

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

2021年7月5日(月)10:00~17:30  
Zoomによるオンライン開催

第17回 スーパーコンピュータ「不老」  
利用型講習会  
ライブラリ利用講習会（初級）

名古屋大学情報基盤センター 片桐孝洋



|

「不老」利用型ライブラリ講習会



# プログラム

---

- ▶ 9:30 - 10:00 Zoom接続開始
- ▶ 10:00 - 10:30 端末設定など(演習)
  - ▶ スーパーコンピュータ「不老」Type1サブシステム(以降、FX1000と表記)へのログイン
  - ▶ FX1000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
  - ▶ FX1000の計算機構成、利用方法
  - ▶ サンプルプログラムの実行
- ▶ 13:30 - 14:30 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:
    - ▶ 2 ▶ 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

---

- ▶ 14:45 - 15:45 BLAS演習(演習)
  - ▶ FX1000を利用したBLAS演習
- ▶ 16:00 - 17:00 LAPACK／ScaLAPACK演習(演習)
  - ▶ 粒子間熱伝導問題の説明
  - ▶ FX1000を利用したLAPACK演習
  - ▶ オプション:FX1000を利用したScaLAPACK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

# スーパーコンピュータ「不老」の役割

## 1. 全国共同利用・共同研究拠点として学内外へ計算資源提供

- ▶ 全国共同利用・共同研究拠点として国が位置づけ
- ▶ 全国の研究者の世界トップレベル研究を強かに支援

世界トップレベル  
研究の支援

HPCI (High Performance  
Computing Infrastructure) 利用者

国策スパコン  
利用支援

名大拠点利用者

JHPCN利用者

名大「不老」  
Type システム  
(富岳型ノード)



簡便な  
移行支援

スーパー  
コンピュータ  
「富岳」

導入支援／高性  
能化／特殊処理  
／長時間実行

世界トップレベル  
研究成果創出

## 2. ものづくり企業支援(地域イノベーションコア形成)

- ▶ 産業利用制度(公開、非公開)
- ▶ 計算機利用型講習会による並列処理・大規模計算普及(地域特有の中小企業支援)

## 3. 新しい計算需要に向けたサービス開拓

- ▶ データサイエンス(ビッグデータ)、AI基盤の提供による新サービスの開拓

## 4. 指定国立大学として重要な役割

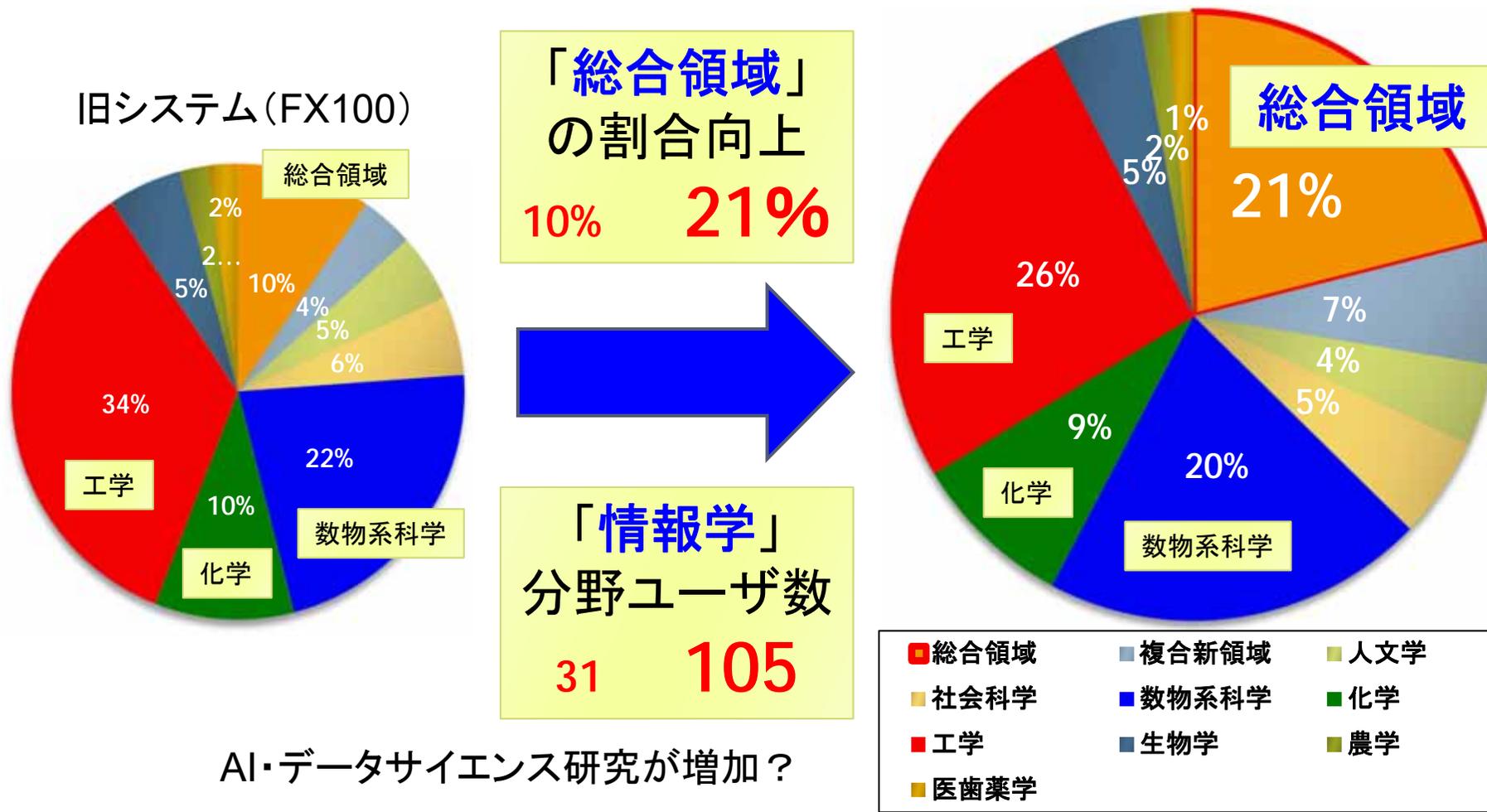
- ▶ 数理・データ科学教育
- ▶ 人材育成・研究力強化・社会との連携

- ・数理データ科学分野の人材育成
- ・AI技術基盤提供(大規模AI計算基盤、大規模ストレージ、サイバーフィジカル)
- ・技術コンサルティング・・・等による社会貢献

# ユーザの研究分野の変遷

2020年10月現在

## スーパーコンピュータ「不老」



# スーパーコンピュータ「不老」の特徴

- ▶ 以下の新しいスパコン利用法を提供しています:
  - ▶ ① Type サブシステム(「富岳」型)による超並列処理
  - ▶ ② Type サブシステム(GPUクラスタ)による大規模機械学習
  - ▶ ③ 数値計算 + AI 処理
  - ▶ ④ ①～③のシームレスなデータ可視化
  - ▶ ⑤ ①～④で必要な大規模・堅牢なデータ蓄積
- ▶ ④に必要なType サブシステム(大規模共有メモリ(48TB))、精細／遠隔可視化装置が、伝統的に名古屋大学は充実
- ▶ ⑤のコールドストレージ(100年保存可能な光ディスク)搭載スパコンは業界初

# 名古屋大学情報基盤センターの スパコン利用の特徴

## ● 計画書提出なし / 論文出版等の義務なく利用可能

- ▶ 名古屋大学拠点利用の場合（HPCI、JHPCN利用を除く）
- ▶ 年間随時募集（ただし、提供資源がなくなった場合を除く）
- ▶ アカデミックの方
  - ▶ 研究目的（平和利用）、予算確保（運営費、科研費等）が明確であれば、必ず利用承認されます
    - 研究目的の記載は数行
  - ▶ 成果報告書は1ページ
    - 事実上、発表・出版論文などの情報の登録をお願いするのみ
  - ▶ 申込書提出後、1週間程度で承認、アカウント発行
- ▶ 民間利用の方
  - ▶ 課題の計画書提出が必要
  - ▶ 審査委員会で審議の上で承認（1～2週間程度）。その後、アカウント発行。
  - ▶ 報告書はアンケート程度

# スーパーコンピュータ「富岳」との連携

## 1. 同型計算機・同一ソフトウェアスタック

- ▶ 「不老」Type サブシステムはCPUが「富岳」と同型
- ▶ OS・コンパイラ等のソフトウェアスタックも同一と想定
  - ▶ ☞「不老」のほうがより進んだバージョンが提供される可能性があります

## 2. 国策ソフトウェアの「不老」Type サブシステムの プリインストール・講習会実施

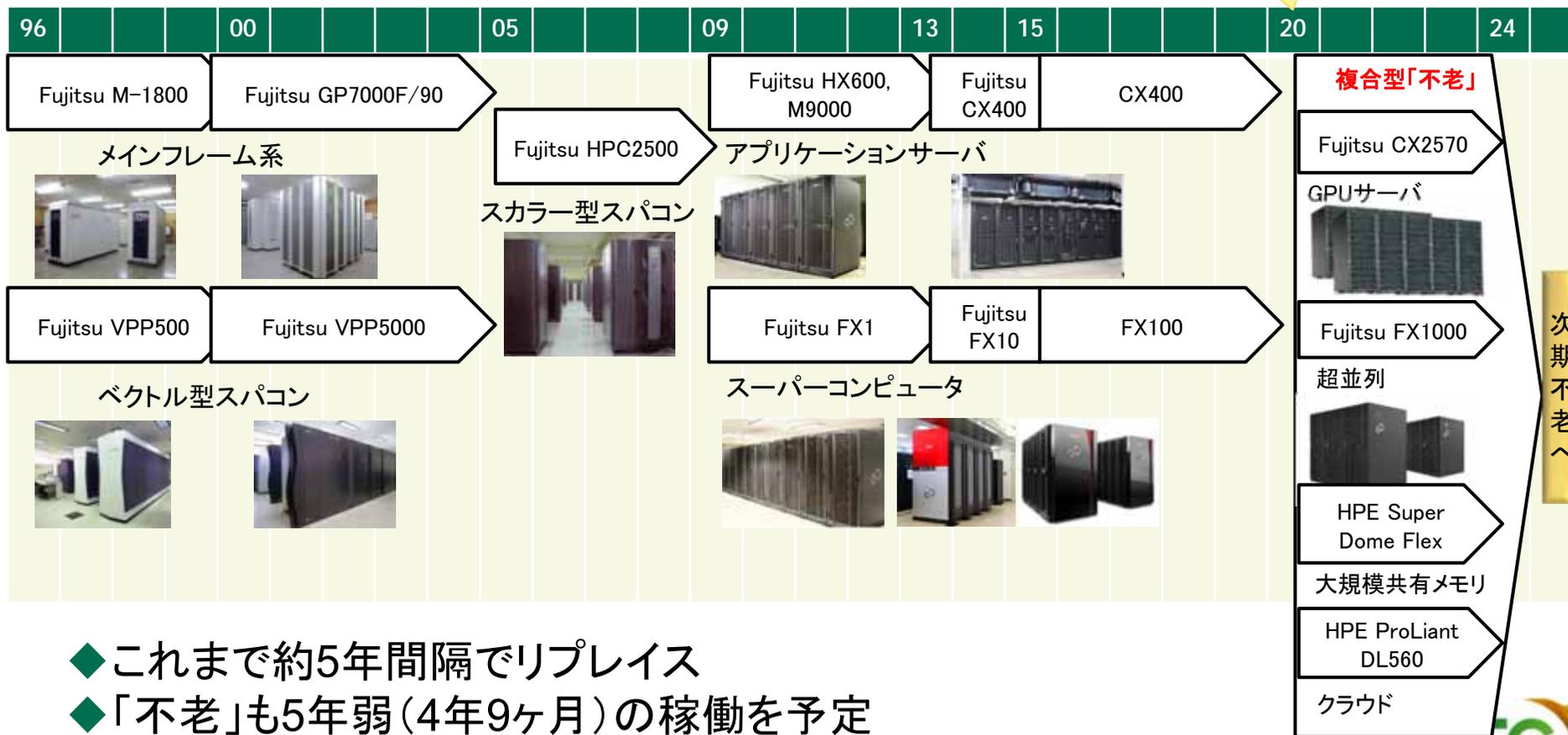
- ▶ 国費開発ソフトウェアで「富岳」で動作するソフトウェアのいくつかを、一般財団法人 高度情報科学技術研究機構(RIST)の協力のもと、「不老」Type で提供、ハンズオン講習会も実施
  - ▶ ☞講習会予定をご覧ください

## 3. 「富岳」向けチューニングと「富岳」への移行支援

- ▶ コンサルティング・教員との共同研究で「富岳」向けと想定されるコードチューニング(☞「不老」Type 向けと等価)を支援します

# 名古屋大学情報基盤センターの スパコンの歴史

スーパーコンピュータ  
「不老」導入



- ◆これまで約5年間隔でリプレイス
- ◆「不老」も5年弱(4年9ヶ月)の稼働を予定

# 導入の背景

---

- ▶ **研究のデジタル化 (デジタルサイエンス)**
  - ▶ コンピューティングを活用した研究の広まり
- ▶ **AI/機械学習研究の増大**
  - ▶ 自動運転、医療、創薬
- ▶ **シミュレーション研究の増加**
  - ▶ 異常気象、津波など国民の安全に密接にかかわる現象
  - ▶ 生命・宇宙などの基礎科学
- ▶ **データの爆発的増大**
  - ▶ 元データ、解析結果、AI学習結果など
  
- ▶ **従来のスパコンでは明らかな能力不足**

# 設置状況

- ▶ 7月1日、スーパーコンピュータ「不老」が稼働開始しました。現在も順調に稼働中です。
- ▶ 名古屋大学 情報基盤センター 本館地下1階の様子



# スーパーコンピュータ「不老」 主な構成要素

Type I, II, III, クラウドの合計で**15.886PFLOPS**  
(旧システムの約4倍)



7.782 PF

## Type Iサブシステム

FUJITSU Supercomputer FX1000  
「富岳」型



7.489 PF

## Type IIサブシステム

FUJITSU Server PRIMERGY CX2570 M5  
GPUスパコン



77.414 TF

## Type IIIサブシステム

HPE Superdome Flex  
大容量メモリ・可視化



537.6 TF

## クラウドシステム

HPE ProLiant DL560  
バッチ&インタラクティブ



30 PB

## ホットストレージ

FUJITSU PRIMERGY RX2540 M5  
FUJITSU ETERNUS AF250 S2  
DDN SFA18KE  
DDN SS9012



6 PB

## コールドストレージ

SONY PetaSite 拡張型 Library

# 性能諸元（主要サブシステム群）

	Type I	Type II	Type III	クラウド	
ノードあたり	CPU	A64FX × 1 (Armv8.2-A + SVE) 48+2コア、2.2GHz	Xeon Gold 6230 × 2 (Cascade Lake) 20コア、2.10-3.90 GHz	Xeon Platinum 8280M × 16 (Cascade Lake) 28コア、2.70-4.00 GHz	Xeon Gold 6230 × 4 (Cascade Lake) 20コア、2.10-3.90 GHz
	メインメモリ	HBM2, 32GB	DDR4, 384GB	DDR4, 24TB	DDR4, 384GB
	GPU	-	Tesla V100 × 4 (Volta) HBM2, 32GB	Quadro RTX6000 × 4 (Turing) GDDR6, 24GB	-
	理論性能	3.3792 TFLOPS(DP) 1,024 GB/s	・CPU 1.344 TFLOPS(DP) × 2 140.784 GB/s × 2 ・GPU 7.8 TFLOPS(DP) × 4 900 GB/s × 4	・CPU 2.4192 TFLOPS(DP) × 16 140.784 GB/s × 16	1.344 TFLOPS(DP) × 4 140.784 GB/s × 4
ノード数	2,304	221	2	100	
ノード間接続	TofuインターコネクトD	InfiniBand EDR × 2	InfiniBand EDR	InfiniBand EDR	
総理論性能	7.782 PFLOPS(DP) 2.359 PB/s	7.489 PFLOPS(DP) 857.8 TB/s	77.414 TFLOPS(DP) 2.253 TB/s	537.6 TFLOPS(DP) 56.314 TB/s	
冷却方式	水冷	水冷	空冷	空冷	

# 消費電力・省電力対策

## ▶ 最大消費電力

サブシステム名	消費電力
TypeIサブシステム	628.1kVA
TypeIIサブシステム	393.5kVA
TypeIIIサブシステム	21.6kVA
クラウドシステム	93.0kVA
ストレージ	49.9kVA
フロントエンド	19.6kVA
運用管理システム他	52.3kVA
冷却設備	641.9kVA
合計	1,899.9kVA

## ▶ 電力可視化



## ▶ 湧水を用いた冷却

- ▶ 地下の湧水を活用したら総合評価時加点
- ▶ 屋外チラーに散水して冷却



# 湧水による冷却システム

- ▶ 情報基盤センターの地下は夏季でも18 程度の湧き水が毎分30L程度湧く
- ▶ この湧き水は、地下からポンプで吸い上げて雨水扱いで捨てていた
- ▶ 今回の仕様で、湧き水を冷熱源として使用する場合は加点
  - ▶ 冷却水としての利用許可・水質検査済み

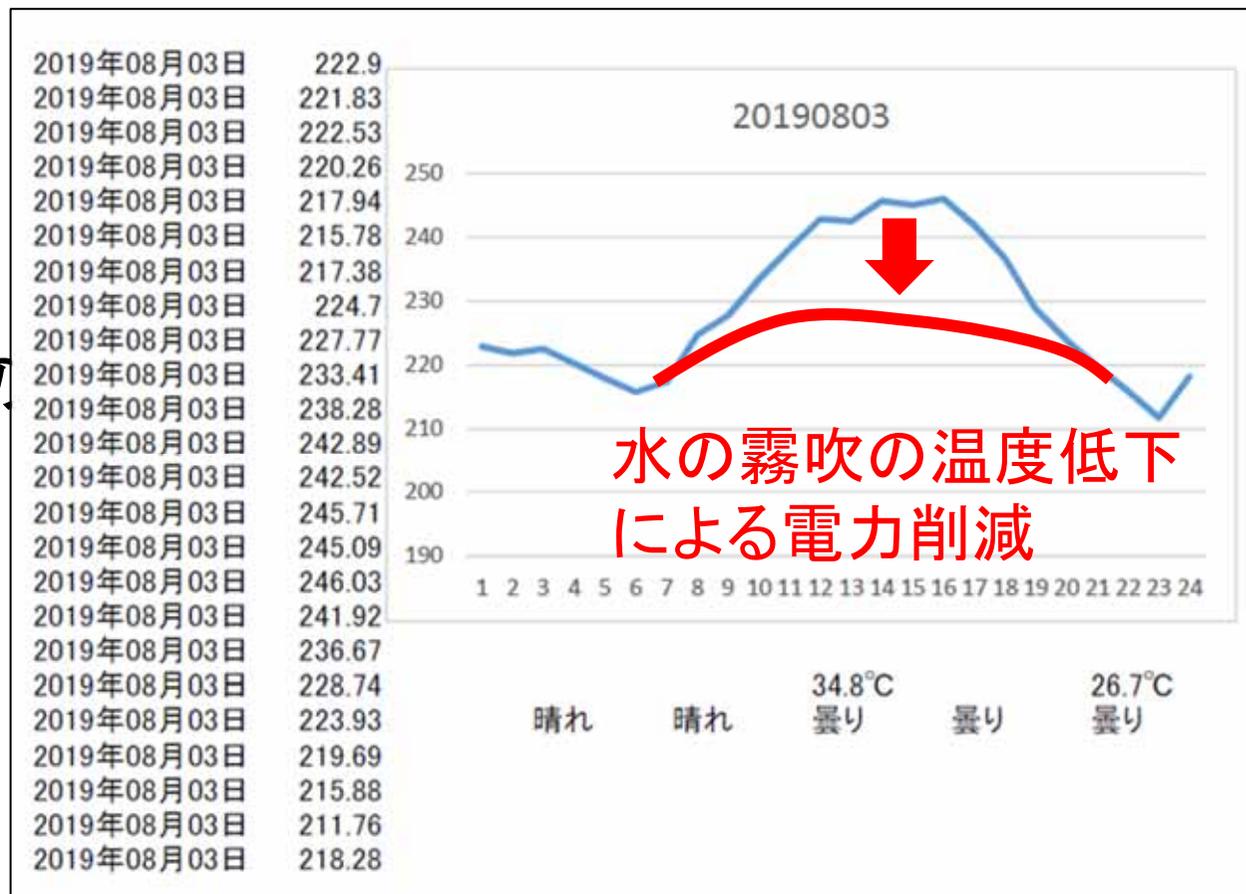


# 湧水による冷却システム



# 湧水による冷却システム

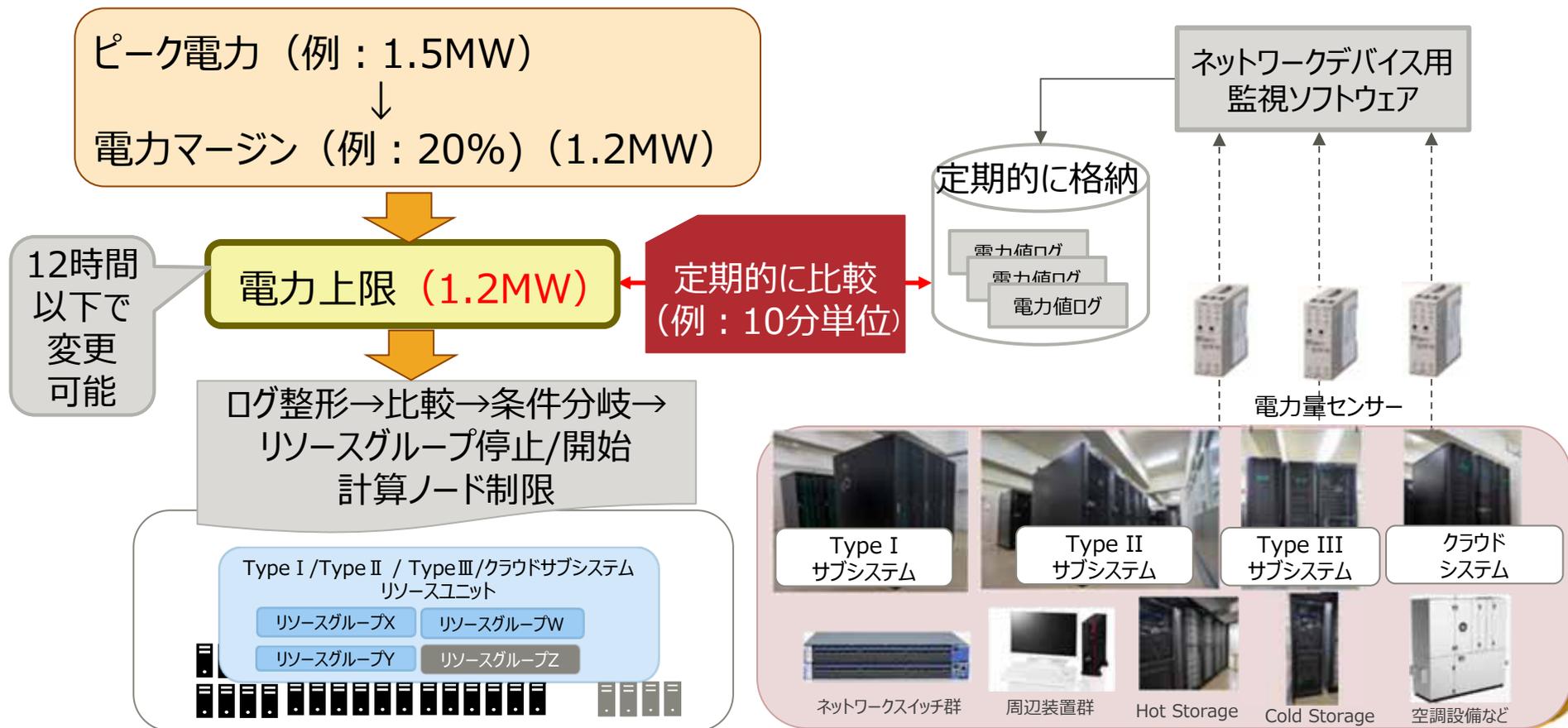
- ▶ 気温の高い4月から11月の間で利用
- ▶ 夏季の1日(2019年8月3日)の(旧)FX100システムの水冷チラーの電気使用量(KW)



年間数百万円程度の電気代削減を予想

# 使用最大電力の動的制御機構

- 監視ソフトウェアから一定時間毎に電力値を取得
- 出力された電力値と、あらかじめ規定したシステム全体の使用最大電力の上限値を比較し、最大電力の上限を超えないよう、計算ノードやジョブ実行可能範囲を制限



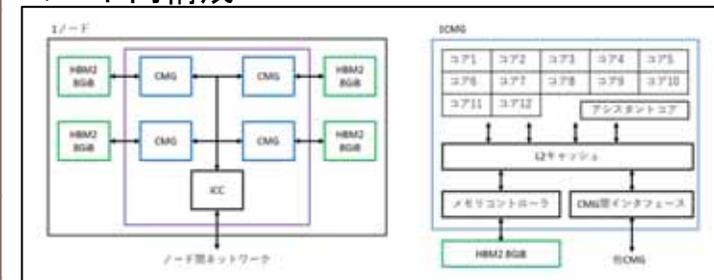
# 各サブシステムの仕様と特徴： Type I サブシステム



機種名		FUJITSU Supercomputer PRIMEHPC FX1000
計算ノード	CPU	A64FX (Armv8.2-A + SVE), 48コア+2アシスタントコア (I/O兼計算ノードは48コア+ 4アシスタントコア), 2.2GHz, 4ソケット相当の NUMA
	メインメモリ	HBM2, 32GiB
	理論演算性能	倍精度 3.3792 TFLOPS, 単精度 6.7584 TFLOPS, 半精度 13.5168 TFLOPS
	メモリバンド幅	1,024 GB/s (1CMG=12コアあたり256 GB/s, 1CPU=4CMG)
ノード数、総コア数		2,304ノード, 110,592コア (+4,800アシスタントコア)
総理論演算性能		7.782 PFLOPS
総メモリ容量		72 TiB
ノード間インターコネクト		TofuインターコネクトD 各ノードは周囲の隣接ノードへ同時に合計 40.8 GB/s × 双方向で通信可能 (1リンク当たり 6.8 GB/s × 双方向, 6リンク同時通信可能)
ユーザ用ローカルストレージ		なし
冷却方式		水冷

