JAERI-Data/Code
97-038

X線結晶構造解析プログラムの並列化

1997年10月

渡部　弘・南　正雪*・山本昭二**

日本原子力研究所
Japan Atomic Energy Research Institute

本レポートは，日本原子力研究所が不定期に公刊している研究報告書です。
入手の問合わせは，日本原子力研究所研究情報部研究情報課（〒319-11 茨城県那珂郡東海村）あて，お申し越しください。なお，このほかに財団法人原子力弘済会資料センター（〒319-11 茨城県那珂郡東海村日本原子力研究所内）で複写による実費頒布をおこなっております。

This report is issued irregularly.
Inquiries about availability of the reports should be addressed to Research Information Division, Department of Intellectual Resources, Japan Atomic Energy Research Institute, Tokai-mura, Naka-gun, Ibaraki-ken, 319-11, Japan.

編集兼発行　日本原子力研究所
印　　刷　いばらき印刷㈱

JAERI-Data/Code 97-038

X線結晶構造解析プログラムの並列化

日本原子力研究所計算科学技術推進センター
渡部　弘・南　正雪*・山本　昭二**

(1997年9月8日受理)

X線結晶構造解析プログラムのベクトル化及び並列化について報告する。ベクトル化ではベクトル長の長い多次元離散フーリエ変換ルーチンを開発導入し，オリジナル版の12.0倍の高速化を達成した。さらに上記ルーチンをＳＸ－４のマイクロタスク機能により，共有メモリ内での並列化を行った。その結果，多次元離散フーリエ変換ルーチン部分は14並列で13.7倍，全体でも3.0倍の高速化が図られた。オリジナルであるスカラ版ＩＣＰＵによるプログラム経過時間と14並列版のプロラム経過時間を比べると，35.9倍の高速化に成功している。

日本原子力研究所：〒153　東京都目黒区中目黒２－２－54
*　株式会社ＮＥＣ情報システムズ
**　無機材質研究所

JAERI-Data/Code 97-038

Parallelization for X-ray Crystal Structural Analysis Program

Hiroshi WATANABE, Masayuki MINAMI* and Akiji YAMAMOTO**

Center for Promotion of Computational Science and Engineering
Japan Atomic Energy Research Institute
Nakameguro, Meguro-ku, Tokyo

(Received September 8, 1997)

In this report we study vectorization and parallelization for X-ray crystal structural analysis program. The target machine is NEC SX-4 which is a distributed/ shared memory type vector parallel supercomputer. X-ray crystal structural analysis is surveyed, and a new multi-dimensional discrete Fourier transform method is proposed. The new method is designed to have a very long vector length, so that it enables to obtain the 12.0 times higher performance result that the original code. Basides the above-mentioned vectorization, the parallelization by micro-task functions on SX-4 reaches 13.7 times acceleration in the part of multi-dimensional discrete Fourier transform with 14 CPUs, and 3.0 times acceleration in the whole program. Totally 35.9 times acceleration to the original 1CPU scalar version is achieved with vectorization and parallelization on SX-4.

Keywords: X-ray, Crystal Structural Anlysis, Vectorization, Parallelization, SX-4

---

* NEC Informatec Systems, Ltd.

** National Institute for Research in Inorganic Materials

目　　次

Contents

1 はじめに

研究開発分野における大規模数値計算の需要は止まる所を知らない。特に近年の研究は人工的な仮定を排し、より根元的な原理から現象を理解しようとする方向に進んでおり、そのため、より高速で記憶容量の大きいコンピュータが強く望まれている。例としてはプラズマ粒子シミュレーション、物性研究における第一原理計算や分子動力学的手法、流体計算における直接数値シミュレーション (DNS) 等が挙げられる。これらの手法は簡明な基本原理に忠実なアルゴリズムに基づいており、従来の人為的なモデル化や経験値を必要とする手法よりも、はるかに高精度で信頼性の高い結果を与えている。一般にこれら新手法は、従来の手法よりアルゴリズムは簡単であるが、そのかわり必要とされる計算量及び記憶容量は膨大になる。そのためこれらの分野では、強力な数値計算能力を有するスーパーコンピュータの利用が必要不可欠であり、計算機のパワーが研究の進展を大きく左右するといっても過言ではない。

これらの需要に応えるためコンピュータメーカ各社は、最新のテクノロジーによるスーパーコンピュータの開発に凌ぎを削っている。特に近年では、単一 CPU による性能向上に限界が見えはじめてきたことから、開発の重心は複数の CPU を用いた並列計算機に移っており、各メーカが各々特色のある並列計算機を商用化している。また各研究現場でも並列計算機の性能向上や低価格化に伴い、積極的に並列計算機を導入しつつある。

