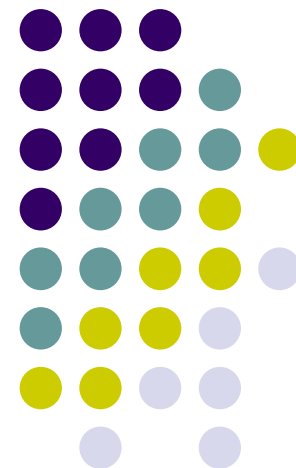


2014年度 実践的並列コンピューティング 第4回

OpenMPによる
共有メモリプログラミング (2)

遠藤 敏夫

endo@is.titech.ac.jp





前回の復習

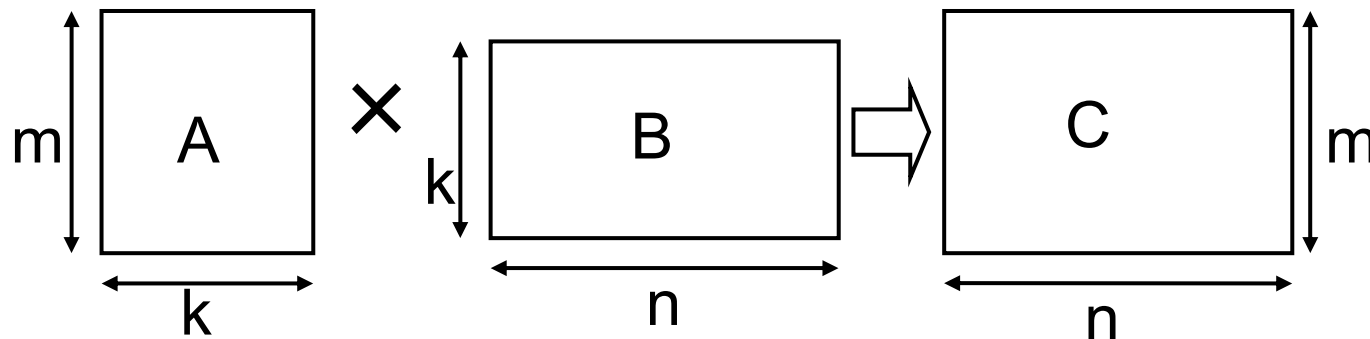
- OpenMP: 共有メモリ並列プログラミングモデルの一つ
- サンプルプログラムはTSUBAMEの
~endo-t-ac/ppcomp/14/
 - 前回はpiとそのOpenMP版を解説
- プログラム中に
 - `#pragma omp parallel`
と書くと、その次のブロック・文が並列に実行される (並列region)
- 並列region中に
 - `#pragma omp for`
と書くと、その次のfor文は、スレッド間で分担して実行される



サンプルプログラム:mm

($m \times k$)行列と($k \times n$)行列の積

- 三重のforループで記述
- 動的な長さの配列. 二次元を一次元で表現
 - Column major format
- 実行オプション: `./mm [m] [n] [k]`
- 計算量: $O(mnk)$
 - 必要な浮動小数演算数は $2mnk$
 - サンプルでは計算速度を $2mnk/t$ (t は実行時間)で計算



C言語の残念な点： 配列長を動的に決められない



- `int a[n];` (これはエラー)を実現するには？
- 重要な関数: `void *malloc(size_t size);`
⇒ `size`バイトのメモリをヒープ領域より確保し, そのポインタを返す.
- 領域が不要になったら, `free`関数で解放する.

固定長さの場合

```
int a[5];
```

…この間は, `a[i]`を
自由に使える…

```
int *a;  
a = (int *)malloc(sizeof(int)*n);  
if (a == NULL) {  
    printf( “メモリ不足¥n” );  
    exit(0);  
}
```