- 世界初正式運用のスーパーコンピュータ「富岳」型システム
- 自己開発のMPIプログラム向き
- 超並列処理用
- AIツールも提供

## ノード内構成



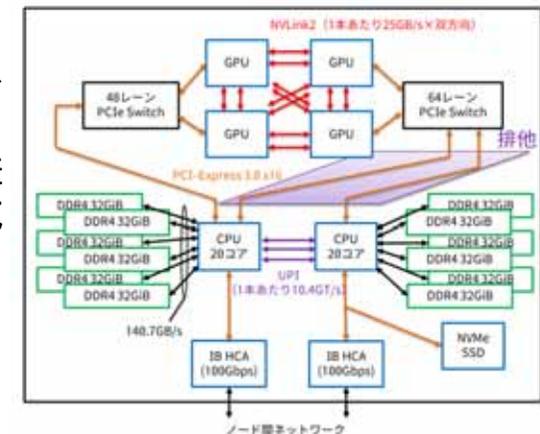
# 各サブシステムの仕様と特徴： Type II サブシステム



機種名	FUJITSU Server PRIMERGY CX2570 M5	
計算ノード	CPU	Intel Xeon Gold 6230, 20コア, 2.10 - 3.90 GHz × 2 ソケット
	GPU	NVIDIA Tesla V100 (Volta) SXM2, 2,560 FP64コア, up to 1,530 MHz × 4ソケット
	メモリ	メインメモリ(DDR4 2933 MHz): 384 GiB (32 GiB × 6 枚 × 2 ソケット) デバイスメモリ(HBM2): 32 GiB × 4 ソケット
	理論演算性能	倍精度 33.888 TFLOPS (CPU 1.344 TFLOPS × 2 ソケット, GPU 7.8 TFLOPS × 4 ソケット)
	メモリバンド幅	メインメモリ 281.5 GB/s (23.464 GB/s × 6 枚 × 2 ソケット) デバイスメモリ 900 GB/s × 4 ソケット
	GPU間接続	NVLINK2 (1GPUから他の3GPUに対してそれぞれ50GB/s × 双方向)
	CPU-GPU間接続	PCI-Express 3.0 (x16)
ノード数、総コア数	221ノード、8,840 CPUコア + 2,263,040 FP64 GPUコア	
総理論演算性能	7.489 PFLOPS (CPU 0.594 PFLOPS, GPU 6.895 PFLOPS)	
総メモリ容量	メインメモリ 82.875 TiB、デバイスメモリ 28.288 TiB	
ノード間インターコネクト	InfiniBand EDR 100 Gbps × 2, 200 Gbps	
ユーザ用ローカルストレージ	NVMe SSD 6.4TB, 一部ノードにて BeeGFS/BeeOND/NVMesh (ローカルストレージを使用した共有ファイルシステム) を提供	
冷却方式	水冷	

- データサイエンス研究、機械学習用のGPUクラスタ型
- 最新GPU (Volta) 4台/ノード
- 充実したAIツール
- 高速SSDローカルディスク

ノード内構成



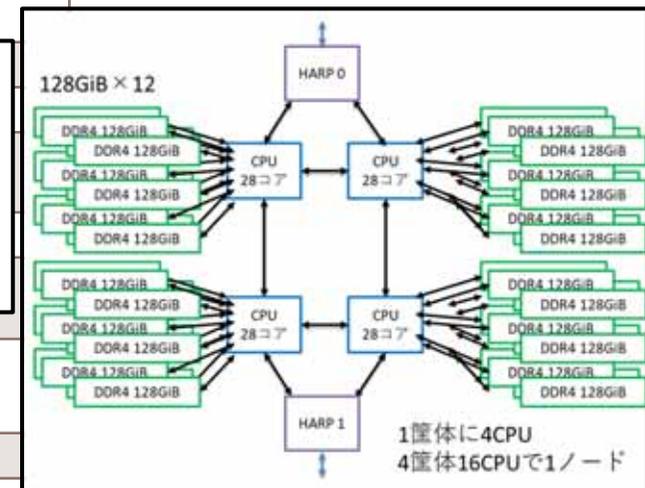
# 各サブシステムの仕様と特徴： Type III サブシステム



機種名	HPE Superdome Flex	
計算ノード	CPU	Intel Xeon Platinum 8280M, 28コア, 2.70 - 4.00 GHz × 16ソケット
	GPU	NVIDIA Quadro RTX6000 × 4
	メモリ	メインメモリ(DDR4 2933 MHz): 24 TiB (128 GiB × 12枚 × 16ソケット) デバイスメモリ(GDDR6): 24 GiB × 4
	理論演算性能	倍精度 38.7072 TFLOPS (CPU 2419.2 TFLOPS × 16ソケット)
	メモリバンド幅	メインメモリ 2252.544 GB/s (23.464 GB/s × 12枚(6チャンネル) × 16ソケット)
	CPU-GPU間接続	PCI-Express 3.0 (x16)
ノード数	2	
総理論演算性能	77.414 TFLOPS (38.7072 TFLOPS × 2ノード)	
総メインメモリ容量	48 TiB	
ノード間インターコネク	InfiniBand EDR 100 Gbps	
ユーザ用ローカルストレージ	一方のノードに102.4 TB SSD、もう一方のノードに1008 TB 共有ストレージを接続	
冷却方式	空冷	

- 大規模共有メモリ(24TiB)
- プリポスト処理用・可視化処理用
- NICE DCVを用いたリモート可視化

ノード内構成



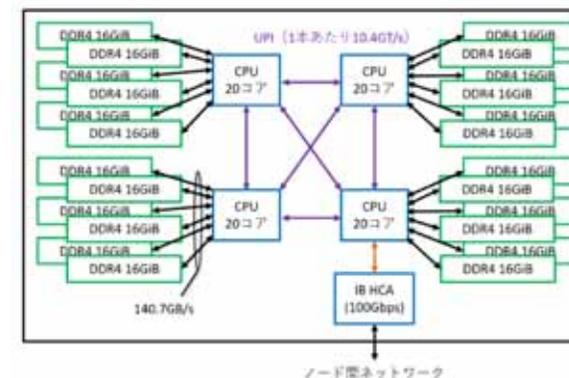
# 各サブシステムの仕様と特徴： クラウドシステム



機種名		HPE ProLiant DL560
計算 ノード	CPU	Intel Xeon Gold 6230, 20コア, 2.10 - 3.90 GHz × 4 ソケット
	メモリ	メインメモリ(DDR4 2933 MHz) 384 GiB (16 GiB × 6 枚 × 4 ソケット)
	理論演算性能	倍精度 5.376 TFLOPS (1.344 TFLOPS × 4 ソケット)
	メモリバンド幅	メインメモリ 563.136 GB/s (23.464 GB/s × 6 枚 × 4 ソケット)
ノード数		100
総理論演算性能		537.6 TFLOPS (5.376 TFLOPS × 100 ノード)
総メインメモリ容量		37.5 TiB
ノード間インターコネクト		InfiniBand EDR 100 Gbps
ユーザ用ローカルストレージ		なし
冷却方式		空冷

- 研究室クラスタから移行しやすい  
Intel CPU搭載システム
- 高いノードあたりCPU性能(4ソケット)
- 時刻を指定してのバッチジョブ・インタラクティブ  
利用が可能

ノード内構成



# ホットストレージの仕様と特徴

メタデータサーバ(MDS)	
機種名	FUJITSU PRIMERGY RX2540 M5
CPU	Intel Xeon Gold 5222 (3.80GHz, 4コア) × 2
メインメモリ	DDR4 192 GiB
HDD	SAS 900 GB 10krpm × 2 (RAID1)
Interconnect	InfiniBand EDR × 2
SAN	FibreChannel 32 Gbps &times; 2
OS	RedHat Enterprise Linux
ノード数	4台
メタデータストレージサーバ(MDT)	
機種名	FUJITSU ETERNUS AF250 S2
SSD	RAID1+0 [4D+4M] × 2 + 2HS RAID1+0 [3D+3M] × 1 + 2HS
ノード数	1台

データストレージ(OSS/OST)	
機種名	DDN SFA18KE × 1台 DDN SS9012 × 10台
HDD	NL-SAS 14TB 7.2krpm × 730、RAID6 [8D+2P] 30 Device × 24 DCR Pool + 10HS
Interconnect	InfiniBand EDR × 8
搭載セット数	4
総容量	
物理容量	40.32 PB (Global Spareを除く)
実効容量	約 <b>30.44PB</b>

- HDD RAID
- 大容量: 30.44 PB (実効容量)
- 超高速アクセス性能: 384 GB/s



# コールドストレージの仕様と特徴

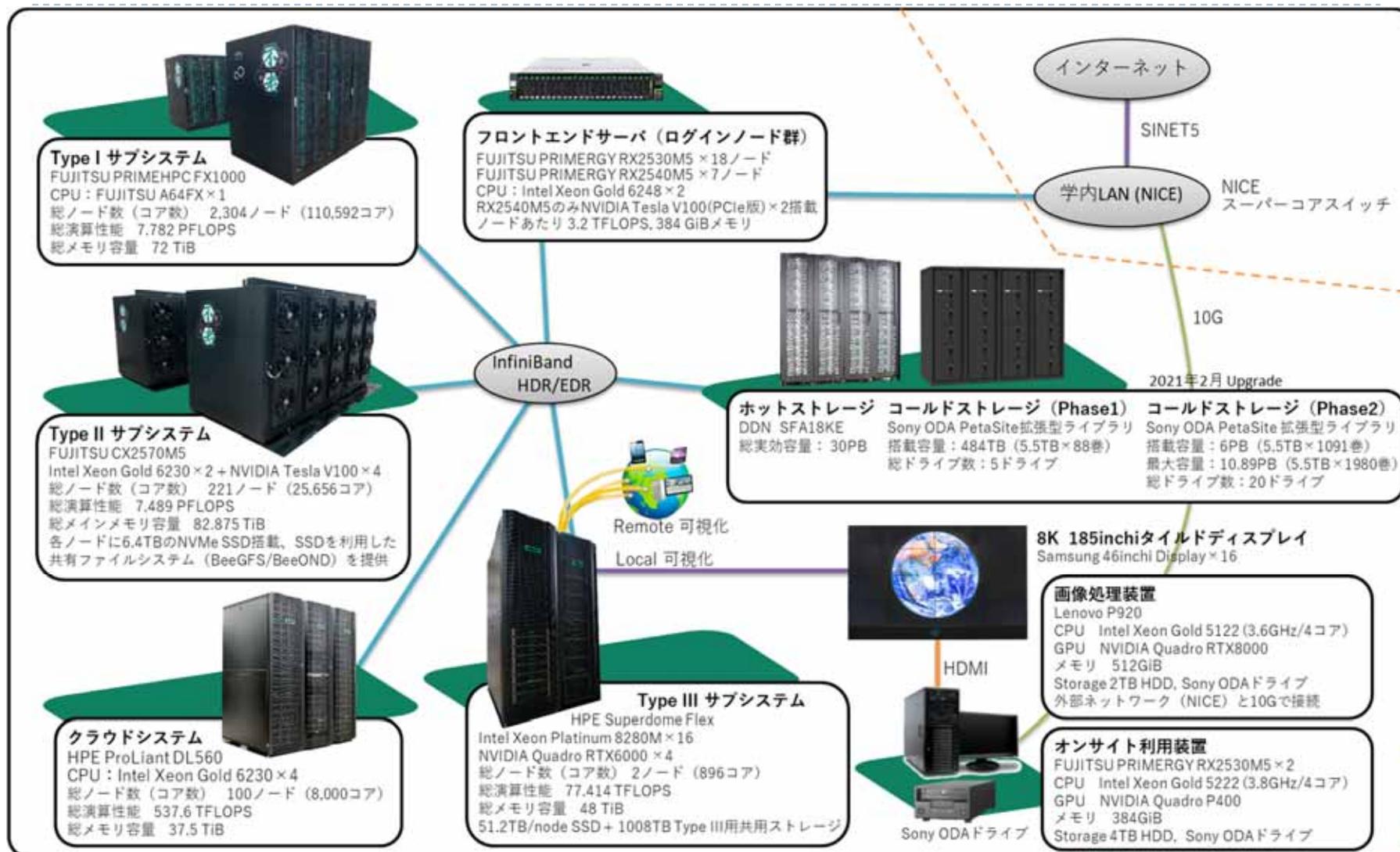
フェーズ2: 2021年2月1日より稼働

機種名	PetaSite拡張型 Library
総スロット数 (最大搭載可能カートリッジ数)	1,980巻
総物理容量 / 最大搭載可能容量	6 PB / 10.89 PB
総ドライブ数	20
ODAサーバ数	4



- 6PBの大容量(スパコンで初)
- 1度書き込み(追記)のみの光ディスクストレージ
- 実験データ等の長期データ保存用
- 理論上100年データ保持可能
- 水にぬれても読み出せる
- サービス終了後ユーザに光ディスクを返却

# 全体システム構成



# 高精細可視化システム

## Type IIIサブシステム (HPE Superdome Flex)

77.4TFLOPS/48TiB MEM

Intel Xeon Platinum 8280M(2.7GHz,28Core) × 16CPU × 2 24TiB × 2  
 NVIDIA Quadro RTX6000 × 4 × 2  
 HDD:実効容量500TB(RAID6) × 2, NVMe:51.2TB × 2



## スーパーコンピュータ共有ストレージ



ホットストレージ  
 DDN SFA18KE  
 総実行容量: 30PB



コールドストレージ  
 Sony ODA PetaSite 拡張型ライブラリ  
 搭載容量: 6PB (5.5TB × 1091巻)  
 最大容量: 10.89PB (5.5TB × 1980巻)

## 8Kタイルドディスプレイ

185inchi 8K高精細大画面タイルドディスプレイ  
 (総解像度: 7680 × 4320)  
 Samsung 46inchi Display × 16



## 画像処理装置 (Windows 可視化サーバ)



Intel Xeon Gold 5122  
 3.6GHz 4 Cores, 512 GiB MEM  
 NVIDIA Quadro RTX8000  
 SATA SSD 2TB  
 光Disk 5.5TBドライブ ODS-D380U

# スーパーコンピュータ「不老」 ソフトウェア利用環境

## プログラム開発環境など

		フロントエンドシステム	Type I サブシステム	Type II サブシステム	Type III サブシステム	クラウドシステム	画像処理サーバ	オンサイト利用装置	企業利用
Intel Parallel Studio Computing Suite	コンパイラ (Fortran, C/C++), プロファイラ/デバッガ, MPI, 数値計算ライブラリ	○		○	○	○			●
NVIDIA HPC SDK	コンパイラ (Fortran, C/C++, OpenACC, CUDA Fortran), プロファイラ/デバッガ, MPI, 数値計算ライブラリ	○		○					●
FUJITSU Technical Computing Suite	コンパイラ (Fortran, C/C++), プロファイラ/デバッガ, MPI, 数値計算ライブラリ	○	○						●
Arm Forge Professional	プロファイラ/デバッガ/最適化	○		○	○				●
NVIDIA CUDA SDK	GPU統合開発環境	○		○					●
Singularity	コンテナ環境	○		○					●
その他	GV, Gfortran, GCC, perl, Python, Ruby, R, Emacs, vi, nkf, etc.	○	○	○	○	○		○	●
	OpenGL	○		○	○	○		○	●

# スーパーコンピュータ「不老」 ソフトウェア利用環境

## ライブラリ利用環境

		フロントエンドシステム	Type I サブシステム	Type II サブシステム	Type III サブシステム	クラウドサブシステム	画像処理サーバ	オンサイト利用装置	企業利用
数値計算 ライブラリ	FFTW, SuperLU, SuperLU MT, SuperLU DIST, METIS, MT-METIS, ParMETIS, Scotch, PT- Scotch, PETSc, MUMPUS, Xabclib  ppOpen-HPCライブラリ: ppOpen-APPL, ppOpen-AT, ppOpen- MATH  精度保障ライブラリ: LINSYS_V, DHPMM_F	○	○	○	○	○			●
入出力 フォーマット ライブラリ	NetCDF, Parallel netCDF, HDF5, JHPCN-DF	○	○	○	○	○			●
画像処理 ソフトウェア	OpenCV, Geant4	○	○	○	○	○			●
機械学習 ソフトウェア	Caffe, Chainer, Keras, PyTorch, TensorFlow, Theano, Mxnet, ONNX  パッケージ: conda, Numpy, Scipy, scikit- image, pillow, matplotlib, jupyterlab	○	○	○	○	○			●
「不老」利用型ライブラリ講習会									

# スーパーコンピュータ「不老」 ソフトウェア利用環境

## 解析ソフトウェア利用環境

•\*1 Type III サブシステムの会話型ノード(lm01)で利用可。

•\*2 CPU並列版の他にGPU対応版が利用可。

		フロントエンドシステム	Type I サブシステム	Type II サブシステム	Type III サブシステム	クラウドサブシステム	画像処理サーバ	オンサイト利用装置	企業利用
流体解析	OpenFOAM, FrontFlow blue/red		○	○	○	○			●
構造解析	LS-Dyna			○					
	FrontISTR		○	○	○	○			●
計算化学解析	AMBER		○	○*2		○			
	Gaussian, Gamess, Gromacs, LAMMPS, NAMD		○	○*2		○			●
	Modylas		○	○		○			●
メッシュャー	Pointwise	○			○*1				
統合ソフトウェア	HyperWorks			○*2		○			

# スーパーコンピュータ「不老」 ソフトウェア利用環境

## 可視化ソフトウェア利用環境

\*1 Type III サブシステムの会話型ノード(lm01)で利用可。

		フロント エンド システム	Type I サブシス テム	Type II サブシス テム	Type III サブシス テム	クラウド サブシス テム	画像処理 サーバ	オンサイト 利用装置	企業利用
リモート可 視化	NICE DCV	○			○*1				●
可視化ソ フトウェア	FieldView	○			○		○	○	
	AVS/Expre ss, Paraview, POV-Ray, VMD	○			○		○	○	●
	3D AVS Player, ffmpeg, ffplay	○			○*1		○	○	●
	IDL, ENVI	○			○		○		
	MicroAVS						○		●
	3dsMax, Visual Studio Pro						○		

---

# Fujitsu PRIMEHPC FX1000 の計算機アーキテクチャ

# FX100とFX1000のアーキテクチャ比較

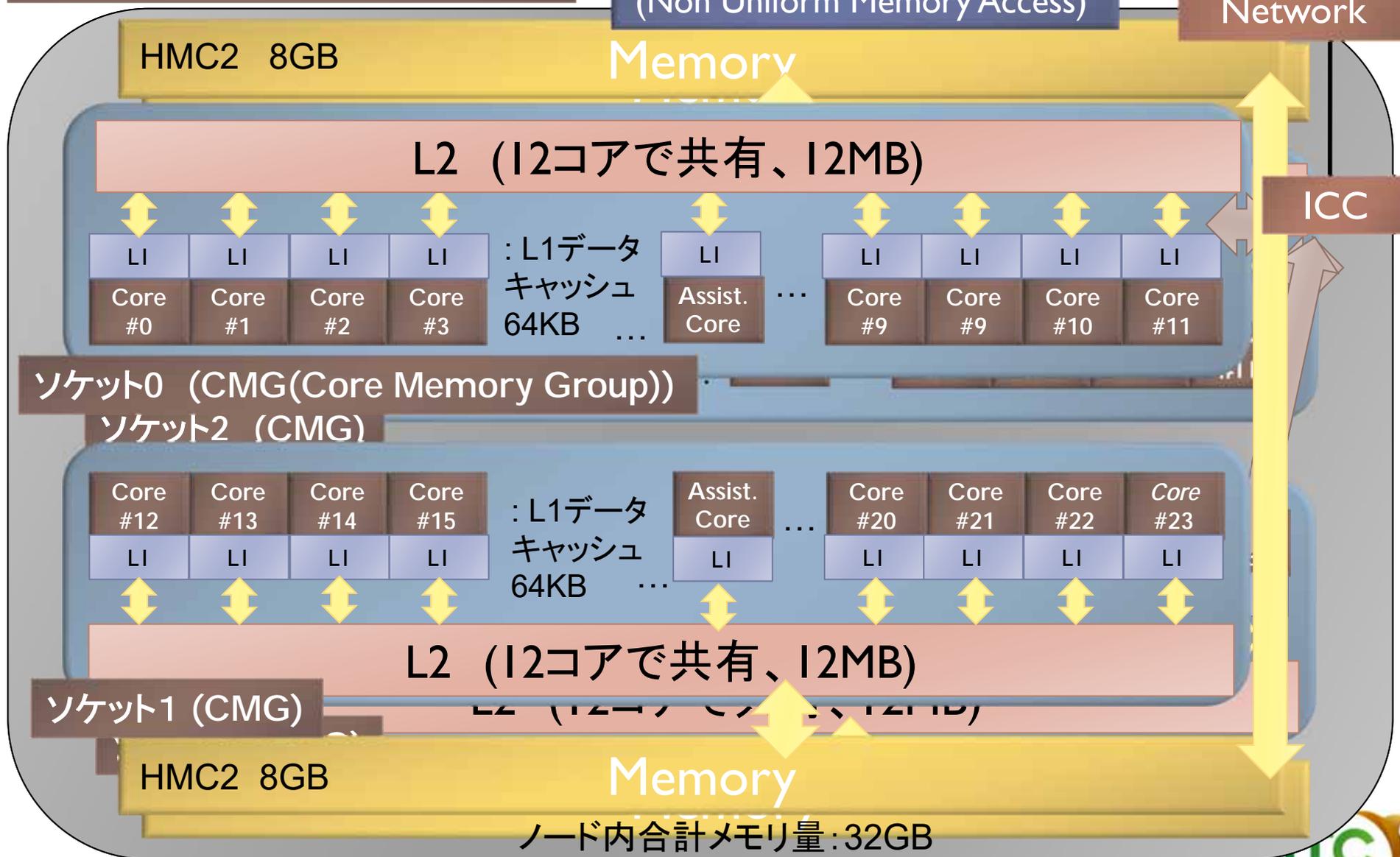
	FX100	FX1000(「不老」Type1)
演算能力／ノード	倍精度: 1.011 TFLOPS 単精度: 2.022 TFLOPS	倍精度 3.3792 TFLOPS, 単精度 6.7584 TFLOPS, 半精度 13.5168 TFLOPS
演算コア数	32	48
アシスタントコア	2	2
SIMD幅	256	512
SIMD命令	整数演算、ストライド & 間接ロード／ストアを強化	・8/16/32ビット整数演算 ・Combined Gather (128バイトアラインブロック単位)
L1Dキャッシュ／コア	64KB、4ウェイ	64KB、4ウェイ
L2キャッシュ／ノード	24MB	32MB、16ウェイ/CMG
メモリバンド幅	480GB/秒	1024GB/秒

出典: [https://www.sskn.gr.jp/MAINSITE/event/2015/20151028-sci/lecture-04/SSKEN\\_sci2015\\_miyoshi\\_presentation.pdf](https://www.sskn.gr.jp/MAINSITE/event/2015/20151028-sci/lecture-04/SSKEN_sci2015_miyoshi_presentation.pdf)  
<https://monoist.atmarkit.co.jp/mn/articles/1905/07/news013.html>

# FX1000計算ノードの構成

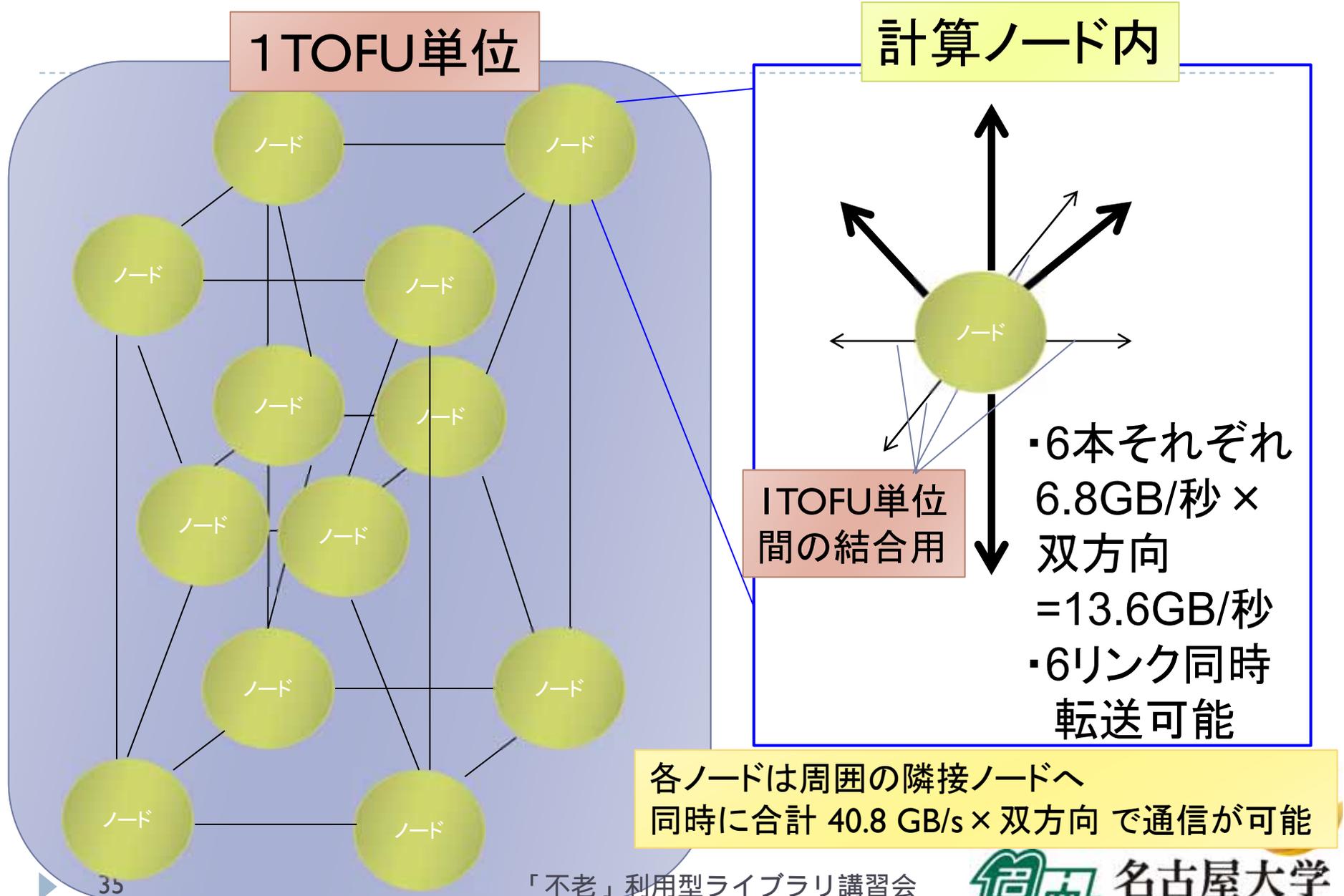
4ソケット相当、NUMA  
(Non Uniform Memory Access)