このように" 並列計算機" はかなり身近なものになってきたが、しかし" 並列計算" 自体についてはどうであろうか。残念ながら、一般の研究者が強力な並列計算機のパワーを充分に活用し、自分の研究に役立てているとは言い難い。その理由は既存の逐次型プログラムを並列化することが、想像以上に困難だからである。かつてベクトル型スーパーコンピュータが広くユーザに受け入れられたのは、その性能もさることながら、自動ベクトル化コンパイラにより比較的容易に既存プログラムを高速化できたことが大きい。一方、並列計算における自動並列化技術は未だ発展途上にあり、並列化により大幅な性能向上を達成するにはユーザの努力が必要不可欠である。

以上のような状況で重要なのは、研究開発分野で用いられている実用プログラムを実際に並列化しながら、並列化の具体的な手法やノウハウを蓄積することであり、さらにそれらを広く普及させることである。

このような目的の下、日本原子力研究所と無機材質研究所は平成 7 年 10 月から共同研究を行い、実際に研究で用いられている各種プログラムの並列化を行ってきた。本報告は上記共同研究の一環であり、MEM によるＸ線結晶構造解析プログラムの並列化に関するものである。ターゲットマシンは比較的並列化が容易なこと、巨大な配列を確保できること、さらには筆者等のアクセスのしやすさを考慮して、ＮＥＣ製共有／分散メモリ型ベクトル並列マシンＳＸ－４とした。

本報告書の構成は以下の通りである。第 2 章はＸ線回折による結晶構造解析の原理を述べる。第 3 章ではＸ線回折観測データから結晶構造を統計的に推定する手法として、MEM(Maximum Entropy Method:最大エントロピー法) を紹介し、あわせてＸ線結晶構造解析プログラムのアルゴリズムを示す。第 4 章ではプログラムの中で最も計算時間コストの大きい多次元離散フーリエ変換部分について説明し、高速化のための方法を論じる。第 5 章で新規に開発したベクトル化多次元離散フーリエ変換ルーチンを導入した結果を示した後、第 6 章で同ルーチンの並列化について述べる。

以上の一連の作業の結果、並列化プログラムはオリジナルのプログラムに比べ、最大 35.9 倍の高速化を達成している。

2 X線結晶構造解析

X線結晶構造解析とは、物質によるX線の回折現象を利用して、原子、分子、イオン、電子の分布状態を研究する手法である。対象は固体、液体、気体を問わないが、結晶に対し特に有力であり、固体物理学、分子科学、分子生物学、薬学、鉱物学、金属学等の物質科学の広い分野で、欠くことのできない研究手段となっている。ここでは本研究で使用されたプログラムの内容を理解するために、必要最小限のX線結晶構造解析の原理を [1] に従って説明する。散乱対象は電子である。

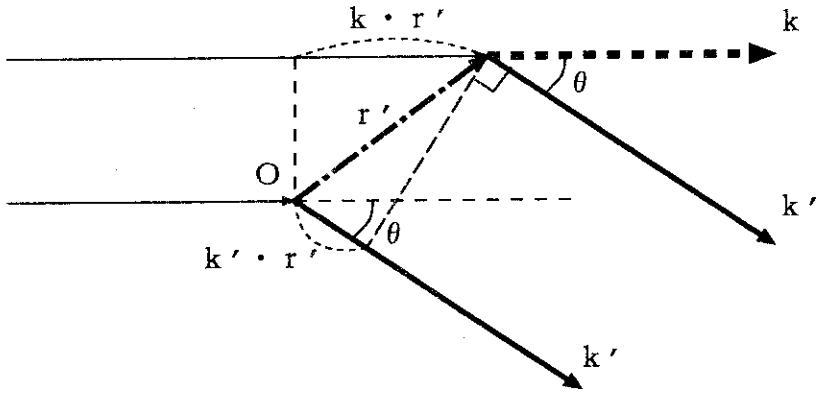

図 1: 原点及び原点から r' の位置に存在する電子による散乱

入射波を $u_0(r) = A_0 e^{ik\cdot r}$ で表される平面波とすると、原点 $r = 0$ にある電子による散乱は、散乱角 θ、原点から充分遠方の距離 $R = |r|$ の地点で次のように与えられる (図 1 参照)。

$$u_{s0}(r) = A_0 \frac{\varphi(\theta)}{R} e^{iKR} \qquad (K = |k|) \tag{1}$$

これは原点を中心とする球面波である。$\varphi(\theta)$ は入射X線の偏りベクトル e から、次式で計算できることが知られている。

$$\varphi(\theta) = \frac{|e|^2}{4\pi\epsilon_0 mc^2} (e \times r/|R|) \tag{2}$$

原点から r'の位置に存在する電子による散乱波 $u_s(r)$ は、図 1 からわかるように、

$$k' \cdot r' - k \cdot r' = b \cdot r' \qquad (b = k' - k) \tag{3}$$

だけ位相差があるので、

$$u_s(r) = A_0 \frac{\varphi(\theta)}{R} e^{iKR - ib\cdot r'} \tag{4}$$

となる。電子が多数存在する場合は、電子密度分布を $\rho_e(r')$ として

$$u_s(r) = A_0 \frac{\varphi(\theta)}{R} e^{iKR} \int_{space} \rho_e(r') e^{-ib\cdot r'} dr' \tag{5}$$

