「情報システム学特別講義３」
プログラム高速化の基礎（その２）

東京大学情報基盤センター准教授　片桐孝洋

２０１４年４月２２日（火）１４：４０－１６：１０

東京大学情報基盤センター
Information Technology Center,The University of Tokyo

講義日程
（情報システム学特別講義３）

レポートおよびコンテスト課題
（締切：
2014年8月11日（月）24時　厳守

1. ~~４月８日：ガイダンス~~
2. ~~４月15日~~
   - ~~プログラム高速化の基礎（その１）~~
3. ４月22日
   - プログラム高速化の基礎（その2）
4. ５月１３日
   - MPIの基礎
5. ５月２０日
   - OpenMPの基礎
6. ５月２７日
   - Hybrid並列化技法
     （MPIとOpenMPの応用編）
7. ６月３日
   - プログラム高速化の応用
8. ６月１０日
   - 行列ーベクトル積の並列化
9. ６月１７日
   - べき乗法の並列化
10. ６月２４日
    - 行列−行列積の並列化
11. ７月８日
    - LU分解の並列化
12. ７月１５日
    - 非同期通信
    - 疎行列反復解法の並列化
13. ７月２２日
    - ソフトウェア自動チューニング
14. ８月５日（補講日）
    - エクサフロップスコンピューティング
      に向けて

ブロック化

小さい範囲のデータ再利用

ブロック化によるアクセス局所化

- キャッシュには大きさがあります。
- この大きさを超えると、たとえ連続アクセスしても、キャッシュからデータは追い出されます。
- データが連続してキャッシュから追い出されると、メモリから転送するのと同じとなり、高速なアクセス速度を誇るキャッシュの恩恵がなくなります。
- そこで、高速化のためには、以下が必要です
  1. キャッシュサイズ限界までデータを詰め込む
  2. 詰め込んだキャッシュ上のデータを、何度もアクセスして再利用する

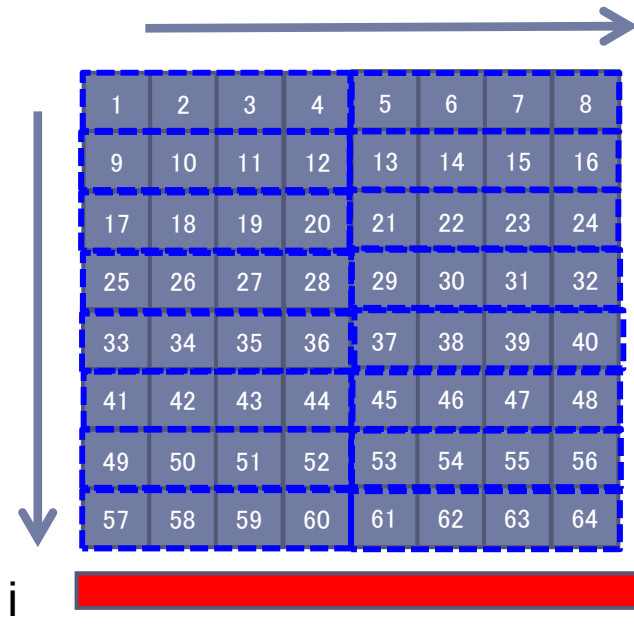
ブロック化によるキャッシュミスヒット削減例

- 行列ー行列積
- 行列サイズ：８×８
  - double A[8][8];
- キャッシュラインは4つ
- １つのキャッシュラインに4つの行列要素が載る
  - キャッシュライン：4×8バイト(double)=32バイト
- 配列の連続アクセスは行方向（C言語）
- キャッシュの追い出しアルゴリズム：
  Least Recently Used (LRU)

配列とキャッシュライン構成の関係

- この前提の、＜配列構成＞と＜キャッシュライン＞の関係
  - ここでは、キャッシュライン衝突は考えません

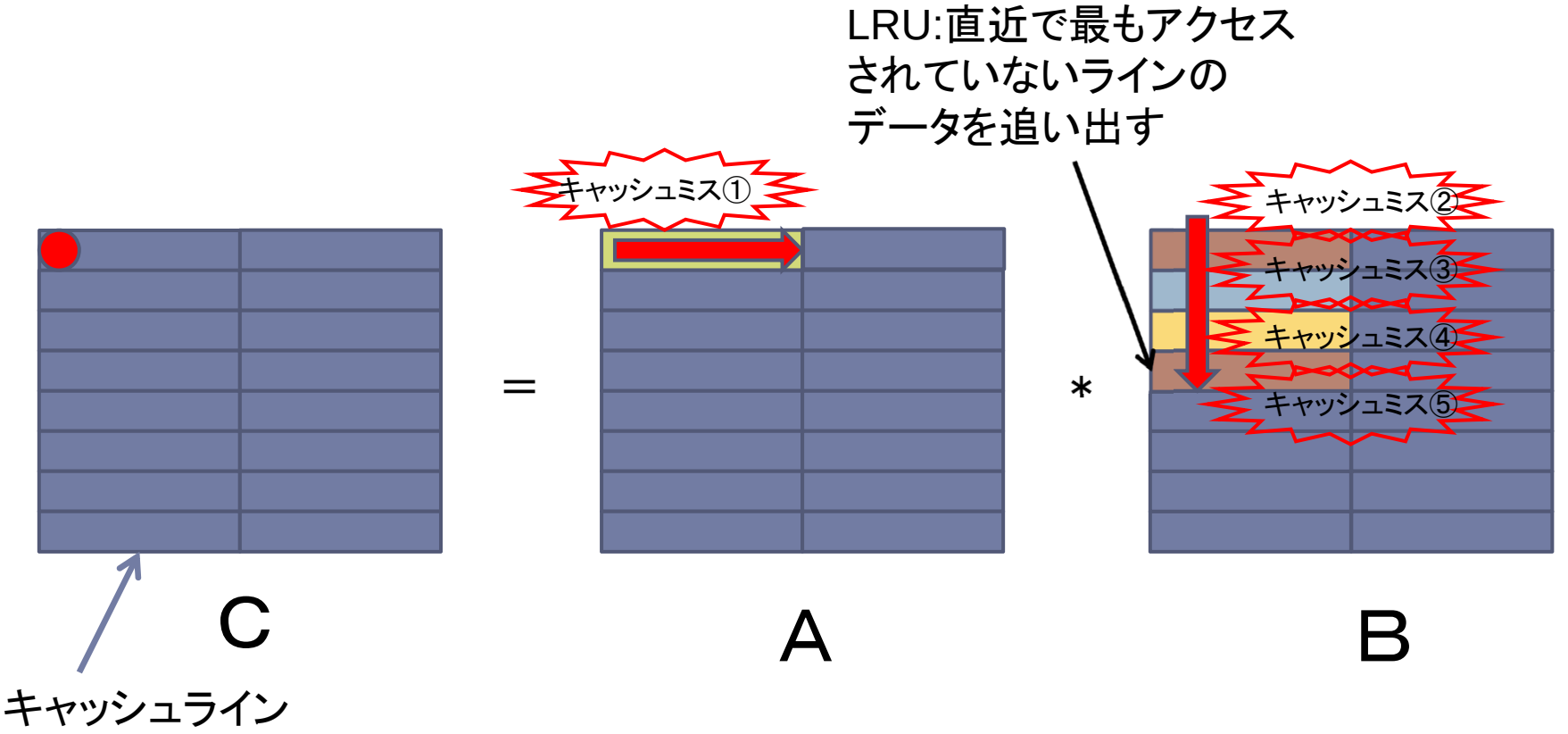
- C言語の場合
  配列A[i][j]、B[i][j]、C[i][j]



