内容に関するご質問は
ida@cc.u-tokyo.ac.jp
まで、お願いします。

第62回　お試しアカウント付き
並列プログラミング講習会
「ライブラリ利用：科学技術計算の効率化入門」

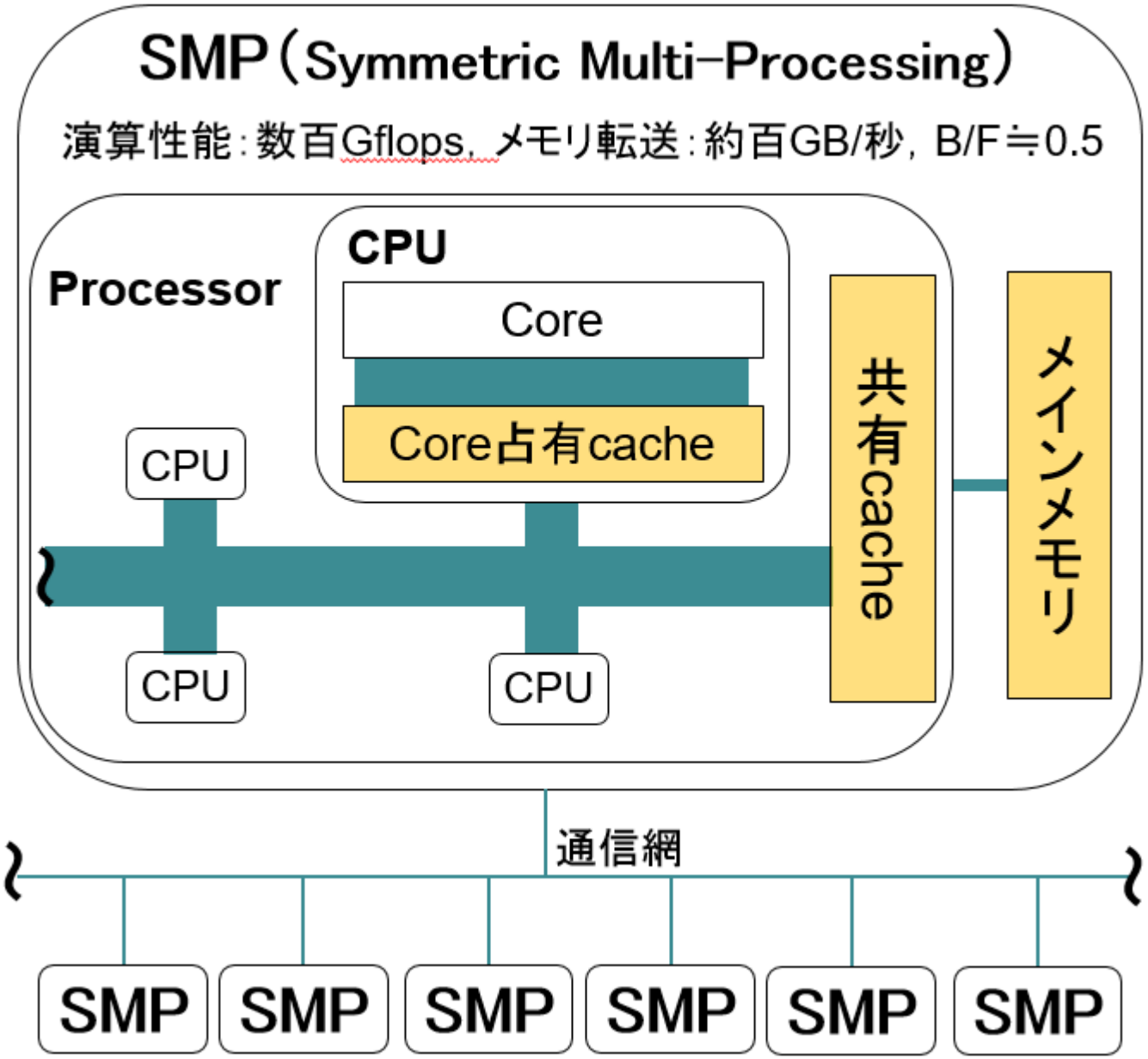
プログラム実習＆座学
（BLAS, LAPACK、ScaLAPACK）

東京大学情報基盤センター　特任准教授　伊田 明弘

チュートリアルの流れ

1. 現代のスパコンと並列計算
2. BLAS, LAPACK, ScaLAPACKの説明
3. プログラム実習 Ⅰ
   - BALS/DGEMMサンプルプログラム実行
4. プログラム実習 Ⅱ
   - LAPACK/DGESVサンプルプログラム実行
   - ScaLAPACK/PDGESVサンプルプログラム実行

SMPクラスタ型並列計算システムの概念図



■転送速度

Core占有cache

∨ 約10倍

共有cache

∨ 約10倍

メインメモリ

∨ 約10倍

通信網

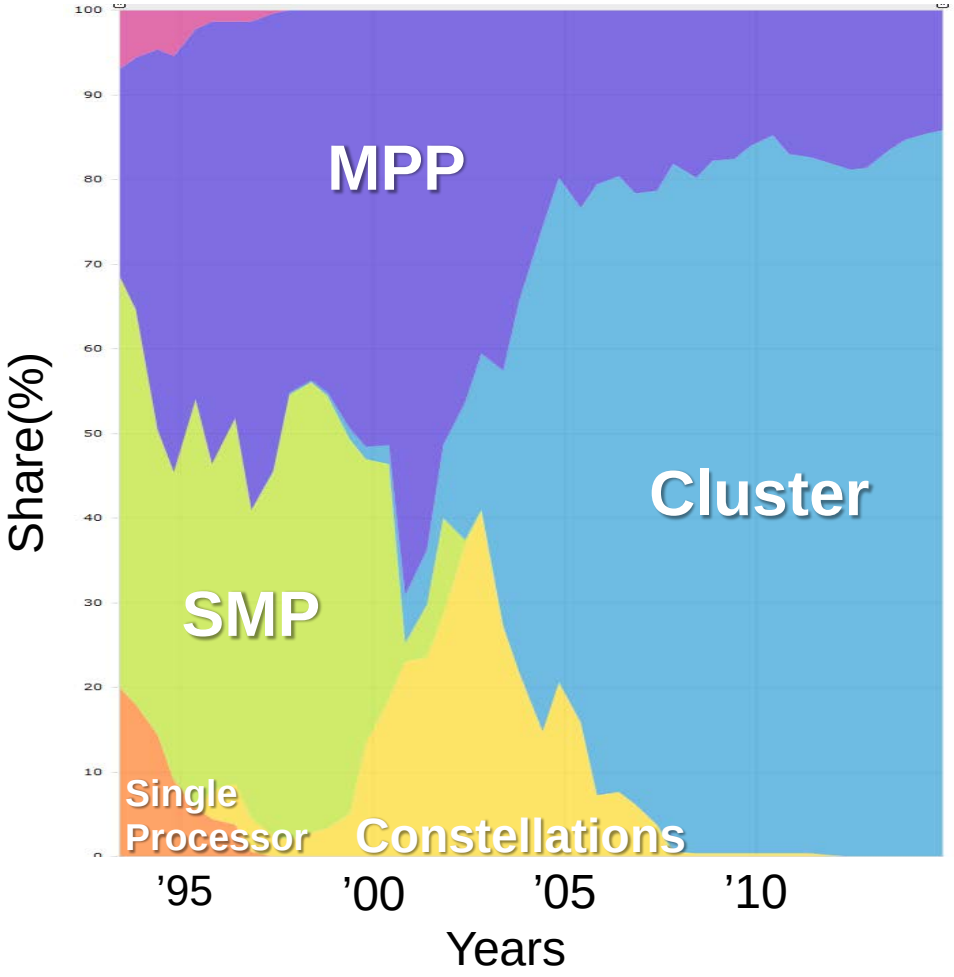
■容量

・Core占有cache：
数十～数百KB

・共有cache：
数十MB

・メインメモリ：
数十GB

スーパーコンピュータの変遷



Top 500 list of Supercomputers (http://www.top500.org)