が得られる。ここで構造因子 $F(b)$ を

$$F(b) = \int_{space} \rho_e(r) e^{ib\cdot r} dr \tag{6}$$

と定義する。すると散乱波は $u_s(r)$ は

$$u_s(r) = A_0 \frac{\varphi(\theta)}{R} e^{iKR} F(-b) \tag{7}$$

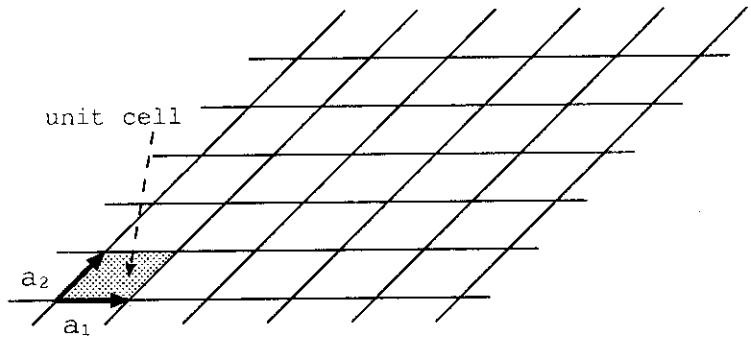

図 2: 結晶構造内の単位胞と基本ベクトル (2 次元の場合)

と表される。X線回折により観測されるのは $u_s(r)$ なので、構造因子 $F(b)$ が観測される物理量と考えてよい (正確には $|F(b)|$)。

ここまでは一般論であるが、結晶の場合はさらに空間的な周期構造を持つことから、より詳しい議論が可能である。結晶構造は 3 つの基本ベクトル a_1, a_2, a_3を 3 辺とする平行 6 面体の単位胞 (unit cell) を各々の方向に並べた構造を持つ (図 2 参照)。

ある平行 6 面体の 1 頂点を原点にとると、他の頂点の位置は整数 n_1, n_2, n_3を用いて、$n_1a_1 + n_2a_2 + n_3a_3$ と表される。従って結晶全体の構造因子は

$$F(b) = F_c(b) \sum_{(n_1,n_2,n_3)} exp(ib \cdot (n_1a_1 + n_2a_2 + n_3a_3)) \tag{8}$$

となる。ただし $F_c(b)$ は結晶構造因子と呼ばれ、次式で与えられる。

$$F_c(b) = \int_{unit\ cell} \rho_e(r) e^{ib \cdot r} dr \tag{9}$$

結晶が充分小さければ、X線の散乱強度の角分布 $|F(b)|^2$は

$$|F(b)|^2 = |F_c(b)|^2 \mid \sum_{(n_1,n_2,n_3)} exp(ib \cdot (n_1a_1 + n_2a_2 + n_3a_3)) \mid^2 = |F_c(b)|^2 L(b) \tag{10}$$

で決定できる。ここで $L(b)$ はラウエの回折関数と呼ばれ、$b \cdot a_1, b \cdot a_2, b \cdot a_3$の全てが 2πの整数倍の時のみ大きな値をもち、それ以外の時はほとんど 0 であるような関数である。ここで逆格子ベクトル b_1, b_2, b_3を

$$b_1 = 2\pi \frac{a_2 \times a_3}{a_1 \cdot (a_2 \times a_3)} \quad b_2 = 2\pi \frac{a_3 \times a_1}{a_2 \cdot (a_3 \times a_1)} \quad b_3 = 2\pi \frac{a_1 \times a_2}{a_3 \cdot (a_1 \times a_2)} \tag{11}$$

で定義すると、容易にわかるように

$$\begin{aligned} a_i \cdot b_j &= 2\pi \quad (i = j) \\ &= 0 \quad (i \neq j) \end{aligned} \tag{12}$$

である。従って h, k, l を整数とすると、bが

$$b = hb_1 + kb_2 + lb_3 \tag{13}$$

の場合のみ、散乱強度は大きな値となることがわかる。

電子密度分布$\rho_e(r)$ は周期ベクトル (a_1, a_2, a_3) を持つ周期関数であり、フーリエ級数展開が可能である。周期ベクトル (a_1, a_2, a_3) を持つ最も基本的な関数は、$exp(-i(hb_1 + kb_2 + lb_3) \cdot r)$ である。

$$exp(-i(hb_1 + kb_2 + lb_3) \cdot (r + a_i)) = exp(-i(hb_1 + kb_2 + lb_3) \cdot r) \quad (i = 1, 2, 3) \tag{14}$$

従ってこの関数を基底として、電子密度分布を級数展開できる。

$$\rho_e(r) = \sum_{(h,k,l)} C_{hkl}\, exp(-i(hb_1 + kb_2 + lb_3)\cdot r) \tag{15}$$