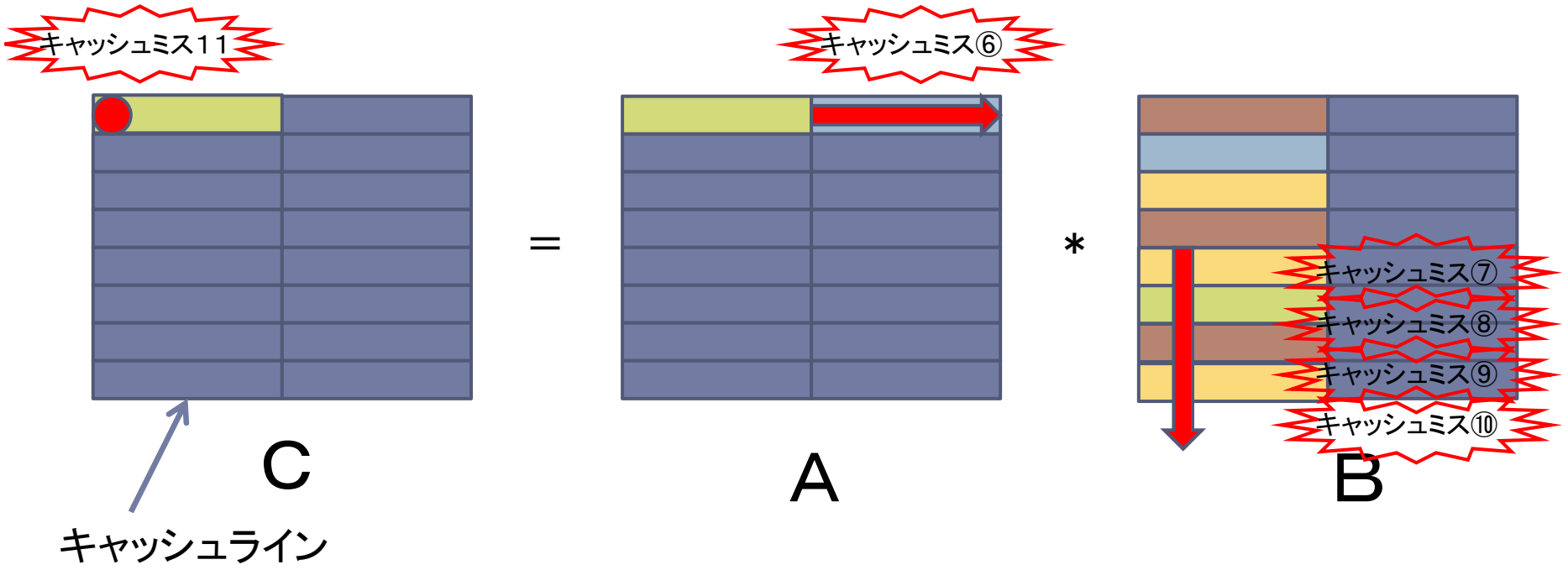
- １×４の配列要素が、キャッシュラインに乗る
- どのキャッシュラインに乗るかは、<配列アクセスパターン> と <置き換えアルゴリズム>依存で決まる

