筑波大学計算科学研究センター CCS HPCサマーセミナー 「並列数値計算アルゴリズム」

高橋大介
daisuke@cs.tsukuba.ac.jp
筑波大学大学院システム情報工学研究科
計算科学研究センター

講義内容

- 高速フーリエ変換（Fast Fourier Transform, FFT）の並列アルゴリズム
- 共役勾配法（Conjugate Gradient Method, CG法)の並列アルゴリズム

高速フーリエ変換（FFT）

- 高速フーリエ変換（FFT）は，離散フーリエ変換（DFT）を高速に計算するアルゴリズム.
- 理学分野での応用例
  - 偏微分方程式の解法
  - 畳み込み，相関の計算
  - 第一原理計算における密度汎関数法
- 工学分野での応用例
  - スペクトラムアナライザ
  - CTスキャナ・MRIなどの画像処理
  - 地上波デジタルテレビ放送や無線LANで用いられているOFDM（直交周波数多重変調）では，変復調処理にFFTを用いている.

離散フーリエ変換（DFT）

- 離散フーリエ変換（DFT）の定義

$$y(k) = \sum_{j=0}^{n-1} x(j)\omega_n^{jk}$$

$$0 \le k \le n-1, \quad \omega_n = e^{-2\pi i/n}$$

行列によるDFTの定式化（1/4）

- $n = 4$ のとき，DFTは以下のように計算できる.

$$y(0) = x(0)\omega^0 + x(1)\omega^0 + x(2)\omega^0 + x(3)\omega^0$$
$$y(1) = x(0)\omega^0 + x(1)\omega^1 + x(2)\omega^2 + x(3)\omega^3$$
$$y(2) = x(0)\omega^0 + x(1)\omega^2 + x(2)\omega^4 + x(3)\omega^6$$
$$y(3) = x(0)\omega^0 + x(1)\omega^3 + x(2)\omega^6 + x(3)\omega^9$$

行列によるDFTの定式化（2/4）

- 行列を用いると，より簡単に表すことができる.

$$\begin{bmatrix} y(0) \\ y(1) \\ y(2) \\ y(3) \end{bmatrix} = \begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \omega^3 \\ \omega^0 & \omega^2 & \omega^4 & \omega^6 \\ \omega^0 & \omega^3 & \omega^6 & \omega^9 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix}$$

- n^2 回の複素数の乗算と，$n(n-1)$ 回の複素数の加算が必要.

行列によるDFTの定式化（3/4）

- $\omega_n^{jk} = \omega_n^{jk \bmod n}$ の関係を用いると，以下のように書き直すことができる．

$$\begin{bmatrix} y(0) \\ y(1) \\ y(2) \\ y(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 \\ 1 & \omega^2 & \omega^0 & \omega^2 \\ 1 & \omega^3 & \omega^2 & \omega^1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix}$$

行列によるDFTの定式化（4/4）

- 行列の分解により，乗算回数を減らすことができる．

$$\begin{bmatrix} y(0) \\ y(2) \\ y(1) \\ y(3) \end{bmatrix} = \begin{bmatrix} 1 & \omega^0 & 0 & 0 \\ 1 & \omega^2 & 0 & 0 \\ 0 & 0 & 1 & \omega^1 \\ 0 & 0 & 1 & \omega^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & \omega^0 & 0 \\ 0 & 1 & 0 & \omega^0 \\ 1 & 0 & \omega^2 & 0 \\ 0 & 1 & 0 & \omega^2 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix}$$