Tofu D  
Network



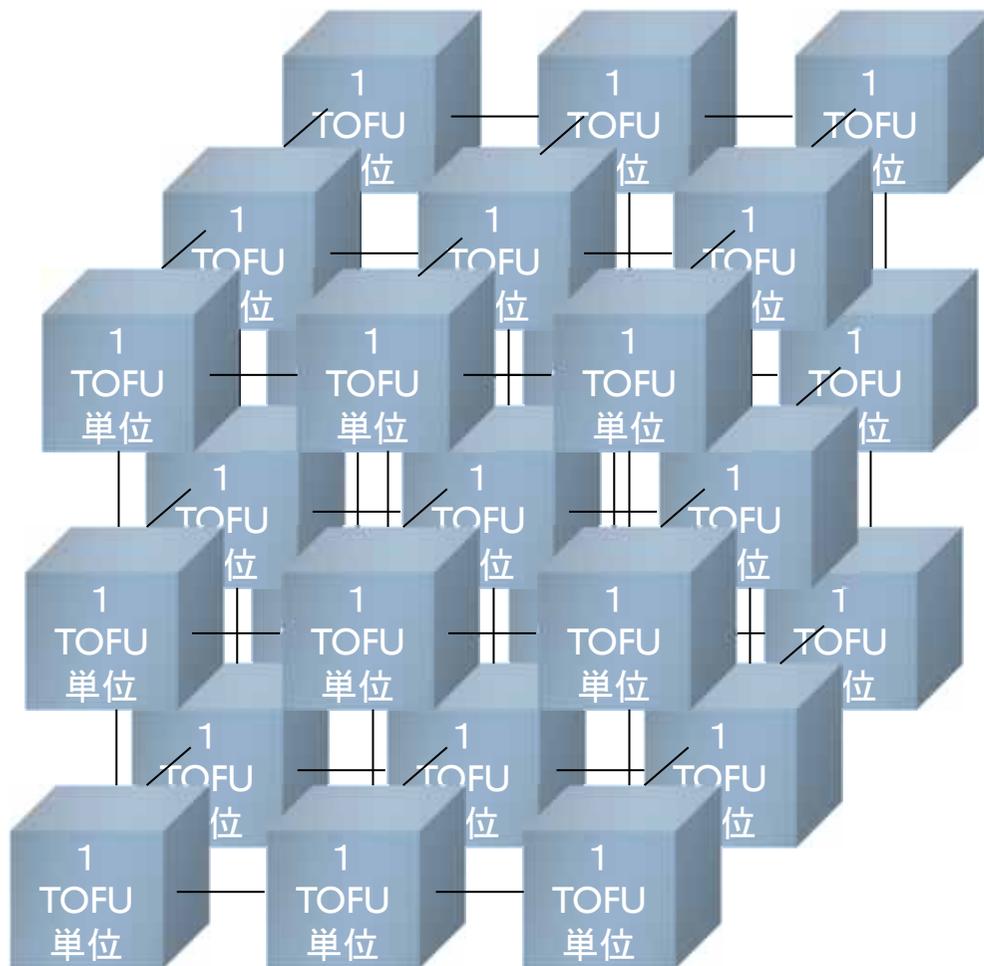
読み込み: 512GB/秒  
 書き込み: 512GB/秒 = 合計: 1024GB/秒

# FX1000通信網：TofuインターコネクトD



# FX1000通信網：TofuインターコネクトD

## 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、繋がって見えます（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

# スーパーコンピュータ「不老」 課金体系 (1/8)

---

- ▶ **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(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サブシステムと可視化システムを利用可能**
  - ▶ Type I、Type II、Type III、クラウド 全て共通

# スーパーコンピュータ「不老」 課金体系 (2/8)

- ▶ アカデミックユーザ(大学、研究機関など所属)向けプラン
  - ▶ **基本負担金**
    - ▶ **利用登録1名につき年額10,000円(登録料)**
    - ▶ **10,000利用ポイントを付加(他ユーザへの譲渡不可)**
  - ▶ **追加負担金**
    - ▶ 1,000円単位で追加が可能
    - ▶ 50万円未満: 1円あたり1ポイント付加
    - ▶ 50万円以上: **1円あたり1.25ポイント付加**

# スーパーコンピュータ「不老」 課金体系 (3/8)

- ▶ Type1サブシステム(「富岳」型ノード)消費ポイント
  - ▶ 計算課金:  $\text{利用ノード数} \times \text{経過時間[s]} \times 0.0056$ 
    - ▶ 基本負担金1万円=1万ポイント付加で利用可能な目安
      - 1ノードを 約21日
      - 4ノードを 約 5日
    - ▶ 10万円(基本利用料金1万円、追加料金9万円)  
=10万ポイント付加で利用可能な目安
      - 1ノードを 約207日
      - 4ノードを 約 52日
      - 8ノードを 約 26日
    - ▶ 1ノードの年間利用額: 約16万9000円
      - 保守日等を考慮し年間350日利用できると仮定、以下同様

# スーパーコンピュータ「不老」 課金体系 (4/8)

- ▶ Typellサブシステム(GPUノード)消費ポイント
  - ▶ 計算課金:  $\text{利用GPU数} \times \text{経過時間[s]} \times 0.007$ 
    - ▶ 基本負担金1万円  
= 1万ポイント付加で利用可能な目安
      - 1ノード(1GPU)を 約17日
      - 1ノード(4GPU)を 約 4日
    - ▶ 10万円(基本利用料金1万円、追加料金9万円)  
= 10万ポイント付加で利用可能な目安
      - 1ノード(1GPU)を 約165日
      - 1ノード(4GPU)を 約 41日
      - 4ノード(16GPU)を 約 10日
    - ▶ 1ノード(4GPU)の年間利用額: 約84万7000円

# スーパーコンピュータ「不老」 課金体系 (5/8)

- ◆ Type III: 1ソケット当たり28コア
- ◆ クラウド: 1ソケット当たり20コア

## ▶ TypeIIIサブシステム(大規模共有メモリノード)、 およびクラウドシステムの消費ポイント

- ▶ 計算課金: **利用ソケット数 × 経過時間[s] × 0.002**
  - ▶ 基本負担金1万円  
= 1万ポイント付加で利用可能な目安
    - 1ソケットを 約58日
    - 4ソケットを 約14日
  - ▶ 10万円(基本利用料金1万円、追加料金9万円)  
= 10万ポイント付加で利用可能な目安
    - 4ソケットを 約145日
    - 32ソケットを 約 18日
  - ▶ 2ソケットの年間利用額: 約12万1000円

# スーパーコンピュータ「不老」 課金体系 (6/8)

---

- ▶ **会話型利用:**  
ログインノード上での処理の消費ポイント
  - ▶ **課金: 無料** (令和3年度改定)
    - ▶ 主にインストール作業での利用想定です。  
計算利用はご遠慮ください。
  - ▶ **Type IIIサブシステム(会話型処理)の課金はありますのでご注意ください。**
  - ▶ **各計算ノードの備えるinteractiveジョブクラスはバッチジョブ扱いの課金です**  
(fx-interactive, cx-interactive, cl-interactive)

# スーパーコンピュータ「不老」 課金体系 (7/8)

## ▶ ホットストレージ

### ▶ ファイル課金

- ▶ 1TB 以下の場合(Home + Large): 徴収しない
- ▶ ファイルの使用容量が1TB 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) 2TB(2000GB)利用: 1000GBが課金対象  
10ポイント/日      300円/30日、3, 500円/350日  
(保守などで停止する日については徴収しない)

128TB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【予定】。

128TBを超える容量が必要な場合は、事前に相談ください。

# スーパーコンピュータ「不老」 課金体系 (8/8)

## ▶ コールドストレージ

### ▶ ファイル課金

#### ▶ 1口: 50TB

□ 1回だけ書き込める(追記可能)の  
光ディスク×10枚(1枚約5TB)

#### ▶ ファイル負担経費(初回利用時のみ必要):

1口 190,000円

#### ▶ ファイル管理経費(毎年必要、基本負担金とは別):

1口 10,000円

- 事前納入の光ディスク6PB売り切り後は、ユーザによる持ち込みのみとなります(上限10PBまで)。
- 管理料1万円/年と格安です!
- ご購入はお早めに!

ユーザの利用終了時、もしくは、「不老」運用終了時に、  
光ディスクを持ち帰りいただけます。

# コールドストレージ サービス紹介

2021年2月より光ディスクを使ったコールドストレージサービスを開始していますが、2021年5月に光ディスクカートリッジが利用できる単体ドライブの貸し出しサービスも開始しました。



コールドストレージ内部

Windows/Mac OSを搭載したPCに、USB3.2又はUSB3.0で単体ドライブを接続して、専用のFilerソフトを使って利用できます。



光ディスクカートリッジ



単体ドライブ

輸送には専用のジェラルミンケースをご用意しています。



輸送用ジェラルミンケース

コールドストレージシステム概要:

詳細は、以下をご覧ください:

<https://icts.nagoya-u.ac.jp/ja/sc/pdf/pamphlet-cold-202012.pdf>

# スーパーコンピュータ「不老」 新サービス：ノード準占有利用

---

- ▶ ノード準占有利用
  - ▶ 1時間以内のジョブ実行開始を保証
  - ▶ バッチ利用のみ
- ▶ 1ノード、1ヶ月間の利用負担金
  - ▶ Type IIサブシステム：  
210,000円（通常価格の約2.8倍）
  - ▶ クラウドシステム：  
62,000円（通常価格の約2.8倍）

# スーパーコンピュータ「不老」 新サービス：クラウドノード予約利用

## ▶ クラウドノード予約利用

- ▶ 専用の予約システム「UNCAI」(Webブラウザで操作)でノードを予約して利用する

## ▶ 利用料金

- ▶ 計算課金：**利用コア数 × 経過時間[s] × 0.0001**  
(ソケット当たりコア数を考えればバッチ実行と同等)
  - ▶ 1ソケット当たり20コア、10コア(0.5ソケット)から利用可能
  - ▶ 利用可能なメモリ容量もコア数に比例
  - ▶ **基本負担金1万円でXeon Gold 20コア1ソケットを約58日間使用可能**

# スーパーコンピュータ「不老」 新サービス：グループ利用

---

## ▶ グループ利用

- ▶ 1口10人まで、10万円で100,000ポイント付与  
(令和3年度改定)
- ▶ 登録料なし
- ▶ 個人利用(個別に1万円×10人が個別に基本負担金を払う)との違い
  - ▶ 100,000ポイントを10人で共有利用できます
  - ▶ 個人利用では、購入した10,000ポイントを他者と共有できません

# 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)

- ▶ 名大学内外の学部・大学院等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)

# スーパーコンピュータ「不老」 民間利用制度（産業利用）

▶ 書類での審査があります。追加負担金も同額です。

## ▶ 公開型

▶ 10アカウントまで**20万円**

▶ アカデミック利用の料金の  
**2倍(20万円当たり100,000ポイント)**

▶ 企業名、課題名、報告書をWebで公開(延期制度あり)

## ▶ 非公開型

▶ 10アカウントまで**40万円**

▶ アカデミック利用の料金の  
**4倍(40万円当たり100,000ポイント)**

▶ 外部に情報は非公開(ただし内部会議では情報が出ます)

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。

# 優先ジョブクラス（アカデミック・民間）

- ▶ **Type1、Type11、クラウドの各サブシステム**
  - ▶ ポイントを**通常の2倍**消費することで利用可能なジョブクラス
  - ▶ 専用のキューにジョブを投げることで利用
  - ▶ 通常のジョブクラスが混んでいるときでも早く実行したい、  
というユーザの利用を想定
- ▶ （優先ジョブクラスも混んでしまったらすいません）

# 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( ZOOMによる遠隔面談も可)
    - ▶ 実際に画面を見ながらなど、電話やメールでは伝えにくいご質問やご相談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てください。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から

# 可視化設備

- ▶ 情報基盤センター可視化室（本館1階）
  - ▶ 可視化室の利用は予約制となります（無料）

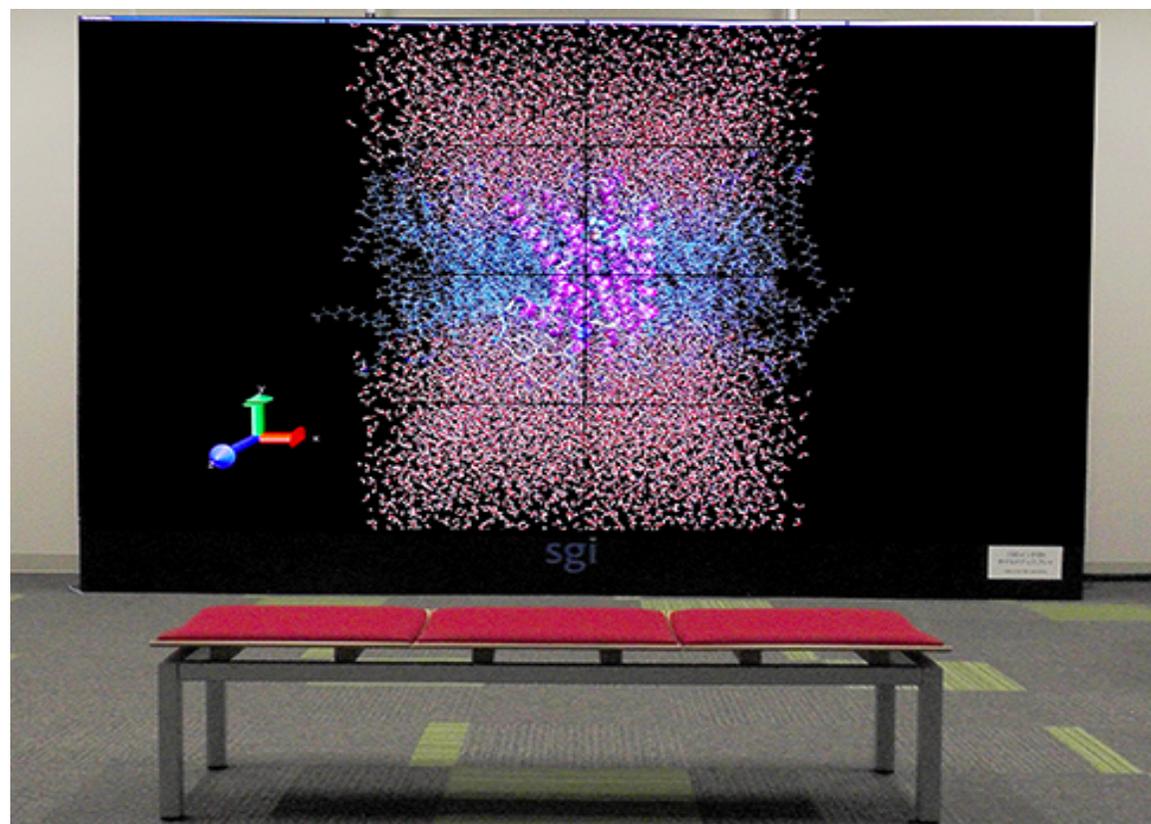


- 8K 185型タイルドディスプレイ
- 全天周映像視聴システム
- 円偏光立体視システム など

# 可視化設備

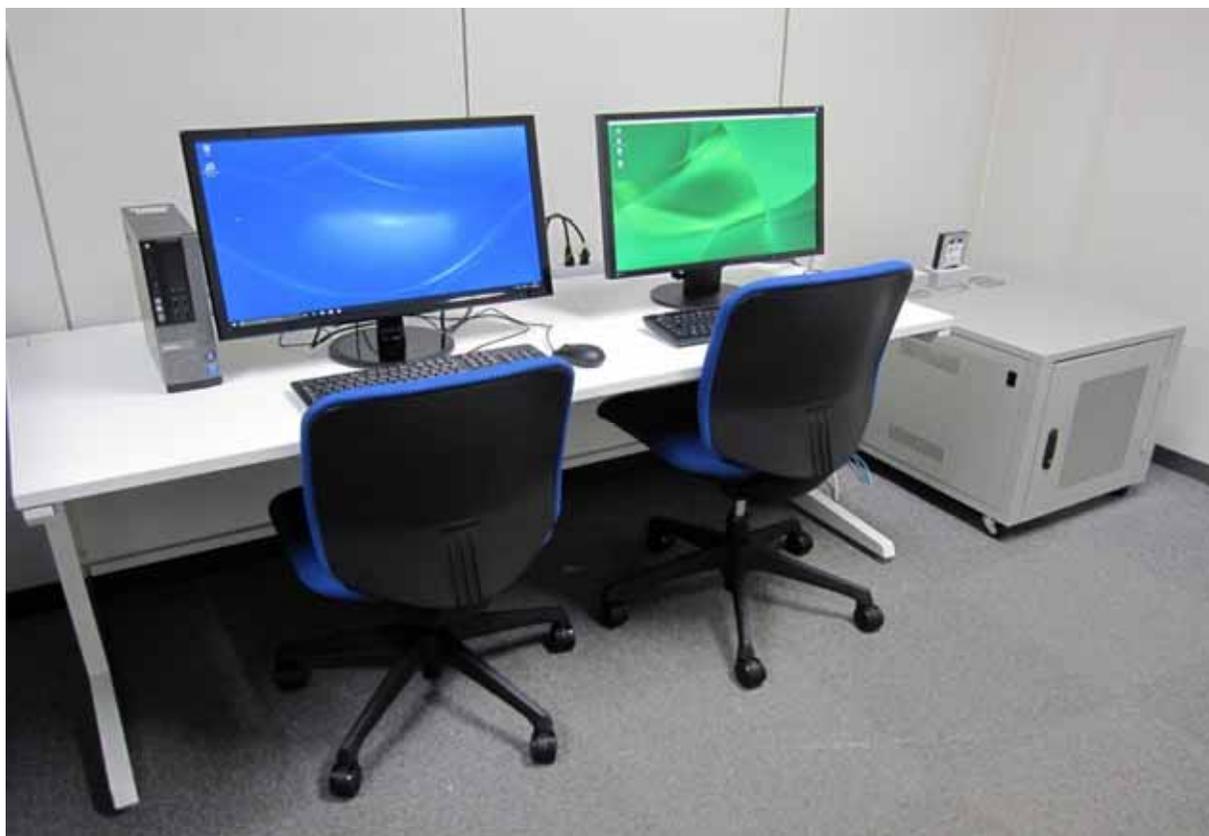
- ▶ 情報基盤センター可視化室（本館1階）
  - ▶ 可視化室の利用は予約制となります（無料）

8K 185型  
タイルド  
ディスプレイ



# 利用者支援室（2部屋）

- ▶ 情報基盤センター利用者支援室（本館3階）
  - ▶ 利用者支援室の利用は予約制となります（無料）



# 訪問者が現地で利用可能な機材

## ▶ 画像処理装置 (1F可視化室)

- Windows10が動作するデスクトップパソコン(1台)
- 対話型のプリ・ポスト処理や共有ファイルストレージの大容量ファイル取り扱い用
- 10Gbps SINETによる高速インターネットが利用可能
- USB外付けHDDやディスクメディア (Blu-rayディスク) を持ち込んでホットストレージに対するデータの読み書きが可能
- 可視化室に設置された185インチ8K高精細ディスプレイに接続、大規模データの高品位なプリポスト処理やコンテンツ生成に活用可能

画像処理装置



ODA単体ドライブ



- コールドストレージの光ディスクアーカイブと互換性のあるUSB接続ドライブ
- 画像処理装置・オンサイト利用装置に接続して利用可能

## ▶ オンサイト利用装置 (3F利用者支援室)

- Linuxが動作する計算サーバ(2台)
- 対話型のプリ・ポスト処理、スーパーコンピュータシステムとの大容量ファイル取り扱い用
- USB外付けHDDや光ディスクメディア (Blu-rayディスク) を持ち込んでホットストレージに対するデータの読み書きが可能
- 予約制の部屋貸し切りで、ハードディスク持ち込みで大規模データの入力、回収が可能

オンサイト利用装置



- ODA単体ドライブの貸し出し  
できます (宅配便で郵送できる  
専用ケースも有)
- ご相談ください

---

# スパコン利用の方法

# マニュアルなどの入手方法

- ▶ スパコンへのログインなど初歩的なことや最新のTips(便利な使い方)、最適化のためのお役立ち情報等
  - ▶ 情報基盤センターのWebページで誰でも入手可能
  - ▶ 「スーパーコンピュータ「不老」基本マニュアルおよび関連資料」
  - ▶ <http://www.icts.nagoya-u.ac.jp/ja/sc/usage.html>
- ▶ 各サブシステムのマニュアルなど
  - ▶ HPC Portalからダウンロード
  - ▶ 「不老」のアカウントを持つ利用者のみがアクセス可能
  - ▶ <https://portal.cc.nagoya-u.ac.jp/>

# 基本的な使い方

---

- ▶ SSH公開鍵の設定、マニュアルの閲覧：  
HPC PortalにWebアクセス
- ▶ バッチジョブ：SSHで接続し、バッチジョブシステムにジョブを登録
- ▶ クラウドシステムの時間指定ジョブ実行：Webシステム（UNCAI）から予約、時間になったらSSHで接続して利用
- ▶ 多くのスパコンと同様、「不老」もバッチジョブシステムでプログラムの実行を管理
  - ▶ 多数のユーザの多数のジョブを効率よく処理するため
- ▶ 利用者はSSH接続環境を整備しジョブスクリプトの書き方を習得する必要がある

# 接続先

## (ログインノードとHPC PortalとUNCAI)

### ▶ SSH接続先

対象計算サブシステム	SSH接続先	備考
Type Iサブシステム	flow-fx.cc.nagoya-u.ac.jp	
Type Iサブシステム (HPCI)	hpcifx.cc.nagoya-u.ac.jp	
Type IIサブシステム	flow-cx.cc.nagoya-u.ac.jp	GPU搭載
Type IIサブシステム (HPCI)	hpcicx.cc.nagoya-u.ac.jp	GPU搭載
Type IIIサブシステム	flow-lm.cc.nagoya-u.ac.jp	GPU搭載
Type IIIサブシステム (可視化用)	post.cc.nagoya-u.ac.jp	リモート可視化用
クラウドシステム	flow-cloud.cc.nagoya-u.ac.jp	

- ▶ HPC Portal: <https://portal.cc.nagoya-u.ac.jp/>
- ▶ UNCAI: <https://portal.cc.nagoya-u.ac.jp/reserve/>

# 共有ファイルシステム上の 大規模ファイルの収納について

- ▶ /home領域は容量制限があります。性能的にも良くありません。
  - ▶ 1TB以上のデータは容量制限があり、デフォルト設定では置くことができません。
- ▶ 大規模ファイル、もしくは、高性能が必要なファイルは、以下に収納してください
  - ▶ /data/group 1 /  
の自分のID名があるフォルダの下
- ▶ Type サブシステム利用の場合は、ノードローカルのメモリ(SSD)を一時的に使うことで高速化できる可能性があります。

---

以降、Cygwinなどのターミナルを利用する人に特化されています

MobaXtermなどを利用の方は、送付済みの別資料をご覧ください。

## 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)

---

# ノートパソコンの設定： 鍵の生成、ログイン

# 無線LANの設定

---

- ▶ 各自のパソコンにおいて、無線LANの設定をしてください
- ▶ 詳細は会場にある無線LAN情報をご覧ください

# 鍵の作成

---

1. ターミナルを起動する
2. 以下を入力する

```
$ ssh-keygen -t rsa
```

3. 鍵の収納先を聞かれるので、リターンを押す
4. 鍵を使うためのパスワードを聞かれるので、**センターのパスワードではない**、自分の好きなパスワードを入れる(**パスフレーズ**とよぶ)
5. もう一度、上記のパスフレーズを入れる
6. 鍵が生成される

# 鍵の利用 (1 / 2)

---

1. 生成した鍵は、以下に入っている  
`.ssh/`
2. 以下を入力する  
`$ cd .ssh/`
3. 以下を入力すると、ファイルが見える  
`$ ls`  
`id_rsa id_rsa.pub known_hosts`

▶ ここで、以下のファイルを区別する

`id_rsa` : 秘密鍵  
`id_rsa.pub` : 公開鍵

この公開鍵の収納ディレクトリを覚えておく(後で使います)