キャッシュラインの構成

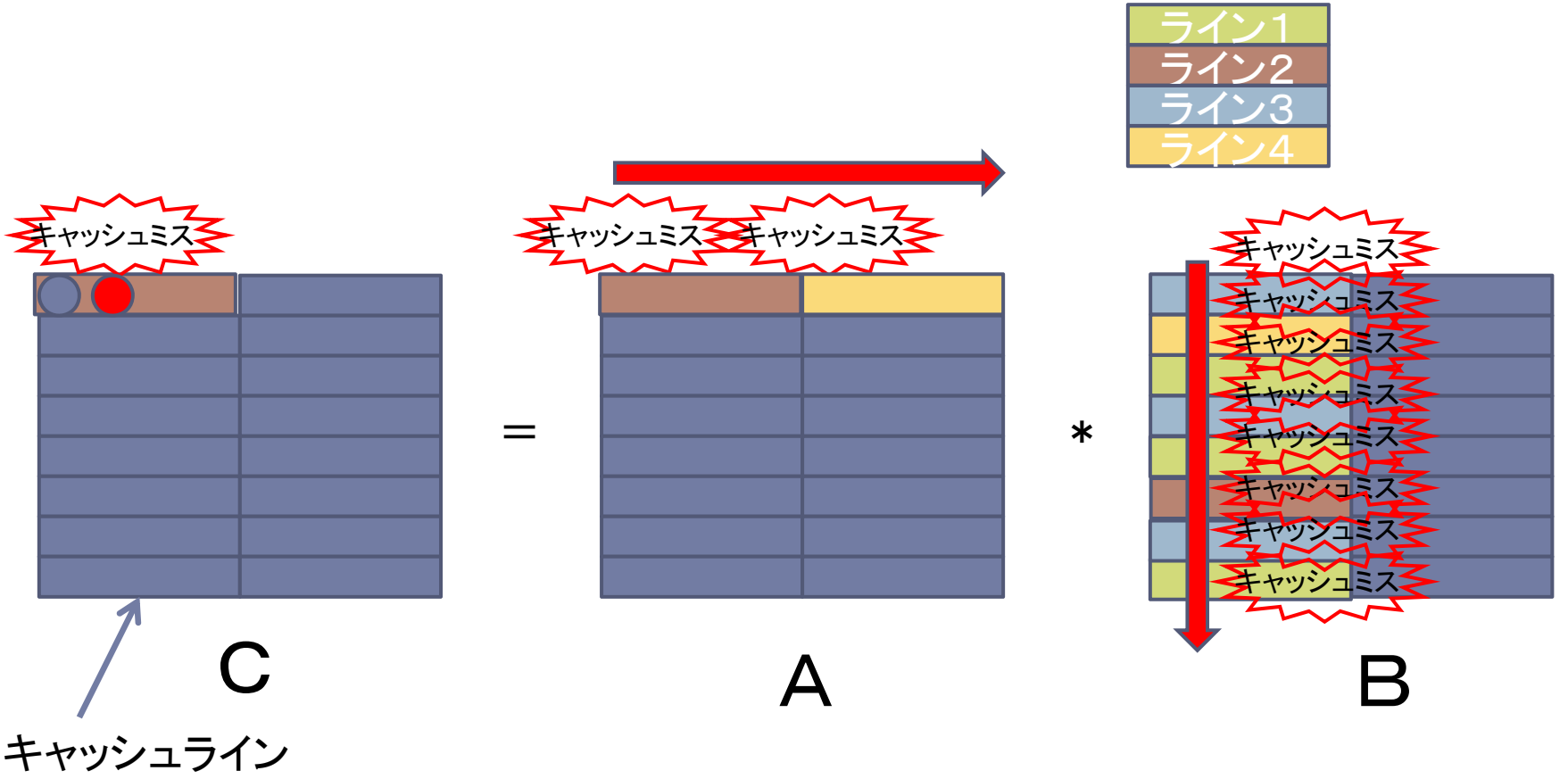
行列-行列積の場合（ブロック化しない）



行列-行列積の場合（ブロック化しない）



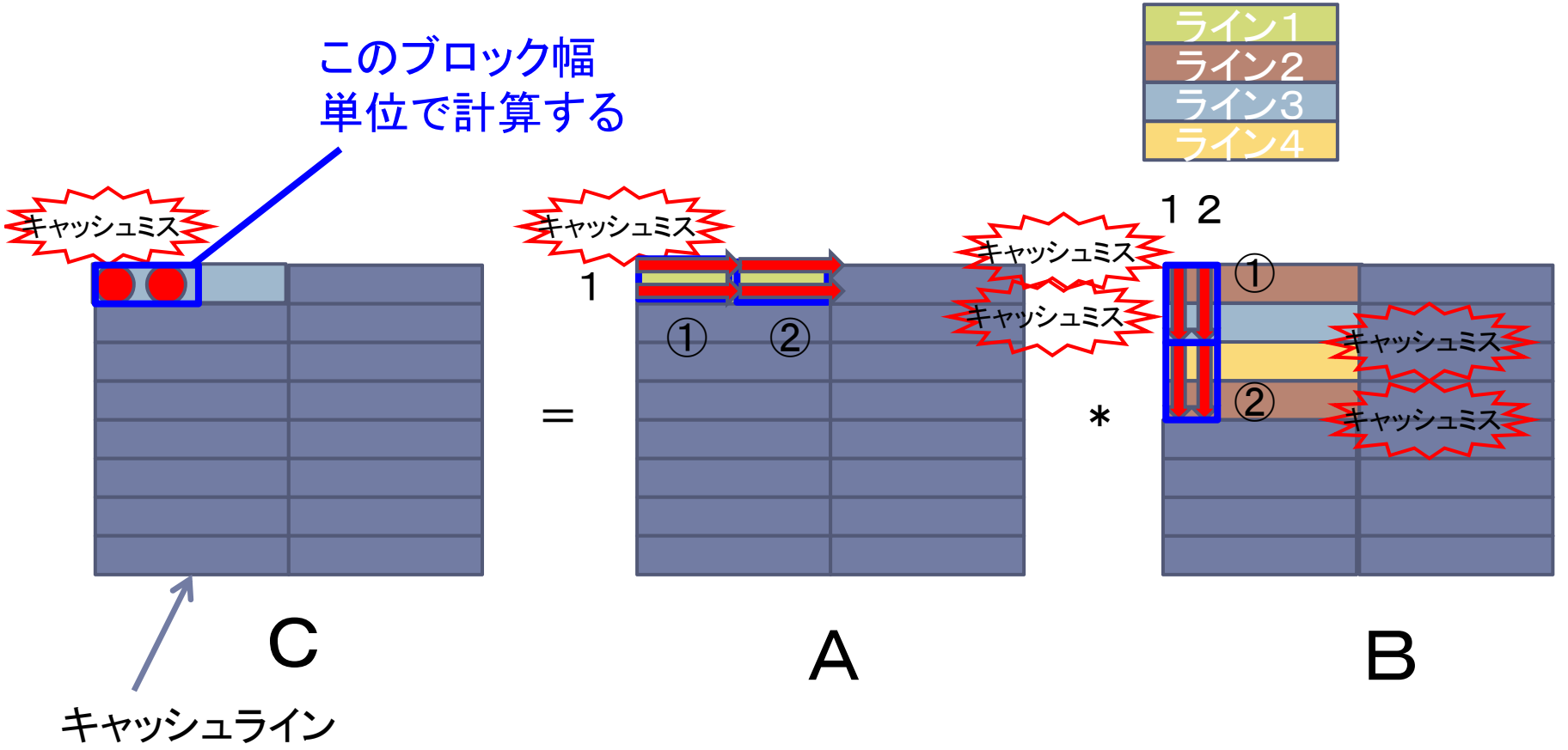
行列-行列積の場合（ブロック化しない）



※2要素計算するのに、
キャッシュミスヒット22回

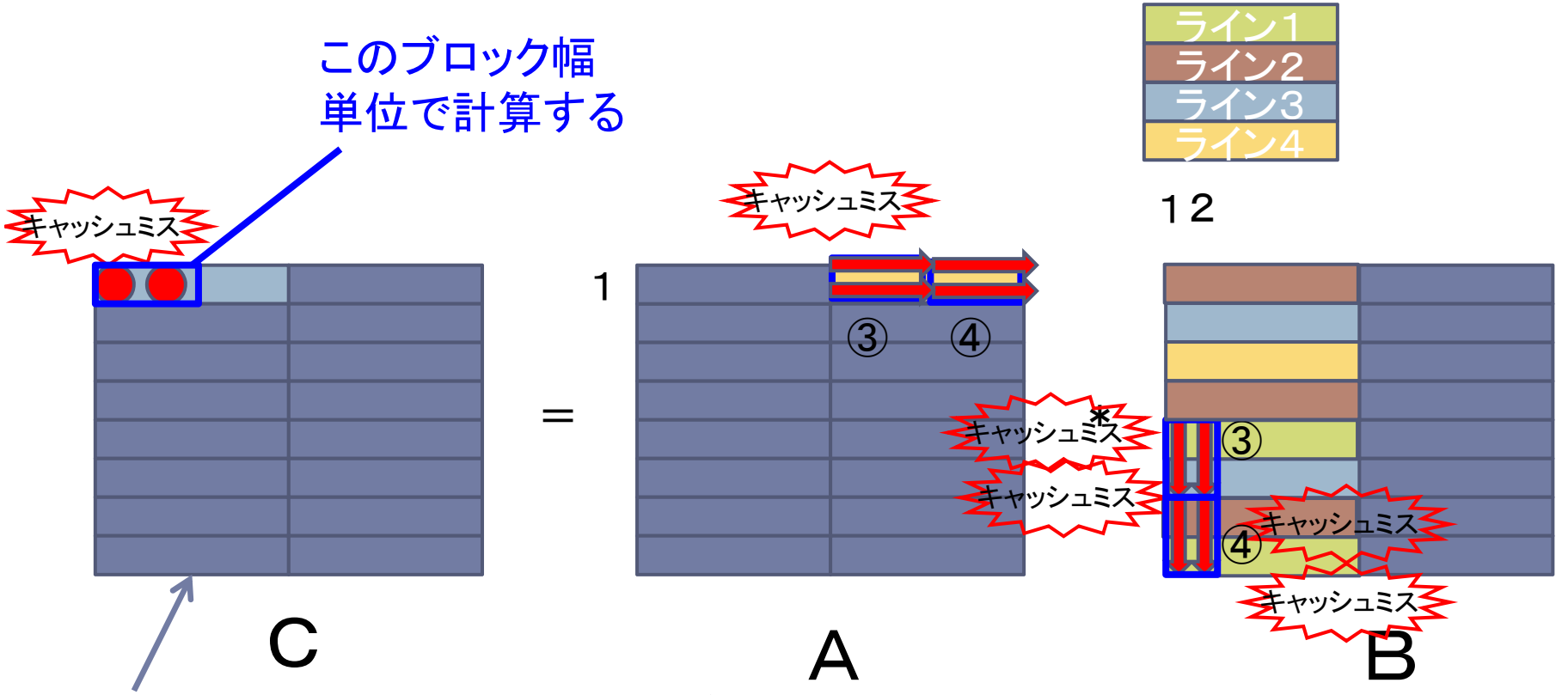
※キャッシュライン4つ、
置き換えアルゴリズム
LRUの場合

行列-行列積の場合（ブロック化する：2要素）



※キャッシュライン4つ、
置き換えアルゴリズム
LRUの場合

行列-行列積の場合（ブロック化する：2要素）



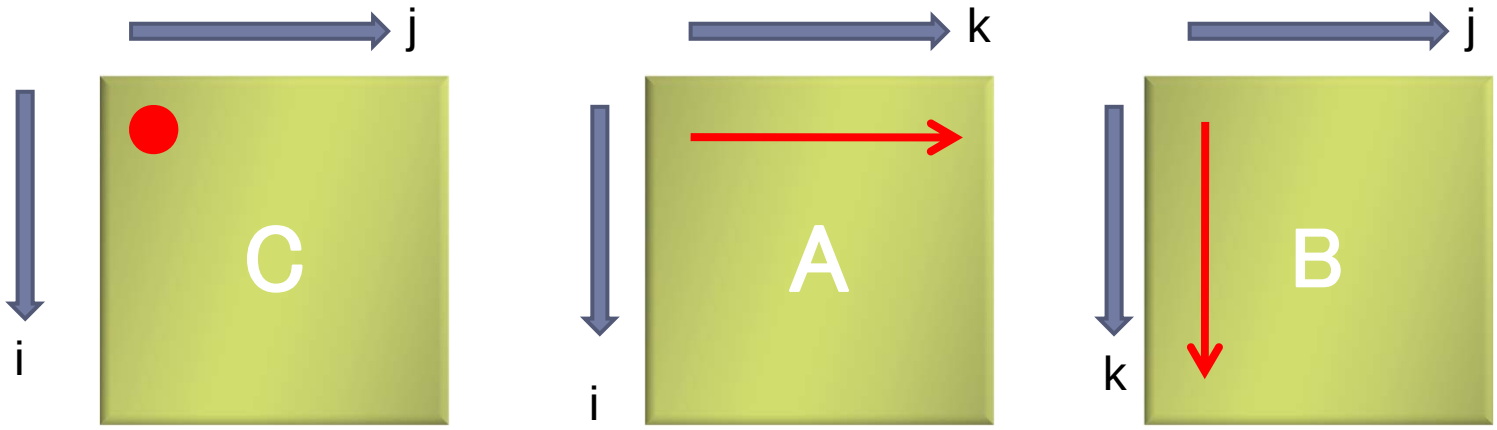
※2要素計算するのに、
キャッシュミスヒット10回

※キャッシュライン4つ、
置き換えアルゴリズム