このフーリエ係数C_{hkl}を求めよう。(15)式を(9)式に代入すると

$$F_c(b) = \sum_{(h,k,l)} C_{hkl} \int_{unit\ cell} exp(i(b - (hb_1 + kb_2 + lb_3))\cdot r)dr \tag{16}$$

となる。特に$b = h'b_1 + k'b_2 + l'b_3$とすると

$$F_c(h'b_1 + k'b_2 + l'b_3) = C_{h'k'l'}\ v_c \tag{17}$$

が得られる。ただしv_cは単位胞の体積である。これから電子密度分布は

$$\rho_e(r) = \frac{1}{v_c} \sum_{(h,k,l)} F_c(hb_1 + kb_2 + lb_3) e^{-i(hb_1+kb_2+lb_3)\cdot r} \tag{18}$$

となる。すなわち結晶内の周期的な電子密度分布のフーリエ係数は、結晶構造因子を単位胞体積で割ったものに等しい。このことから多数の結晶構造因子を測定することにより、結晶内の電子密度分布が推定できることがわかる。

上式の逆変換はaを格子定数として

$$F_c(hb_1 + kb_2 + lb_3) = a \sum_r \rho_e(r) e^{i(hb_1+kb_2+lb_3)\cdot r} \tag{19}$$

で与えられる。

3 最大エントロピー法

(18) 式に従って結晶内の電子密度分布を求めようとすると、無限個の正確な結晶構造因子の測定が必要となる。しかし実際に測定できるのは有限個の誤差を含んだ結晶構造因子であり、そのため何らかの統計的処理が必要となる。本研究で用いられるプログラムでは、最大エントロピー法 (Maximum Entropy Method : MEM) により、精度の高い電子密度分布の推定を可能にしている。本章では MEM がどのように結晶構造解析に適用されるかを説明する ([2],[3],[4],[5])。

MEM([6]) は 1967 年地震波解析に関して Burg により提案され、その後

- 少ないデータからも推定が可能
- 推定精度が極めて高い

という特徴から、強力な統計的解析手法として急速に広範な分野に浸透した。

MEM とは" 情報エントロピーを最大にするように真の値を推定する" 手法であり、統計的に見て最も偏見なく結晶構造を推定することが可能である。以下では簡単に情報理論を復習し、MEM の概要について述べる。

ρ_iを事象 i が生起する確率とすると、情報理論における情報量 P は $P = -log\rho_i$で定義される。これは、確率の小さい事象 ($\rho_i \to 0$) が生起した場合に得られる情報量は大きく ($P \to \infty$)、確率の大きい事象 ($\rho_i \to 1$) が生起した場合に得られる情報量は小さく ($P \to 0$) なるよう、定義されている。

情報エントロピー S は、1 回の試行で得られるであろう情報量の期待値として定義される。

$$S = -\sum_i \rho_i \, log\rho_i \tag{20}$$

例として生起確率が p, q (ただし $p+q=1$) である事象群の情報エントロピー S の様子を図 3 に示す。この図から分るように、情報エントロピーは $p = q = 1/2$ の時最大になる。すなわち情報エントロピーが最大になるのは、事象の生起確率にかたよりが無い場合である。

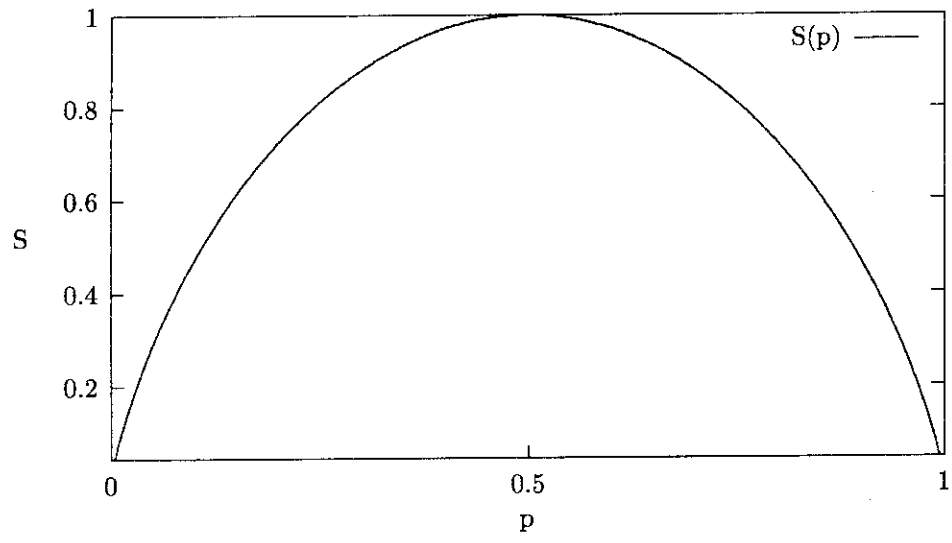

図 3: 生起確率 (p) と情報エントロピー (S) の関係

次に、ある基準となる事象が生起したという条件下での情報エントロピー (条件付きエントロピー)S'を

$$S' = -\sum_i \rho_i \, log\frac{\rho_i}{\tau_i} \tag{21}$$

で定義する ([7])。τ_iは基準となる事象の確率である。

ここで事象の確率として、結晶内の電子の存在確率を対応させる。$\rho(r)$ は結晶格子を適当に分割した場合の位置 r での電子密度、$\tau(r)$ は同じ位置での基準とする電子密度 (初期推定値) とする。すると、位置 r における電子の存在確率$\rho'(r)$ およびその基準存在確率$\tau'(r)$ は、次式で定義できる。