## 鍵の利用（2 / 2）

---

4. 以下を入力して、公開鍵を表示する

```
$ cat id_rsa.pub
```

<公開鍵が表示される>

5. “ssh-rsa ...”で始まる部分を、マウスでカットアンドペーストし、公開鍵の登録に使う。

# 「不老」への公開鍵の登録

---

- ▶ 以下をアクセスする。

<https://portal.cc.nagoya-u.ac.jp/cgi-bin/hpcportal.ja/index.cgi>

- ▶ ユーザ名とパスワードを聞かれるので、  
センターから発行されたユーザ名とパスワード  
を入れる。

# ポータル画面（ログイン前）

HPC Portal

[ English/Japanese ]

ログイン

## HPC Portal

HPC Portal Ver.3.0 by FUJITSU LIMITED

### ログイン

ユーザ名とパスワードを入力して [Login] ボタンをクリックしてください。

ユーザ名:  パスワード:

LOGIN RESET

#### お知らせ

- HPC PortalがサポートするOS (JREは、ファイルのUpload)

OS	Windows 7/8.1
ブラウザ	Internet Explorer
Java	Java Runtime Env

- JREは [こちら](#) (Oracleのサ)

センターから配られた  
利用者番号 と パスワード  
を入れる

# 鍵の登録

---

1. 「ログイン」ボタンを押す
  2. 成功すると、ログインメッセージがでる
  3. 左側メニューの「SSH公開鍵登録」をクリックする
  4. 「公開鍵」の画面に、公開鍵を  
カットアンドペーストする
  5. 「新規登録」ボタンを押す
- 
- ▶ 2度は鍵登録できません
  - ▶ 1度登録すれば「不老」全サブシステムで使えます

# ポータル画面（ログイン後）

HPC Portal

ユーザ名: z43403z ログアウト

About

SSH公開鍵登録

マニュアル

言語製品

利用手引書

ドキュメント

## 概要

HPC Portalは、Webブラウザから簡易なGUIでスーパーコンピュータシステム不老を利用できるWebシステムです。

## クライアントの条件

• HPC Portalがサポートするクライアント環境は以下の通りです。

OS	Windows 8.1/10
ブラウザ	Internet Explorer 11, Edge, FireFox

## 機能

Copyright 2012 - 2020 FUJITSU LIMITED

ここをクリック

75 Fujitsu HPC Portal

# ポータル画面（公開鍵登録）：

HPC Portal

ユーザ名: z43403z

ログアウト ヘルプ

About

パスワード変更

SSH公開鍵登録

マニュアル

言語製品

利用手引き

ドキュメント

## SSH公開鍵登録

登録者: z43403z

登録先: /home/z43403z/.ssh/authorized\_keys

公開鍵:

# 公開鍵をペースト

※公開鍵の中に改行文字が入らないようにご注意ください。  
※一度の操作で、1つの公開鍵を登録します。

新規 入力フィールドのクリア

公開鍵の再作成など、公開鍵に対するお問い合わせは下記にてご連絡ください。

[QAシステム利用相談](#)

公開鍵登録の際、以下の点にご注意下さい

- 改行文字が含まれていないこと。(特に末尾に改行が含まれていないことに注意してください)
- ヘッダ(ssh-rsa, ecdsa-sha2-nistp256, ecdsa-sha2-nistp384, ecdsa-sha2-nistp521, ssh-ed25519)が正しいこと。
- RSA公開鍵の場合、2048bit 以上で公開鍵を作成していること。
- ECDSA公開鍵の場合、256bit、384bitもしくは521bit で公開鍵を作成していること。
- Ed25519公開鍵の場合、256bit で公開鍵を作成していること。
- 全角文字などの不正文字が含まれないこと。

Copyright 2012 - 2020 FUJITSU LIMITED

76 Fujitsu HPC Portal

「不老」利用型ライブラリ講習会

ITC 富士大学 UNIVERSITY

V03L02

# 「不老」 Type サブシステムへログイン

---

- ▶ ターミナルから、以下を入力する  
`$ ssh flow-fx.cc.nagoya-u.ac.jp -l w490xxa`  
「-l」はハイフンと小文字のL、  
「aYYxxx」は利用者番号(数字)  
“aYYxxx”は、利用者番号を入れる
- ▶ 接続するかと聞かれるので、 yes を入れる
- ▶ 鍵の設定時に入れた  
自分が決めたパスワード(パスフレーズ)  
を入れる
- ▶ 成功すると、ログインができる

# 「不老」 TypeIサブシステム上のデータをPCに取り込む

- ▶ ターミナルから、以下を入力する

```
$ scp aYYxxx@flow-fx.cc.nagoya-u.ac.jp:~/a.f90 ./
```

「aYYxxx」は利用者番号(数字)

“aYYxxx”は、利用者番号を入れる

- ▶ 「不老」上に“a.f90”というファイルが無いとだめです。
- ▶ 「不老」上のホームディレクトリにある”a.f90”を、PCのカレントディレクトリに取ってくる
- ▶ ディレクトリごと取ってくるには、“-r” を指定

```
$ scp -r aYYxxx@flow-fx.cc.nagoya-u.ac.jp:~/SAMP ./
```

- ▶ 「不老」上のホームディレクトリにあるSAMPフォルダを、その中身ごと、PCのカレントディレクトリに取ってくる

# PCのファイルを「不老」Type1 サブシステム上に置く

- ▶ ターミナルから、以下を入力する

```
$ scp ./a.f90 aYYxxx@flow-fx.cc.nagoya-u.ac.jp:
```

「aYYxxx」は利用者番号(数字)

“aYYxxx”は、利用者番号を入れる

- ▶ PCのカレントディレクトリにある”a.f90”を、「不老」上のホームディレクトリに置く
- ▶ ディレクトリごと置くには、“-r” を指定

```
$ scp -r ./SAMP aYYxxx@flow-fx.cc.nagoya-u.ac.jp:
```

- ▶ PCのカレントディレクトリにあるSAMPフォルダを、その中身ごと、「不老」上のホームディレクトリに置く

## 参考：emacs の tramp機能（必要な人のみ）

- ▶ emacs が自分のパソコンに入っている人は、Tramp機能により、遠隔のファイルが操作できます
- ▶ 「不老」の秘密鍵を、SSHに登録します
- ▶ emacs を立ち上げます
- ▶ ファイル検索モードにします  
`^x ^f` (^はcontrol)
- ▶ “Find file:”の現在のパス名部分を消し、以下を入れます（ただし、t~は自分のログインIDにする）  
[Find file:/ssh:aYYxxx@flow-fx.cc.nagoya-u.ac.jp:](#)
- ▶ パスフレーズを入れると、ローカルファイルのようにF「不老」上のファイルが編集できます。

---

# 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



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
```



# 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);
```

```
}
```

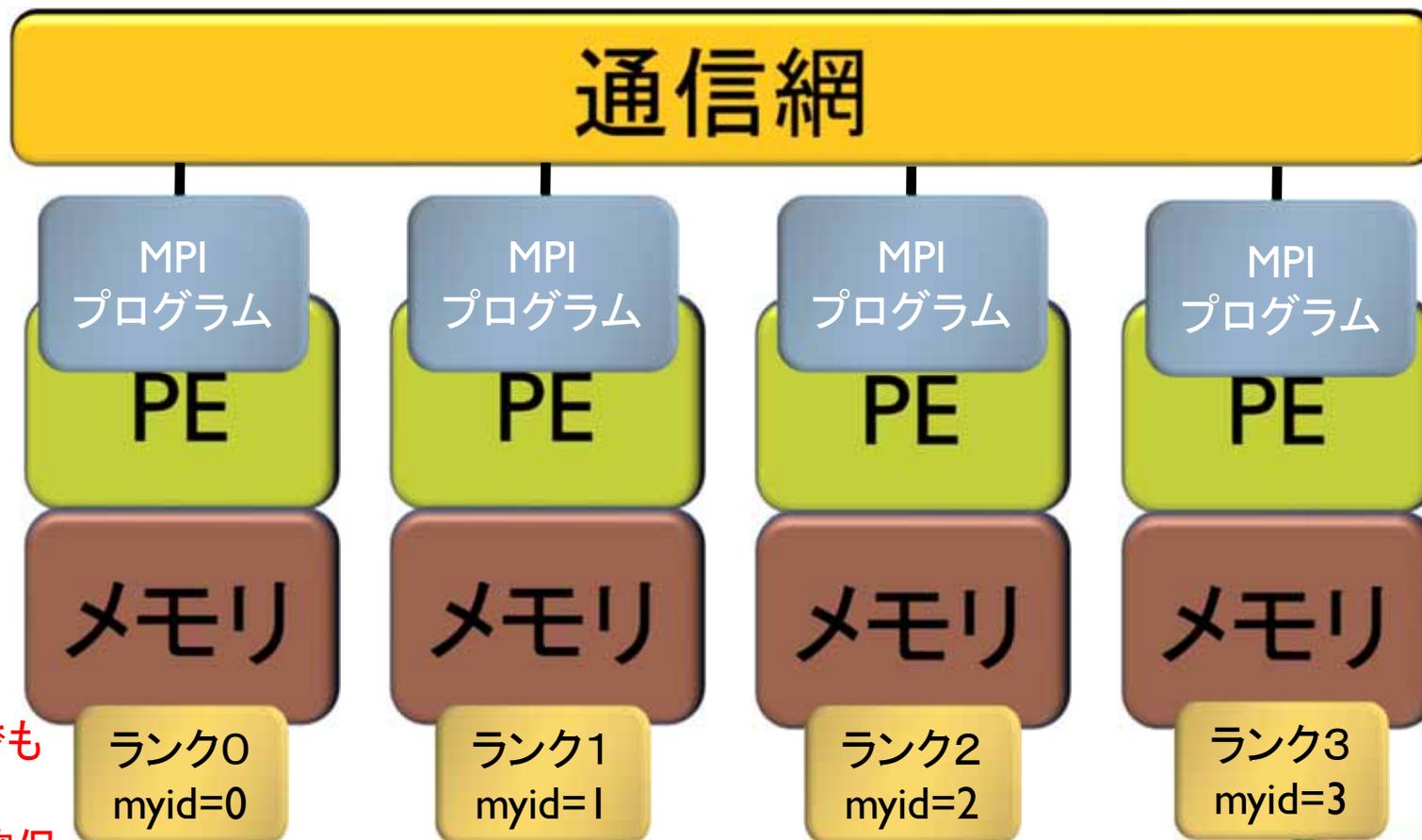
MPIの初期化

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
```

MPIの初期化

```
integer myid, numprocs  
integer ierr
```

自分のID番号を取得  
:各PEで値は異なる

```
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
```

全体のプロセッサ台数  
を取得  
:各PEで値は同じ

```
call MPI_FINALIZE(ierr)
```

MPIの終了

```
stop  
end
```

# 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

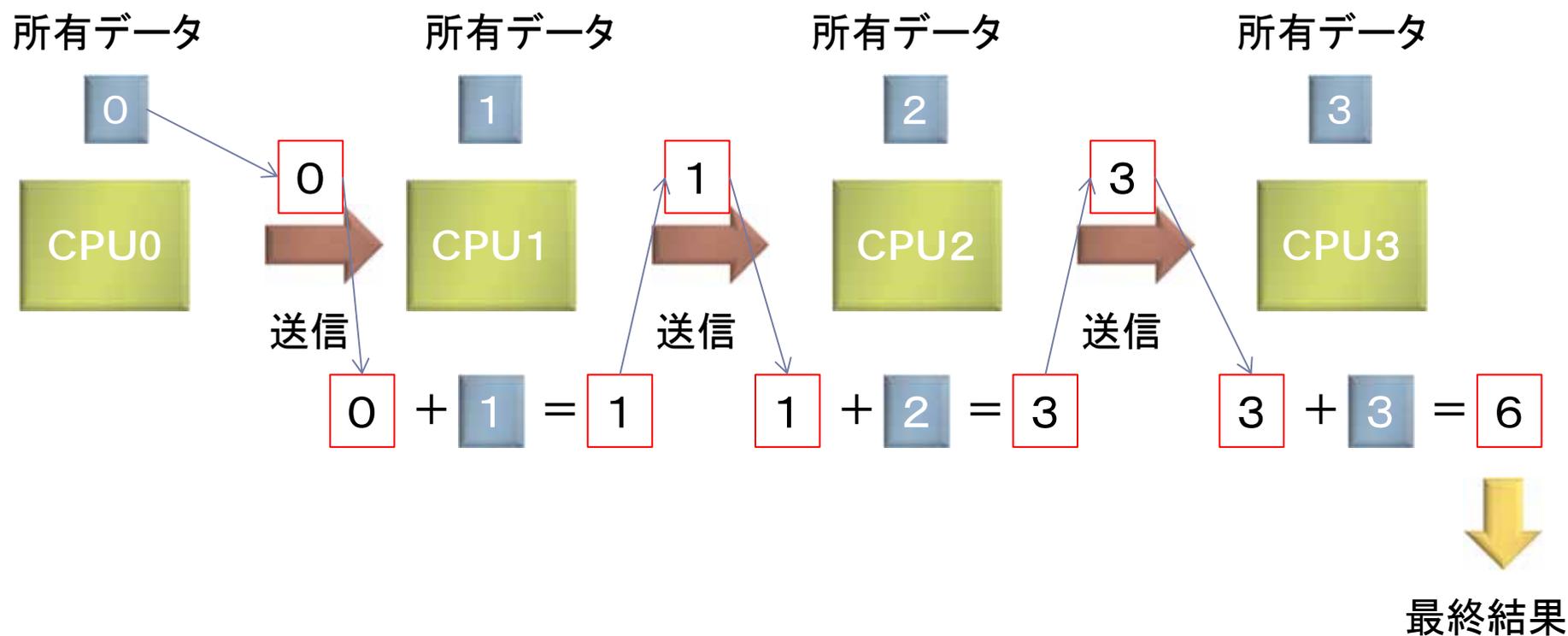
Hello parallel world! Myid:2

- 4プロセスなので、表示が4個である  
(1000プロセスなら1000個出力ができる)
- 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型データ一つを  
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  
一つを送信

# 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 ( $i=0; i < \log_2(\text{nprocs}); i++$ )
3. if ( ( $\text{myid} \& k$ ) ==  $k$ )
  - ▶ ( $\text{myid} - k$ )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
  - ▶ ( $\text{myid} + k$ )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）

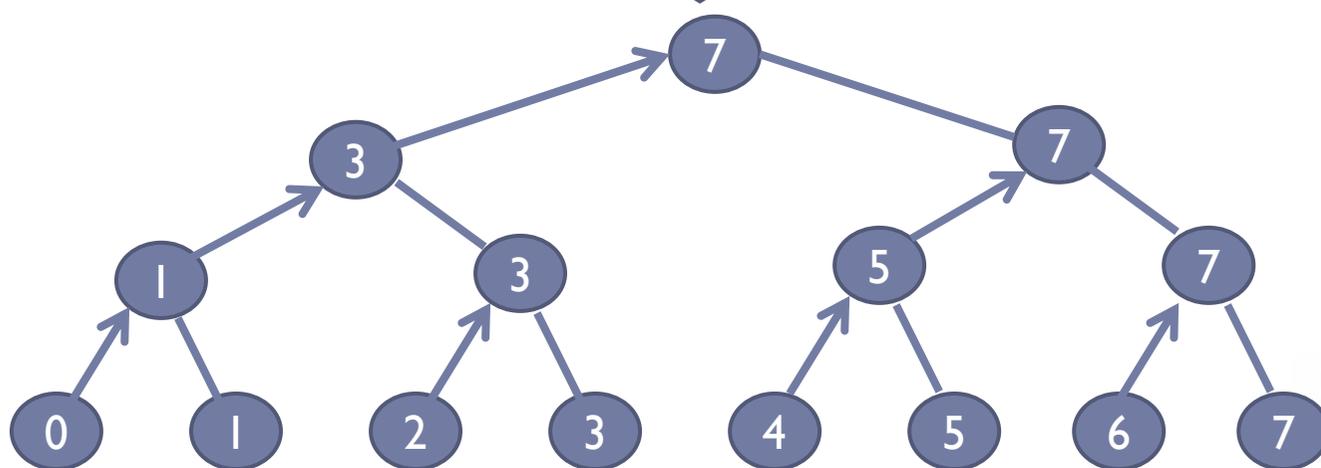
3段目 =  $\log_2(8)$  段目



2段目



1段目



# 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、 $\log_2(nprocs)$  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）

# 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  
— 並列処理とMPIの学習 —」
  - ▶ 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

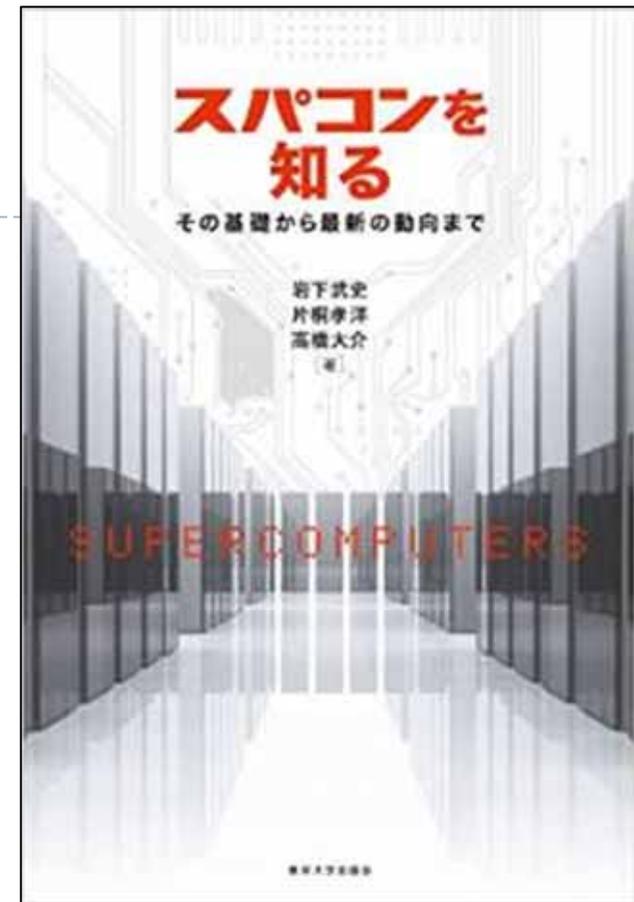
▶ 【本書の特徴】

- ▶ 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  
(株)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%)

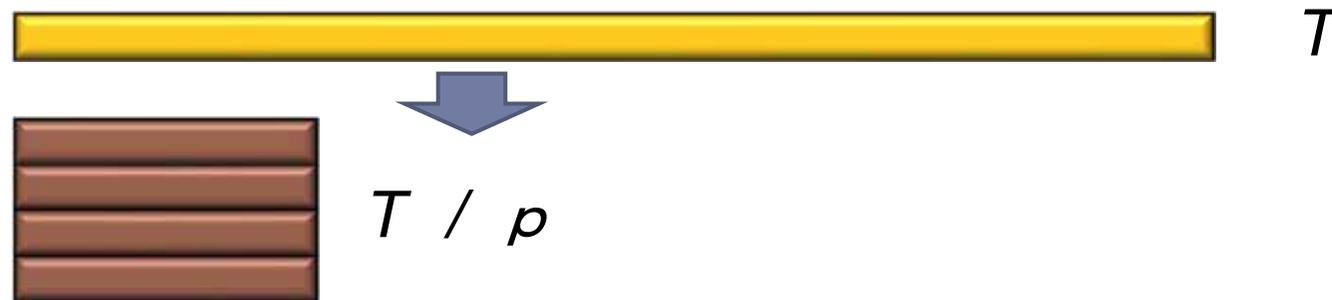
### ▶ 【本書の特徴】

- ▶ 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

# 並列と並行

## ▶ 並列 (Parallel)

- ▶ 物理的に並列 (時間的に独立)
- ▶ ある時間に実行されるものは多数



## ▶ 並行 (Concurrent)

- ▶ 論理的に並列 (時間的に依存)
- ▶ ある時間に実行されるものは1つ (= 1プロセッサで実行)



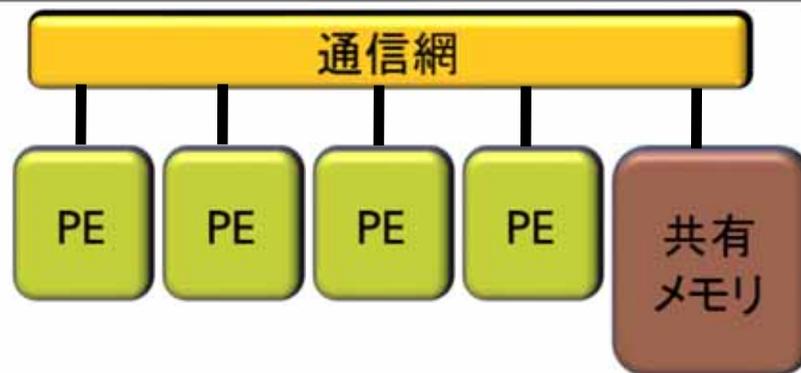
- ▶ 時分割多重、疑似並列
- ▶ OSによるプロセス実行スケジューリング (ラウンドロビン方式)

# 並列計算機の種類

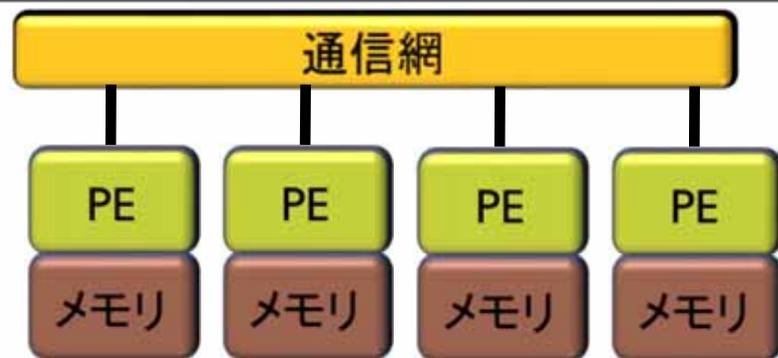
- ▶ Michael J. Flynn教授(スタンフォード大)の種類(1966)
- ▶ 単一命令・単一データ流  
(SISD, Single Instruction Single Data Stream)
- ▶ 単一命令・複数データ流  
(SIMD, Single Instruction Multiple Data Stream)
- ▶ 複数命令・単一データ流  
(MISD, Multiple Instruction Single Data Stream)
- ▶ 複数命令・複数データ流  
(MIMD, Multiple Instruction Multiple Data Stream)

# 並列計算機のメモリ型による分類

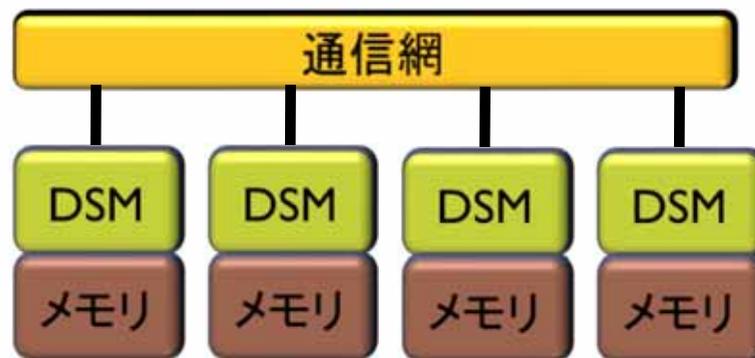
1. 共有メモリ型  
(SMP、Symmetric Multiprocessor)



2. 分散メモリ型  
(メッセージパッシング)

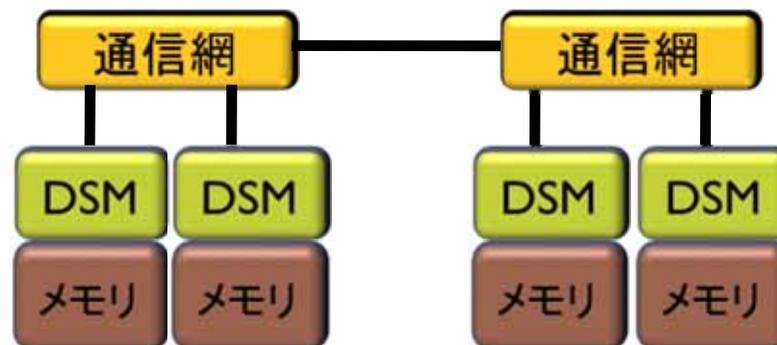


3. 分散共有メモリ型  
(DSM、Distributed Shared Memory)



# 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)

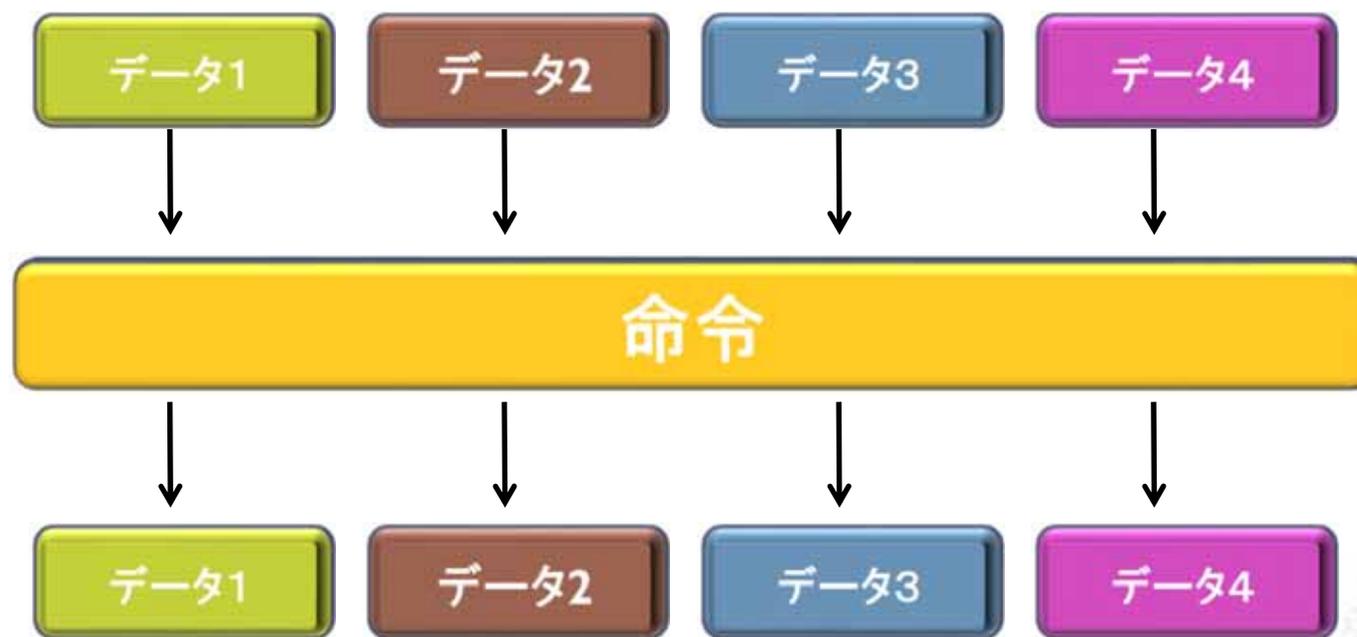


# 並列計算機の分類とMPIとの関係

- ▶ 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

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

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-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	$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{pmatrix}$	=	$\begin{pmatrix} 1*9+2*6+3*3 & 1*8+2*5+3*2 & 1*7+2*4+3*1 \end{pmatrix}$
CPU1	$\begin{pmatrix} 4 & 5 & 6 \end{pmatrix}$			$\begin{pmatrix} 4*9+5*6+6*3 & 4*8+5*5+6*2 & 4*7+5*4+6*1 \end{pmatrix}$
CPU2	$\begin{pmatrix} 7 & 8 & 9 \end{pmatrix}$			$\begin{pmatrix} 7*9+8*6+9*3 & 7*8+8*5+9*2 & 7*7+8*4+9*1 \end{pmatrix}$

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: カレールーを入れる

### ● 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

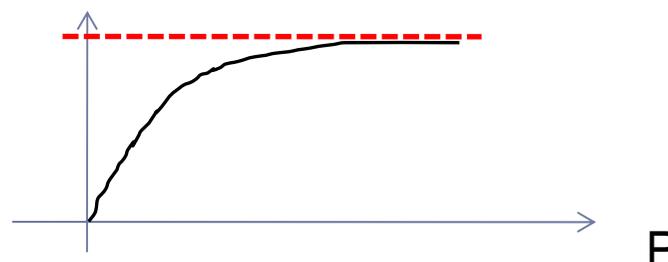
- ▶ 式:  $S_p = T_S / T_p$  ( $0 \leq S_p$ )
- ▶  $T_S$  : 逐次の実行時間、  $T_p$  : P台での実行時間
- ▶ P台用いて  $S_p = P$  のとき、理想的な(ideal)速度向上
- ▶ P台用いて  $S_p > P$  のとき、スーパーニア・スピードアップ
  - ▶ 主な原因は、並列化により、データアクセスが局所化されて、キャッシュヒット率が向上することによる高速化

## ▶ 並列化効率

- ▶ 式:  $E_p = S_p / P \times 100$  ( $0 \leq E_p$ ) [%]

## ▶ 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  $\alpha$  とする。
- ▶ このとき、台数効果は以下のようなになる。

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

- ▶ 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 ( $P \rightarrow \infty$ )、台数効果は、高々  $1 / (1-\alpha)$  である。

## (アムダールの法則)

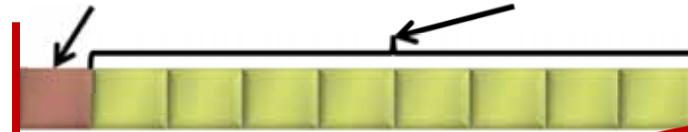
- ▶ 全体の90%が並列化できたとしても、無限大の数のプロセッサをつかっても、 $1 / (1-0.9) = 10$  倍 にしかない！

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

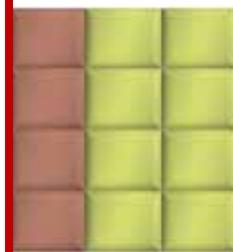
並列化できない部分(1ブロック) 並列化できる部分(8ブロック)

● 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  $\neq$  6倍

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

# 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、特定のソフトウェ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ーラップ
      - 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:メーカー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

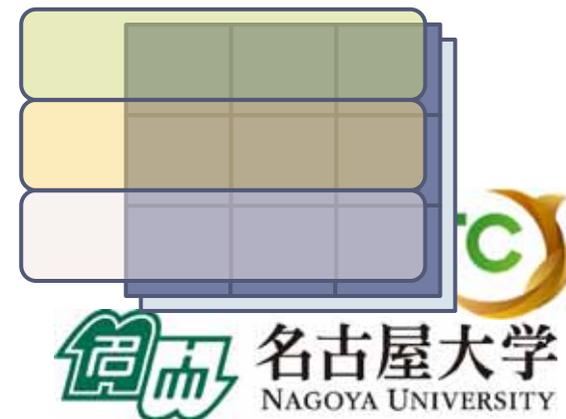
- ▶ MPI\_Reduce; MPI\_Allreduce; MPI\_Barrier;

## ▶ 時間計測関数

- ▶ MPI\_Wtime

# コミュニケータ

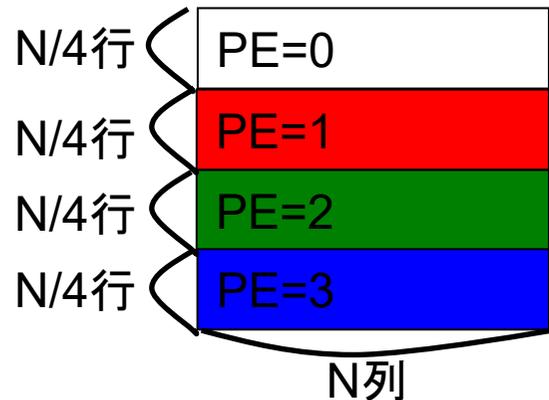
- ▶ MPI\_COMM\_WORLDは、**コミュニケータ**とよばれる概念を保存する変数
- ▶ コミュニケータは、操作を行う対象のプロセッサ群を定める
- ▶ 初期状態では、**0番～numprocs - 1番**までのプロセッサが、1つのコミュニケータに割り当てられる
  - ▶ この名前が、“**MPI\_COMM\_WORLD**”
- ▶ プロセッサ群を分割したい場合、**MPI\_Comm\_split** 関数を利用
  - ▶ メッセージを、一部のプロセッサ群に放送するとき利用
  - ▶ “マルチキャスト”で利用



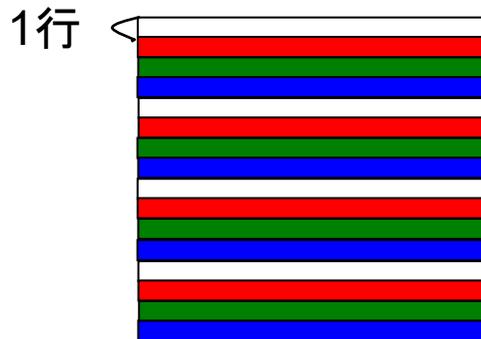
# 基本演算

- ▶ 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

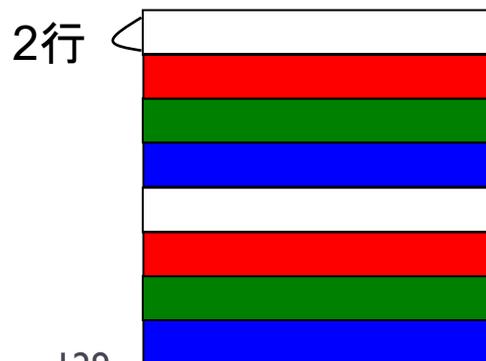
# 1次元分散



- (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

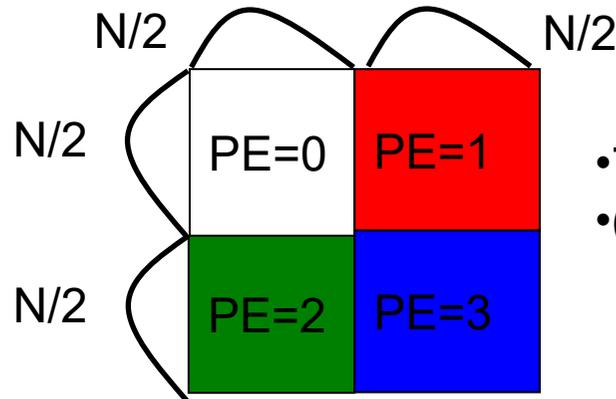


- (行方向)ブロック・サイクリック分割方式
- (Cyclic(2), \*) 分散方式

この例の「2」: <ブロック幅>とよぶ

「不老」利用型ライブラリ講習会

# 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

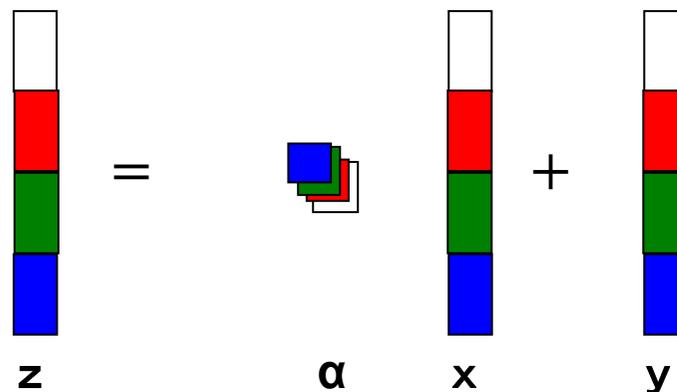
「不老」利用型ライブラリ講習会

# ベクトルどうしの演算

- ▶ 以下の演算

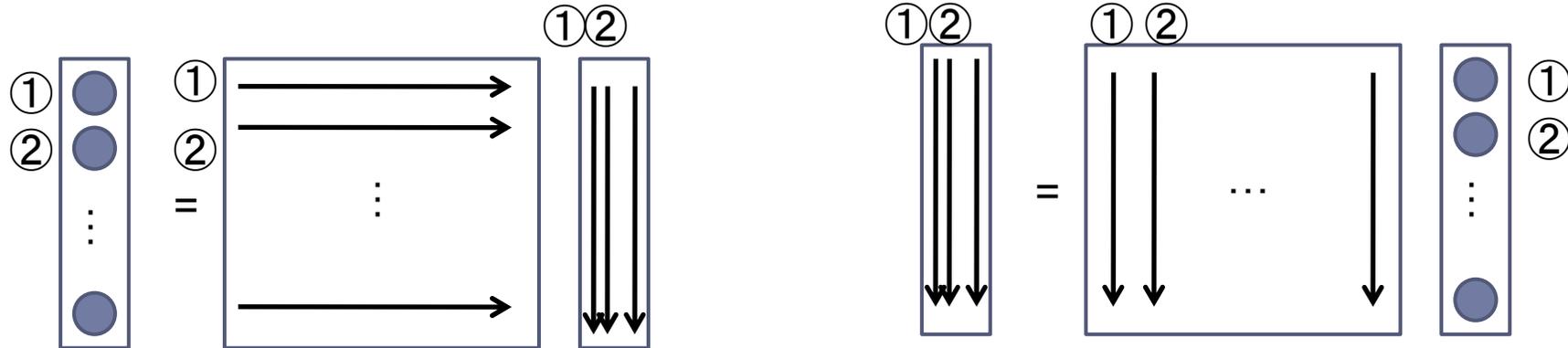
$$z = \alpha x + y$$

- ▶ ここで、 $\alpha$ 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  $\alpha$  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

132