スーパーコンピュータのアーキテクチャの変遷

■クラスタ型が現代では主流

■クラスタ型計算システムの性能を発揮させるのは大変

- キャッシュの有効利用
- SIMDの活用
- 計算負荷バランス
- データ競合回避
- 計算順序保障
- ハイパースレッドの活用
- 通信量の削減
- 効率的な通信パターン
- 同期回数の削減

⇨ ライブラリを活用することで
プログラミングコスト削減

SMPクラスタ型並列計算システムの例

SMPクラスタ名		Reedbush-U @東大基盤センタ	Oakleaf-FX @東大基盤センタ
全体	SMP数	420	4800
	SMP間 データ転送性能	25Gbit/s × 4	40Gbit/s × 2
SMP	Processor数	2	1
	理論ピーク演算性能	1209.6 Gflops	236.5 GFlops
	メモリ容量	256GB	32GB
	メモリ転送速度	153.6 GB/sec	85 GB/sec
Processor	Processor名	Intel Xeon E5™	SPARC64™ IXfx
	Core数	18	16
	理論演算性能(Core)	33.6 GFlops	14.78 GFlops
	Core占有cache	L1:32KB L2:256KB	32 KB
	共有cache	45MB	12 MB

■GB:Giga Byteの略, 1GB=8Gbit

■flops : 1秒間に行われる浮動小数点演算の回数
計算機の性能を表す指標によく使用される

並列計算の必要性と種類

■逐次コードで使える計算資源は、スパコン全体の極々一部
- スパコンは、普通のCPUの集合に過ぎずちっともスーパーではない
- 単体CPUの性能は、普通のパソコンと大差ない

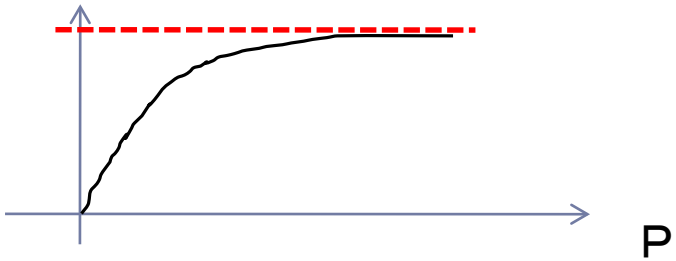
■並列計算の種類

① 逐次コードの単純並列実行
- 小規模計算を大量のパラメータに対して行う
- 何も工夫することなく、最高の並列計算効率
- アルゴリズムの工夫やCasheの使い方などが高速化の鍵

② 並列コード
- 複数のCPUを連携させて、大規模計算を行う
- 並列化効率を向上させるため、うまく連携させる工夫が必要
- Casheの使い方など逐次コードで鍵となった技術もやはり重要

性能評価指標－台数効果

- 台数効果
  - 式：　$S_P = T_S / T_P \quad (0 \leq S_p)$
  - 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：　$E_P = S_P / P \times 100 \quad (0 \leq E_p)\ [\%]$
- 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 α とする。
- このとき、台数効果は以下のようになる。

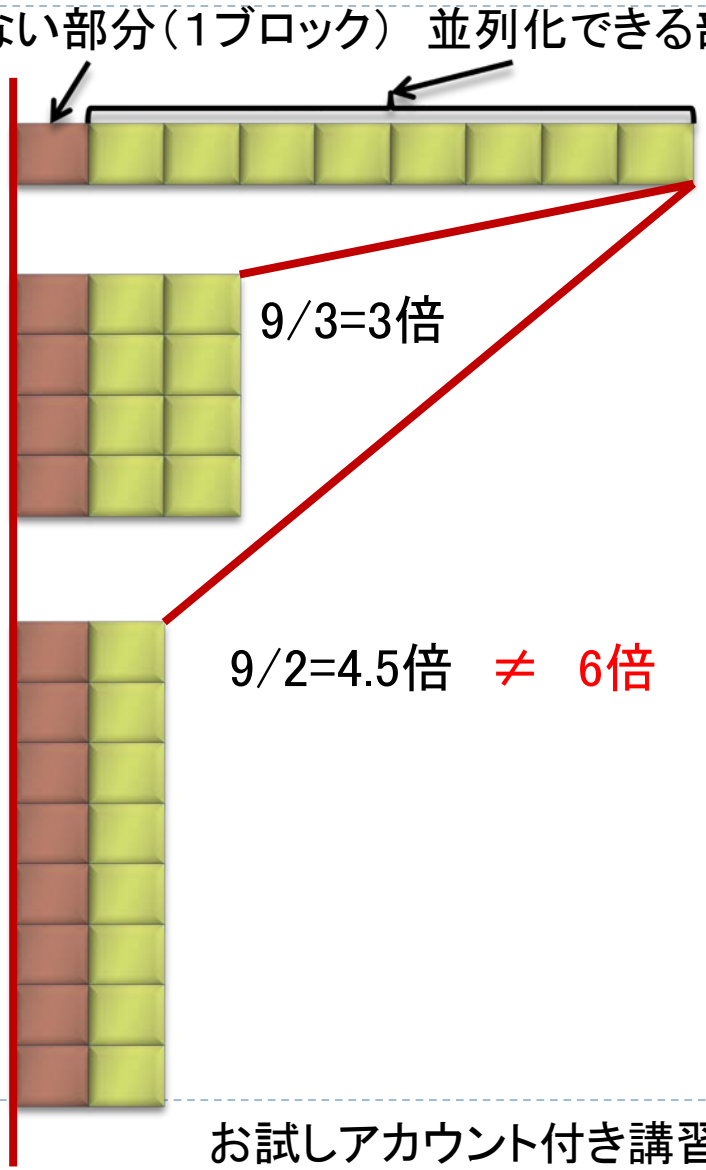
$$S_P = K/(K\alpha/P + K(1-\alpha))$$

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

- 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

並列コードの種類

MPIプロセスとOpenMPスレッドを用いた並列化が二大メジャー手段

■プロセス並列

- **MPI（Message Passing Interface）**
- HPF（High Performance Fortran）

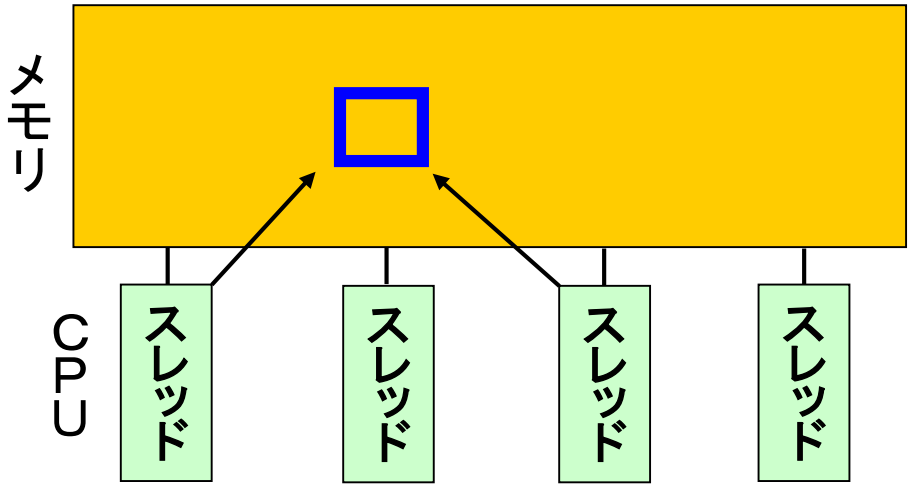
■スレッド並列

- **Open MP**
  - ユーザが並列化指示行を記述
- Open ACC
- Pthread (POSIX スレッド)
- CILK
- Java

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」

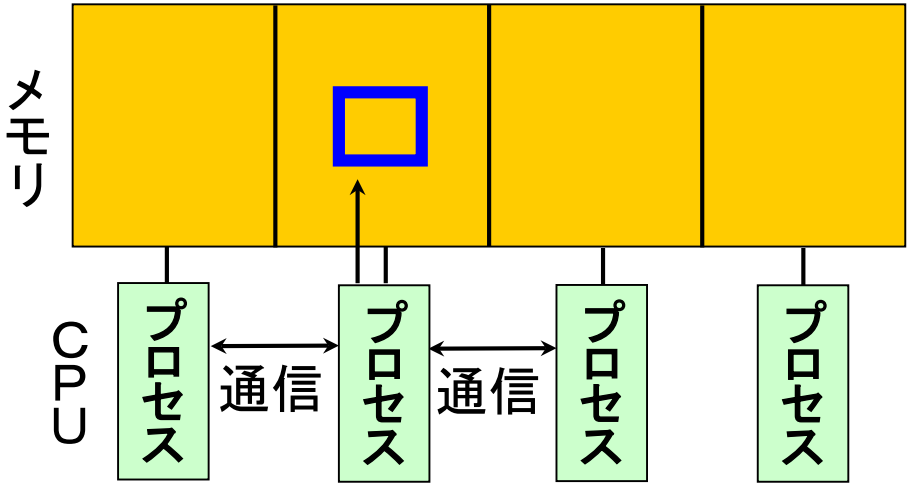
スレッド並列とプロセス並列

■スレッド並列



- SMPのように、メモリが多数のCPUで共有されていることが前提
- 任意の情報へどのスレッドからも直接アクセス可能
- 通信網の先にある他のSMPが持つ情報へはアクセス不能