…この間は, `a[i]`を自由に使える…

```
free(a);
```

大きさが動的に決まる多次元配列 を作りたい場合

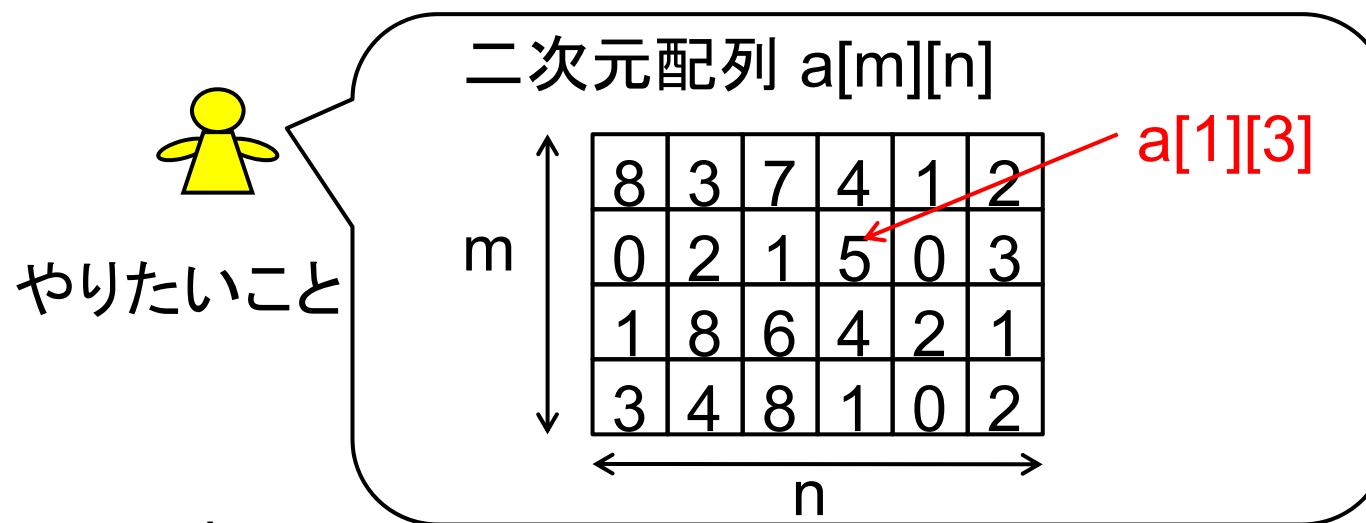


`int a[m][n];` (これはエラー)を実現するには？

素直にはできないので、以下のいずれか

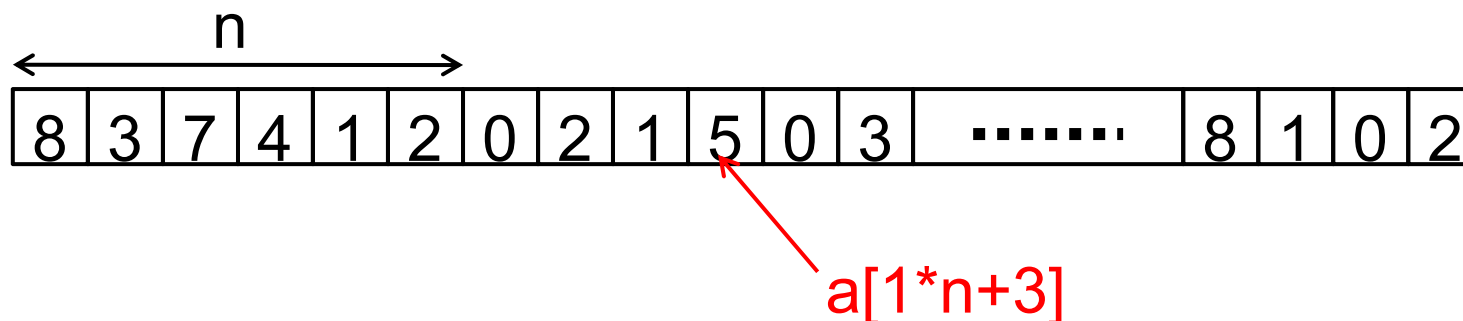
- ポインタのポインタにする
各行の一次元配列をmallocし、それらへのポインタをまとめた動的配列をmallocする
- あきらかに、長さ $m \times n$ の一次元配列にする
(サンプルプログラムではこちらを採用)
`a[i][j]` の代わりに、`a[i*n+j]`とする