```
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];
    }
}
```

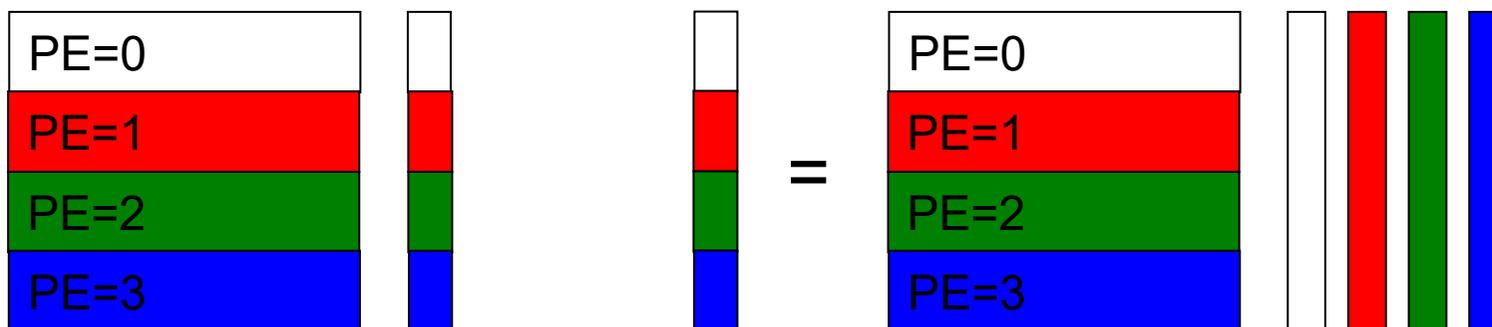
**<列方式>**： Fortran言語向き

「不老」利用型ライブラリ講習会

# 行列とベクトルの積

## ＜行方式の場合＞

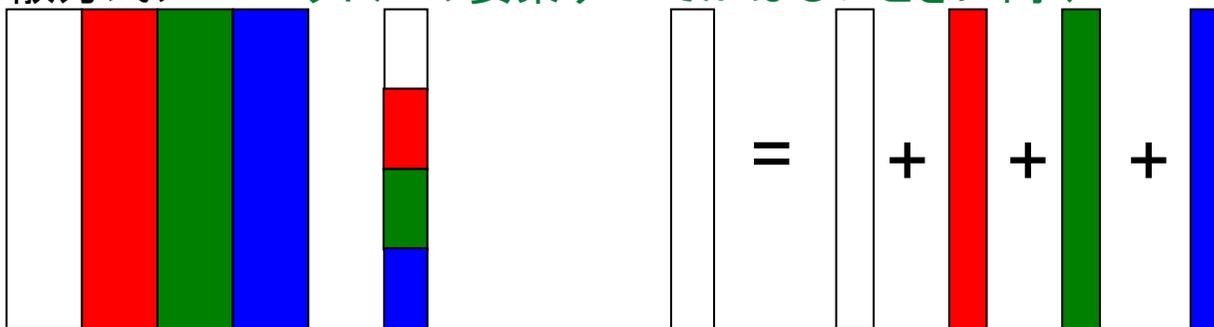
＜行方向分散方式＞ : 行方式に向く分散方式



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



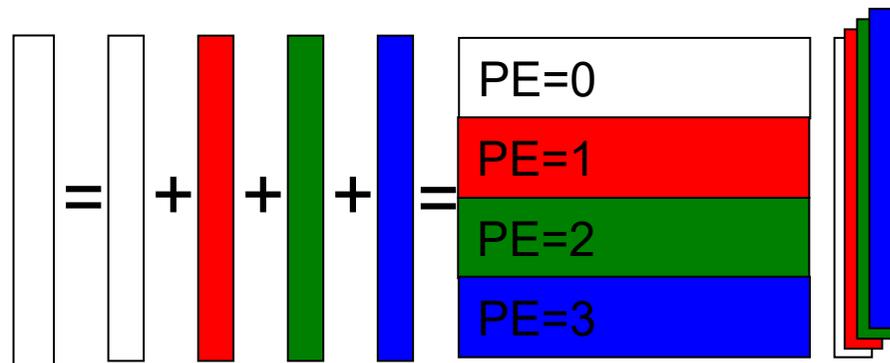
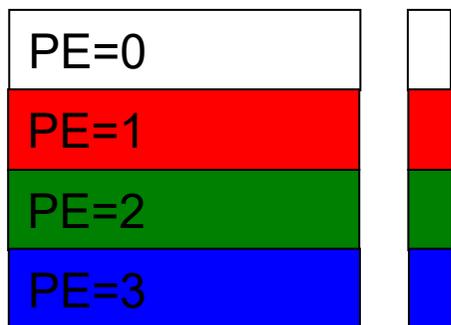
各PE内で行列-ベクトル積  
を行う

`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＞

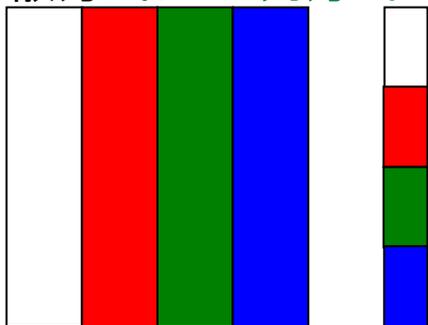
＜行方向分散方式＞ : 無駄が多く使われない



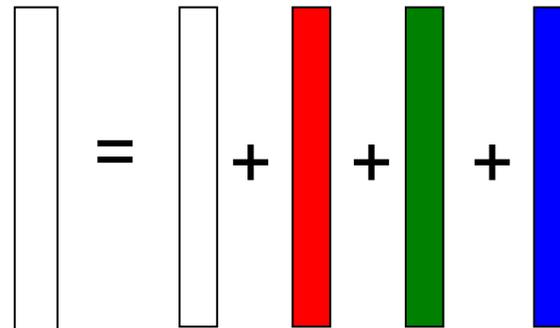
右辺ベクトルを `MPI_Allgather`関数  
を利用して、全PEで所有する

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)

---

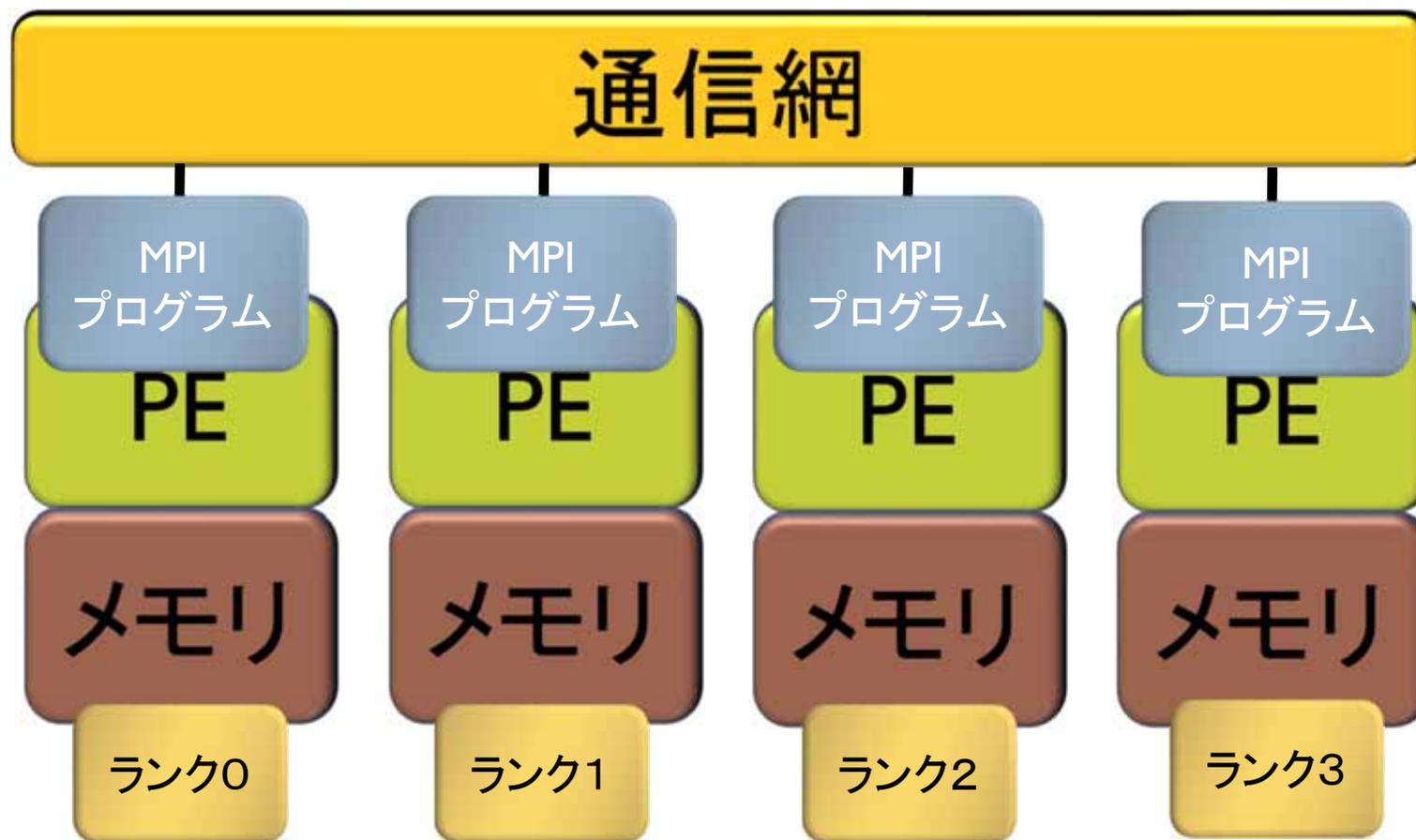
# 基本的なMPI関数

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(Processor 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



# 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言語インター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,  source,  
                 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)

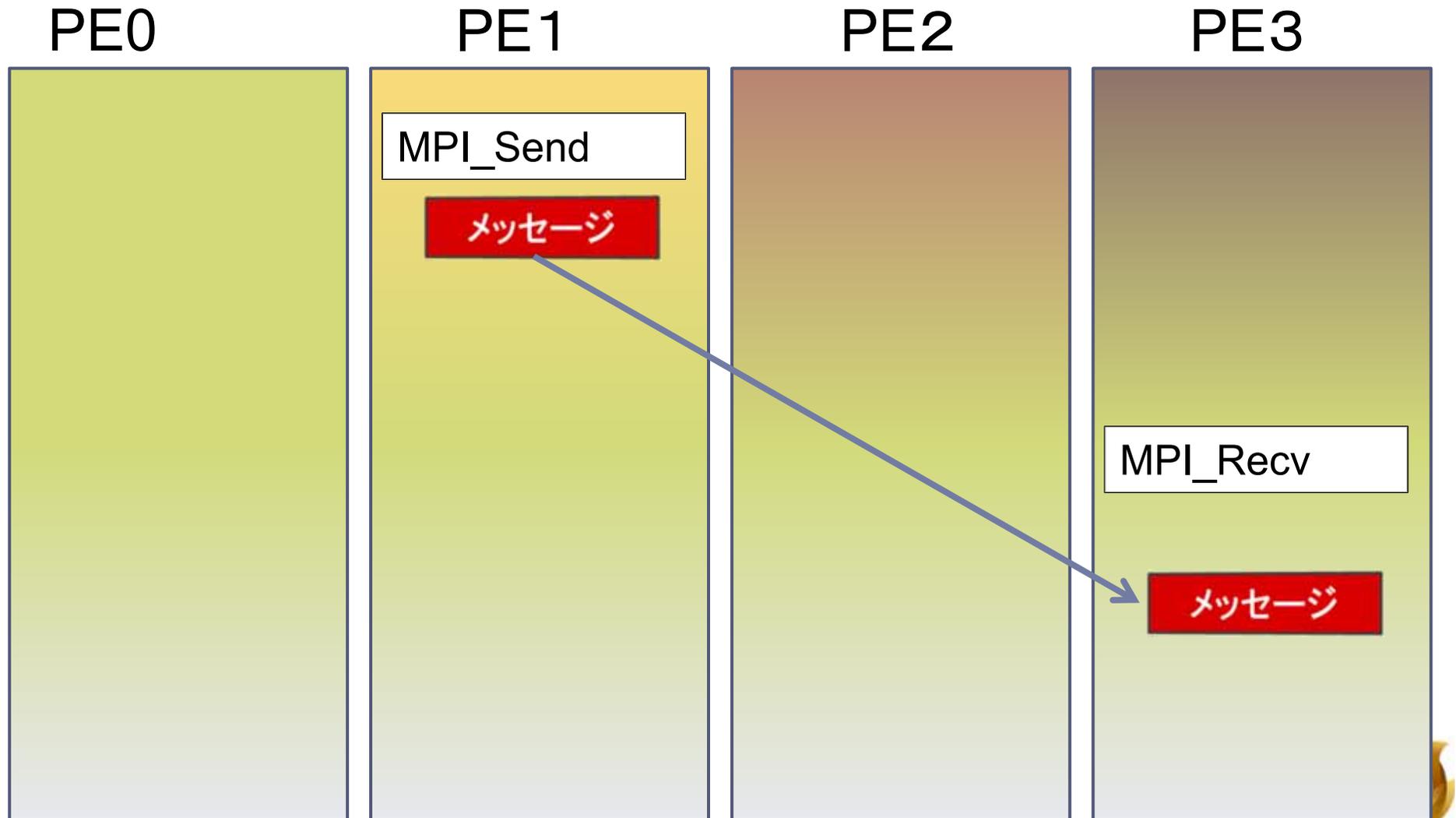
- ▶ **itag** : 整数型。受信したいメッセージに付いているタグの値を指定。
  - ▶ 任意のタグ値のメッセージを受信したいときは、**MPI\_ANY\_TAG** を指定。
- ▶ **icomm** : 整数型。PE集団を認識する番号であるコミュニケータを指定。
  - ▶ 通常では**MPI\_COMM\_WORLD** を指定すればよい。
- ▶ **istatus** : MPI\_Status型(整数型の配列)。受信状況に関する情報が入る。**かならず専用の型宣言をした配列を確保すること。**
  - ▶ 要素数が**MPI\_STATUS\_SIZE**の整数配列が宣言される。
  - ▶ 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)

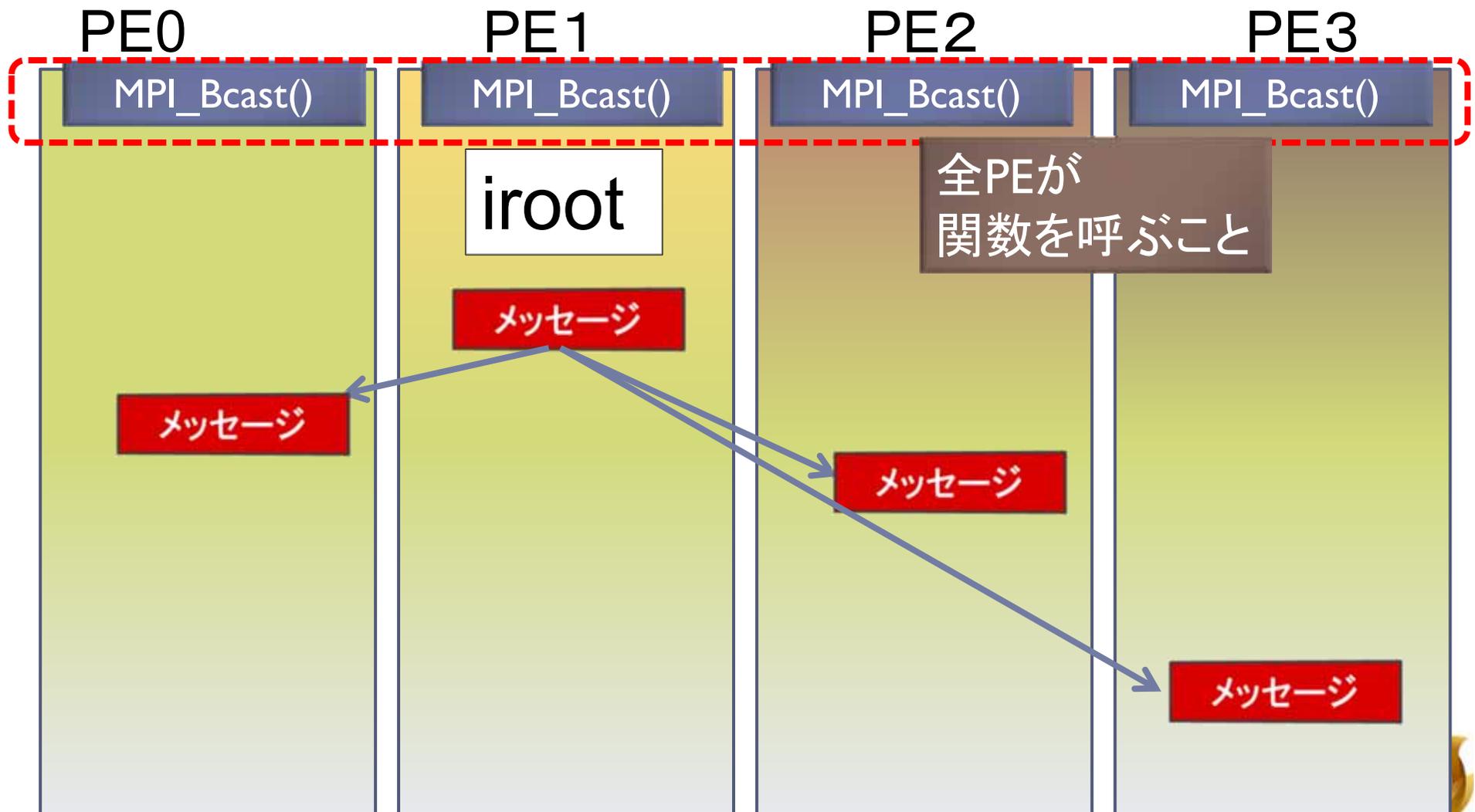


# 基礎的なMPI関数—MPI\_Bcast