これを再帰的に行うと，演算量を $O(n \log n)$ にできる（データ数 n は合成数である必要がある）

DFTとFFTの演算量の比較

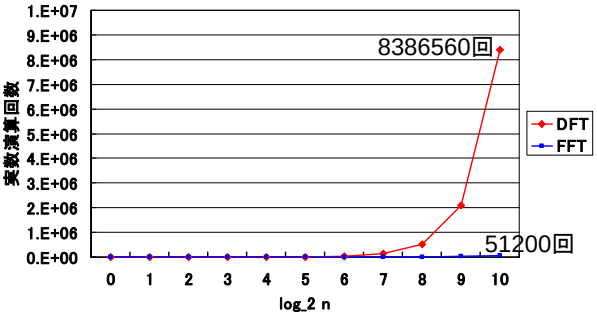
- DFTの実数演算回数

$$T_{DFT} = 8n^2 - 2n$$

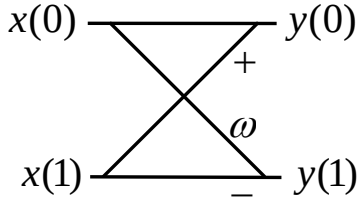
- FFTの実数演算回数
（n が2のべきの場合）

$$T_{FFT} = 5n \log_2 n$$

DFTとFFTの演算量の比較

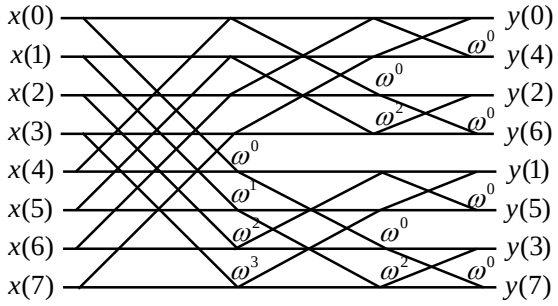


バタフライ演算

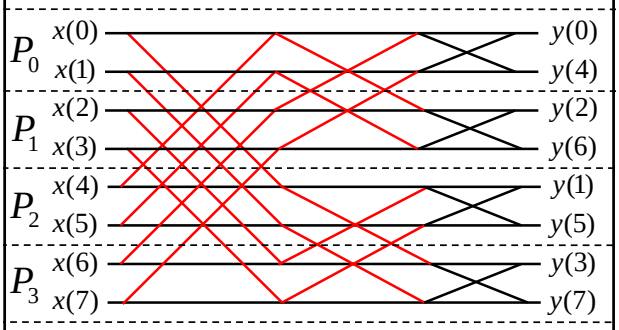


$$y(0) = x(0) + x(1)$$
$$y(1) = \omega\{x(0) - x(1)\}$$

Cooley-Tukey FFTの信号流れ図



Cooley-Tukey FFTの並列化



並列Cooley-Tukey FFTの通信量

- ノード数を P とすると，並列Cooley−Tukey FFTでは，$\log_2 P$ ステージの通信が必要になる．
- 各ステージでは n/P 個の倍精度複素数データの通信（MPI_Send，MPI_Recv）が行われるため，合計の通信量は，

$$T_{Cooley-Tukey} = \frac{16n}{P}\log_2 P \quad \text{（バイト）}$$

$n = n_1 n_2$ に対するFFTアルゴリズム

- $n = n_1 n_2$ で与えられるとする．

$$j = j_1 + j_2 n_1 \quad j_1 = 0, 1, \cdots, n_1 - 1 \quad j_2 = 0, 1, \cdots, n_2 - 1$$
$$k = k_2 + k_1 n_2 \quad k_1 = 0, 1, \cdots, n_1 - 1 \quad k_2 = 0, 1, \cdots, n_2 - 1$$

- 上記の表現を用いると，DFTの定義式を以下のように書き換えることができる．

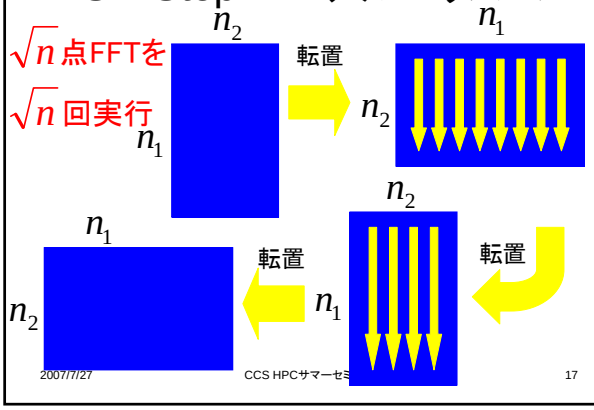
$$y(k_2, k_1) = \sum_{j_1=0}^{n_1-1}\left[\sum_{j_2=0}^{n_2-1} x(j_1, j_2)\omega_{n_2}^{j_2 k_2}\omega_{n_1 n_2}^{j_1 k_2}\right]\omega_{n_1}^{j_1 k_1}$$