■プロセス並列



- メモリが多数のCPUで共有されていたとしてもメモリ空間を分離
- 他のプロセスが持つ情報へは「通信」することでアクセス
- 「通信」することにより、他のSMPが持つ情報へもアクセス可能

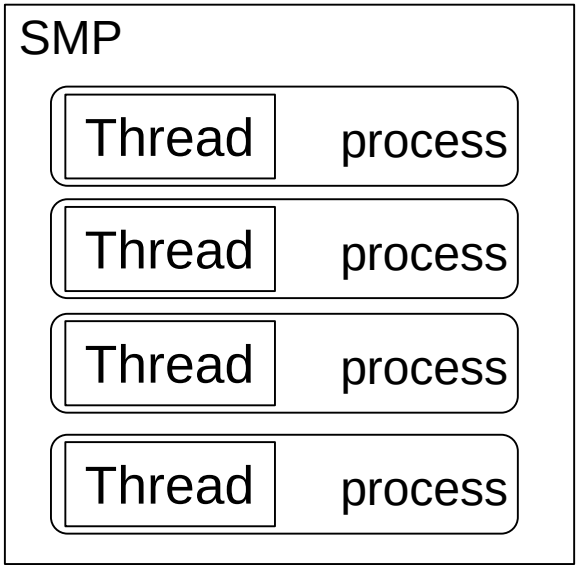
MPI＋OpenMPハイブリッドプログラミングモデル

■SMPクラスタを考えるとプロセス並列とスレッド並列の組み合わせが合理的

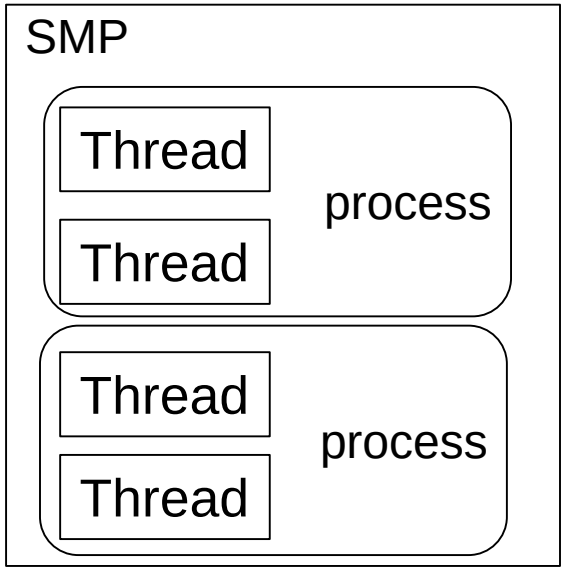
- 各MPI-processが、1つ以上,SMP-core数以下の固定数のOpenMP-Threadを持つ
- 演算の多くはOpenMP-Threadが共有メモリ下で実行する
- 各MPI-processは独立したメモリ空間を有し，process間通信でデータをやりとりする
- 各MPI-processが何Thread持つのが良いかは計算システムと実装で決まる

■SMP-core数が4つの場合の組み合わせ

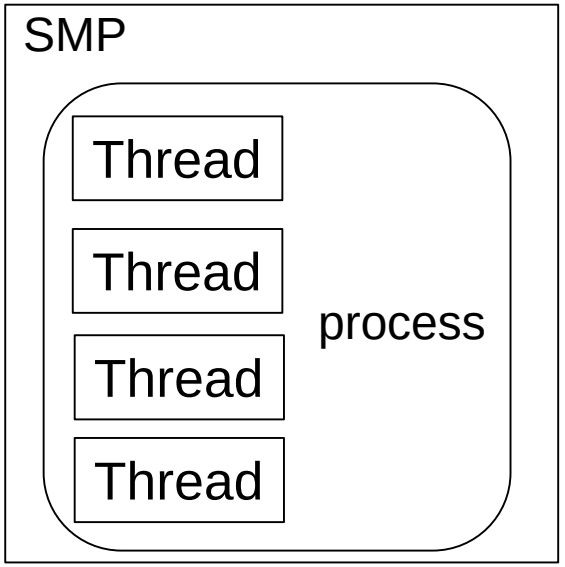
1process1thread(準pure-MPI)



1process2thread



1process4thread



MPIの概要と特徴

■ 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（クラスタ）に対応可能

- 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、性能が良い

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

MPIによるコードの並列化

- 基本は逐次計算を行った後で通信ルーチンを呼ぶ
- 1つのコードで、呼ぶ側と呼ばれる側を表現する SPMD（Single Program Multiple Data）が一般的

```
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
```

OpenMPの概要と特徴

- 共有メモリ型並列計算機用のDirectiveの統一規格
- ディレクティヴ（指示行）の形で利用
  - プログラムを並列に実行するための動作をユーザーが明示
  - 挿入直後のループが並列化される等
  - コンパイラがサポートしていなければ，コメントとみなされる
- 逐次コードからの並列化が比較的
  - 指示行（ディレクティヴ）を挿入するだけで手軽に並列化が可能
  - ループ単位など少しずつ段階的に並列化が可能
- OpenMPがMPIより簡単ということはない
  - OpenMP実行環境は，依存関係，衝突，デッドロック，競合条件，結果としてプログラムが誤った実行につながるような問題に関するチェックは要求されていない
  - プログラムが正しく実行されるよう構成するのはユーザーの責任

OpenMPによるコードの並列化

- 基本は「!omp parallel do」～「!omp end parallel do」
- 変数について，グローバルな変数（shared）と，Thread内でローカルな「private」な変数に分けられる
  - デフォルトは「shared」
  - 内積を求める場合は「reduction」を使う

```
!$omp parallel do private(ip,iS,iE,i)
!$omp&            reduction(+:RHO)
      do ip= 1, PEsmpTOT
        iS= STACKmcG(ip-1) + 1
        iE= STACKmcG(ip  )
        do i= iS, iE
          RHO= RHO + W(i,R)*W(i,Z)
        enddo
      enddo
!$omp end parallel do
```

W(:,:), R, Z, PEsmpTOT などはグローバル変数

スパコン上でのライブラリ利用

■ライブラリ・ソフトウェア使えばスパコンの性能を楽に引き出せる

■並列化の各ステージごとに様々なソフトウェアが提供されている

① SMPクラスタ対応パッケージソフト

・格子データや定義ファイルを用意するだけで大規模計算が可能

・ユーザは並列化や通信をあまり意識する必要がない

・OpenFOAM，FrontFlow, ppOpen-APPL/BEMなど

② SMPクラスタ対応ライブラリ、クラスタ対応ライブラリ

・通信機能を持つルーチンを呼ぶことで大規模計算を行う

・ライブラリルーチンが行う計算以外の並列計算はユーザが行う

・MPI，Super-LU，ScaLAPACKなど

④ SMP対応ライブラリ、逐次計算用ライブラリ

・逐次計算コードで高速化したい部分の機能を有するルーチンを呼ぶ

・複数SMPを使いたければ、通信をユーザが行う必要あり

・Intel MKL，富士通SSL2，BLAS，LAPACKなど

BLAS, LAPACK, ScaLAPACKの説明

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/

Netlib ：科学技術計算用ソフトウェア・リポジトリ
（AT＆T、ベル研究所、テネシー大学、
オークリッジ国立研究所が共同管理）

BLASとは

- 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。
  - 普通は、密行列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＞というものあるが、まだ定着していない。