二次元配列の一次元配列による表現



C言語での表現

```
int *a; a = malloc(sizeof(int)*m*n);
```



配列要素 $a_{i,j}$ を、 $a[i*n+j]$ と表す

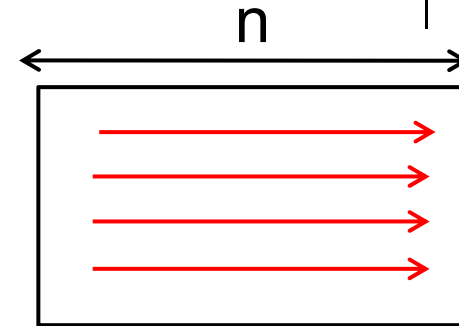
二次元データの並びの呼び方



Row major order

- C言語の二次元配列

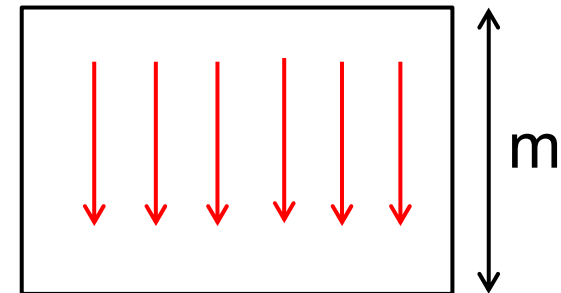
$$a_{i,j} \Rightarrow a[i*n+j]$$



Column major order

- Fortranの二次元配列
- BLASライブラリ
- Mmサンプル

$$a_{i,j} \Rightarrow a[i+j*m]$$

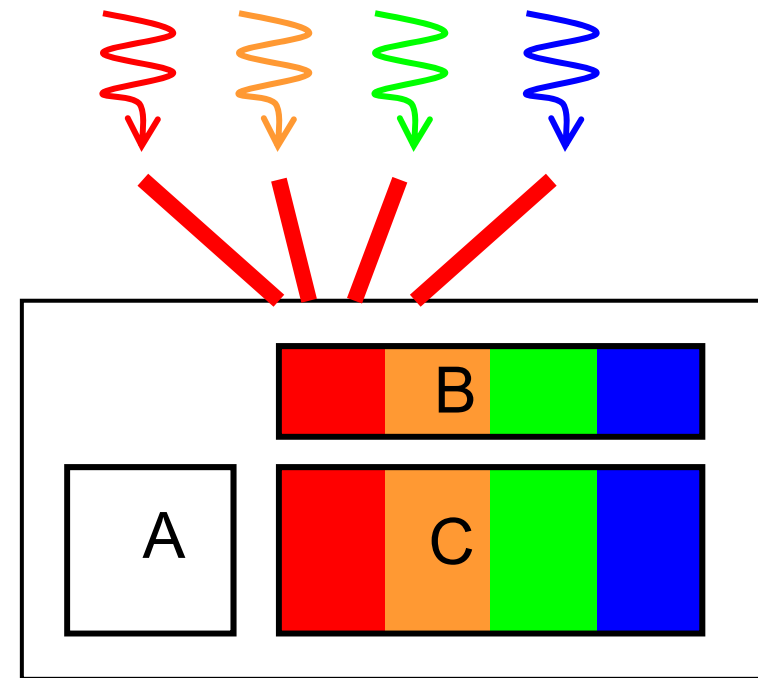


- 三次元以上でも、プログラマが並びを考える必要あり



MmのOpenMPによる並列化

- 三重ループの最外ループを並列化
 - #pragma omp parallel for
 - nをスレッド間で分割することになる



行列Aは全スレッドによってアクセスされる



変数のデータ共有属性(1)