- n 点FFTを n_1 点FFTと n_2 点FFTに分解している．

Six-Step FFTアルゴリズム

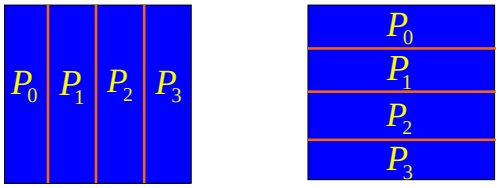
1. 行列の転置
2. n_2 組の n_1 点 multicolumn FFT
3. ひねり係数（$\omega_{n_1 n_2}^{j_1 k_2}$）の乗算
4. 行列の転置
5. n_1 組の n_2 点 multicolumn FFT
6. 行列の転置

Six-Step FFTアルゴリズム



配列の分割方法（1/4）

- MPIで並列化を行う際には，各ノードで配列を分割して持つようにすると，メモリを節約することができる．
- ブロック分割
  - 連続する領域をノード数で分割

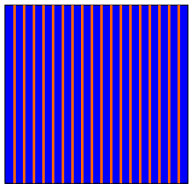


列方向に分割したブロック分割　行方向に分割したブロック分割

配列の分割方法(2/4)

- サイクリック分割
  - 1列(または1行)ごとに分割
  - ブロック分割に比べてロードバランスが取りやすい

0123012301230123…

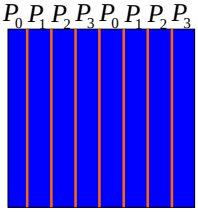


列方向に分割したサイクリック分割

配列の分割方法(3/4)

- ブロックサイクリック分割
  - 複数列(または複数行)ごとに分割
  - ブロック分割とサイクリック分割の中間

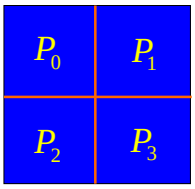
$P_0\ P_1\ P_2\ P_3\ P_0\ P_1\ P_2\ P_3$



列方向に分割したブロックサイクリック分割

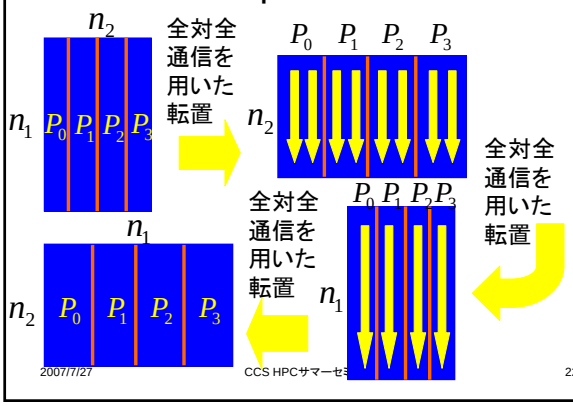
配列の分割方法(4/4)

- 二次元分割
  - 行方向と列方向の両方を分割
  - 一次元分割よりも通信量が減ることがある
  - 二次元のブロック分割，サイクリック分割，ブロックサイクリック分割が考えられる．



二次元ブロック分割

並列Six-Step FFTアルゴリズム



全対全通信(MPI_Alltoall)を用いた行列の転置

並列Six-Step FFTの通信量

- ノード数を P とすると，並列Six-Step FFTでは，3回の全対全通信が必要になる．
- 全対全通信では，各ノードは n/P^2 個の倍精度複素数データを，自分以外の $P-1$ ノードに送ることになるため，合計の通信量は