BLASの意義

- 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、ソフトウエア開発効率が悪い
    - ＜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、数学ソフトウエアにおける＜ソフトウエア工学＞のはしり

BLASサブルーチンの分類

- 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
- **なるべく高いレベルのルーチンを使用するのが効果的**

レベル１　BLAS

- ベクトル内積、ベクトル定数倍の加算、など
  - 例：　$y \leftarrow \alpha x + y$
- 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
  - 使用による性能向上が、あまり期待できない
  - ほとんど、計算機ハードウエアの演算性能
- 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

レベル２ BLAS

- 行列–ベクトル積などの演算
  - 例: $y \leftarrow \alpha A x + \beta y$
- 前進/後退代入演算、$T x = y$（Tは三角行列）をxについて解く演算、を含む
- レベル1BLASのみの実装よる、データ再利用性の喪失を回避する目的で提案
  - データアクセス時間を、実装法により短縮
  - レベル1BLASに比べ性能を出しやすい（が十分でない）

レベル３ BLAS

- 行列−行列積などの演算
  - 例： $C \leftarrow \alpha A B + \beta C$
- レベル２ BLASより更にデータ再利用を追及
- 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。

BLASの命名規則

- 関数名：XYYYY
  - X：　データ型
    S:単精度、D：倍精度、C：複素、Z：倍精度複素
  - YYYY：　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

BLASルーチン一覧

LEVEL1 BLAS

zrotg	zdcal	drotg	drot	drotm	zdrot	zswap
dswap	zdscal	dscal	zcopy	dcopy	zaxpy	daxpy
ddot	zdotc	zdotu	dznrm2	dnrm2	dasum	izasum
idamax	dzabs1					

LEVEL2 BLAS

zgemv	dgemv	zgbmv	dgbmv	zhemv	zhbmv	zhpmv	dsymv
dsbmv	ztrmv	zgemv	dgemv	zgbmv	dgemv	zhemv	zhbmv
zhpmv	dsymv	dsbmv	dspmv	ztrmv	dtrmv	ztbmv	ztpmv
dtpmv	ztrsv	dtrsv	ztbsv	dtbsv	ztpsv	dger	zgeru
zgerc	zher	zhpr	zher2	zhpr2	dsyr	dspr	dsyr2
dspr2							

LEVEL3 BLAS

zgemm	dgemm	zsymm	dsymm	zhemm	zsyrk	dsyrk	zherk
zsyr2k	dsyr2k	zher2k	ztrmm	dtrmm	ztrsm	dtrsm	

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*op(A)*op(B) + beta*C　の計算をする
  - op(X) = X　もしくは　op(X) = X’（Xの転置行列）
- 引数
  - TRANSA（入力） - CHARACTER*1
    - TRANSA は op(A) の操作を指定する。以下の文字列を指定。
      - TRANSA = ‘N’ もしくは ’n’,　op(A) = A
      - TRANSA = ‘T’ もしくは ’t’,　op(A) = A’
      - TRANSA = ’C’ or ’c’,　op(A) = A ‘
  - TRANSB（入力） - CHARACTER*1
    - TRANSB は 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×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×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×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×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）

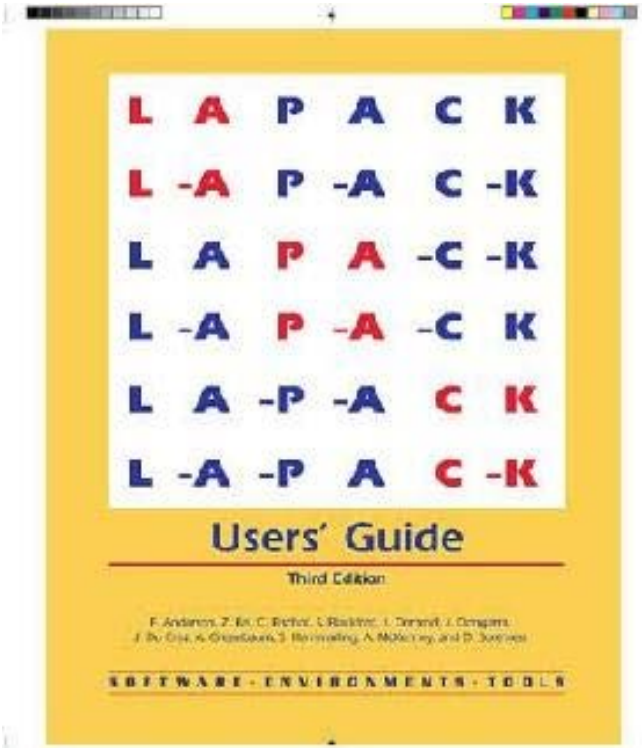
- BETA（入力）– DOUBLE PRECISION
  - スカラ値　BETAの値を設定する。
- C（入力／出力）– DOUBLE PRECISION
  - 行列Cの配列。
  - 入力時、m × n の配列に行列Cを入れる。配列には、BETAが0でない限り、行列Cを入れる。この場合は、Cの入力は必要ない。
  - 出力時、この配列に、m × n行列の演算結果
    (alpha*op(A)*op(B) + beta*C)が上書きされて戻る。
- LDC（入力）– INTEGER
  - 行列Cの最初の次元数を入れる。LDC は　max(1, m)でなくてはならない。

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

LAPACKとは

- Linear Algebra PACKage
  （線形代数パッケージ）
  - BLASを組み合わせて使用することで、より高度な線形代数計算を実現
  - 線形方程式系の解法
  - 固有値問題の解法
  - 特異値計算
  - 最小二乗法
- 使いたいものを探すのが大変なほど数多くのルーチンが用意されている
- HP： http://www.netlib.org/lapack/



LAPACKの命名規則

- 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、…

LAPACKルーチン一覧

■先頭にデータ型(“s”,“d”,”c”,”z”)のいづれかを付けるとルーチン名になる

bdsdc bdsqr disna gbbrd gbcon gbequ gbequb gbrfs gbrfsx gbsv gbsvx gbsvxx gbtrf gbtrs gebak gebal gebrd gecon geequ geequb gees geesx geev geevx gehrd gejsv gelqf gels gelsd gelss gelsy geqlf geqp3 geqpf geqrf geqrfp gerfs gerfsx gerqf gesdd gesv gesvd gesvj gesvx gesvxx getrf getri getrs ggbak ggbal gges ggesx ggev ggevx ggglm gghrd gglse ggqrf ggrqf ggsvd ggsvp gtcon gtrfs gtsv gtsvx gttrf gttrs hgeqz hsein hseqr opgtr opmtr orgbr orghr orglq orgql orgqr orgrq orgtr ormbr ormhr ormlq ormql ormqr ormrq ormrz ormtr pbcon pbequ pbrfs pbstf pbsv pbsvx pbtrf pbtrs pftrf pftri pftrs pocon poequ poequb porfs porfsx posv posvx posvxx potrf potri potrs ppcon ppequ pprfs ppsv ppsvx pptrf pptri pptrs pstrf ptcon pteqr ptrfs ptsv ptsvx pttrf pttrs sbev sbevd sbevx sbgst sbgv sbgvd sbgvx sbtrd sfrk spcon spev spevd spevx spgst spgv spgvd spgvx sprfs spsv spsvx sptrd sptrf sptri sptrs stebz stedc stegr stein stemr steqr sterf stev stevd stevr stevx sycon syequb syev syevd syevr syevx sygst sygv sygvd sygvx syrfs syrfsx sysv sysvx sysvxx sytrd sytrf sytri sytrs tbcon tbrfs tbtrs tfsm tftri tfttp tfttr tgevc tgexc tgsen tgsja tgsna tgsyl tpcon tprfs tptri tptrs tpttf tpttr trcon trevc trexc trrfs trsen trsna trsyl trtri trtrs trttf trttp tzrzf

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×N*行列で、*X* と *B* は *N×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 >= 0。