OpenMPでは、変数はスレッドによって共有されるか否か、注意
基本的には

- 並列Region外で宣言された変数 ⇒ 共有変数
- 並列Region内で宣言された変数 ⇒ プライベート変数

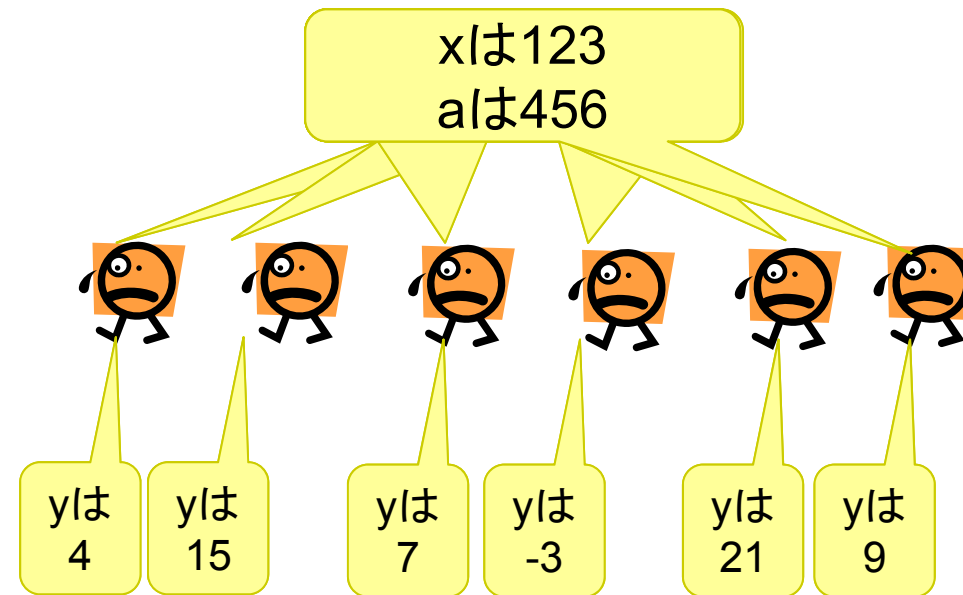
```
{  
    int s = 1000; shared  
    #pragma omp parallel  
    {  
        int i; private  
        i = func(s, omp_get_thread_num());  
        printf( "%d¥n" , i);  
    }  
}
```

```
int func(int a, int b)  
{  
    int rc = a+b; private  
    return rc;  
}
```



変数のデータ共有属性(2)

- 変数xとaが共有
- 変数yがプライベートのとき



- 共有変数を誰かが書き換えると、ほかのスレッドにも伝わる



共有/プライベートの落とし穴

- 二重以上のループがあり、並列性を持つプログラム
- 外側ループを並列化することにした
- 以下の「いかにもokそうなプログラム」はバグあり
 - コンパイルは通るのに、期待通りに動かないやっかいなケース

```
int i, j;  
#pragma parallel for  
for (i = 0; i < m; i++) {  
    for (j = 0; j < n; j++) {  
        ...  
    }  
}
```

i,jの宣言がparallelの外側にあるので共有変数。**j**がまずい



共有/プライベートの落とし穴(2)

- 修正方法

```
int i;  
#pragma parallel for  
for (i = 0; i < m; i++) {  
    int j;    // j はプライベート  
    for (j = 0; j < n; j++) {  
        ...  
    } }  

```

```
int i, j;  
#pragma parallel for private(j)  
// jは外側で宣言されているがプライベートに  
for (i = 0; i < m; i++) {  
    for (j = 0; j < n; j++) {  
        ...  
    } }  

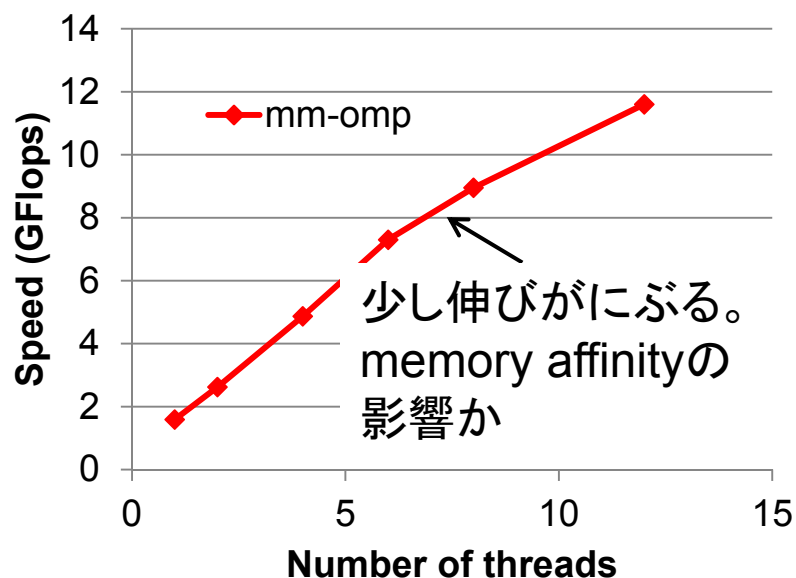
```