$$T_{Six-Step} = 3\cdot(P-1)\cdot\frac{16n}{P^2}\ \text{(バイト)}$$

並列Cooley-Tukey FFTと並列Six-Step FFTの通信量の比較

- 並列Cooley-Tukey FFTの通信量

$$T_{Cooley-Tukey} = \frac{16n}{P}\log_2 P$$

- 並列Six-Step FFTの通信量

$$T_{Six-Step} = 3\cdot(P-1)\cdot\frac{16n}{P^2}$$

- 両者を比較すると，$P>8$ の場合には，並列Six-Step FFTの方が通信量が少なくなる．

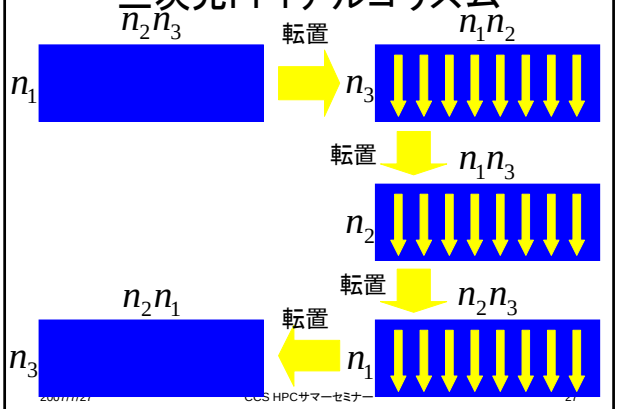
三次元FFT

- 三次元離散フーリエ変換（DFT）の定義

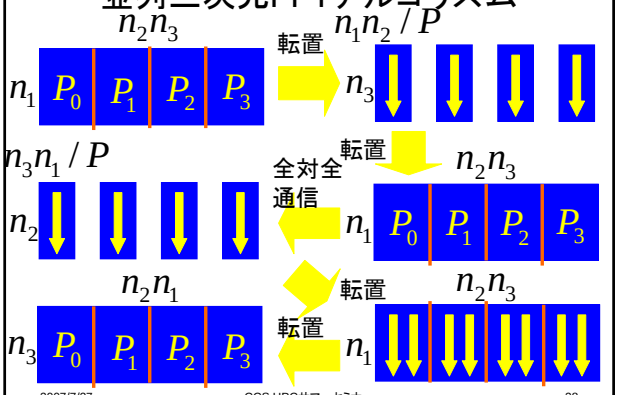
$$y(k_1,k_2,k_3)=\sum_{j_1=0}^{n_1-1}\sum_{j_2=0}^{n_2-1}\sum_{j_3=0}^{n_3-1}x(j_1,j_2,j_3)\,\omega_{n_3}^{j_3k_3}\omega_{n_2}^{j_2k_2}\omega_{n_1}^{j_1k_1}$$

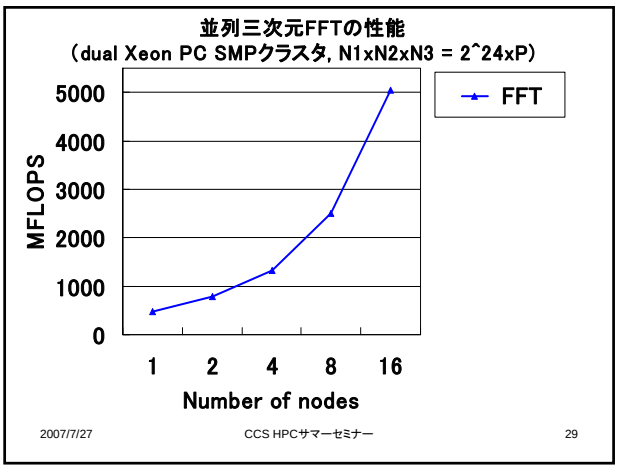
$$0\le k_r\le n_r-1,\ \ \omega_{n_r}=\exp(-2\pi i/n_r)$$