$$\rho'(r) = \frac{\rho(r)}{\sum_r \rho(r)} \qquad \tau'(r) = \frac{\tau(r)}{\sum_r \tau(r)} \tag{22}$$

これより結晶構造解析での条件付きエントロピー S'は

$$S' = -\sum_r \rho'(r) log \frac{\rho'(r)}{\tau'(r)} \tag{23}$$

となる。S'を最大にするということは、基準となる電子密度$\tau(r)$ が与えられている条件の下で、最もかたより無く電子密度$\rho(r)$ を推定することに他ならない。

ここまでの議論では実験的に求められた結晶構造因子

$$F_{obs}(h,k,l) = F_{obs}(hb_1 + kb_2 + lb_3) \tag{24}$$

の情報がなんら考慮されていない。そこで関数 C を次式で定義する。

$$C = \sum_{(h,k,l)} \frac{|F_{obs}(h,k,l) - F_{true}(h,k,l)|^2}{\sigma(h,k,l)^2} \tag{25}$$

ここで $F_{true}, \sigma(h,k,l)$ は各々結晶構造因子の真値と測定値の標準偏差である。結晶構造因子の測定誤差が偶発誤差のみによるものならば、その誤差はσ程度であることから、N を測定データ数として

$$C \approx N \tag{26}$$

となる。$F_{true}(h,k,l)$ を適当な推定値 $F_{cal}(h,k,l)$ で置き換えると、偶発誤差の他に真の値と推定値の誤差も加わるので、一般に

$$C > N \tag{27}$$

である。これらを考慮した新しいエントロピー Q を次式で与える。

$$Q = -\sum_r \rho'(r) log \frac{\rho'(r)}{\tau'(r)} - \frac{\lambda}{2} \sum_{(h,k,l)} \frac{|F_{obs}(h,k,l) - F_{cal}(h,k,l)|^2}{\sigma(h,k,l)^2} \tag{28}$$

ここでλはラグランジュの未定乗数である。

結晶構造解析における MEM とは、この Q を最大にする電子密度分布を求めることに相当する。そのためには

$$\frac{\partial Q}{\partial \rho(r)} = 0 \tag{29}$$

が必要条件である。これらの式および適当な近似を用いることにより

$$\rho(r) = \tau(r)\, exp\left(\lambda \sum_{(h,k,l)} \frac{(F_{obs}(h,k,l) - F_{cal}(h,k,l)) \cdot e^{-i(hb_1+kb_2+lb_3)\cdot r}}{\sigma(h,k,l)^2} \right) \tag{30}$$

が得られる。

この式から MEM による電子密度分布は、次のようなアルゴリズムで計算できる ([8])。

○MEMによる結晶構造解析アルゴリズム

1. 結晶構造因子の測定値 $F_{obs}(h,k,l)$ 及び標準偏差$\sigma(h,k,l)$ を入力する。

2. 電子密度分布の初期状態$\tau(r)$ を決定する。

3. 電子密度分布状態$\tau(r)$ から結晶構造因子 $F_{cal}(h,k,l)$ を (19) 式から求める。すなわち、

$$F_{cal}(h,k,l) = a\sum_{r}\tau(r)e^{i(hb_1+kb_2+lb_3)\cdot r} \tag{31}$$

4. ラグランジュ項Λを次式により計算する。

$$\Lambda = \lambda \sum_{(h,k,l)} \frac{(F_{obs}(h,k,l) - F_{cal}(h,k,l))\cdot e^{-i(hb_1+kb_2+lb_3)\cdot r}}{\sigma(h,k,l)^2} \tag{32}$$

5. 新しい電子密度分布$\rho(r) = \tau(r)exp(\Lambda)$ を求める。

6. 次式から C を計算し、$C < N$ であれば計算を終了する。そうでなければ、$\rho(r)$ を$\tau(r)$ とした上で 3. に戻る。

$$C = \sum_{(h,k,l)} \frac{|F_{obs}(h,k,l) - F_{cal}(h,k,l)|^2}{\sigma(h,k,l)^2} \tag{33}$$

このアルゴリズムで最も計算時間コストの大きい部分は、明らかに (31) 式及び (32) 式の多次元離散フーリエ変換及び逆変換である。実際次章でみるように、本プログラムの中で多次元離散フーリエ変換及び逆変換が全計算時間の 80 %以上を占めている。