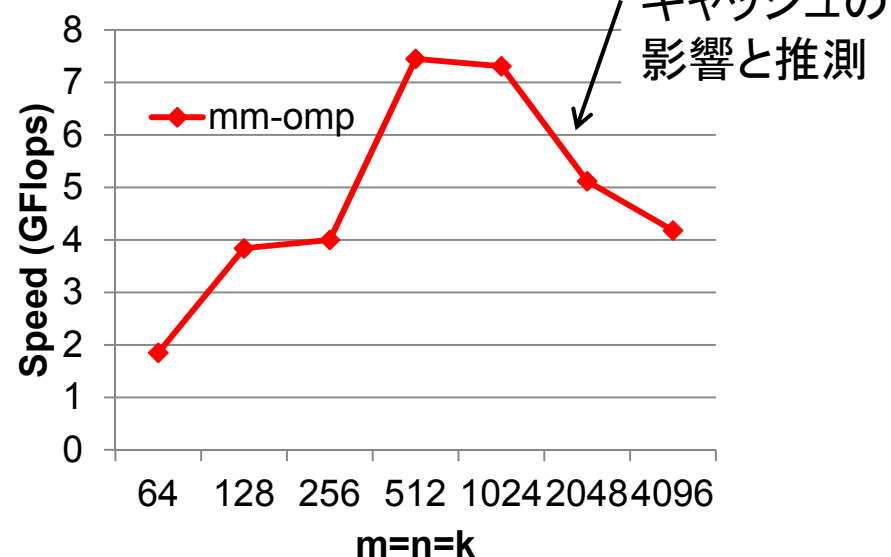
Mm-ompの性能

- TSUBAME2ノード上(Xeon X5670 2.93GHz 12core)
- OMP_NUM_THREADS環境変数によりスレッド数指定
- (2mnk/経過時間)にてFlops単位の色度を取得

