MPI_FLOAT (実数型)、 MPI_DOUBLE(倍精度実数型)
  - isource : 整数型。受信したいメッセージを送信するPEのランクを指定する。
    - 任意のPEから受信したいときは、MPI_ANY_SOURCE を指定する。

基礎的なMPI関数—MPI_Recv（２／２）

- itag： 整数型。受信したいメッセージに付いているタグの値を指定する。
  - 任意のタグ値のメッセージを受信したいときは、MPI_ANY_TAG を指定する。
- icomm： 整数型。PE集団を認識する番号であるコミュニケータを指定する。
  - 通常ではMPI_COMM_WORLD を指定すればよい。
- istatus： MPI_Status型（整数型の配列）。受信状況に関する情報が入る。専用の配列を宣言すること。
  - 要素数がMPI_STATUS_SIZEの整数配列が宣言される。
  - 受信したメッセージの送信元のランクが istatus[MPI_SOURCE]、タグが istatus[MPI_TAG] に代入される。
- ierr(戻り値)： 整数型。エラーコードが入る。

実装上の注意

- タグ(itag)について
  - MPI_Send(), MPI_Recv()で現れるタグ（itag）は、任意のint型の数字を指定してよいです。
  - ただし、同じ値（0など）を指定すると、どの通信に対応するかわからなくなり、誤った通信が行われるかもしれません。
  - 循環左シフト通信では、MPI_Send()と MPI_Recv()の対が、2つでてきます。これらを別のタグにした方が、より安全です。
  - たとえば、一方は最外ループの値iloopとして、もう一方をiloop+NPROCSとすれば、全ループ中でタグがぶつかることがなく、安全です。

さらなる並列化のヒント

以降、本当にわからない人のための資料です。
ほぼ回答が載っています。

並列化のヒント

1. 循環左シフトは、PE総数-1回 必要
2. 行列Bのデータを受け取るため、行列B[][]に関するバッファ行列B_T[][]が必要
3. 受け取ったB_T[][] を、ローカルな行列-行列積で使うため、B[][]へコピーする。
4. ローカルな行列-行列積をする場合の、対角ブロックの初期値：ブロック幅*myid。
   ループ毎にブロック幅だけ増やしていくが、
   Nを超えたら0に戻さなくてはいけない。

並列化のヒント（ほぼ回答，C言語）

- 以下のようなコードになる。

```
ib = n/numprocs;
for (iloop=0;  iloop<NPROCS;  iloop++ )  {
    ローカルな行列-行列積  C  =  A * B;
    if (iloop != (numprocs-1) ) {
      if (myid % 2 == 0 ) {
          MPI_Send(B, ib*n, MPI_DOUBLE, isendPE,
                iloop, MPI_COMM_WORLD);
          MPI_Recv(B_T, ib*n, MPI_DOUBLE, irecvPE,
                iloop+numprocs, MPI_COMM_WORLD, &istatus);
      } else {
          MPI_Recv(B_T, ib*n, MPI_DOUBLE, irecvPE,
                iloop, MPI_COMM_WORLD, &istatus);
          MPI_Send(B, ib*n, MPI_DOUBLE, isendPE,
                iloop+numprocs, MPI_COMM_WORLD);
      }
      B[ ][ ]  へ  B_T[ ][ ]  をコピーする;
    }
  }
```

並列化のヒント（ほぼ回答，C言語）

- ローカルな行列-行列積は、以下のようなコードになる。

```
jstart=ib*( (myid+iloop)%NPROCS );
for (i=0;  i<ib;  i++) {
   for(j=0;  j<ib;  j++) {
      for(k=0;  k<n;  k++) {
         C[ i ][ jstart + j ] += A[ i ][ k ] * B[ k ][ j ];
      }
   }
}
```

並列化のヒント（ほぼ回答，Fortran言語）

- 以下のようなコードになる。

```
ib = n/numprocs
do iloop=0,  NPROCS-1
    ローカルな行列-行列積  C  =  A * B
    if (iloop .ne. (numprocs-1) ) then
      if (mod(myid, 2)  .eq. 0 ) then
         call MPI_SEND(B, ib*n, MPI_DOUBLE_PRECISION,  isendPE,
&                iloop, MPI_COMM_WORLD,  ierr)
         call MPI_RECV(B_T,  ib*n,  MPI_DOUBLE_PRECISION,  irecvPE,
&                iloop+numprocs, MPI_COMM_WORLD,  istatus,  ierr)
      else
         call MPI_RECV(B_T,  ib*n,  MPI_DOUBLE_PRECISION,  irecvPE,
&                iloop,  MPI_COMM_WORLD,  istatus,  ierr)
         call MPI_SEND(B,  ib*n,  MPI_DOUBLE_PRECISION,  isendPE,
&                iloop+numprocs,  MPI_COMM_WORLD,  ierr)
      endif
      B  へ  B_T  をコピーする
    endif
  enddo
```

並列化のヒント（ほぼ回答，Fortran言語）

- ローカルな行列-行列積は、以下のようなコードになる。

```
imod = mod( (myid+iloop), NPROCS )
jstart = ib* imod
do i=1, ib
   do j=1, ib
      do k=1, n
         C( i , jstart + j ) = C( i , jstart + j ) +  A( i , k ) * B( k , j )
       enddo
    enddo
enddo
```

おわり

お疲れさまでした