上記のアルゴリズムでは単純な結晶を解析対象にしていたので、多次元離散フーリエ変換の次元数は 3 次元であった。一方、結晶構造解析の分野で近年精力的に研究されている対象として 1984 年に Shechtman 等によって発見された準結晶がある ([9],[12])。準結晶は、X線回折像に結晶構造の反映である鋭いピークが現れるにもかかわらず、結晶構造に空間的な周期性を持たない。その後の研究により準結晶には単位胞が複数個存在し、そのため逆格子ベクトルの個数が空間の次元数より大きいことがわかっている ([10])。本プログラムはこの準結晶構造も解析対象としており、その場合必要とされる離散フーリエ変換の次元数は 4 以上、通常は 5 次元または 6 次元離散フーリエ変換が必要となる。この場合は多次元離散フーリエ変換にかかる比重がさらに大きくなる。

4 多次元離散フーリエ変換

前章で述べたように、本プログラムの高速化の鍵は多次元離散フーリエ変換にある。ここでは我々がベクトル計算機向きに開発した、ベクトル化多次元離散フーリエ変換ルーチンについて説明する。簡単のため3次元の順変換についてのみ説明するが、高次元の場合及び逆変換については全く同様に議論できる。

3次元の実空間データ $\{f(j_1,j_2,j_3)\}$ $(j_1,j_2,j_3=0,1,...n-1)$ から周波数空間データ $\left\{\hat{f}(k_1,k_2,k_3)\right\}$ $(k_1,k_2,k_3=0,1,...n-1)$ を計算する離散フーリエ変換は次式で与えられる。

$$\hat{f}(k_1,k_2,k_3)=\sum_{j_1,j_2,j_3=0}^{n-1} f(j_1,j_2,j_3)\cdot\omega^{-k_1j_1-k_2j_2-k_3j_3} \qquad \left(\omega=exp\left(\frac{2\pi i}{n}\right)\right) \tag{34}$$

本来各次元の要素数は同一である必要はないが、本プログラムでは同一の場合 (一定値 n) しか扱わない。これを実際に計算する際に

$$\hat{f}(k_1,k_2,k_3)=\sum_{j_1=0}^{n-1}\left(\sum_{j_2=0}^{n-1}\left(\sum_{j_3=0}^{n-1} f(j_1,j_2,j_3)\cdot\omega^{-k_3j_3}\right)\omega^{-k_2j_2}\right)\omega^{-k_1j_1} \tag{35}$$

として、一次元離散フーリエ変換を順次行う方法がまず考えられる。図4にその様子を示す。左図は3次元データ配列 $\{f(j_1,j_2,j_3)\}$ を立方体で表しており、j_3軸方向 (1) → j_2軸方向 (2) → j_1軸方向 (3) の順序で一次元離散フーリエ変換が行われることを示している。右図は j_3軸方向 (1) の計算の様子であり、1本の実線矢印が $k_3=0$ における (35) 式の最内側計算を表し、太い破線矢印は同様の計算を $k_3=0,1,2,...$に対して順次行うことを意味している。

この式を素朴に FORTRAN でコーディングすれば、次のようになるであろう。

```
○素朴な形式の j3 軸方向計算
do k3 = 0,n-1
   do j1 = 0,n-1
      do j2 = 0,n-1
         do j3 = 0,n-1
            fhat(j1,j2,k3) = fhat(j1,j2,k3) + f(j1,j2,j3)*OMEGA**(-k3*j3)
         end do
      end do
   end do
end do
```

ベクトル計算の場合には、ベクトル化される最内側ループのループ長 (ベクトル長) は n であること、その最内側ループ計算が (j_1,j_2) 平面上で n^2回、逐次に行われることに注意しよう。図4右図で説明すれば、一本の矢印がベクトル化される最内側計算に相当する。その矢印が (j_1,j_2) 平面で n^2本存在するので、順次 n^2回の最内側計算を繰り返す必要がある。