m=n=k=2000固定、
スレッド数を変化



4スレッド、
m=n=kを変化

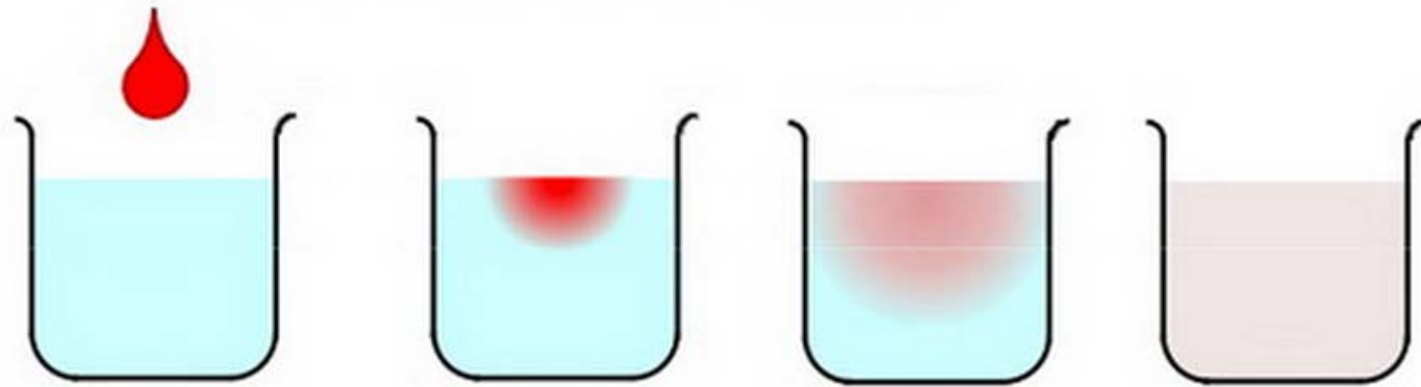




サンプルプログラム: diffusion

拡散現象

コップの中の水に赤インクを落す



次第に拡散して赤インクは拡がって行き、最後は均一な色になる

© 青木尊之

- 各点のインク濃度は、時間がたつと変わっていく → その様子を計算機で計算
 - 天気予報などにも含まれる計算



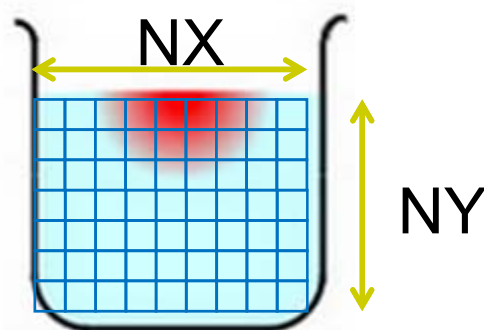
Diffusionの実行方法

- 実行方法: `./diffusion`
- `nx, ny`: 空間サイズ. `nt`: 時間ステップ数
 - サンプルでは、`nx=8192, ny=8192, nt=100`に固定
 - オプション指定などで、可変パラメータに対応できるほうがよい (mm参照)
- 計算量: $O(nx \times ny \times nt)$

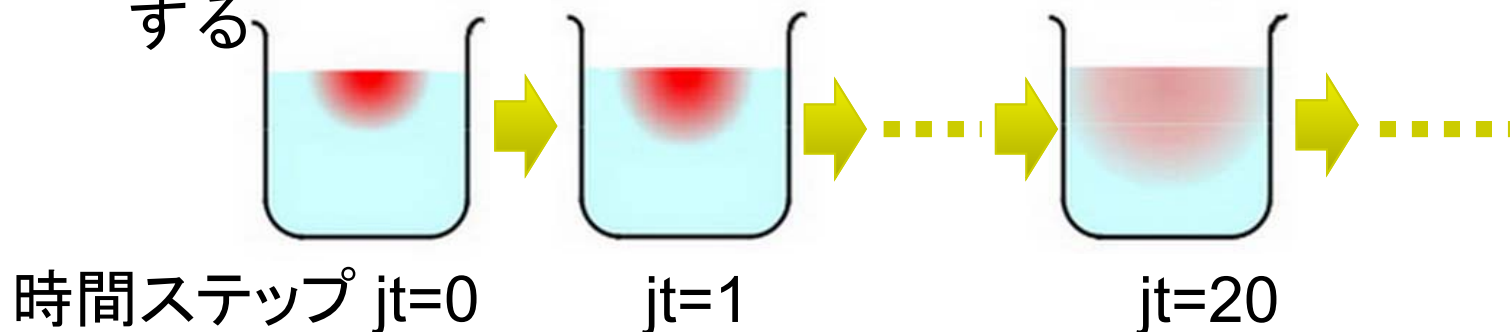


diffusionのデータ構造

- シミュレーションしたい空間をマス目で区切り、配列で表す(本プログラムでは二次元配列)