```
▶ ierr = MPI_Bcast(sendbuf, icount, idatatype,  
    iroot, icommm);
```

- ▶ **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。
- ▶ **icommm** : 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  
インター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

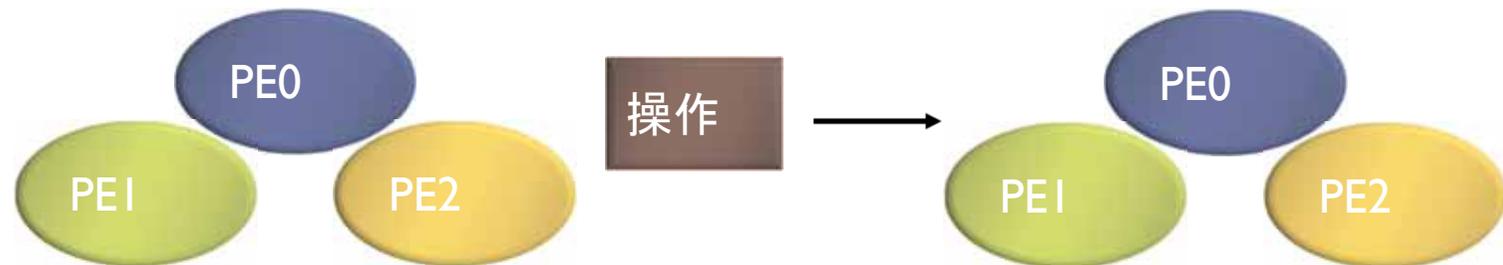
- ▶ **MPI\_Reduce**関数

- ▶ 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);
```

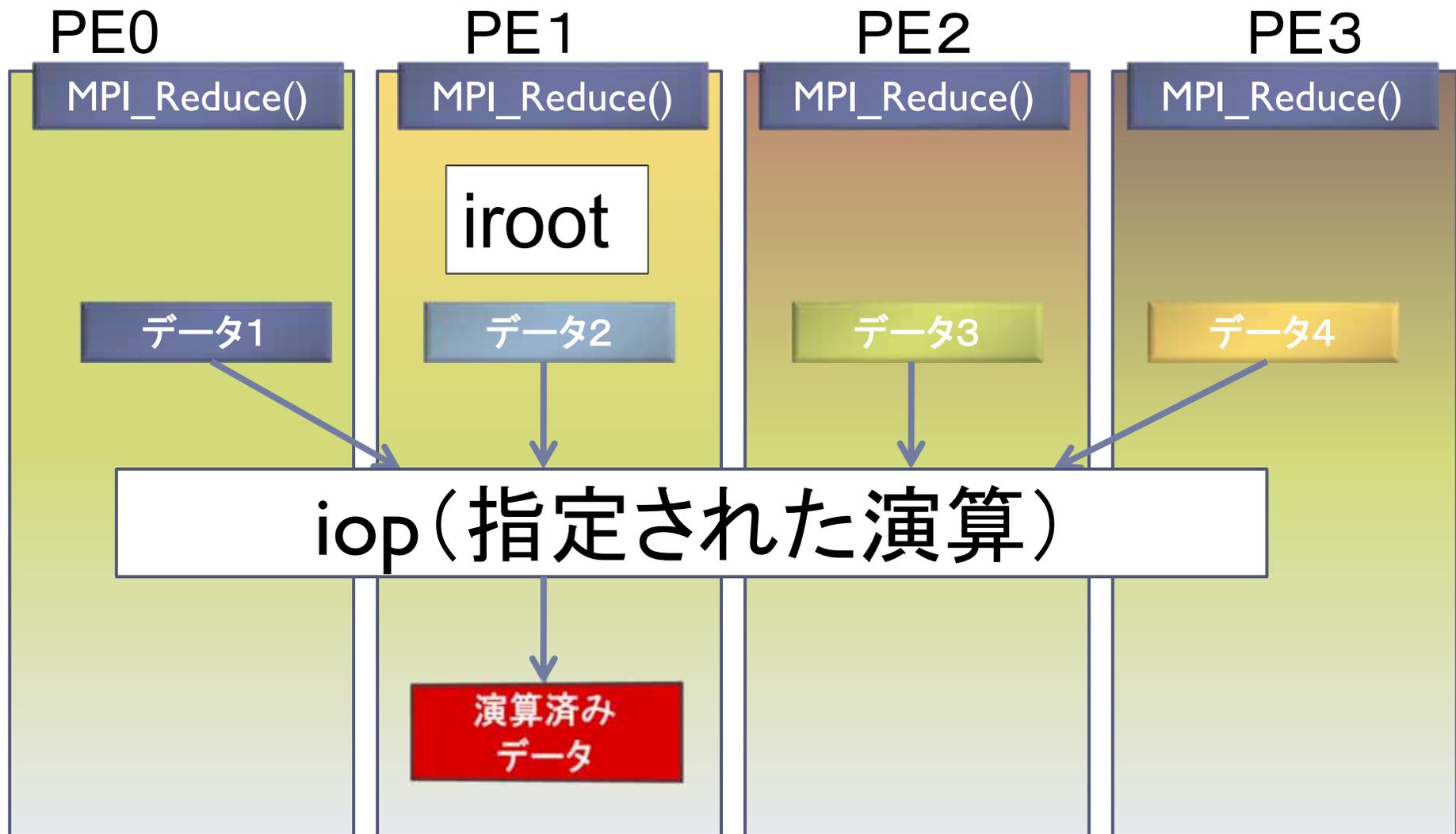
- ▶ **sendbuf** : 送信領域の先頭番地を指定する。
- ▶ **recvbuf** : 受信領域の先頭番地を指定する。iroot で指定したPEのみで書き込みがなされる。  
送信領域と受信領域は、同一であってはならない。  
すなわち、異なる配列を確保しなくてはならない。
- ▶ **icount** : 整数型。送信領域のデータ要素数を指定する。
- ▶ **idatatype** : 整数型。送信領域のデータの型を指定する。
  - ▶ (Fortran) <最小／最大値と位置>を返す演算を指定する場合は、**MPI\_2INTEGER**(整数型)、**MPI\_2REAL**(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

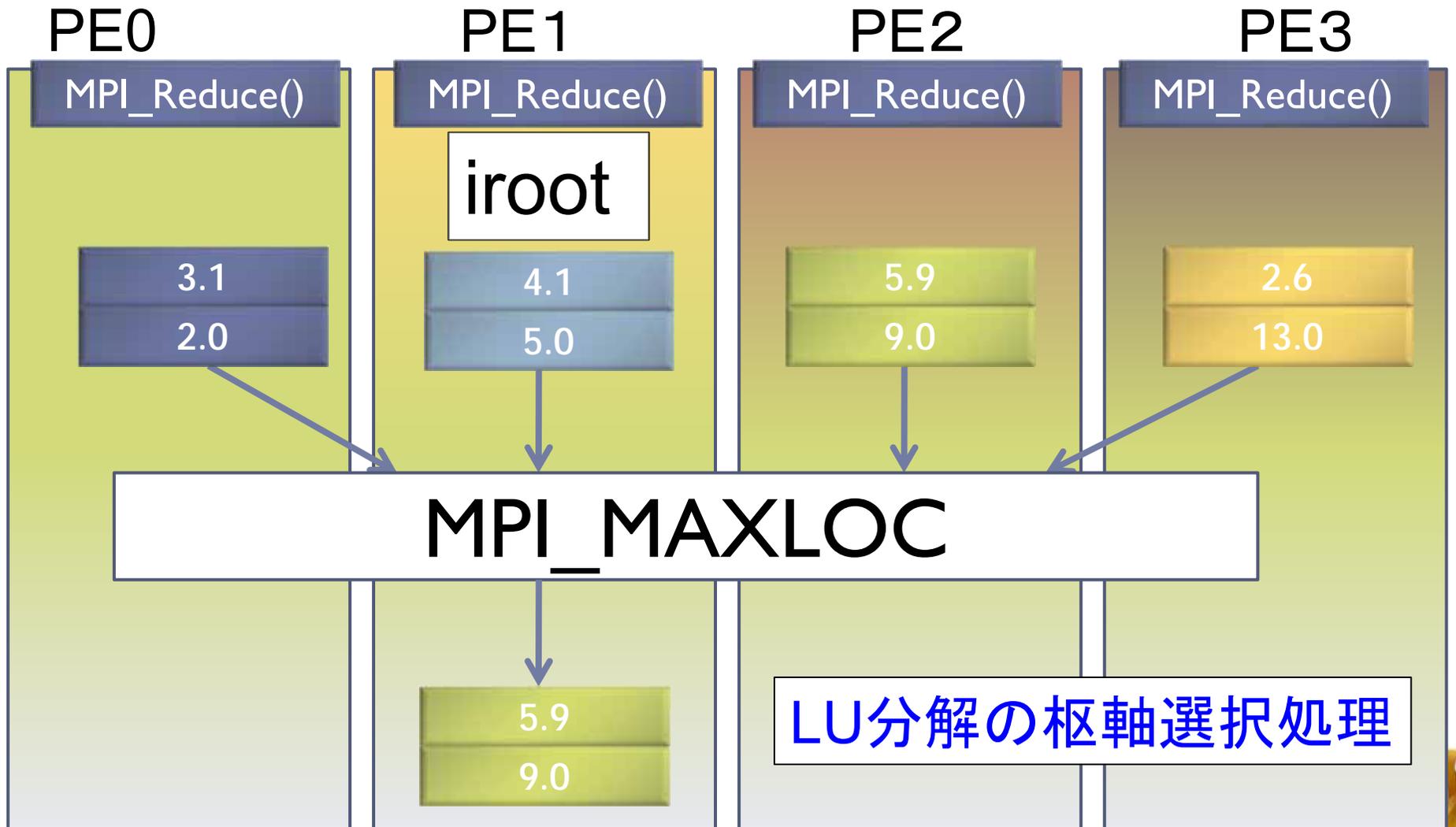
---

- ▶ **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, icommm);
```

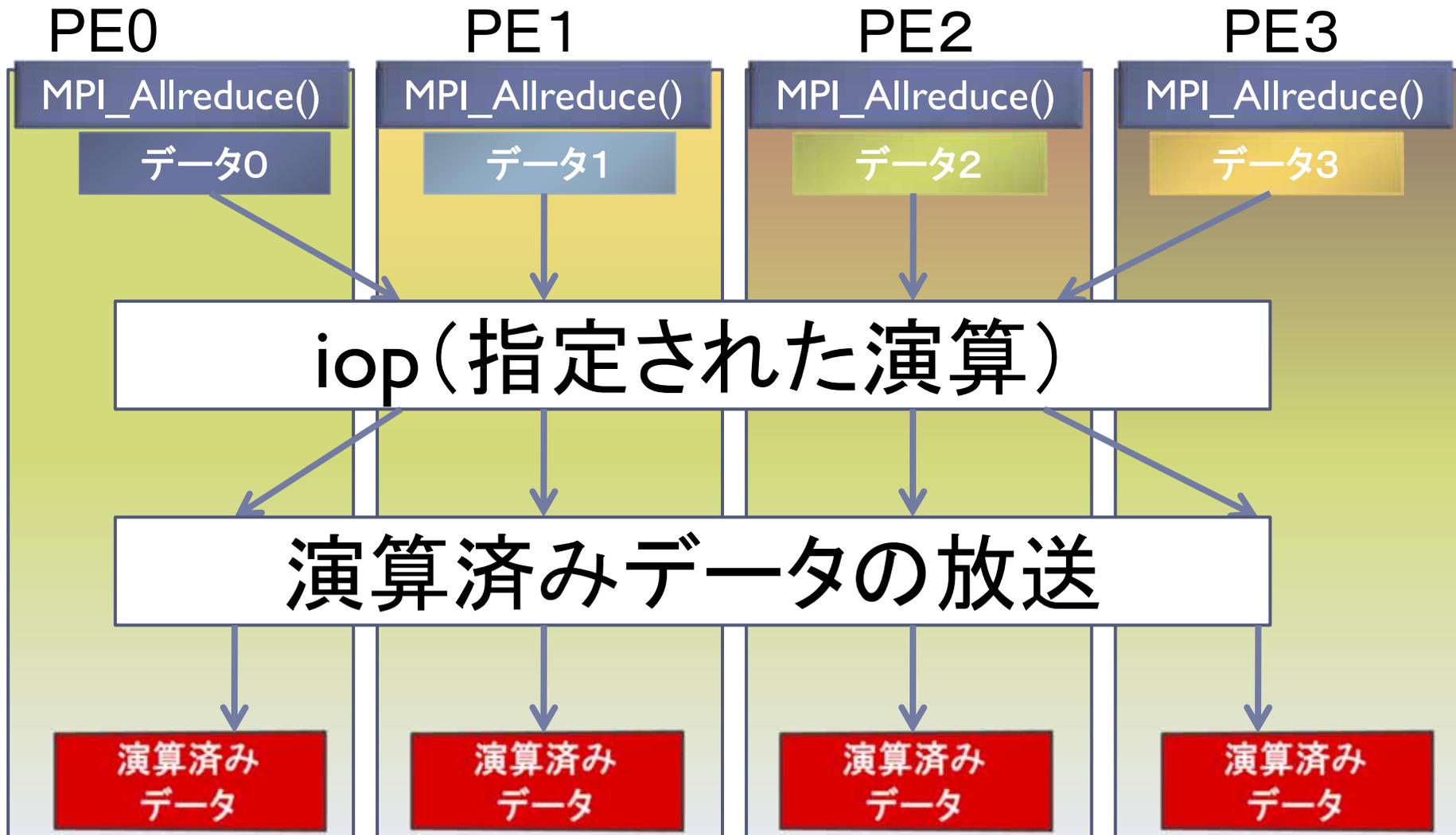
- ▶ **sendbuf** : 送信領域の先頭番地を指定する。
- ▶ **recvbuf** : 受信領域の先頭番地を指定する。iroot で指定したPEのみで書き込みがなされる。  
送信領域と受信領域は、同一であってはならない。  
すなわち、異なる配列を確保しなくてはならない。
- ▶ **icount** : 整数型。送信領域のデータ要素数を指定する。
- ▶ **idatatype** : 整数型。送信領域のデータの型を指定する。
  - ▶ 最小値や最大値と位置を返す演算を指定する場合は、**MPI\_2INT**(整数型)、**MPI\_2FLOAT**(単精度型)、**MPI\_2DOUBLE**(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

---

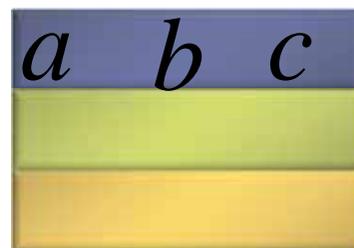
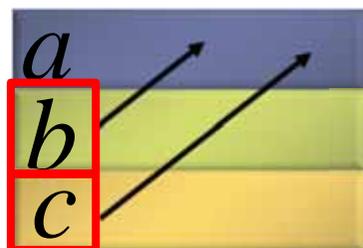
## ▶ 性能について

- ▶ 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

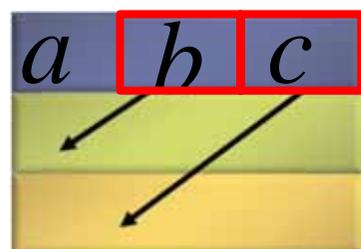
- ▶ 行列  $A$  が (Block, \*) 分散されているとする。
- ▶ 行列  $A$  の転置行列  $A^T$  を作るには、MPIでは次の2通りの関数を用いる

- ▶ 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);
```

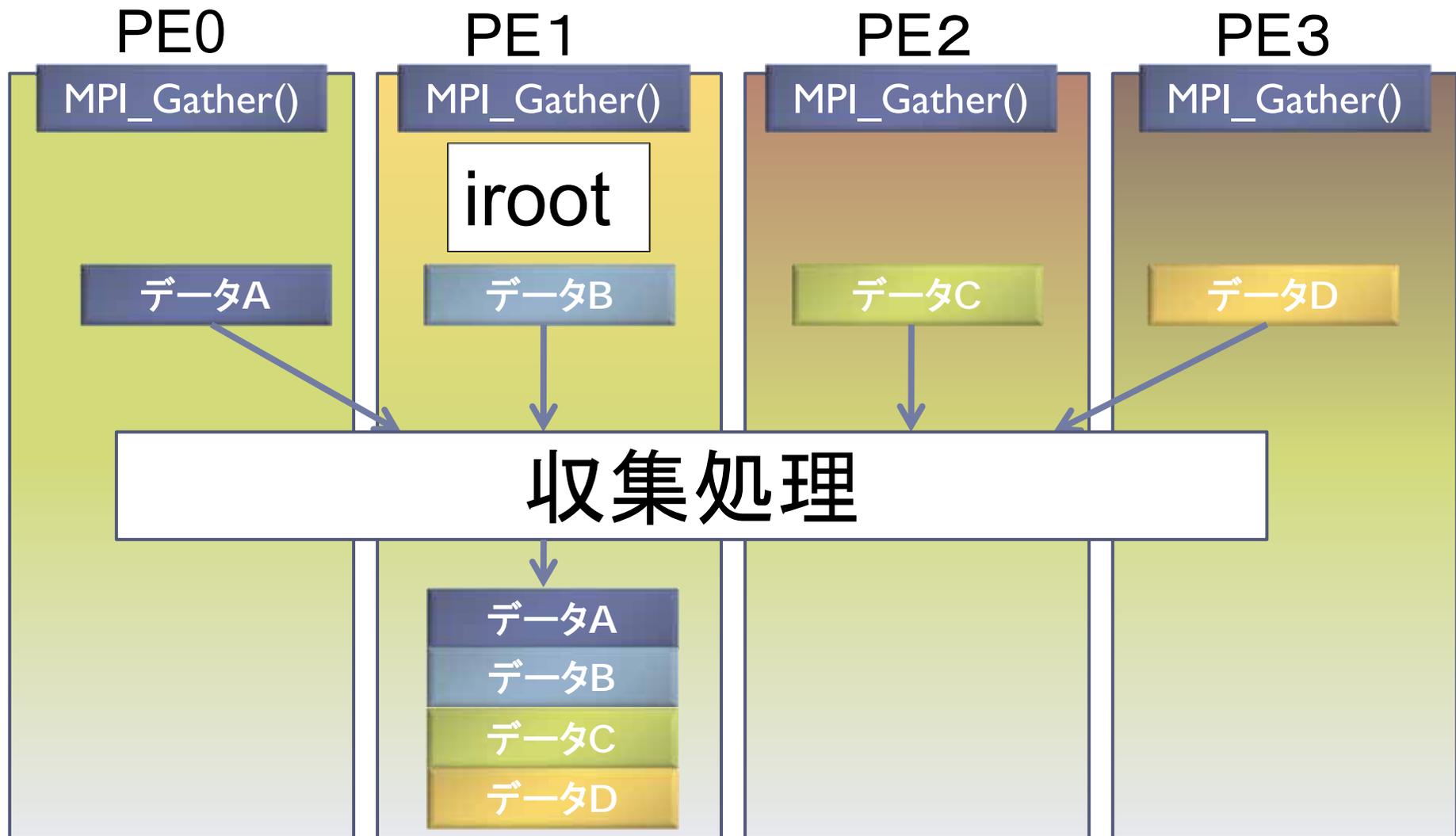
- ▶ **sendbuf** : 送信領域の先頭番地を指定する。
- ▶ **isendcount**: 整数型。送信領域のデータ要素数を指定する。
- ▶ **isendtype** : 整数型。送信領域のデータの型を指定する。
- ▶ **recvbuf** : 受信領域の先頭番地を指定する。iroot で指定したPEのみで書き込みがなされる。
  - ▶ なお原則として、送信領域と受信領域は、同一であってはならない。すなわち、異なる配列を確保しなくてはならない。
- ▶ **irecvcount**: 整数型。受信領域のデータ要素数を指定する。
  - ▶ この要素数は、1PE当たりの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。
- ▶ **irroot** : 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

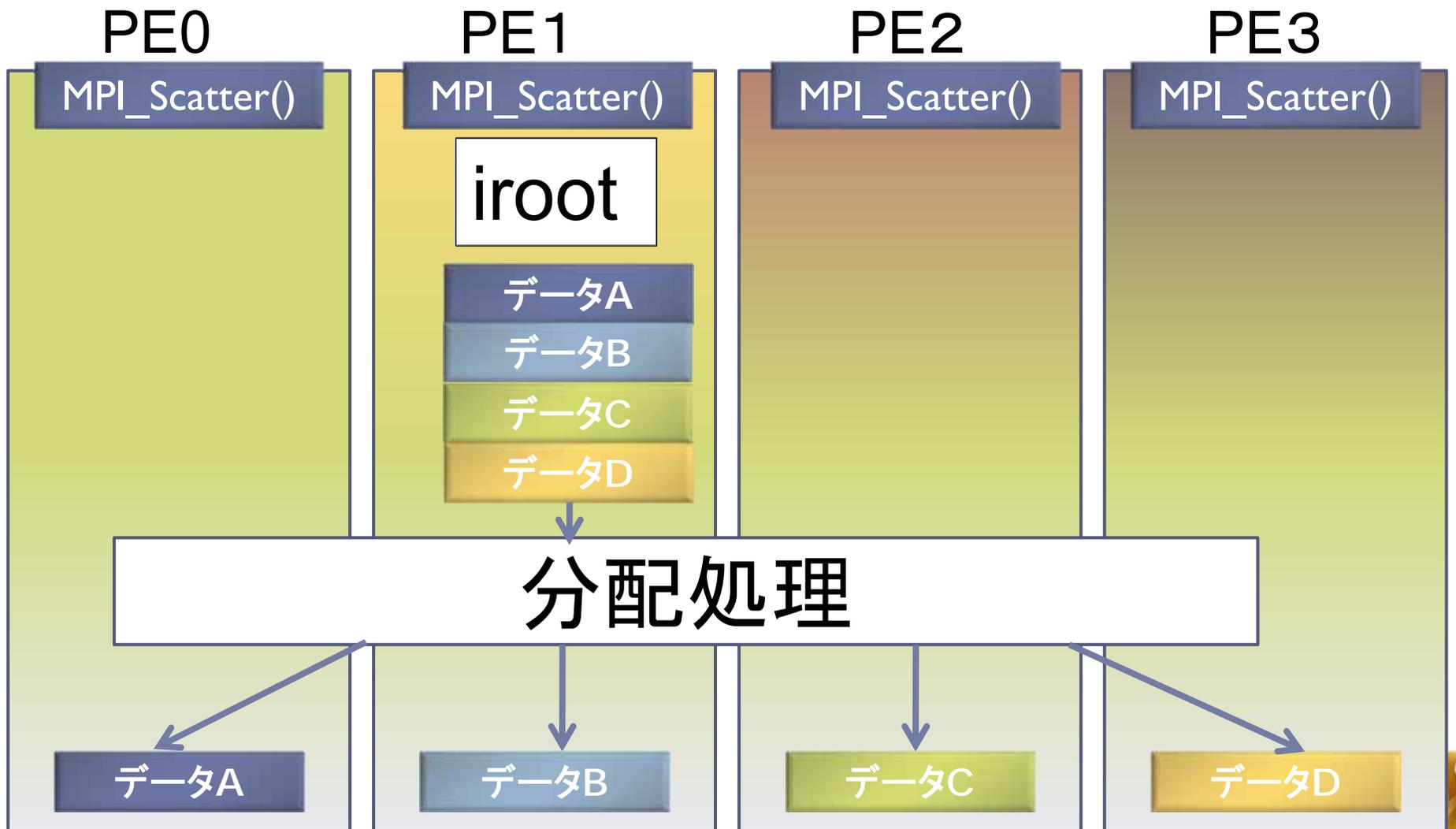
- ▶ `ierr = MPI_Scatter ( sendbuf, isendcount, isendtype, recvbuf, irecvcount, irecvtype, iroot, ictmm);`
- ▶ `sendbuf` : 送信領域の先頭番地を指定する。
- ▶ `isendcount`: 整数型。送信領域のデータ要素数を指定する。
  - ▶ この要素数は、1PEあたりに送られる送信データ数を指定すること。
  - ▶ MPI\_Scatter 関数では各PEで異なる数のデータを分散することはできないので、同じ値を指定すること。
- ▶ `isendtype` : 整数型。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。
- ▶ **irroot** : 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.パチエコ 著 / 秋葉 博 訳
2. Message Passing Interface Forum  
( <http://www.mpi-forum.org/> )
3. 並列コンピュータ工学、富田眞治著、昭晃堂(1996)