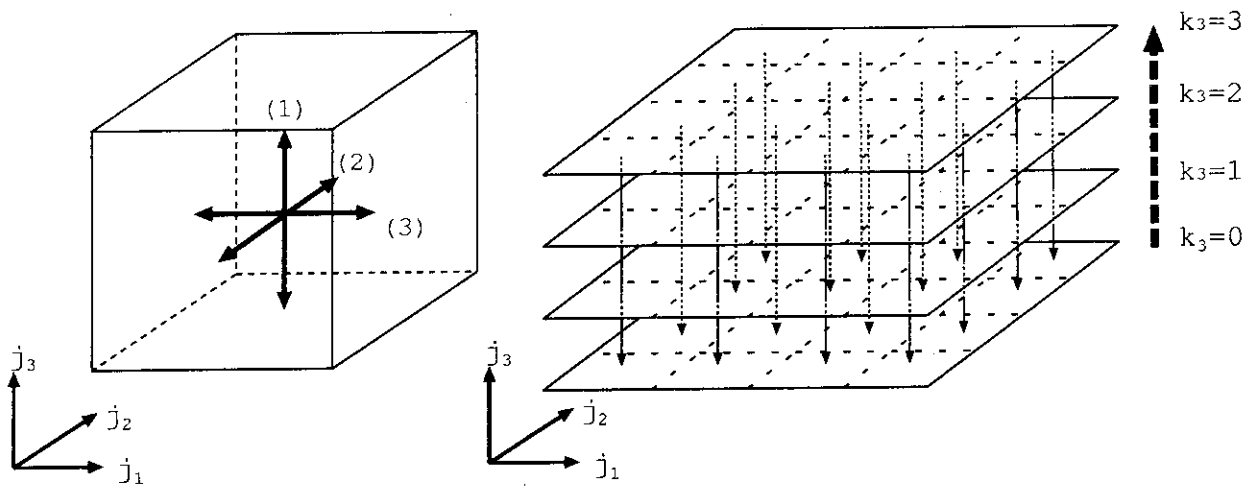

図 4: 多次元離散フーリエ変換の計算順序

このアルゴリズムで高速化を図るには、2 つの方針が考えられる。1 つは一次元離散フーリエ変換に FFT(Fast Fourier Transform) を用いることである。これは (35) 式最内側の計算 (図における 1 本の矢印) の計算量を少なくすることに相当する。本プログラムでも、オリジナルの多次元離散フーリエ変換ルーチンではこの方法を採用しており、スカラ計算機には有効であろう。

しかしこの方針はベクトル計算機の場合には問題が多い。なぜなら、通常の結晶構造解析で必要な 1 方向の分割数 n は 10〜20 程度であり、従って最内側 DO ループのベクトル長が短いからである。ループ長の短い DO ループ内でいくら計算量を減らしても、ベクトルパイプライン立ち上げのオーバーヘッドの方が大きく、大幅な性能向上は望めない。

高速化を図るもう 1 つの方針はベクトル長を長くすることである。我々はこの方針に従い、ベクトル計算機に適したベクトル化多次元離散フーリエ変換ルーチンを開発した。核となるアイデアは、(35) 式の最内側計算が全ての (j_1, j_2) 要素に対し同様に行われることにある。すなわち (j_1, j_2) 平面全体を一時に加算することにより、その平面内の要素数分のベクトル長を確保できる (図 5 参照)。このことを実現するため、j_3方向の総和計算は外側のループに、配列要素 (j_1, j_2) の範囲に関するループは内側のループにする。最内側ループはベクトル化され、そのループ長 (ベクトル長) は n^2となる。特に 6 次元の準結晶構造の場合はベクトル長が n^5となり、大幅な性能向上が期待できる。

```
○ベクトル長を長くした j3 軸方向計算
do k3 = 0,n-1
   do j3 = 0,n-1
      do j12 = 0,n**2-1
         fhat(j12,0,k3) = fhat(j12,0,k3) + f(j12,0,j3)*OMEGA**(-k3*j3)
      end do
   end do
end do
(do j12 = 0,n**2-1 は (j1,j2) が (0,0)〜(n-1,n-1) の範囲で動くことを表す)
```

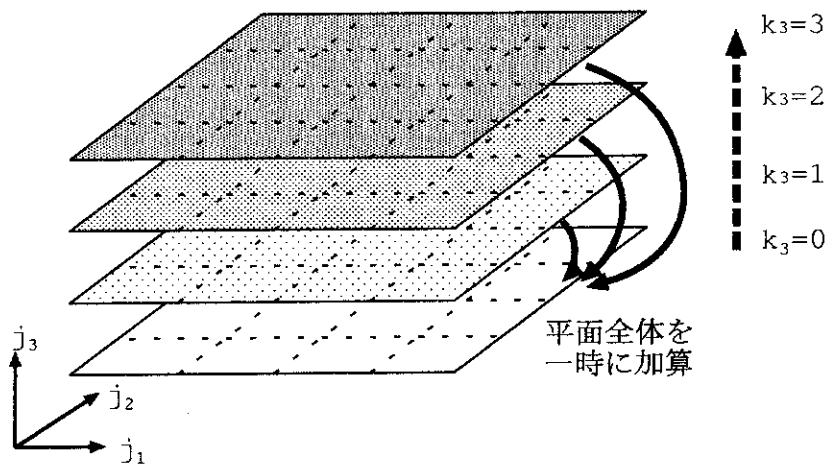

図 5: ベクトル化多次元離散フーリエ変換ルーチンの計算

5 ベクトル化による性能向上

性能向上の指標とするため、SX-4 上でオリジナルな状態のプログラム (各次元方向の計算に 1 次元 FFT を使用) を実行し、各種性能値及び各ルーチンでの時間コストを計測した。

ＳＸ－４では PROGINF と呼ばれるツールにより、ユーザプログラムの性能情報を容易に取得することができる。それを用いることにより次のような性能情報が得られた。なお解析データとしては亜鉛-マグネシウム-イットリウム (Zn-Mg-Y) 準結晶のＸ線回折データを用いており ([11])、格子点数は $12^6 (\approx 3000000)$ とした。