インタフェース例：DGESV　（２／３）

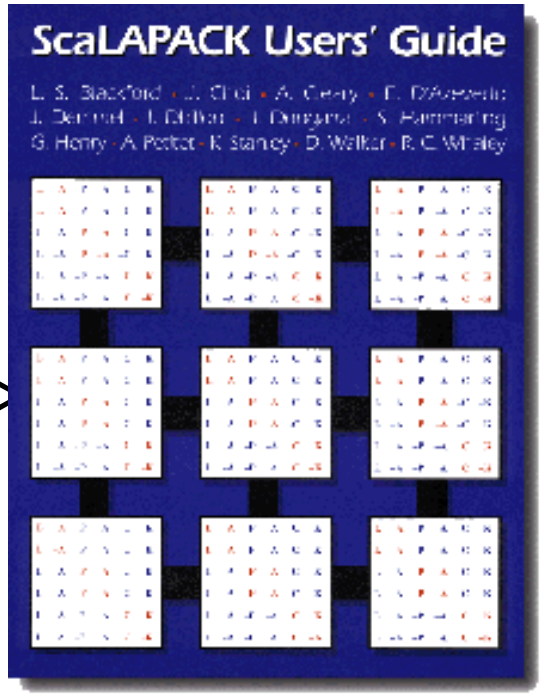
- NRHS（入力）－ INTEGER
  - 右辺ベクトルの数。行列Bの次元数。NRHS >= 0。
- A（入力／出力）－ DOUBLE PRECISION, DIMENSION(:,:)
  - 入力時は、N×Nの行列Aの係数を入れる。
  - 出力時は、Aから分解された行列LとU　= P*L*Uを圧縮して出力する。Lの対角要素は1であるので、収納されていない。
- LDA（入力）－ INTEGER
  - 配列Aの最初の次元の大きさ。LDA　>= max(1,N)。
- IPIVOT（出力）－ DOUBLE PRECISION, DIMENSION(:)
  - 交換行列Aを構成する枢軸のインデックス。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）

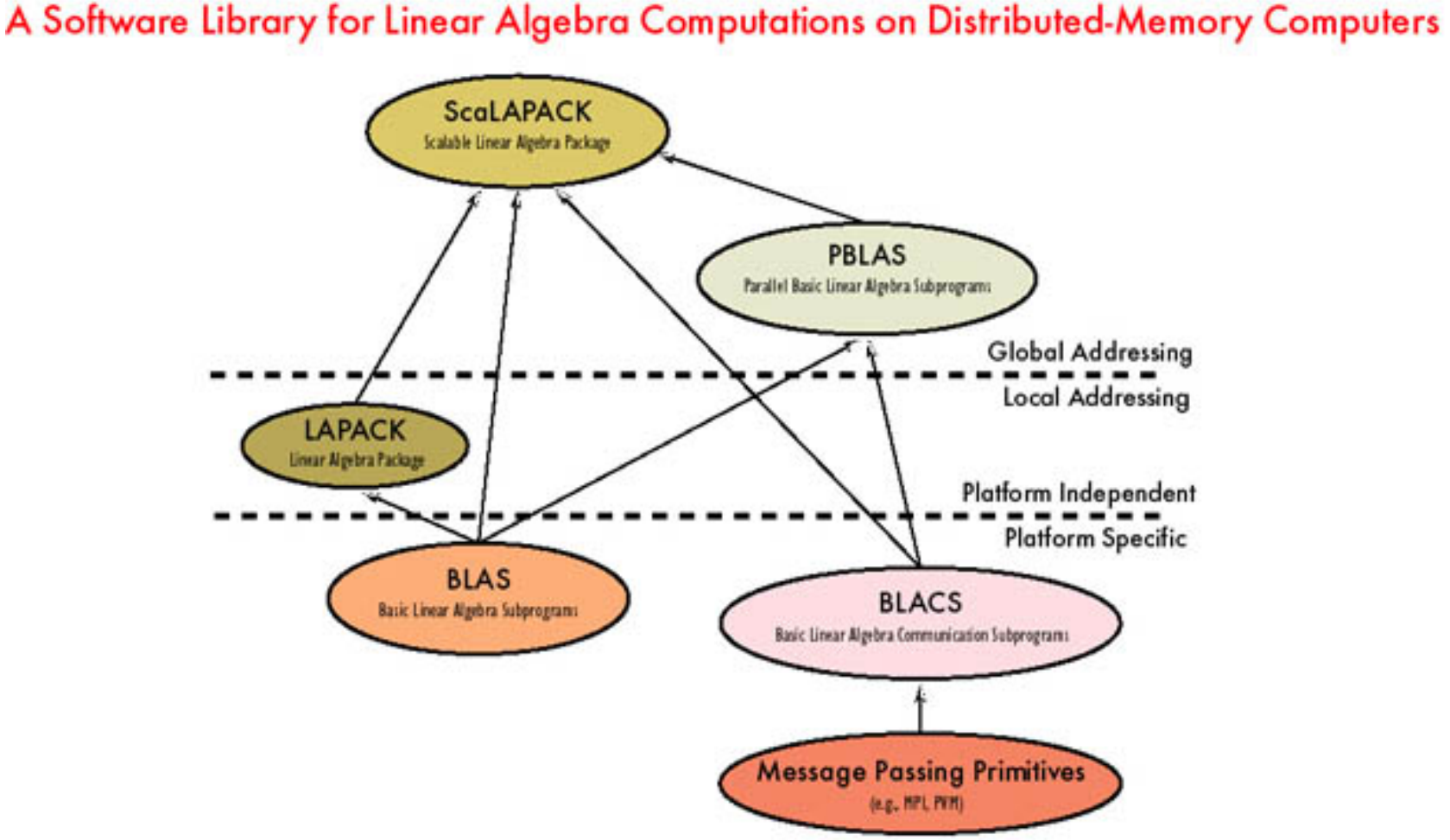
- B（入力／出力）– DOUBLE PRECISION, DIMENSION(:,:)
  - 入力時は、右辺ベクトルの　N×NRHS　行列Bを入れる。
  - 出力時は、もし、INFO = 0　なら、N×NRHS行列である解行列Xが戻る。
- LDB（入力）　ーINTEGER
  - 配列Bの最初の次元の大きさ。LDB >= max(1,N)。
- INFO（出力）ーINTEGER
  - = 0:　正常終了
  - < 0:　もし INFO = –i　なら i–th 行の引数の値がおかしい。
  - > 0:　もし INFO = i　なら U(i,i) が厳密に0である。分解は終わるが、U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
- ユーザインタフェースはLAPACKに＜類似＞
- ソフトウエアの＜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ソフトウエア構成図



（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
  - 各社が提供するMPIライブラリの利用を想定しており、ScaLAPACK内でコード修正せずに使うことを目的とする
    - 通信ライブラリのラッパー的役割でScaLAPACK内で利用
    - 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*×*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*×*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 >= 0
- NRHS（大域入力）– INTEGER
  - 右辺ベクトルの数。行列Bの次元数。NRHS >= 0
- A（局所入力／出力）– DOUBLE PRECISION, DIMENSION(:,:)
  - 入力時は、N×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×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×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。

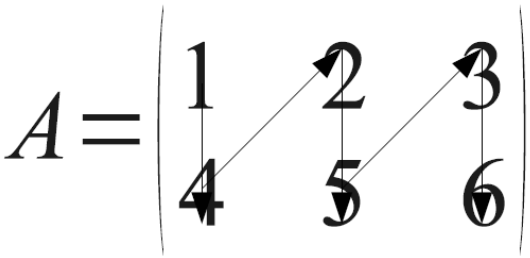
インタフェース例：PDGESV　（3 / 4）

- 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。

C言語でＢＬＡＳ，LAPACKを使う時の注意

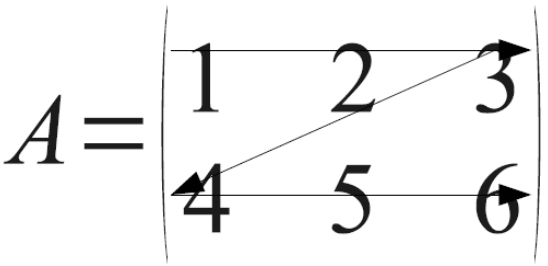
■BLAS, LAPACKはFortranで書かれている

- ルーチン名が変えれている（DGEMM⇒dgemm_）
- 引数はポインタ引き渡し
- 配列インデックスは1−オリジン
- ２次元配列はColomnMajor



ColomnMajor

Fortran



Row Major

C言語

プログラム実習Ⅰ (BLAS/DGEMM)