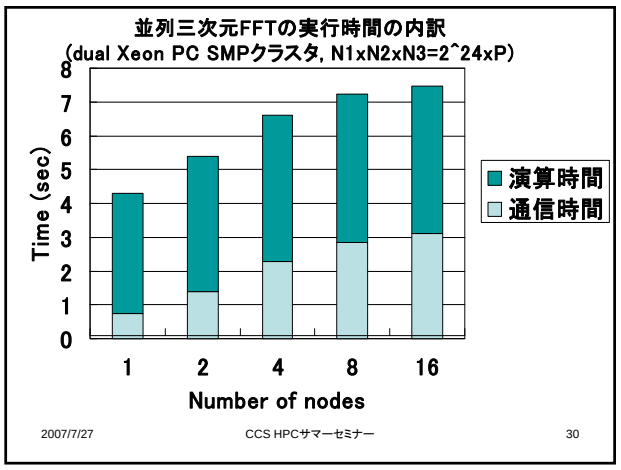
三次元FFTアルゴリズム



並列三次元FFTアルゴリズム







連立一次方程式の解法

- 科学技術計算の多くは連立一次方程式

$$A\mathbf{x} = \mathbf{b}$$

を解くことに帰着される.
- 大規模な連立一次方程式を解く際には，多大な時間を要することから，これまでにさまざまな解法が提案されている.
- 並列処理は，実行時間を短縮する上で必要不可欠.

連立一次方程式の解法

- 連立一次方程式の解法には大きく分けて，
  - 直接法（ガウスの消去法，LU分解など）
  - 反復法（ガウス・ザイデル法，ヤコビ法，共役勾配法など）

  がある.
- 今回は反復法である共役勾配法を取り上げる.

共役勾配法（CG法）

- 連立一次方程式の係数行列が対称正定値な行列 A（任意のベクトル$\mathbf{x}$ に対して $\mathbf{x}^T A\mathbf{x} > 0$ ）に対しては

$$A\mathbf{x} = \mathbf{b}$$

の解が

$$f(\mathbf{x}) = \frac{1}{2}(\mathbf{x}, A\mathbf{x}) - (\mathbf{x}, \mathbf{b})$$

という関数を最小にするという性質を利用したのが共役勾配法（Conjugate Gradient Method，CG法）である.

共役勾配法（CG法）のアルゴリズム

初期ベクトル$\mathbf{x}_0$を用意する

$\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$，　$\mathbf{p}_0 = \mathbf{r}_0$

for k = 0, 1, …

$\alpha_k = \dfrac{(\mathbf{p}_k, \mathbf{r}_k)}{(\mathbf{p}_k, A\mathbf{p}_k)}$ ・内積（ddot），行列ベクトル積（dgemv）

$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$ ・ベクトルの定数倍とベクトルの和（daxpy）

$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k A\mathbf{p}_k$ ・行列ベクトル積（dgemv），ベクトルの定数倍とベクトルの和（daxpy）

収束判定

$\beta_k = \dfrac{(\mathbf{r}_{k+1}, A\mathbf{p}_k)}{(\mathbf{p}_k, A\mathbf{p}_k)}$ ・行列ベクトル積（dgemv）と内積（ddot）

$\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$ ・ベクトルの定数倍（dscal）とベクトルの和

end

共役勾配法（CG法）に必要な計算

- ベクトルの内積（ddot）
- 行列ベクトル積（dgemv）
- ベクトルの定数倍とベクトルの和（daxpy）
- ベクトルの定数倍（dscal）
- 上記のルーチンは自分で書いてもよいが，よく使われるので，ライブラリになっている.

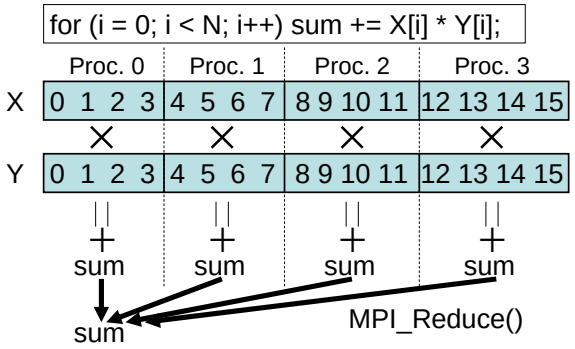
BLASとは

- BLAS（Basic Linear Algebra Subprograms）は，ベクトルと行列の基本演算を行うサブルーチン群.
- 呼び出し方法が統一されているので，異機種間での移植性に優れている.
- 各プロセッサに高度に最適化されており，高い性能を発揮する.
  - Intel MKL，AMD ACML，IBM ESSL，GOTO BLAS