LRUの場合

行列積コード（C言語）：キャッシュブロック化なし

- コード例

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



行列-行列積のブロック化のコード（C言語）

- nがブロック幅（ibl=16）で割り切れるとき、以下のような6重ループのコードになる

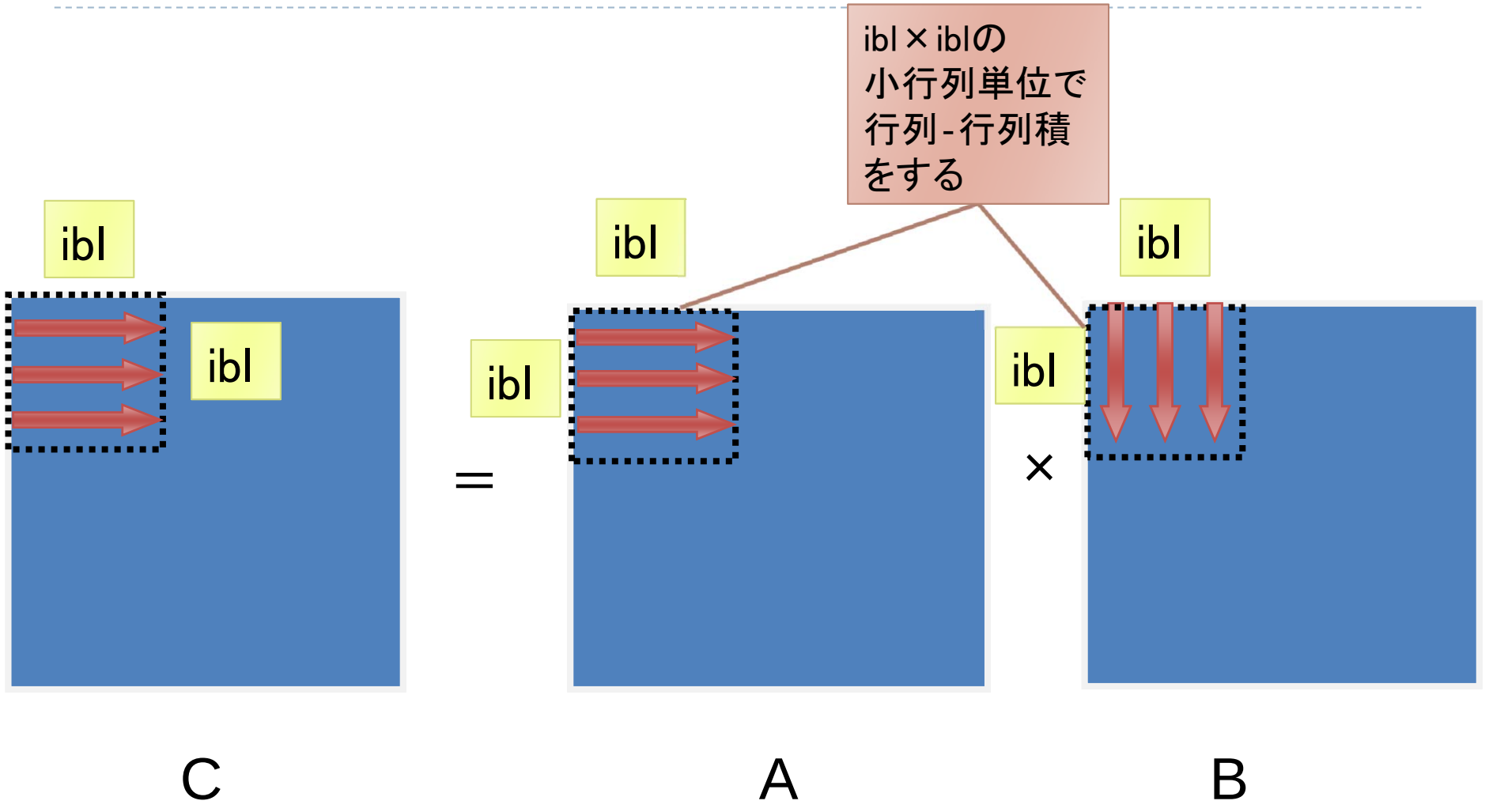
```
ibl = 16;
for ( ib=0; ib<n; ib+=ibl ) {
  for ( jb=0; jb<n; jb+=ibl ) {
    for ( kb=0; kb<n; kb+=ibl ) {
      for ( i=ib; i<ib+ibl; i++ ) {
        for ( j=jb;  j<jb+ibl; j++ ) {
          for ( k=kb; k<kb+ibl; k++ ) {