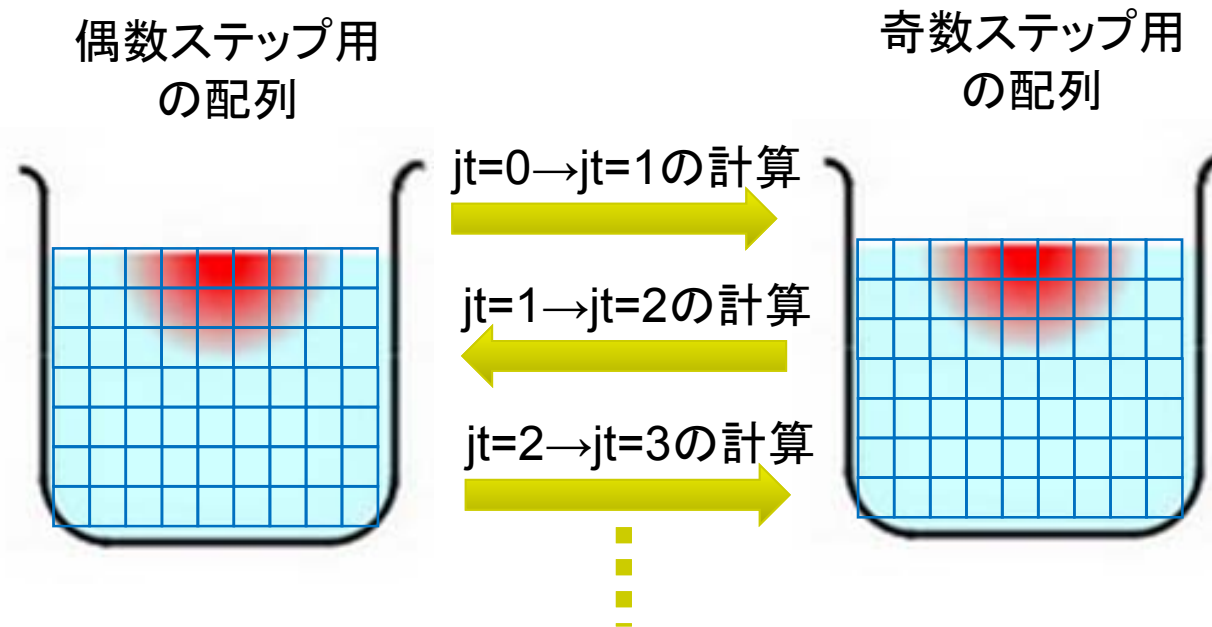
- 時間を少しずつ、パラパラ漫画のように進めながら計算する



ダブルバッファリング技術



- 全時間ステップの配列を覚えておくとメモリ容量を食い過ぎる
- ニステップ分だけ覚えておき、二つの配列(ダブルバッファ)を使いまわす

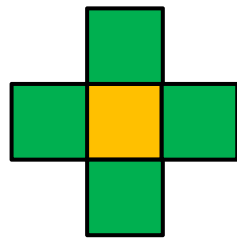


※ サンプルプログラムでは、大域変数
`float data[2][NY][NX];`
で表現

diffusionの計算: ステンシル計算



- 時間 t における点 (i,j) を計算するには？
- 時間 $t-1$ における下記を利用
 - 点 (i,j) の値
 - 点 (i,j) の近傍の値 (このサンプルでは上下左右)



時間 $t-1$



時間 t

- このタイプの演算を**ステンシル計算**と呼ぶ
- 以下が既知とする
 - 時間0の全点の温度(**初期条件**)
 - 各時間における、領域の「端」の点の温度(**境界条件**)



本来の「ステンシル」



Diffusionの並列化について

これを並列化するには??

- 空間ループをomp forで並列化が良い. 結果的に空間を分割して, スレッドたちで分担することになる.
- 時間ループにomp forをつけてはいけない! なぜか?

For指示文の補足情報: #pragma omp forが書ける条件



- 直後のfor文が「canonical form (正準形)」であること

```
#pragma omp for  
  for (var = lb; var rel-op b; incr-expr)  
    body
```

ここでincr-exprは ++var, --var, var++, var--, var+=c, var-=cなど

for (i = 0; i < n; i++) ⇒ For 指示文可能 !

for (p = head; p != NULL; p = p->next) ⇒ For 指示文不可

Canonical formであっても、プログラムの挙動の正しさは
やはりプログラマの責任

For指示文のオプション: スケジューリング



- 通常, スレッドに割り当てられる反復回数は均等分割
 - 例: 1000回を4スレッドでなら、250ずつ
- 各反復の仕事量が違うと非効率. たとえば三角行列の演算 ⇒ 様々なスケジューリング手法が用意されている

#pragma omp for **schedule(...)**

`schedule(static)`

均等分割(default)