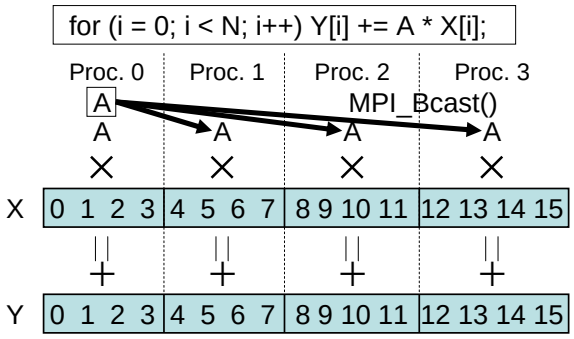
BLASの種類

- BLASには，大きく分けてLevel 1，Level 2，Level 3の3種類がある．
- Level 1（ベクトルーベクトル演算）
  - dot（内積），axpy（ベクトルの定数倍とベクトルの和）など
- Level 2（行列ーベクトル演算）
  - gemv（行列ベクトル積）など
- Level3（行列ー行列演算）
  - gemm（行列積）など
- サブルーチンの名前の前にs（単精度実数），d（倍精度実数），c（単精度複素数），z（倍精度複素数）が付く．

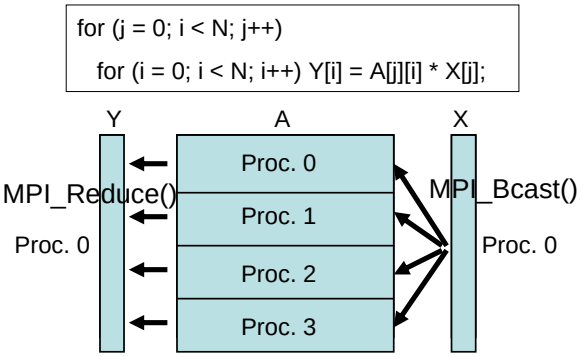
内積の並列化



daxpyの並列化



行列ベクトル積の並列化

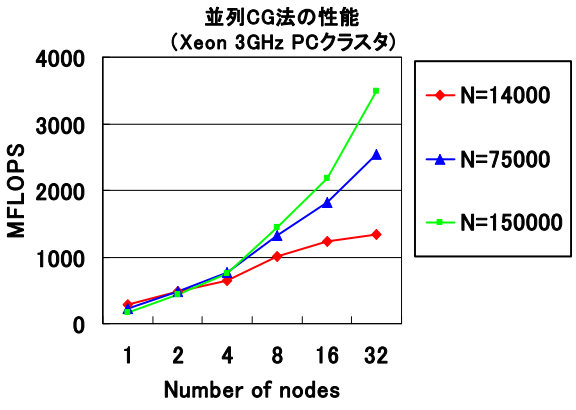


並列CG法のプログラム（一部）

```
do j=1,lastrow-firstrow+1
   sum = 0.0d0
   do k=rowstr(j),rowstr(j+1)-1
      sum = sum + a(k)*p(colidx(k))
   enddo
   w(j) = sum
end do

do i = l2npcols, 1, -1
   call mpi_irecv(…)
   call mpi_send(…)
   call mpi_wait(…)
   do j=send_start,send_start + reduce_recv_lengths(i) - 1
      w(j) = w(j) + q(j)
   end do
 end do
```

疎行列の非ゼロ要素だけを一次元配列に格納



まとめ

- 並列数値計算アルゴリズムとして，
  - FFT（高速フーリエ変換）
  - CG法による連立一次方程式の解法

  を取り上げた．
- 問題の領域をどのように分割するかが鍵となる．
  - ブロック分割，サイクリック分割，ブロックサイクリック分割
- 科学技術計算には並列性を含んだものが多いので，並列化はそれほど難しくない（ものもある）．
  - 逐次ルーチンとMPIの集団通信ルーチンを使えば，かなりの部分は記述できる．