            C[i][j]  +=  A[i][k]  *  B[k][j];
          } } } } } }
```

行列-行列積のブロック化のコード（Fortran言語）

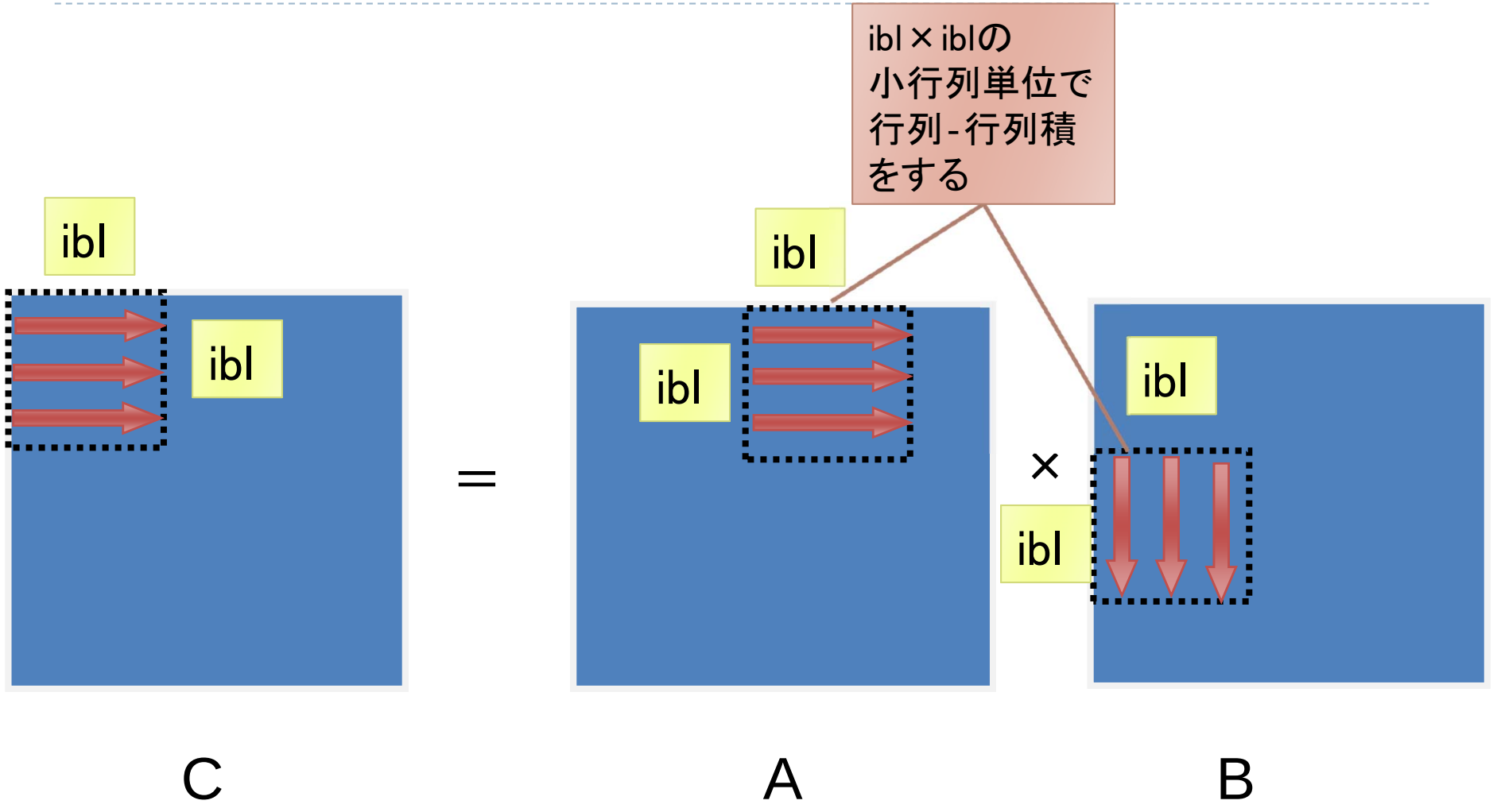
- nがブロック幅（ibl=16）で割り切れるとき、以下のような6重ループのコードになる

```
ibl = 16
do ib=1, n, ibl
   do jb=1, n, ibl
      do kb=1, n, ibl
         do i=ib, ib+ibl-1
            do j=jb, jb+ibl-1
               do k=kb, kb+ibl-1
                  C(i, j) = C(i, j) + A(i, k)  *  B(k, j)
enddo; enddo; enddo; enddo; enddo; enddo;
```