---

# BLAS演習（演習）

# 演習内容

---

- ▶ BLASとは
- ▶ GOTO BLASとは
- ▶ LAPACKとは
- ▶ ScaLAPACKとは
- ▶ BLASの利用法と実習 (DGEMM)

# 1.8 BLASとPBLAS

---

- ▶ **BLAS (Basic Linear Algebra Subprograms、基本線形代数副プログラム集)**
  - ▶ 線形代数計算で用いられる、基本演算を標準化 (API化) したものの。
  - ▶ 普通は、密行列用の線形代数計算用の基本演算の副プログラムを指す。
  - ▶ 疎行列の基本演算用の **<スパースBLAS>** というものがあるが、まだ定着していない。

## 1.8 BLASとPBLAS

---

- ▶ 高性能なライブラリの〈作成の手間〉と、〈プログラム再利用性〉を高める目的で提案
- ▶ BLAS演算の性能改善を、個々のユーザが、個々のプログラムで独立して行うのは、ソフトウェア開発効率が悪い
  - ▶ 〈工学的〉でない
- ▶ 性能を改善するチューニングは、経験のないユーザには無理
  - ▶ 〈職人芸〉 ≠ 〈科学、工学〉
- ▶ BLASは、数学ソフトウェアにおける〈ソフトウェア工学〉のはしり

# 1.8 BLASとPBLAS

---

- ▶ BLASでは、以下のように分類わけをして、サブルーチンの命名規則を統一
  1. 演算対象のベクトルや行列の型(整数型、実数型、複素型)
  2. 行列形状(対称行列、三重対角行列)
  3. データ格納形式(帯行列を二次元に圧縮)
  4. 演算結果が何か(行列、ベクトル)
- ▶ 演算性能から、以下の3つに演算を分類
  - ▶ レベル1 BLAS: ベクトルとベクトルの演算
  - ▶ レベル2 BLAS: 行列とベクトルの演算
  - ▶ レベル3 BLAS: 行列と行列の演算

# 1.8 BLASとPBLAS

---

## ▶ レベル1 BLAS

### ▶ ベクトル内積、ベクトル定数倍の加算、など

▶ 例:  $y = \alpha x + y$

▶ データの読み出し回数、演算回数がほぼ同じ

▶ データの再利用(キャッシュに乗ったデータの再利用によるデータアクセス時間の短縮)がほとんどできない

▶ 実装による性能向上が、あまり期待できない

▶ ほとんど、計算機ハードウェアの演算性能

▶ レベル1BLASのみで演算を実装すると、演算が本来持っているデータ再利用性がなくなる

▶ 例: 行列-ベクトル積を、レベル1BLASで実装

# 1.8 BLASとPBLAS

## ▶ レベル2 BLAS

### ▶ 行列-ベクトル積などの演算

▶ 例:  $y = \alpha Ax + \beta y$

### ▶ 前進/後退代入演算、 $Tx = y$ ( $T$ は三角行列)を $x$ について解く演算、を含む

### ▶ レベル1BLASのみの実装による、データ再利用性の喪失を回避する目的で提案

### ▶ 行列とベクトルデータに対して、データの再利用性あり

▶ データアクセス時間を、実装法により短縮可能

▶ (実装法により)性能向上がレベル1BLASに比べしやすい(が十分でない)

## 1.8 BLASとPBLAS

---

### ▶ レベル3 BLAS

#### ▶ 行列-行列積などの演算

▶ 例:  $C = \alpha AB + \beta C$

▶ 共有記憶型の並列ベクトル計算機では、レベル2 BLASでも性能向上が達成できない。

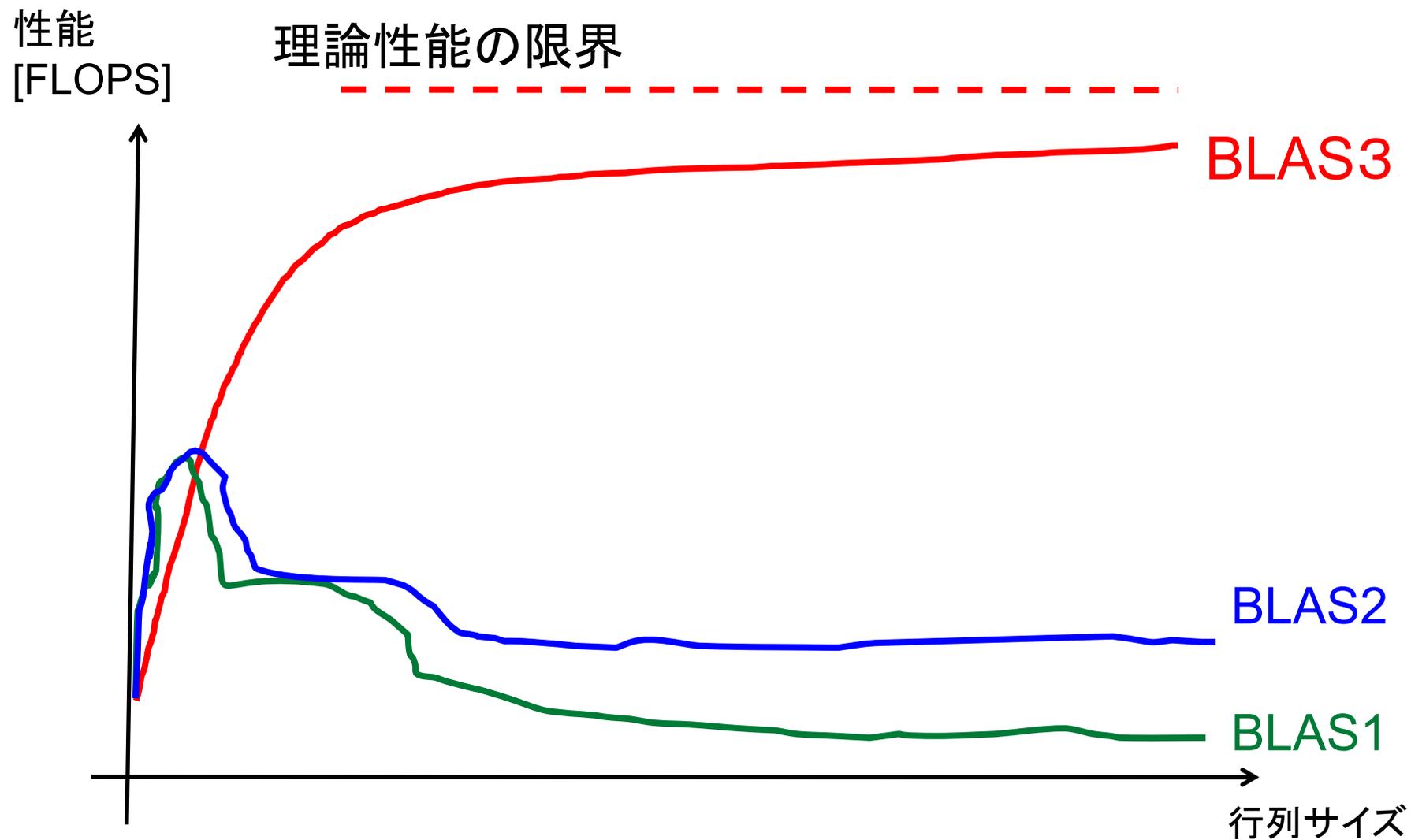
▶ 並列化により1PE当たりのデータ量が減少する。

▶ より大規模な演算をとり扱わないと、再利用の効果がない。

▶ 行列-行列積では、行列データ  $O(n^2)$  に対して演算は  $O(n^3)$  なので、データ再利用性が原理的に高い。

▶ 行列積は、アルゴリズムレベルでもブロック化できる。さらにデータの局所性を高めることができる。

# 典型的なBLASの性能



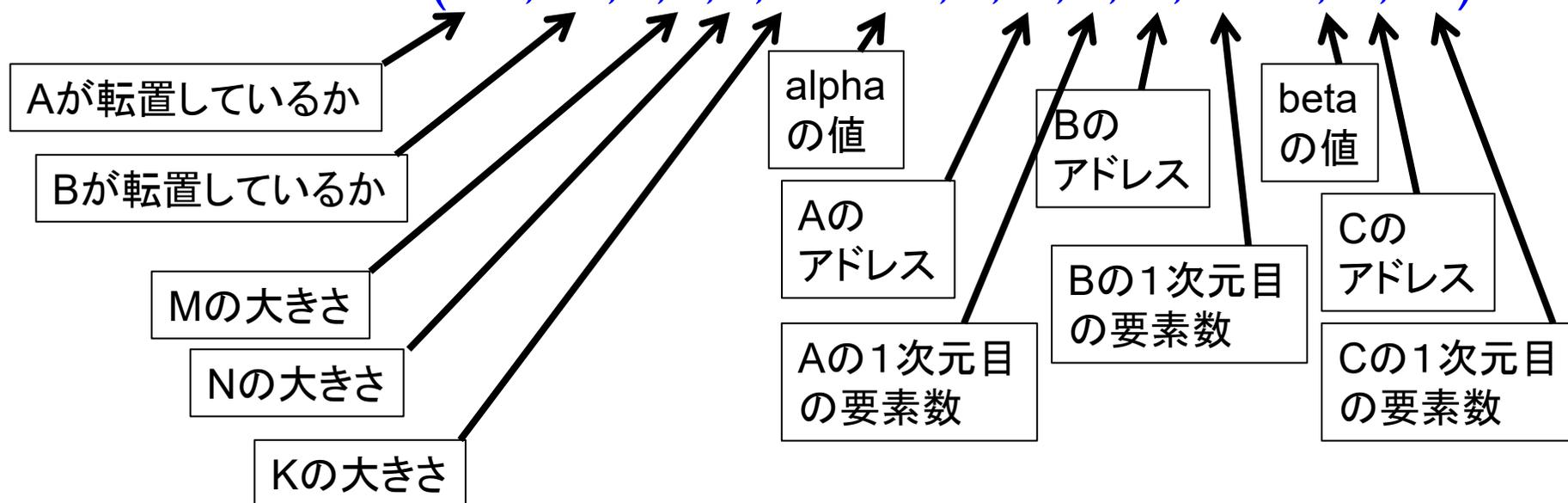
# BLAS利用例

## ▶ 倍精度演算BLAS3

$C := \alpha * op(A) * op(B) + \beta * C$

A: M\*K; B: K\*N; C: M\*N;

CALL DGEMM( 'N', 'N', n, n, n, ALPHA, A, N, B, N, BETA, C, N )



引数が多すぎ！ とても使いにくい！

## 1.8 BLASとPBLAS

---

### ▶ PBLAS

- ▶ 並列版のBLAS
- ▶ BLASとほぼ同じインタフェースをもつ
- ▶ BLAS利用者が、容易に移行できる
- ▶ ライブラリ再利用の目的で開発
  - ▶ 連立一次方程式用ライブラリ: *LINPACK* (リン・パック)
  - ▶ 固有値計算用ライブラリ: *EISPACK* (アイス・パック)
  - ▶ これらを統合したライブラリ: *LAPACK* (エル・エー・パック)
    - アルゴリズムレベルでブロック化して、レベル3 BLASを利用するアルゴリズムを新規開発
  - ▶ LAPACKを分散メモリ並列化: *ScaLAPACK* (スカラ・パック)
    - 並列版BLASとして、PBLASを利用

# BLASの機能詳細

---

- ▶ 詳細はHP: <http://www.netlib.org/blas/>
- ▶ 命名規則: 関数名: **XYYYYY**
  - ▶ **X**: データ型  
S:単精度、D:倍精度、C:複素、Z:倍精度複素
  - ▶ **YYYYY**: 計算の種類
    - ▶ レベル1:  
例: AXPY: ベクトルをスカラー倍して加算
    - ▶ レベル2:  
例: GEMV: 一般行列とベクトルの積
    - ▶ レベル3:  
例: GEMM: 一般行列どうしの積

# インタフェース例：DGEMM (1 / 4)

## ▶ DGEMM

(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)

- ▶  $C := \alpha * \text{op}(A) * \text{op}(B) + \beta * C$  の計算をする
- ▶  $\text{op}(X) = X$  もしくは  $\text{op}(X) = X'$  (Xの転置行列)

## ▶ 引数

### ▶ TRANSA(入力) CHARACTER\*1

- ▶ TRANSA は  $\text{op}(A)$  の操作を指定する。以下の文字列を指定。
  - $\text{TRANSA} = 'N'$  もしくは 'n',  $\text{op}(A) = A$
  - $\text{TRANSA} = 'T'$  もしくは 't',  $\text{op}(A) = A'$
  - $\text{TRANSA} = 'C'$  or 'c',  $\text{op}(A) = A'$

### ▶ TRANSB(入力) CHARACTER\*1

- ▶ TRANSB は  $\text{op}(B)$  の操作を指定する。以下同様。

# インタフェース例：DGEMM (2 / 4)

- ▶ **M(入力) - INTEGER**
  - ▶ op(A) と 行列 Cの行の大きさを指定する。
- ▶ **N(入力) - INTEGER**
  - ▶ op(B) と 行列 Cの列の大きさを指定する。
- ▶ **K(入力) - INTEGER**
  - ▶ op(A) の列の大きさ、および op(B) の行の大きさを指定する。
- ▶ **ALPHA(入力) - DOUBLE PRECISION**
  - ▶ スカラ値 ALPHAの値を設定する。
- ▶ **A(入力) - DOUBLE PRECISION**
  - ▶ 行列Aの配列。大きさは (LDA, ka) で、ka はTRANSA = 'N' or 'n' のときは k。そうでないときは、m。
  - ▶ TRANSA = 'N' or 'n' のときは、 $m \times k$ の配列の要素に行列Aを含まないといけない。そうでないときは、 $k \times m$ の配列に行列Aの転置を入れる。

# インタフェース例：DGEMM (3 / 4)

## ▶ LDA(入力) - INTEGER

- ▶ 行列Aの最初の次元数を入れる。TRANSA = 'N' もしくは 'n' なら、LDA は  $\max(1, m)$  でなくてはならない。そうでないなら、LDA は  $\max(1, k)$  でなくてはならない。

## ▶ B(入力) - DOUBLE PRECISION

- ▶ 行列Bの配列。大きさは (LDB, kb) で、kb はTRANSA = 'N' or 'n' のときは n。そうでないときは、k。
- ▶ TRANSA = 'N' or 'n' のときは、 $k \times n$  の配列の要素に行列Bを含まないといけない。そうでないときは、 $n \times k$  の配列に行列Bの転置を入れる。

## ▶ LDB(入力) - INTEGER

- ▶ 行列Bの最初の次元数を入れる。TRANSA = 'N' もしくは 'n' なら、LDA は  $\max(1, k)$  でなくてはならない。そうでないなら、LDB は  $\max(1, n)$  でなくてはならない。

# インタフェース例：DGEMM (4 / 4)

---

## ▶ BETA (入力) - DOUBLE PRECISION

- ▶ スカラ値 BETAの値を設定する。

## ▶ C (入力／出力) - DOUBLE PRECISION

- ▶ 行列Cの配列。
- ▶ 入力時、 $m \times n$  の配列に行列Cを入れる。配列には、BETAが0でない限り、行列Cを入れる。この場合は、Cの入力は必要ない。
- ▶ 出力時、この配列に、 $m \times n$ 行列の演算結果  $(\alpha * \text{op}(A) * \text{op}(B) + \text{beta} * C)$  が上書きされて戻る。

## ▶ LDC (入力) - INTEGER

- ▶ 行列Cの最初の次元数を入れる。LDC は  $\max(1, m)$  でなくてはならない。

# BLASの問題点

## ▶ BLASの問題点

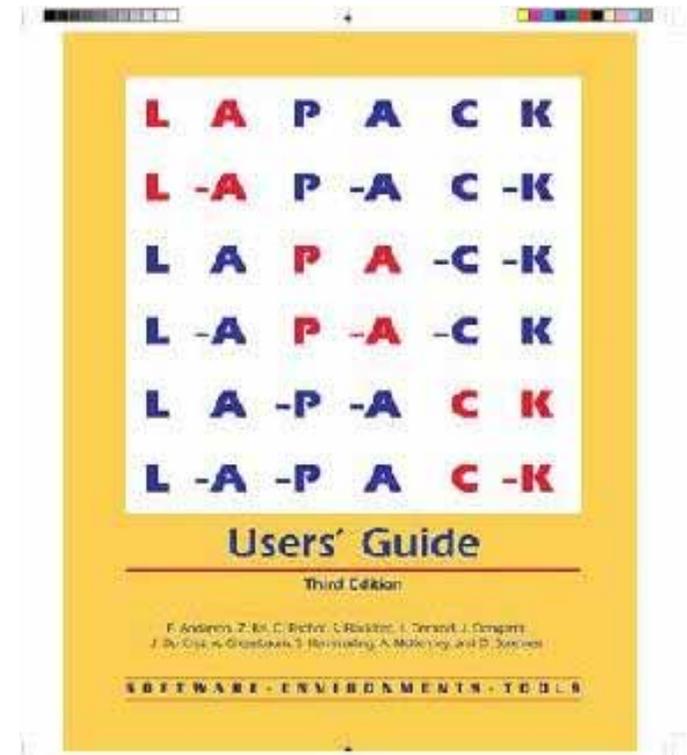
1. BLASやPBLASを用いると、データ再利用性や並列性が低下するかもしれない
  - ▶ 例:レベル1BLASにおける行列-ベクトル積
2. インタフェースに合わせるため、無駄な処理(配列への代入等)が必要になる場合も
  - ▶ <メモリ浪費>や<演算性能低下>の要因に
3. ソースコードが読みにくくなる
  - ▶ BLASのインタフェースを熟知しないと、かえって処理が理解できない
    - まあ、再利用性・性能とのトレードオフでしょうが

# GOTO BLASとは

- ▶ 後藤和茂 氏により開発された、ソースコードが無償入手可能な、高性能BLASの実装(ライブラリ)
- ▶ 特徴
  - ▶ マルチコア対応がなされている
  - ▶ 多くのコモディティハードウェア上の実装に特化
    - ▶ Intel Nehalem and Atom systems
    - ▶ VIA Nanoprocessor
    - ▶ AMD Shanghai and Istanbul
- 等
- ▶ テキサス大学先進計算センター(TACC)で、GOTO BLAS2として、ソースコードを配布している
  - ▶ HP : <http://www.tacc.utexas.edu/tacc-projects/gotoblas2/>

# LAPACK

- ▶ 密行列に対する、  
連立一次方程式の解法、  
および固有値の解法の  
“標準”アルゴリズムルーチンを  
無償で提供
- ▶ その道の大学の専門家が集結
  - ▶ カリフォルニア大バークレー校：  
James Demmel教授
  - ▶ テネシー大ノックスビル校：  
Jack Dongarra教授
- ▶ HP  
<http://www.netlib.org/lapack/>



# LAPACKの命名規則

---

## ▶ 命名規則： 関数名：**XYYZZZ**

### ▶ **X**: データ型

S:単精度、D:倍精度、C:複素、Z:倍精度複素

### ▶ **YY**: 行列の型

BD:二重対角、DI:対角、GB:一般帯行列、GE:一般行列、  
HE:複素エルミート、HP:複素エルミート圧縮形式、SY:対称  
行列、....

### ▶ **ZZZ**: 計算の種類

TRF: 行列の分解、TRS: 行列の分解を使う、CON: 条件数  
の計算、RFS: 計算解の誤差範囲を計算、TRI: 三重対角行  
列の分解、EQU: スケーリングの計算、...

# インタフェース例：DGESV (1 / 3)

## ▶ DGESV

(N, NRHS, A, LDA, IPIVOT, B, LDB, INFO)

- ▶  $A X = B$  の解の行列  $X$  を計算をする
- ▶  $A * X = B$ 、ここで  $A$  は  $N \times N$  行列で、 $X$  と  $B$  は  $N \times NRHS$  行列とする。
- ▶ 行交換の部分枢軸選択付きのLU分解で  $A$  を  $A = P * L * U$  と分解する。ここで、 $P$  は交換行列、 $L$  は下三角行列、 $U$  は上三角行列である。
- ▶ 分解された  $A$  は、連立一次方程式  $A * X = B$  を解くのに使われる。

## ▶ 引数

### ▶ N (入力) - INTEGER

- ▶ 線形方程式の数。行列  $A$  の次元数。  $N \geq 0$ 。

# インタフェース例：DGESV (2 / 3)

## ▶ NRHS (入力) – INTEGER

- ▶ 右辺ベクトルの数。行列Bの次元数。NRHS  $\geq 0$ 。

## ▶ A (入力／出力) – DOUBLE PRECISION, DIMENSION(:, :)

- ▶ 入力時は、 $N \times N$ の行列Aの係数を入れる。
- ▶ 出力時は、Aから分解された行列Lと $U = P * L * U$ を圧縮して出力する。Lの対角要素は1であるので、収納されていない。

## ▶ LDA (入力) – INTEGER

- ▶ 配列Aの最初の次元の大きさ。LDA  $\geq \max(I, N)$ 。

## ▶ IPIVOT (出力) – DOUBLE PRECISION, DIMENSION(:)

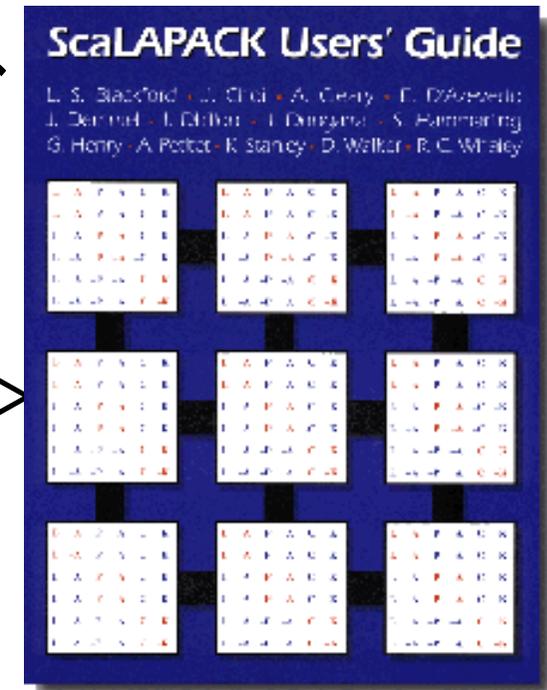
- ▶ 交換行列Aを構成する枢軸のインデックス。行列のi行がIPIVOT(i)行と交換されている。

# インタフェース例：DGESV (3 / 3)

- ▶ **B (入力／出力) – DOUBLE PRECISION, DIMENSION(:,:)**
  - ▶ 入力時は、右辺ベクトルの  $N \times NRHS$  行列Bを入れる。
  - ▶ 出力時は、もし、 $INFO = 0$  なら、 $N \times NRHS$ 行列である解行列Xが戻る。
- ▶ **LDB (入力) – INTEGER**
  - ▶ 配列Bの最初の次元の大きさ。  $LDB \geq \max(1, N)$ 。
- ▶ **INFO (出力) – INTEGER**
  - ▶  $= 0$ : 正常終了
  - ▶  $< 0$ : もし  $INFO = -i$  なら  $i$ -th 行の引数の値がおかしい。
  - ▶  $> 0$ : もし  $INFO = i$  なら  $U(i,i)$  が厳密に0である。分解は終わるが、Uの分解は特異なため、解は計算されない。

# ScaLAPACK

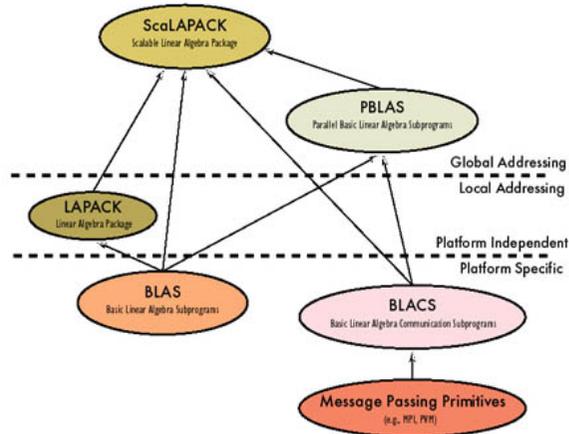
- ▶ 密行列に対する、連立一次方程式の解法、および固有値の解法の“標準”アルゴリズムルーチンの並列化版を無償で提供
- ▶ ユーザーインターフェースはLAPACKに<類似>
- ▶ ソフトウェアの<階層化>がされている
  - ▶ 内部ルーチンはLAPACKを利用
  - ▶ 並列インターフェースはBLACS
- ▶ データ分散方式に、2次元ブロック・サイクリック分散方式を採用
- ▶ HP: <http://www.netlib.org/scalapack/>



# ScaLAPACKソフトウェア構成図

## ScaLAPACK

A Software Library for Linear Algebra Computations on Distributed-Memory Computers



### AVAILABLE SOFTWARE:

#### Dense, Band, and Tridiagonal Linear Systems

- general
- symmetric positive definite

#### Full-Rank Linear Least Squares

#### Standard and Generalized Orthogonal Factorizations

#### Eigensolvers

- SEP: Symmetric Eigenproblem
- NEP: Nonsymmetric Eigenproblem
- GSEP: Generalized Symmetric Eigenproblem

#### SVD

#### Prototype Codes

- HPF interface to ScaLAPACK
- Matrix Sign Function for Eigenproblems
- Out-of-core solvers (LU, Cholesky, QR)
- Super LU
- PBLAS (algorithmic blocking and no alignment restrictions.)