サンプルプログラムの説明（BLAS）

- 1コア（逐次）実行版です
- スレッド並列化には、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。
  - 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
    - ジョブスクリプト内に使用スレッド情報を記載
      - Oakleaf-fxの場合

        export OMP_NUM_THREADS=16
      - Reedbushの場合

        #PBS -l select=1:mpiprocs=1:ompthreads=1

実習を行うあたっての注意点(BLAS DGEMM)

- C言語版、Fortran言語版のファイル名（共通）
  Mat-Mat-BLAS-fx.tar (Oakleaf-fx)
  Mat-Mat-BLAS-rb.tar (Reedbush)
- ジョブスクリプトファイルmat-mat-blas.bash
  中のキュー名を
  lecture から tutorial に変更してから
  pjsub してください。
  - lecture : 実習時間外のキュー（同時実行数１）
  - tutorial : 実習時間内のキュー（同時実行数4+）

BLAS DGEMMのサンプルプログラムの実行（Oafleaf-fx, C言語版/Fortran言語版共通）

- 以下のコマンドを実行する

```
$ cp /home/z30107/ Mat-Mat-BLAS-fx.tar ./
$ tar xvf Mat-Mat-BLAS-fx.tar
$ cd Mat-Mat-BLAS
$ cd C  //C言語の人
```

- `$ cd F  //Fortran言語の人`

```
$ make
$ pjsub mat-mat-blas.bash
```

- 実行が終了したら、以下を実行する

```
$ cat mat-mat-blas.bash.oXXXXXX (XXXXXXは数値)
```

BLAS DGEMMのサンプルプログラムの実行（Reedbush, C言語版/Fortran言語版共通）

- cdwコマンド(Lustre作業用ディレクトリに移動する)を実行してLustreファイルシステムに移動する
- 以下のコマンドを実行する

```
$ cp /lustre/gt00/Mat-Mat-BLAS-rb.tar ./
$ tar xvf Mat-Mat-BLAS-rb.tar
$ cd Mat-Mat-BLAS
$ cd C  //C言語の人
```

- `$ cd F  //Fortran言語の人`

```
$ make
$ qsub mat-mat-blas.bash
```

- 実行が終了したら、以下を実行する

`$ cat mat-mat-blas.bash.oXXXXXX` (XXXXXXは数値)

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 = 1000

Mat-Mat time = 6.759452 [sec.]

295.881965 [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  =  1000
Mat-Mat time[sec.] =  6.788638369180262
MFLOPS =  294.6098887856930
 OK!
```

サンプルプログラムの説明（C言語）

- #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　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×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=1000)

Oakleaf-fxでのBLAS呼び出し

- 富士通コンパイラから、BLAS (SSL II 数学ライブラリ)を呼び出す場合、以下のオプションを付けます。

●C/Fortran言語共通(BLASが逐次(1コア実行))
mpifrtpx -SSL2 <プログラム名> :Fortran言語
mpiccpx -SSL2 <プログラム名> :C言語

●C/Fortran言語共通(BLASがスレッド実行)
mpifrtpx -Kfast,openmp -SSL2BLAMP <プログラム名>
:Fortran言語
mpiccpx -Kfast,openmp -SSL2BLAMP <プログラム名>
:C言語

ReedBushでのBLAS呼び出し

- インテルコンパイラから、BLAS (Intel MKLライブラリ)を呼び出す場合、以下のオプションを付けます。

●BLASが逐次(1コア実行)

mpiifort -mkl <プログラム名> :Fortran言語

mpiicc -mkl <プログラム名> :C言語

●BLASがスレッド実行

mpiifort -mkl=parallel -qopenmp <プログラム名>
:Fortran言語

mpiicc -mkl=parallel -qopenmp <プログラム名>
:C言語

演習課題１　（BLAS）

1. MyMatMat関数（手続き）の行列積コードを、BLAS DEGMMルーチンの呼び出しにより高速化してください
   - コードで行列−行列積の実行部をコメントアウト
   - DEGMMの引数の並びに注意
   - C言語は、DGEMM のFortran手続きを呼び出します。以下に注意してください。
     - 関数名： DGEMM_
     - 全ての引数が ポインタ引き渡し

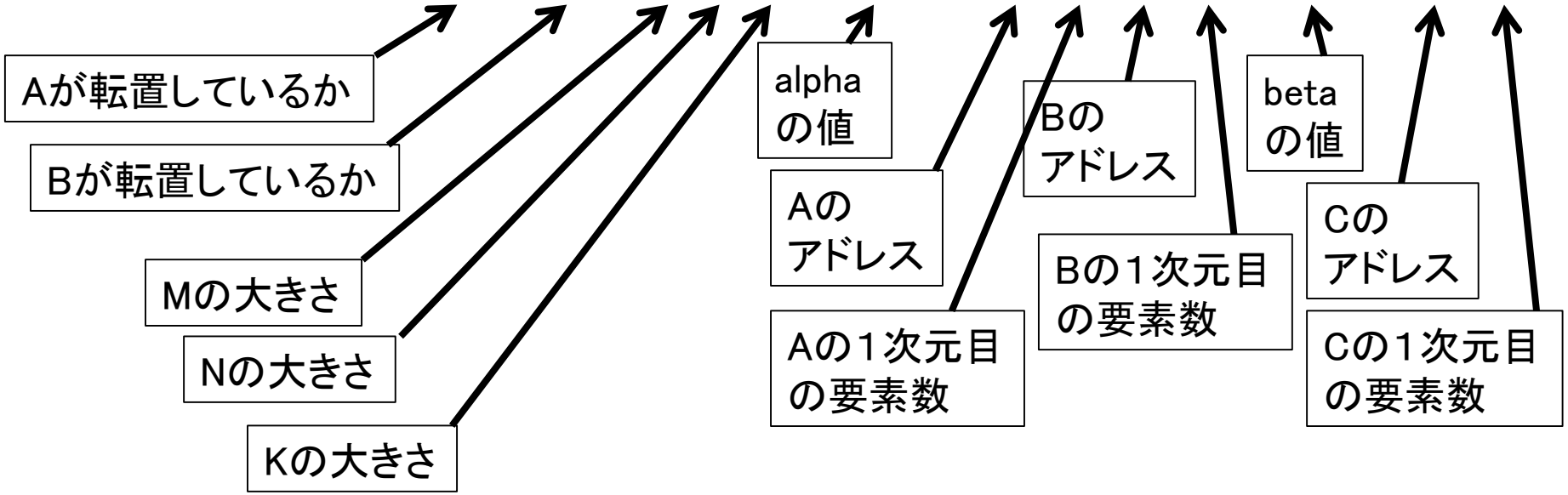
演習課題１のヒント

- 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)



演習課題２ （BLAS）

2-1 スレッド並列コードの実行

スレッド並列版のDGEMMを呼び出し、
いろいろなスレッド数で実行してください。

2-2 台数効果による性能向上の評価

逐次実行に比べ、どれだけ高速化されたか、
性能評価をしてください。

BLAS DGEMM回答

BLAS DGEMM演習問題１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演習問題１の回答（Fotran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)
```