キャッシュブロック化時のデータ・アクセスパターン



キャッシュブロック化時のデータ・アクセスパターン



行列-行列積のブロック化のコードのアンローリング（C言語）

- 行列−行列積の6重ループのコードに加え、さらに各6重ループにアンローリングを施すことができる。
- i−ループ、およびj−ループ2段アンローリングは、以下のようなコードになる。（ブロック幅iblが2で割り切れる場合）

```
ibl = 16;
for (ib=0; ib<n; ib+=ibl) {
  for (jb=0; jb<n; jb+=ibl) {
    for (kb=0; kb<n; kb+=ibl) {
      for (i=ib; i<ib+ibl; i+=2) {
        for (j=jb; j<jb+ibl; j+=2) {
          for (k=kb; k<kb+ibl; k++) {
            C[i  ][j  ] += A[i  ][k] * B[k][j  ];
            C[i+1][j  ] += A[i+1][k] * B[k][j  ];
            C[i  ][j+1] += A[i  ][k] * B[k][j+1];
            C[i+1][j+1] += A[i+1][k] * B[k][j+1];
          } } } } } }
```

行列-行列積のブロック化のコードのアンローリング（Fortran言語）

- 行列−行列積の6重ループのコードに加え、
さらに各6重ループにアンローリングを施すことができる。
- i−ループ、およびj−ループ2段アンローリングは、以下のようなコードになる。（ブロック幅iblが2で割り切れる場合）

```
ibl = 16
do ib=1, n, ibl
  do jb=1, n, ibl
    do kb=1, n, ibl
      do i=ib, ib+ibl, 2
        do j=jb, jb+ibl, 2
          do k=kb, kb+ibl
            C(i  , j  ) = C(i  , j  ) + A(i  , k) * B(k, j  )
            C(i+1, j  ) = C(i+1, j  ) + A(i+1, k) * B(k, j  )
            C(i  , j+1) = C(i  , j+1) + A(i  , k) * B(k, j+1)
            C(i+1, j+1) = C(i+1, j+1) + A(i+1, k) * B(k, j+1)
enddo; enddo; enddo; enddo; enddo; enddo;
```

その他の高速化技術

共通部分式の削除（１）

- 以下のプログラムは、冗長な部分がある。

```
d = a + b + c;
f = d + a + b;
```

- コンパイラがやる場合もあるが、以下のように書く方が無難である。

```
temp = a + b;
d = temp + c;
f = d + temp;
```

共通部分式の削除（２）

- 配列のアクセスも、冗長な書き方をしないほうがよい。

```
for (i=0;  i<n;  i++) {
   xold[i] = x[i];
   x[i] = x[i] + y[i];
}
```

- 以下のように書く。

```
for (i=0;  i<n;  i++) {
  dtemp = x[i];
  xold[i] = dtemp;
  x[i] = dtemp + y[i];
}
```

コードの移動

- 割り算は演算時間がかかる。ループ中に書かない。

```
for (i=0;  i<n;  i++) {
    a[i] = a[i] / sqrt(dnorm);
}
```

- 上記の例では、掛け算化して書く。

```
dtemp = 1.0d0 / sqrt(dnorm);
for (i=0;  i<n;  i++) {
    a[i] = a[i] *dtemp;
}
```

ループ中のＩＦ文

- なるべく、ループ中にIF文を書かない。

```
for (i=0;  i<n;  i++) {
  for (j=0;  j<n;  j++) {
    if ( i != j )  A[i][j] = B[i][j];
     else   A[i][j] = 1.0d0;
  } }
```

- 以下のように書く。

```
for (i=0;  i<n;  i++) {
  for (j=0;  j<n;  j++) {
    A[i][j] = B[i][j];
  } }
for (i=0;  i<n;  i++)  A[i][i] = 1.0d0;
```

ソフトウェア・パイプライニングの強化

- 基のコード
  （2段のアンローリング）

定義ー参照の距離が近い
→ソフトウェア的には
何もできない

```
for (i=0; i<n; i+=2) {
   dtmpb0 = b[i];
   dtmpc0 = c[i];
   dtmpa0 = dtmpb0 + dtmpc0;
   a[i] = dtmpa0;
   dtmpb1 = b[i+1];
   dtmpc1 = c[i+1];
   dtmpa1 = dtmpb1 + dtmpc1;
   a[i+1] = dtmpa1;
}
```

- ソフトウェアパイプライニング
  を強化したコード
  （2段のアンローリング）

定義ー参照の距離が遠い
→ソフトウェアパイプライニング
が適用できる機会が増加！

```
for (i=0; i<n; i+=2) {
   dtmpb0 = b[i];
   dtmpb1 = b[i+1];
   dtmpc0 = c[i];
   dtmpc1 = c[i+1];
   dtmpa0 = dtmpb0 + dtmpc0;
   dtmpa1 = dtmpb1 + dtmpc1;
   a[i] = dtmpa0;
   a[i+1] = dtmpa1;
}
```

数値計算ライブラリの利用

数値計算ライブラリ

- 密行列用ライブラリ
  - 行列の要素に0がない（というデータ構造を扱う）
  - 連立一次方程式の解法、固有値問題、FFT、その他
  - 直接解法（反復解法もある）
  - BLAS、LAPACK、ScaLAPACK、SuperLU、MUMPS、FFTW、など
- 疎行列用ライブラリ
  - 行列の要素に0が多い
  - 連立一次方程式の解法、固有値問題、その他
  - 反復解法
  - PETSc、Xabclib、Lis、ARPACK、など

疎行列用ライブラリの特徴

- 疎行列を扱うアプリケーションはライブラリ化が難しい
  - 疎行列データ形式の標準化が困難
    - COO、CRS(CCS)、ELL、JDS、BCSR、・・・
  - カーネルの演算が微妙に違う、かつ、カーネルは広い範囲に分散
    - 陽解法（差分法）を基にしたソフトウェア
- 数値ミドルウェアおよび領域特化型言語（Domain Specific Language, DSL）
  - 解くべき方程式や離散化方法に特化させることで、処理（対象となるプログラムの性質）を限定
  - 以上の限定から、高度な最適化ができる言語（処理系）の作成（DSL）や、ライブラリ化（数値ミドルウェア）ができる
  - 数値ミドルウェアの例
    - ppOpen-HPC(東大)、PETSc(Argonne National Laboratory, USA.) 、Trilinos (Sandia National Laboratory, USA)、など

BLAS

- 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。
  - 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＞というものあるが、まだ定着していない。
    - スパースBLASはIntel MKL(Math Kernel Library)に入っているが、広く使われているとは言えない。

BLAS

- 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

レベル１　BLAS

- 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
    - 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