`schedule(static, n)`

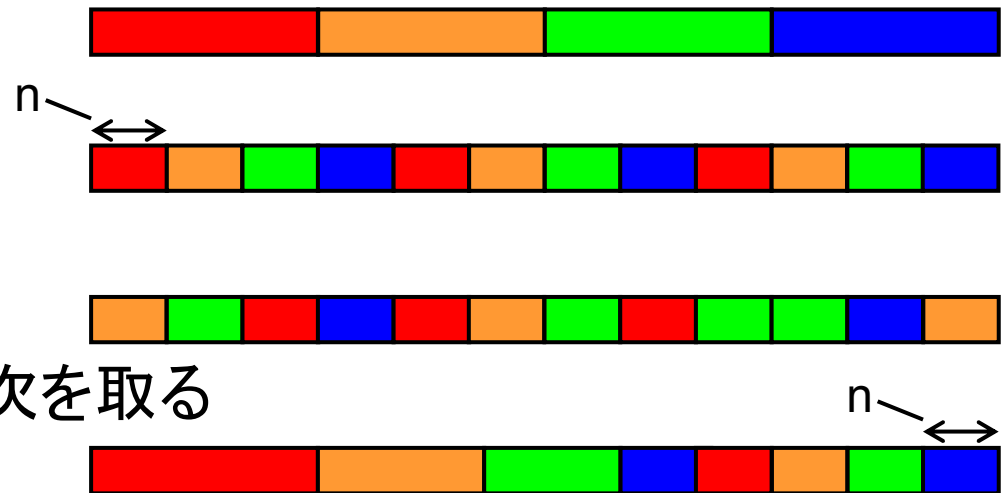
ブロックサイクリック分割

`schedule(dynamic, n)`

仕事が終わったスレッドが次を取る

`schedule(guided, n)`

“chunk size” が等比的に小さく



参考: OpenMPとpthreadの比較



- OpenMPでは
 - 原則的にスレッド数は一定
- Pthreadでは
 - 好きなときにスレッドの生成・終了
 - スレッド内の局所変数はスレッドプライベート

	OpenMP	pthread
クリティカルセクション	あり	あり(pthread_mutex_lockなど)
バリア同期	あり	なし
条件変数による同期 (パイプライン並列など)	なし	あり(pthread_cond_waitなど)
タスク並列	あり(限定的)	なし(明示的に記述)
ワークシェアリング構文	あり (omp for)	なし



本授業のレポートについて

- 各パートで課題を出す。2つ以上のパートのレポート提出を必須とする

予定パート:

- OpenMPパート
- MPIパート
- GPUパート



OpenMPパート課題説明 (1)

以下の[O1]—[O2]のどれか一つについてレポートを提出してください

[O1] diffusionサンプルプログラムを、OpenMPで並列化してください。

オプション:

- 配列サイズや時間ステップ数を可変パラメータにしてみる。引数で受け取って、配列をmallocで確保するようにする、など。
- より良いアルゴリズムにしてみる。ブロック化・計算順序変更でキャッシュミスが減らせないか？



OpenMPパート課題説明 (2)

[O2] 自由課題: 任意のプログラムを, OpenMPを用いて並列化してください.

- 単純な並列化で済む問題ではないことが望ましい
 - スレッド・プロセス間に依存関係がある
 - 均等分割ではうまくいかない、など
- たとえば, 過去のSuperConの本選問題
<http://www.gsic.titech.ac.jp/supercon/>
たんぱく質類似度(2003), N体問題(2001)・・・
入力データは自分で作る必要あり
- たとえば, 自分が研究している問題



課題の注意

- いずれの課題の場合も、レポートに以下を含むこと
 - 計算・データの割り当て手法の説明
 - TSUBAME2などで実行したときの性能
 - プロセッサ(コア)数を様々に変化させたとき. 大規模のほうがよい. XXコア以上で発生する問題に触れているとなお良い
 - 問題サイズを様々に変化させたとき(可能な問題なら)
 - 高性能化のための工夫が含まれているとなお良い
 - 「XXXのためにXXXを試みたが高速にならなかった」のような失敗でも可
 - 作成したプログラムについても、zipなどで圧縮して添付
 - 困難な場合、TSUBAME2の自分のホームディレクトリに置き、置き場所を連絡



課題の提出について

- OpenMPパート提出期限
 - 5/26 (月)
- OCW-i ウェブページから下記ファイルを提出のこと
- レポート形式
 - 本文: PDF, Word, テキストファイルのいずれか
 - プログラム: zip形式に圧縮するのがのぞましい
- OCW-iからの提出が困難な場合、メールでもok
 - 送り先: ppcomp@el.gsic.titech.ac.jp
 - メール題名: ppcomp report



次回: 5/7(水)

- OpenMP(3)
 - スケジュールについてはOCW pageも参照
 - <http://www.el.gsic.titech.ac.jp/~endo/>
→ 2014年度前期情報(OCW) → 講義ノート