### DOCUMENTATION:

#### ScaLAPACK Users' Guide

[http://www.netlib.org/scalapack/slug/scalapack\\_slug.html](http://www.netlib.org/scalapack/slug/scalapack_slug.html)

#### Future Work

- Out-of-core Eigensolvers
- Divide and Conquer routines
- C++ and Java Interfaces

#### Commercial Use

ScaLAPACK has been incorporated into the following software packages:

- NAG Numerical Library
- IBM Parallel ESSL
- SGI Cray Scientific Software Library

and is being integrated into the VNI IMSL Numerical Library, as well as software libraries for Fujitsu, HP/Convex, Hitachi, and NEC.

<http://www.netlib.org/scalapack/>

(参照) <http://www.netlib.org/scalapack/poster.html>

# BLACSとPBLAS

---

## ▶ BLACS

- ▶ ScaLAPACK中で使われる通信機能を関数化したもの。
- ▶ 通信ライブラリは、MPI、PVM、各社が提供する通信ライブラリを想定し、ScaLAPACK内でコード修正せずに使うことを目的とする
  - ▶ いわゆる、通信ライブラリのラッパー的役割でScaLAPACK内で利用
- ▶ 現在、MPIがデファクトになったため、MPIで構築されたBLACSのみ、現実的に利用されている。
  - ▶ なので、ScaLAPACKはMPIでコンパイルし、起動して利用する

## ▶ PBLAS

- ▶ BLACSを用いてBLASと同等な機能を提供する関数群
- ▶ 並列版BLASといってよい。

# ScaLAPACKの命名規則

---

- ▶ 原則：  
LAPACKの関数名の頭に“P”を付けたもの
- ▶ そのほか、BLACS、PBLAS、データ分散を制御するためのScaLAPACK用関数がある。

# インタフェース例：PDGESV (1 / 4)

## ▶ PDGESV

( N, NRHS, A, IA, JA, DESCA, IPIV, B, IB, JB, DESCB, INFO )

- ▶  $sub(A) X = sub(B)$  の解の行列  $X$  を計算をする
- ▶ ここで  $sub(A)$  は  $N \times N$  行列を分散した  $A(IA:IA+N-1, JA:JA+N-1)$  の行列
- ▶  $X$  と  $B$  は  $N \times NRHS$  行列を分散した  $B(IB:IB+N-1, JB:JB+NRHS-1)$  の行列
- ▶ 行交換の部分枢軸選択付きのLU分解で  $sub(A)$  を  $sub(A) = P * L * U$  と分解する。ここで、 $P$  は交換行列、 $L$  は下三角行列、 $U$  は上三角行列である。
- ▶ 分解された  $sub(A)$  は、連立一次方程式  $sub(A) * X = sub(B)$  を解くのに使われる。

## インタフェース例：PDGESV (2 / 4)

- ▶ **N (大域入力) – INTEGER**
  - ▶ 線形方程式の数。行列Aの次元数。  $N \geq 0$ 。
- ▶ **NRHS (大域入力) – INTEGER**
  - ▶ 右辺ベクトルの数。行列Bの次元数。  $NRHS \geq 0$ 。
- ▶ **A (局所入力／出力) – DOUBLE PRECISION, DIMENSION(:, :)**
  - ▶ 入力時は、 $N \times N$ の行列Aの局所化された係数を配列A(LLD\_A, LOCc( JA+N-1))を入れる。
  - ▶ 出力時は、Aから分解された行列Lと  $U = P * L * U$  を圧縮して出力する。Lの対角要素は1であるので、収納されていない。
- ▶ **IA(大域入力) – INTEGER** : *sub(A)*の最初の行のインデックス
- ▶ **JA(大域入力) – INTEGER** : *sub(A)*の最初の列のインデックス
- ▶ **DESCA (大域かつ局所入力) – INTEGER**
  - ▶ 分散された配列Aの記述子。

# インタフェース例：PDGESV (3 / 4)

## ▶ IPIVOT (局所出力) – DOUBLE PRECISION, DIMENSION(:)

- ▶ 交換行列Aを構成する枢軸のインデックス。行列のi行がIPIVOT(i)行と交換されている。分散された配列(LOCr(M\_A)+MB\_A)として戻る。

## ▶ B (局所入力／出力) – DOUBLE PRECISION, DIMENSION(:,:)

- ▶ 入力時は、右辺ベクトルの  $N \times NRHS$  の行列Bの分散されたものを(LLD\_B, LOCc(JB+NRHS-1))に入れる。
- ▶ 出力時は、もし、INFO = 0 なら、 $N \times NRHS$  行列である解行列Xが、行列Bと同様の分散された状態で戻る。

## ▶ IB(大域入力) – INTEGER

- ▶  $sub(B)$  の最初の行のインデックス

## ▶ JB(大域入力) – INTEGER

- ▶  $sub(B)$  の最初の列のインデックス

## ▶ DESCB (大域かつ局所入力) – INTEGER

- ▶ 分散された配列Bの記述子。

# インタフェース例：PDGESHV (3 / 4)

## ▶ INFO (大域出力) —INTEGER

- ▶ = 0: 正常終了
- ▶ < 0:
  - もし  $i$  番目の要素が配列で、その  $j$  要素の値がおかしいなら、 $INFO = -(i*100+j)$  となる。
  - もし  $i$  番目の要素がスカラーで、かつ、その値がおかしいなら、 $INFO = -i$  となる。
- ▶ > 0: もし  $INFO = K$  のとき  $U(IA+K-1, JA+K-1)$  が厳密に0である。分解は完了するが、分解された  $U$  は厳密に特異なので、解は計算できない。

---

# サンプルプログラムの実行 (BLAS DGEMM)

# UNIX防忘録

---

- ▶ emacsの起動: emacs 編集ファイル名
  - ▶  $\text{^x ^s}$  (^はcontrol) : テキストの保存
  - ▶  $\text{^x ^c}$  : 終了  
( $\text{^z}$  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。)
  - ▶  $\text{^g}$  : 訳がわからなくなったとき。
  - ▶  $\text{^k}$  : カーソルより行末まで消す。消した行は一時記憶される。
  - ▶  $\text{^y}$  :  $\text{^k}$ で消した行を、現在のカーソルの場所にコピーする。
  - ▶  $\text{^s 文字列}$  : 文字列の箇所まで移動する。
  - ▶  $\text{^M x goto-line}$  : ( $\text{^M}$ はESC) 指定した行まで移動する。

# UNIX防忘録

---

- ▶ **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 ~** : ルート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。

# BLAS DGEMM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 = 2000**

**Mat-Mat time = 23.236051 [sec.]**

**688.585154 [MFLOPS]**

**OK!**

# BLAS DGEMMのサンプルプログラムの実行 (Fortran言語版)

---

▶ 以下のような結果が見えれば成功

**N = 2000**

**Mat-Mat time[sec.] = 4.223372380249202**

**MFLOPS = 3788.441680972445**

**OK!**

# サンプルプログラムの説明（C言語）

---

- ▶ `#define N 2000`  
の、数字を変更すると、行列サイズが変更  
できます
- ▶ `#define DEBUG 1`  
「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 $N \times N$ 行列AとBの行列積をおこない、  
Double型 $N \times N$ 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NNの宣言は、以下のファイルにあります。

`mat-mat-blas.inc`

- ▶ 行列サイズ変数が、NNとなっています。

`integer NN`

`parameter (NN=2000)`

# サンプルプログラムの説明 (BLAS)

- ▶ **1コア(逐次)実行版です**
- ▶ スレッド並列化版は、対応するスレッド並列化版をリンクしてコンパイルの上、並列実行数の指定をする必要があります。
  - ▶ BLASスレッドでは、**OpenMPのスレッド数指定法と同じ方法**で、実行するスレッド数が指定できます
    - ▶ **OMP\_NUM\_THREADS**環境変数に実行スレッド数を入れる
    - ▶ ジョブスクリプト内に、以下を記載(48スレッド時):  
**export OMP\_NUM\_THREADS=48**

---

# BLAS DGEMM回答

# BLAS DGEMMの回答 (C言語版)

```
double ALPHA, BETA;  
char TEX[1] = {'N'};  
  
ALPHA=1.0;  
BETA=1.0;  
dgemm_(&TEX, &TEX, &n, &n, &n, &ALPHA,  
      A, &n, B, &n, &BETA, C, &n);
```

# BLAS DGEMMの回答 (Fortran言語版)

double precision ALPHA,BETA

ALPHA=1.0d0

BETA=1.0d0

CALL DGEMM('N', 'N', n, n, n, ALPHA,  
& A, n, B, n, BETA, C, n)

## 参考文献（1）

---

### 1. BLAS

<http://www.netlib.org/blas/>

### 2. LAPACK

<http://www.netlib.org/lapack/>

### 3. ScaLAPACK

<http://www.netlib.org/scalapack/>

### 4. スパースBLAS

<http://math.nist.gov/spblas/>

## 参考文献（２）

---

1. MPI並列プログラミング、P.パチエコ 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、  
理化学研究所情報基盤センタ  
( <http://acc.riken.jp/HPC/training/text.html> )
3. Message Passing Interface Forum  
( <http://www.mpi-forum.org/> )
4. MPI-Jマーキングリスト  
(<http://phase.hpcc.jp/phase/mpi-j/ml/>)
5. 並列コンピュータ工学、富田眞治著、昭晃堂(1996)
6. 並列数値処理 一高速化と性能向上のために一、  
金田康正 編著、コロナ社(2010)

---

# LAPACK/ScaLAPACK演習 (演習)

# 演習内容

---

- ▶ 粒子間熱伝導問題の説明(座学)
- ▶ LAPACKの利用法と演習  
(C言語、Fortran言語)
- ▶ ScaLAPACKの利用法と演習  
(Fortran言語のみ)

---

# サンプルプログラムの実行 (LAPACK dgesv)

# LAPACK dgesv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

time = 0.000001 [sec.]

1136788958.46 [MFLOPS]

990 -2.272792e+00

991 -2.276715e+00

992 -2.288410e+00

993 -2.307670e+00

994 -2.334166e+00

995 -2.367479e+00

996 -2.407125e+00

997 -2.452584e+00

998 -2.503330e+00

999 -2.558846e+00

err = 6.274235e+02

連立一次方程式求解部分が  
実装されて無いため、  
時間が極端に短く、  
誤差ERRが大きくな  
っていますが、  
正常な動作です。

# LAPACK dgesvのサンプルプログラムの実行 (Fortran言語版)

## ▶ 以下のような結果が見えれば成功

TIME[sec] = 5.792826414108276E-07

MFLOPS = 1156892943.257929

991 -2.272792E+00

992 -2.276715E+00

993 -2.288410E+00

994 -2.307670E+00

995 -2.334166E+00

996 -2.367479E+00

997 -2.407125E+00

998 -2.452584E+00

999 -2.503330E+00

1000 -2.558846E+00

EPS = 627.4235113914225

連立一次方程式求解部分が  
実装されて無いので、  
時間が極端に短く、  
誤差ERRが大きくな  
っていますが、  
正常な動作です。

# サンプルプログラムの説明 (LAPACK)

---

- ▶ **1コア(逐次)実行版です**
- ▶ スレッド並列化版は、スレッド並列化版をリンクしてコンパイルの上、並列実行数の指定をする必要があります。
- ▶ LAPACKはスレッド並列化のみ対応しているため、複数ノードを通信しつつ利用することはできません
  - ▶ 通信しつつ利用するには、**分散版であるScaLAPACK**を利用します
  - ▶ ノード内のみLAPACKを使い、ノード間はMPIで並列化するような使い方は可能です
    - ▶ たとえば、領域分割法を利用し、ノード内のみLAPACKで**不老** ITC 連立一次方程式を解く場合

# サンプルプログラムの説明 (LAPACK)

---

- ▶ 誤差ERRの計算は、 $10 \times 10 \times 10$ の問題に特化されています。  
それ以外の問題サイズでは機能しません。
- ▶ test\_org.c、test\_org.fは、オリジナルのCG法を用いた反復解法のコードです。

---

# LAPACK dgesv回答

# LAPCK dgesvの回答 (C言語版)

---

```
dgesv_(&nn, &inc, amat2, &nn,  
piv, rhs, &nn, &info);
```

# LAPCK dgesvの回答 (Fortran言語版)

---

```
call DGESV(N, INC, AMAT, N,  
&          PIV, RHS, N, INFO)
```

---

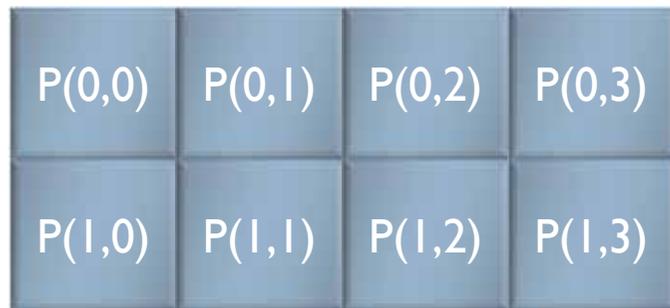
# サンプルプログラムの実行 (ScaLAPACK pdgesv)

# ScaLAPACKにおけるプロセッサ・グリッド

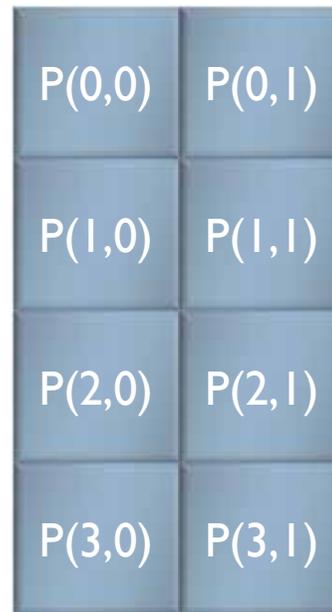
- ▶ MPIで起動する総プロセス数は、プロセッサ・グリッドと呼ばれる2次元プロセッサ構造で管理されます。

- ▶ 例： 8プロセス

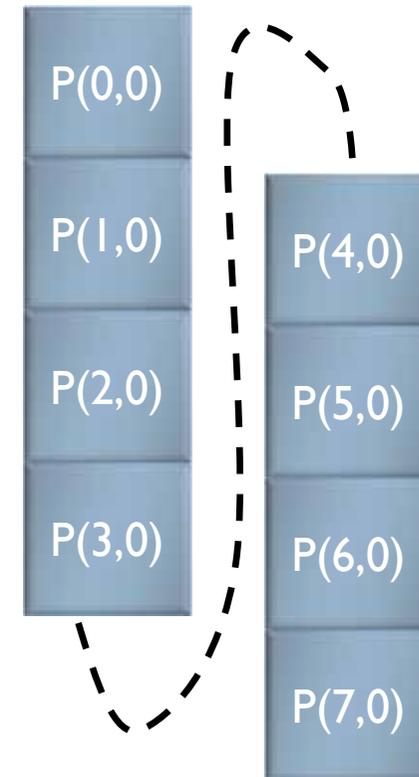
● 2 × 4構成



● 4 × 2構成



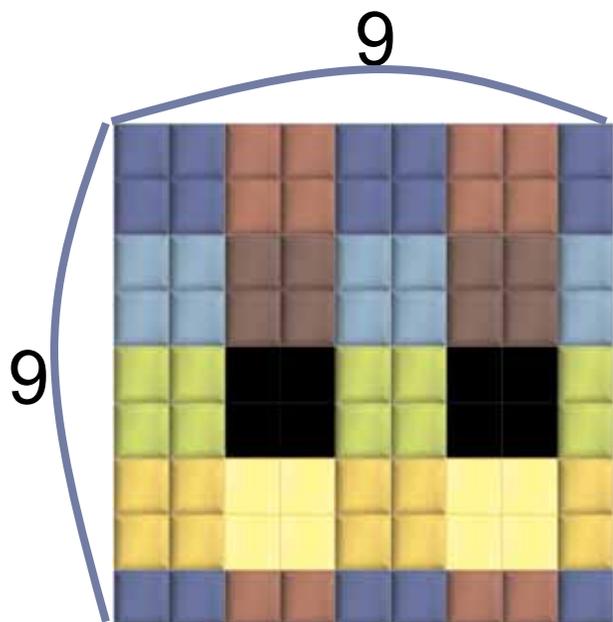
● 8 × 1構成



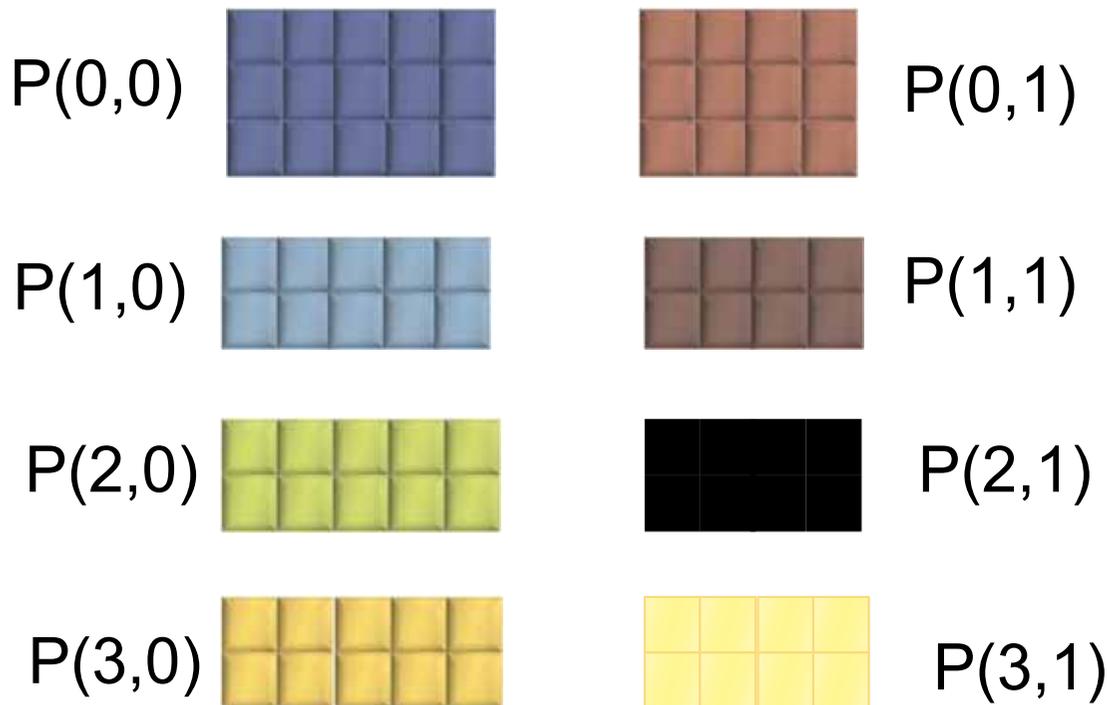
# ScaLAPACKにおけるデータ分散

- ▶ ブロック幅MBの2次元ブロックサイクリック分散となります
- ▶ 例: 8プロセス(2×4プロセッサー・グリッド)、MB=2

行列サイズで割り切れないプロセス数の場合も考慮して、 $\text{sub}(A)$ の確保領域を計算する必要があります



大域ビュー(行列A)



局所ビュー(sub(A))

# コンテキストの定義

## ●コンテキスト:プロセッサ・グリッドに関する情報体

### **BLACS\_GET( ICONTXT, WHAT, VAL )**

#### ●BLACSの内部情報を取得する

##### ●**ICONTXT**: 入力、整数型、

WHATで指定するコンテキストに設定を試みる値(引数)。  
(引数が)無い時は無視される。

##### ●**WHAT**: 入力、整数型、コンテキストに設定する内容

WHAT = 0 : デフォルトのシステムコンテキスト

WHAT = 1 : BLACS メッセージの ID 幅

WHAT = 2 : コンパイルされるBLACS デバックレベル

WHAT = 10: ICONTXTにより制御されるBLACSコンテキストを定義するために  
用いられるシステムコンテキストの参照

WHAT = 11: 現在使っているマルチリング構造のリング数

WHAT = 12: 現在使っている一般化した木構造の枝数

##### ●**VAL**: 出力、整数型、コンテキスト情報

# BLACSグリッドの定義

---

## **BLACS\_GRIDINIT( ICONTXT, ORDER, NPROW, NPCOL )**

● **ICONTXT** : 入力／出力、整数型  
コンテキスト

● **ORDER** : 入力、文字型 \* 1

プロセスをどのようにBLACSのグリッドに割り当てるか指定

● 'R' : 行方向の自然なオーダリングを使う。

● 'C' : 列方向の自然なオーダリングを使う。

● そうでないなら: 行方向の自然なオーダリングを使う。

# BLACSグリッドの情報入手

## **BLACS\_GRIDINFO**

**(ICONTXT, NPROW, NPCOL, MYPROW, MYPCOL)**

- **ICONTXT**: 入力、整数型  
コンテキスト
- **NPROW**: 出力、整数型  
プロセッサ・グリッドの行数
- **NPCOL**: 出力、整数型  
プロセッサ・グリッドの列数
- **MYPROW**: 出力、整数型  
プロセッサ・グリッドにおける自分の行方向の認識番号
- **MYPCOL**: 出力、整数型  
プロセッサ・グリッドにおける自分の列方向の認識番号

## データ分散を容易にする関数

### **PDELSET( A, IA, JA, DESCA, ALPHA )**

- 大域配列の添え字から、局所配列の適する場所にデータを収納する
  - **A**: 局所出力、倍精度型  
収納すべき局所配列
  - **IA**: 大域入力、整数型  
大域的配列における1次元目の添え字
  - **JA**: 大域入力、整数型  
大域的配列における2次元目の添え字
  - **DESCA**: 大域かつ局所入力、整数型配列  
局所配列Aの記述子
  - **ALPHA**: 局所入力、倍精度型  
代入すべき値

## データ分散を容易にする関数:例

大域配列AMATを、適切な局所行列Aに代入

```
DO J = 1, N
  DO I = 1, N
    CALL PDELSET( A, I, J, DESC A, AMAT(I,J))
  ENDDO
  CALL PDELSET( B, J, 1, DESC B, RHS(J))
ENDDO
```

大域ベクトルRHSを、適切な局所ベクトルBに代入

# PDELSET関数による方法の注意点

- ▶ PDELSET関数を用いて行列を分散する方法は容易ですが、**サンプルプログラムのように、大域行列Aを全プロセスで所有していると、メモリに関するスケーラビリティが無くなります。**
  - ▶ つまり、実行できる行列サイズNが、1ノードのメモリ総量で決まる。実行ノード数を増加しても、Nを大きくできない。
- ▶ 本格的な並列プログラムを書くためには
  - ▶ **局所ビューの観点で、行列の値を直接sub(A)に値を代入する**  
プログラムを書く必要があります。
- ▶ そのためには、ブロック・サイクリック分散方式の特徴を理解して、局所配列の確保をする必要があります
  - ▶ **本講習会のMPI基礎編で、関連技術の演習があります。**

# ScaLAPACK pdgesvの サンプルプログラムの実行 (Fortran言語版)

▶ 以下のような結果が見えれば成功

TIME[sec] = 5.895271897315979E-07

MFLOPS = 1560927138.191826

ScaLAPACK Example Program -- Sep, 2010

Solving  $Ax=b$  where  $A$  is a 1000 by 1000 matrix with a block size of  
32

Running on 64 processes, where the process grid is 8 by 8

INFO code returned by PDGESV = 0

EPS = 627.4235113914225

連立一次方程式求解部分が実装されて無いため、時間が  
極端に短く、誤差ERRが大きくなっていますが、正常な動作です。

# サンプルプログラムの説明(ScaLAPACK)

- ▶ **ピュアMPI版です**
  - ▶ スレッド並列を利用したハイブリッドMPI版を利用するには、スレッド並列版をリンクしてコンパイルし、スレッド並列数を指定する必要があります
- ▶ 64プロセス実行(プロセッサ・グリッド $8 \times 8$ )、問題サイズ1000、ブロック幅32に**特化されています**
  - ▶ プロセス数、プロセッサ・グリッド、問題サイズ、ブロック幅を変更する場合、PARAMETER文の定数を変更する必要があります
- ▶ 誤差ERRの計算は、 $10 \times 10 \times 10$ の問題に特化されています。それ以外の問題サイズでは機能しません。

---

# ScaLAPACK pdgesv回答

# ScaLAPACK dgesvの回答 (Fortran言語版)

---

```
CALL PDGESV(  
&   NN, NRHS,  
&   A, IA, JA, DESCA, IPIV,  
&   B, IB, JB, DESCB,  
&   INFO )
```

---

おわり

お疲れさまでした