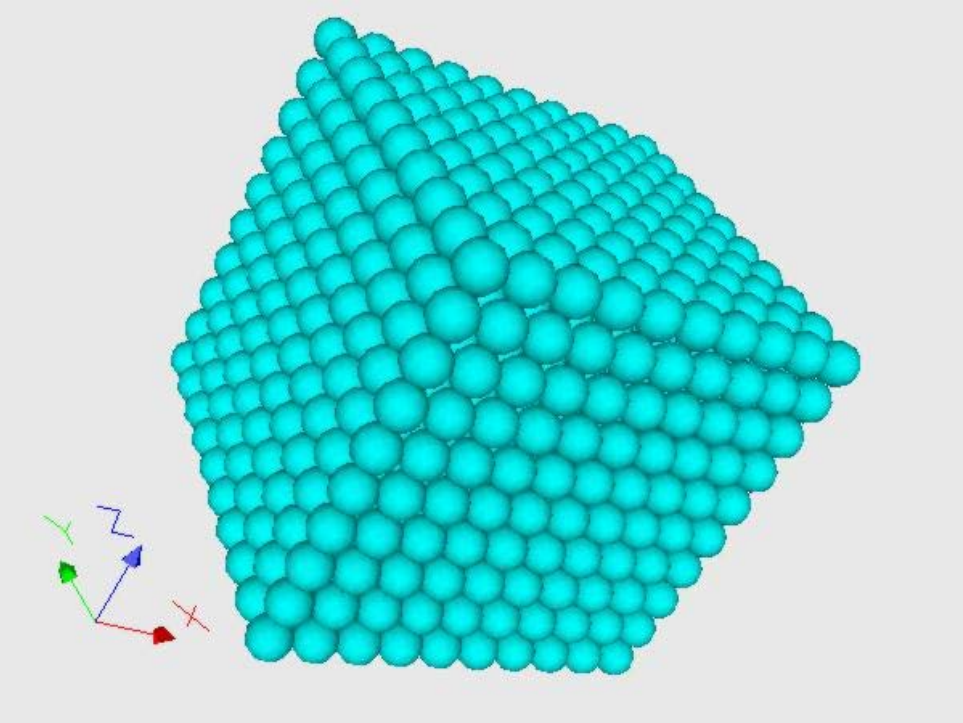
プログラム実習 II
(LAPACK, ScaLAPACK)

内容

- サンプルプログラムの説明
- LAPACK/DGESVサンプルプログラム実行
  （C言語、Fortran言語）
- ScaLAPACK/PDGESVサンプルプログラム実行
  （Fortran言語のみ）

サンプルプログラムで計算する問題

■格子状に並んだ粒子間の熱伝導問題



■詳細資料

http://nkl.cc.u-tokyo.ac.jp/09e/CE23/CE23.pdf

サンプルプログラム・シリアル版の計算手順

- 制御データ入力
- 粒子生成
- 係数マトリクス計算
  - 熱伝導
  - 対流熱伝達
  - 発熱
- CG法による連立一次方程式求解
  （test_org.f, test_org.c）
  - 密行列
  - 点ヤコビ前処理
- 出力
  - MicroAVS用

サンプルプログラムの説明

- 誤差ERRの計算は，10×10×10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、ライブラリを用いず、手製の反復解法（CG法）を用いて、線形方程式系の求解を行うコードです。

サンプルプログラムの実行（LAPACK/DGESV）

サンプルプログラムの説明（ LAPACK版）

- 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で連立一次方程式を解く場合

実習を行うあたっての注意点(LAPACK dgesv)

- C言語版、Fortran言語版のファイル名（共通）
  - lecLAPACK-fx.tar （Oakleaf-fxのみ）
- ジョブスクリプトファイルgo.sh中のキュー名を
  lecture から tutorial に変更してから
  pjsub してください。
  - lecture : 実習時間外のキュー（同時実行数1）
  - tutorial : 実習時間内のキュー（同時実行数4+）

LAPACK dgesvのサンプルプログラムの実行（Oakleaf-fx，C言語版/Fortran言語版共通）

- 以下のコマンドを実行する

```
$ cp /home/z30107/lecLAPACK-fx.tar ./
$ tar xvf lecLAPACK-fx.tar
$ cd sphere-LAPACK
$ cd C  //C言語の人
$ cd F  //Fortran言語の人
$ make
$ pjsub go.bash
```

- 実行が終了したら、以下を実行する

```
$ cat go.bash.oXXXXXX (XXXXXXは数字)
```

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  = 0.000013 [sec.]
 53612532.46 [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。

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] = 1.529976725578308E-05
MFLOPS = 43802496.39070075
     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。

富士通LAPACK呼び出しオプション

- 富士通コンパイラから、LAPACK(SSL II (Scientific Subroutine Library II) 数学ライブラリ)を呼び出す場合、以下のオプションを付けます。

●C/Fortran言語共通(LAPACKが逐次(1コア実行))

mpifrtpx -SSL2 〈プログラム名〉 :Fortran言語

mpiccpx -SSL2 〈プログラム名〉 :C言語

●C/Fortran言語共通(LAPACKがスレッド実行)

mpifrtpx -Kfast, openmp **-SSL2BLAMP** 〈プログラム名〉

: Fortran言語

mpiccpx -Kfast, openmp **-SSL2BLAMP** 〈プログラム名〉

:C言語

演習課題1 （LAPACK）

1. test.c もしくは test.f のメイン関数（手続き）の連立一次方程式の求解を行う部分に、LAPACK dgesvルーチンを呼び出してください
   - 引数の並びに注意してください。
   - C言語は、dgesv のFortran手続きの呼び出しになります。以下に注意してください。
     - 関数名： **dgesv_**
     - 全ての引数がポインタ引き渡し
   - デバッグとして、誤差（ERR）が十分小さいことを確認してください。

演習課題2（LAPACK）

2-1 CG法使用版test_org.c もしくはtest_org.f をコンパイルして、LAPACK版と実行速度を比べてください。

2-2 問題サイズを大きくして、LAPACK版、CG法版の実行速度を比べてください。

- 問題サイズはNX＊ NY＊ NZで与えられ、
  NX, NY, NZの値はファイル“inp”の一行目
  3つの値が対応しています

演習課題 3, 4 (LAPACK)

3. 富士通コンパイラからスレッド並列版のLAPACKを呼び出すように変更してください。
   - 16並列までのスレッド版LAPACKを利用し、性能評価を行ってください。

4. ソース一式をReedbushへ送り、インテルコンパイラを使って計算を実行してください。

ヒント：Makeファイルで使用コンパイラを書き換える

ヒント：Makeファイルでコンパイラオプションを書き換える

ヒント：Reedbush用ジョブスクリプトを作成する

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（Fotran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）

サンプルプログラムの説明(ScaLAPACK版)

- 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（プロセッサー・グリッド8×8）、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、プロセッサー・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，10×10×10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．

実習を行うあたっての注意点 (ScaLAPACK pdgesv)

- Fortran言語版のファイル名

  lecScaLAPACK-fx.tar

  ※ScaLAPACKはC言語版のサンプル はありません

- ジョブスクリプトファイルgo.sh　中のキュー名を

  lecture から tutorial に変更してから

  pjsub　してください。

  - lecture : 実習時間外のキュー（同時実行数1）
  - tutorial : 実習時間内のキュー（同時実行数4+）

ScaLAPACK pdgesvのサンプルプログラムの実行（Fortran言語版のみ）

- 以下のコマンドを実行する

```
$ cp /home/z30107/lecScaLAPACK-fx.tar ./
$ tar xvf lecScaLAPACK-fx.tar
$ cd sphere-ScaLAPACK
$ cd F
$ make
$ pjsub go.bash
```

- 実行が終了したら、以下を実行する

```
$ cat go.bash.oXXXXXX (XXXXXXは数字)
```

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] = 3.399793058633804E-06

MFLOPS = 216189698.2744275

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。

富士通ScaLAPACK呼び出しオプション

- 富士通コンパイラから、ScaLAPACK(SSL II (Scientific Subroutine Library II) 数学ライブラリ)を呼び出す場合、以下のオプションを付けます。

●C/Fortran言語共通(ScaLAPACKが逐次(1コア実行))

mpifrtpx -SCALAPACK -SSL2 〈プログラム名〉 :Fortran言語

mpiccpx -SCALAPACK -SSL2 〈プログラム名〉 :C言語

●C/Fortran言語共通(ScaLAPACKがスレッド実行)

mpifrtpx -Kfast, openmp **-SCALAPACK -SSL2BLAMP** 〈プログラム名〉

: Fortran言語

mpiccpx -Kfast, openmp **-SCALAPACK -SSL2BLAMP** 〈プログラム名〉

: C言語

演習課題1 （ScaLAPACK）

1. test.f のメイン手続きの連立一次方程式求解部分を、ScaLAPACK pdgesv手続きの呼び出しにより、高速化してください
   - 引数の並びに注意してください。
   - デバッグとして、誤差（ERR）が十分小さいことを確認してください。