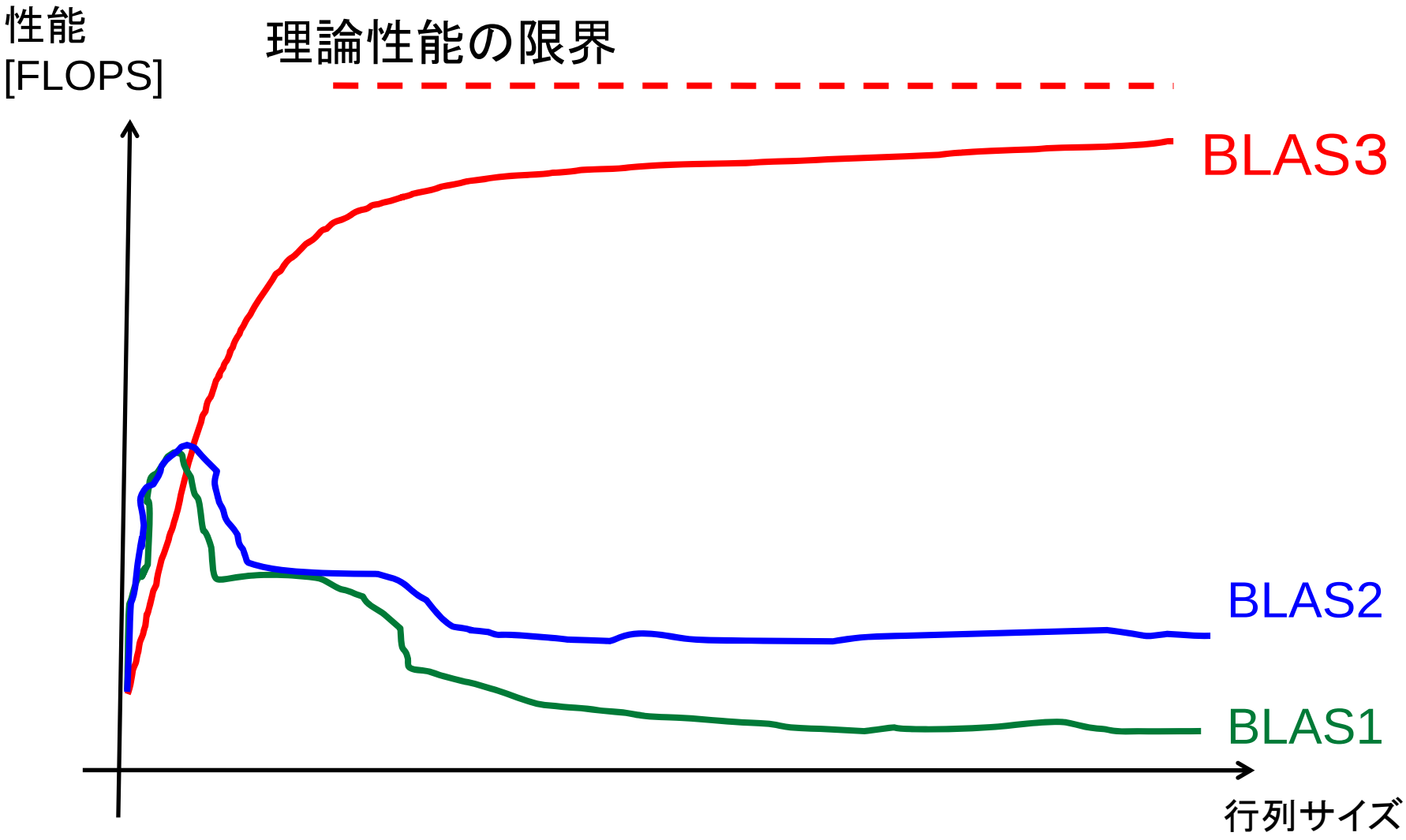
レベル２　BLAS

- 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
  - 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）

レベル３　BLAS

- レベル3　BLAS
  - 行列−行列積などの演算
    - 例：　$C \leftarrow \alpha A B + \beta C$
  - 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



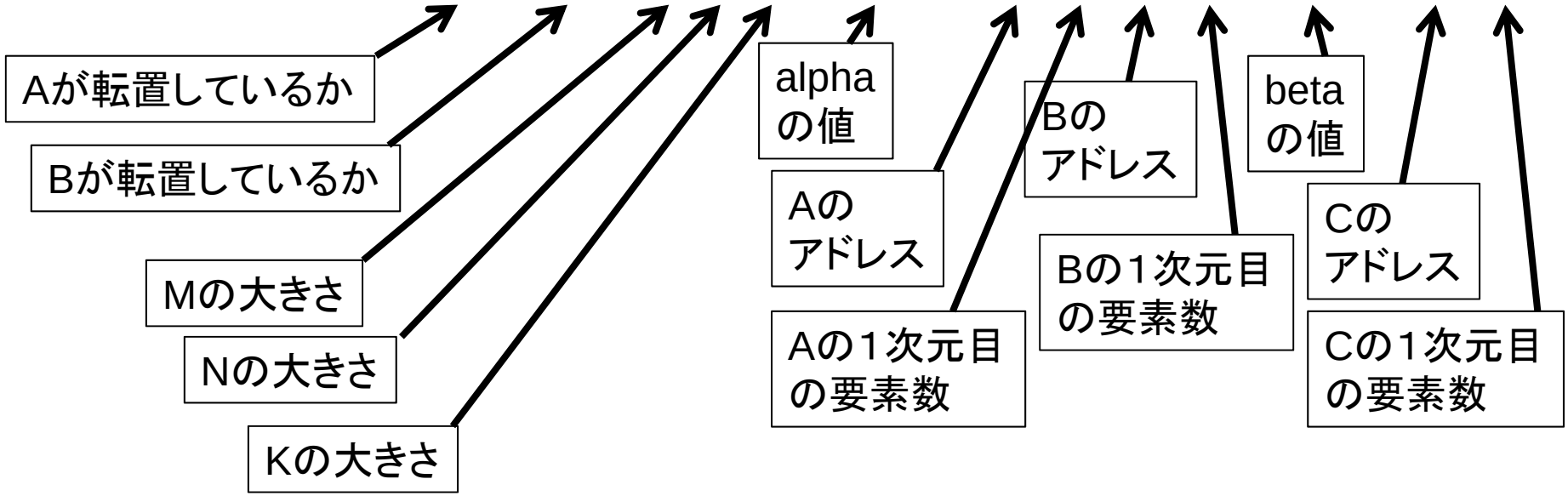
BLAS利用例

- 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)



BLASの機能詳細

- 詳細はHP:　http://www.netlib.org/blas/
- 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

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エア上の実装に特化
    - 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*×*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。