演習課題2（ScaLAPACK）

2. 問題サイズを大きくして実行速度を比べてください。
3. 並列実行数を64並列から変更して実行してください。
   - プロセッサー・グリッドを８×８から変更する必要があります。

- 上記の2、3ともに、データ分散（行列の確保のサイズ）の変更（コードの修正）が必須ですので、注意してください。

演習課題３ （ScaLAPACK）

4. 大域配列matに大域的な係数を代入せず、
局所配列 A = sub(mat) に直接、
各プロセスで必要な係数を代入する
プログラムを作成し、メモリに関する
スケーラビリティがあるように、
プログラムを改良してください。
   - 大域配列matの確保は不要となります
   - 解ベクトルxは分散されて戻ります。
解の検証（解の利用方法）の並列化も必要です。

ScaLAPACK pdgesv回答

ScaLAPCK dgesvの回答（Fortran言語版）

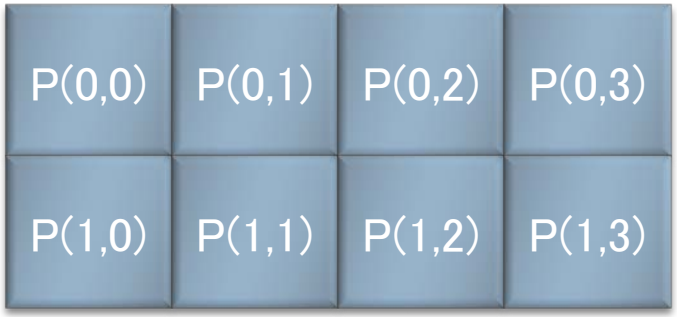
```
      CALL PDGESV(
     &          NN, NRHS,
     &          A, IA, JA, DESCA,  IPIV,
     &          B, IB, JB, DESCB,
     &          INFO )
```

ScaLAPACKを使用するための補足

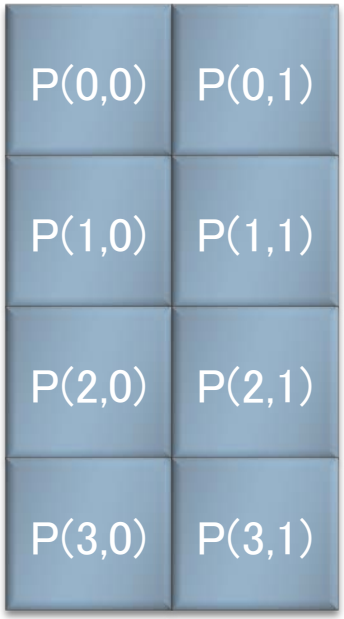
ScaLAPACKにおけるプロセッサー・グリッド

- MPIで起動する総プロセス数は、プロセッサー・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

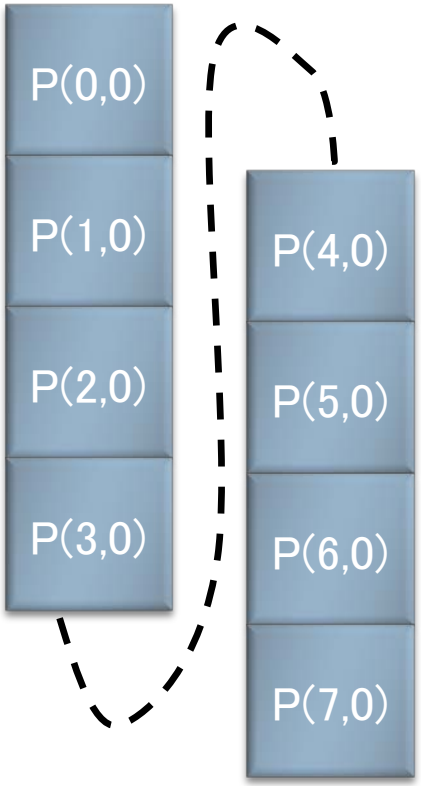
●2×4構成



●4×2構成



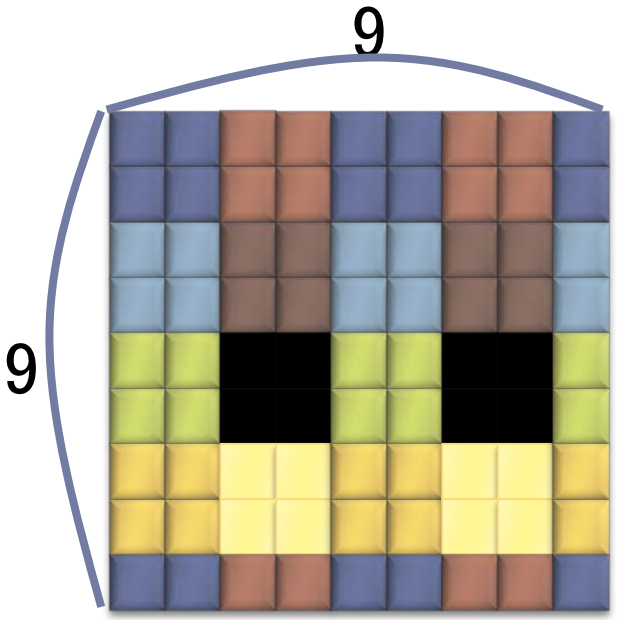
●8×1構成



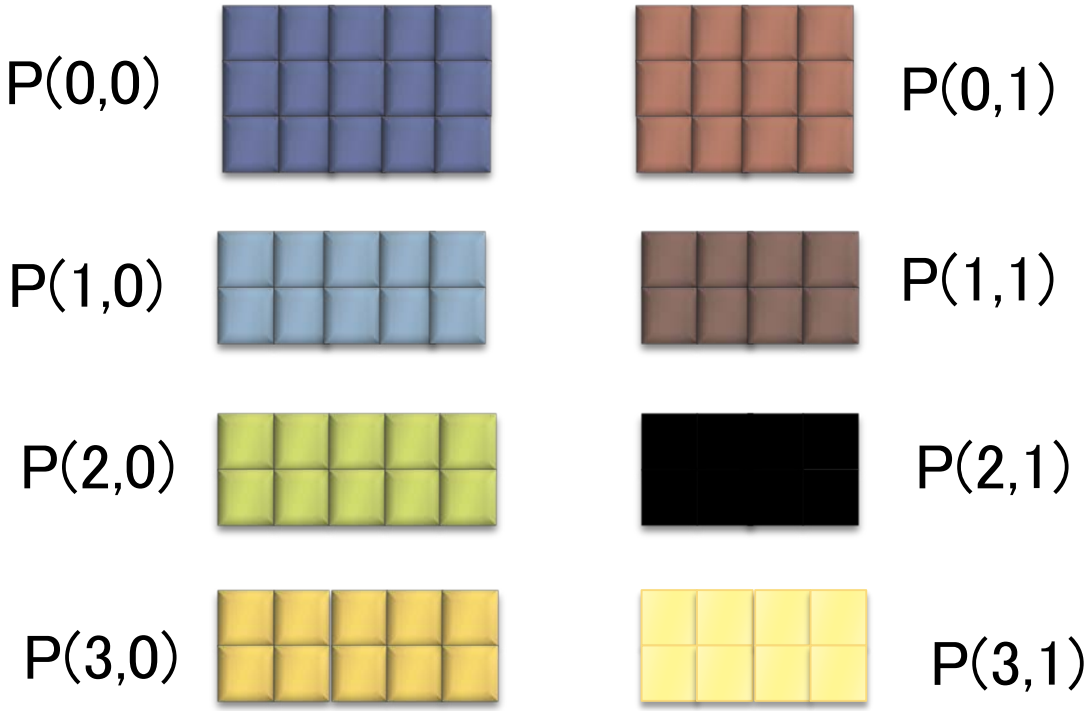
ScaLAPACKにおけるデータ分散

- 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（4×2プロセッサ・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，
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, DESCA, AMAT(I,J))
      ENDDO
      CALL PDELSET( B, J, 1,  DESCB, 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。

お疲れさまでした