東京大学情報基盤センター
Information Technology Center,The University of Tokyo

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）

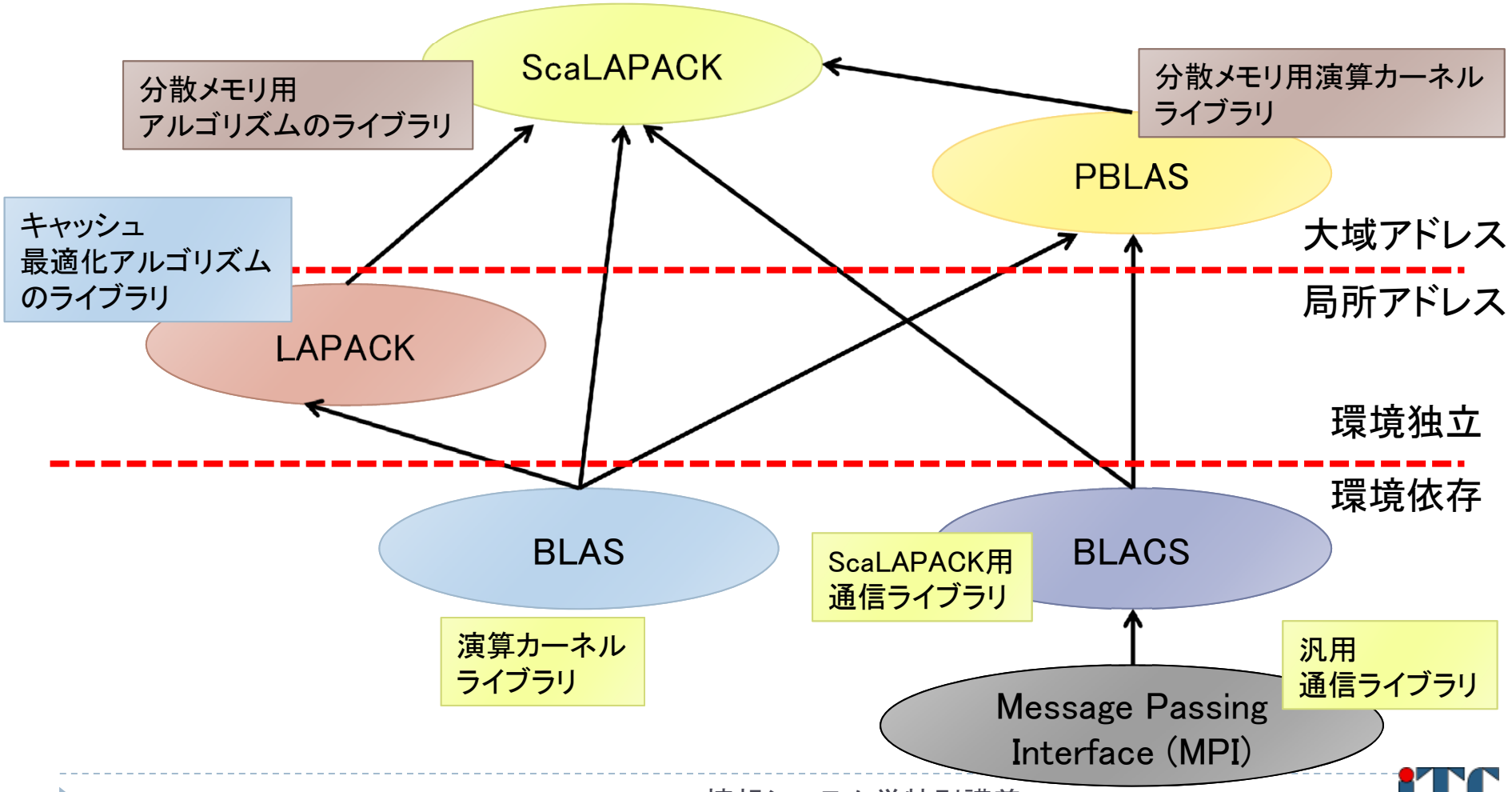
- B（入力／出力）– DOUBLE PRECISION, DIMENSION(:,:)
  - 入力時は、右辺ベクトルの　N×NRHS　行列Bを入れる。
  - 出力時は、もし、INFO = 0　なら、N×NRHS行列である解行列Xが戻る。
- LDB（入力）　—INTEGER
  - 配列Bの最初の次元の大きさ。LDB >= max(1,N)。
- INFO（出力）—INTEGER
  - = 0:　正常終了
  - < 0:　もし INFO = –i　なら i–th 行の引数の値がおかしい。
  - > 0:　もし INFO = i　なら U(i,i) が厳密に0である。分解は終わるが、U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
- ユーザインタフェースは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 （詳細は、「MPI」の講義で説明）
- HP: http://www.netlib.org/scalapack/

ScaLAPACKのソフトウェア構成図

出典：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*×*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。

東京大学情報基盤センター
Information Technology Center,The University of Tokyo

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　（4 / 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。

BLAS利用の注意

- C言語からの利用
  - BLASライブラリは（たいてい）Fortranで書かれている
  - 行列を1次元で確保する
    - Fortranに対して転置行列になるので、BLASの引数で転置を指定
  - 引数は全てポインタで引き渡す
  - 関数名の後に“_”をつける（BLASをコンパイルするコンパイラ依存）
    - 例：dgemm_(...)
- 小さい行列は性能的に注意
  - キャッシュに載るようなサイズ（例えば、100次元以下）の行列については、BLASが高速であるとは限らない
    - BLASは、大規模行列で高性能になるように設計されている
  - 全体の行列サイズは大きくても、利用スレッド数が多くなると、スレッド当たりの行列サイズが小さくなるので注意！
    - 例） N=8000でも、64スレッド並列だと、スレッドあたりN=1000まで小さくなる

レポート課題（その１）

- 問題レベルを以下に設定

問題のレベルに関する記述：
- L00: きわめて簡単な問題。
- L10: ちょっと考えればわかる問題。
- L20: 標準的な問題。
- L30: 数時間程度必要とする問題。
- L40: 数週間程度必要とする問題。複雑な実装を必要とする。
- L50: 数か月程度必要とする問題。未解決問題を含む。

※L40以上は、論文を出版するに値する問題。

- 教科書のサンプルプログラムは以下が利用可能
  - Mat-Mat-noopt-fx.tar

レポート課題（その２）

1. [L15] 行列−行列積のコードをブロック化して性能を評価せよ。このとき、いくつか行列サイズを変化させた上で、ブロックサイズのチューニングを行え。
2. [L10]　身近の計算機にあるBLASを利用する、もしくは、BLASをインストールした上で、性能を評価せよ。
3. [L15] 後藤BLASをインストールし、性能を評価せよ。
4. [L15] 自動チューニング機能付きのBLASライブラリであるATLASをインストールし、性能を評価せよ。
5. [L10] LAPACKを利用して、連立一次方程式の求解を行え。
6. [L15] ScaLAPACKを利用して、連立一次方程式の求解を行え。