オリジナル版のプログラム性能情報 (Zn-Mg-Y)

```
******  Program Information  ******
Real Time (sec)       :       182.106268  (経過時間)
User Time (sec)       :       180.432311  (ユーザプログラム CPU 時間)
Sys  Time (sec)       :         0.463259  (システム CPU 時間)
Vector Time (sec)     :        11.856077
Inst. Count           :      9365800461.
V. Inst. Count        :       150591657.
V. Element Count      :      4872058917.
FLOP Count            :      3826061349. (浮動小数点演算数)
MOPS                  :        78.075083
MFLOPS                :        21.204968  (フロップス値)
VLEN                  :        32.352781  (平均ベクトル長)
V. Op. Ratio (%)      :        34.584839  (ベクトル化率)
Memory Size (MB)      :       277.031250  (使用メモリ量)
MIPS                  :        51.907557
I-Cache (sec)         :         2.814733  (命令キャッシュミス時間)
O-Cache (sec)         :        97.266088  (データキャッシュミス時間)
Bank (sec)            :         0.000000  (メモリバンク競合時間)
```

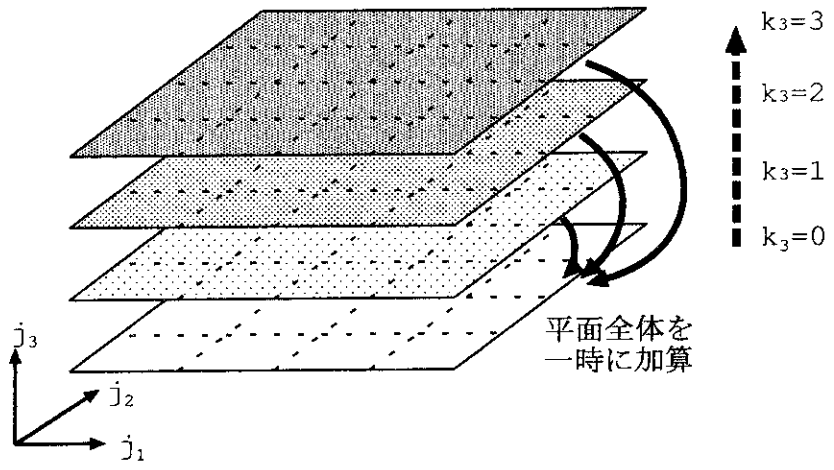

図 5: ベクトル化多次元離散フーリエ変換ルーチンの計算

5 ベクトル化による性能向上

性能向上の指標とするため、SX-4 上でオリジナルな状態のプログラム (各次元方向の計算に 1 次元 FFT を使用) を実行し、各種性能値及び各ルーチンでの時間コストを計測した。

ＳＸ－４では PROGINF と呼ばれるツールにより、ユーザプログラムの性能情報を容易に取得することができる。それを用いることにより次のような性能情報が得られた。なお解析データとしては亜鉛-マグネシウム-イットリウム (Zn-Mg-Y) 準結晶のＸ線回折データを用いており ([11])、格子点数は $12^6 (\approx 3000000)$ とした。

オリジナル版のプログラム性能情報 (Zn-Mg-Y)

```
   ******  Program Information  ******
Real Time (sec)       :       182.106268  (経過時間)
User Time (sec)       :       180.432311  (ユーザプログラム CPU 時間)
Sys  Time (sec)       :         0.463259  (システム CPU 時間)
Vector Time (sec)     :        11.856077
Inst. Count           :      9365800461.
V. Inst. Count        :       150591657.
V. Element Count      :      4872058917.
FLOP Count            :      3826061349. (浮動小数点演算数)
MOPS                  :        78.075083
MFLOPS                :        21.204968  (フロップス値)
VLEN                  :        32.352781  (平均ベクトル長)
V. Op. Ratio (%)      :        34.584839  (ベクトル化率)
Memory Size (MB)      :       277.031250  (使用メモリ量)
MIPS                  :        51.907557
I-Cache (sec)         :         2.814733  (命令キャッシュミス時間)
O-Cache (sec)         :        97.266088  (データキャッシュミス時間)
Bank (sec)            :         0.000000  (メモリバンク競合時間)
```

ＳＸ－４の1CPUは2GFLOPSの性能なので、21.20MFLOPSというフロップス値は満足すべきものとは言えない。問題はベクトル化率の低さ(34.58％)と平均ベクトル長の短さ(32.35)にある。すなわち、そもそもベクトル化できる部分が少なく、ベクトル化できた部分もベクトル長が短いため性能が上がらないという状況であることがわかる。

次にUNIXのprofコマンドでプログラムのどの部分が計算時間を消費しているかを調査した。

prof出力ファイル

%Time	Seconds	Cumsecs	#Calls	msec/call	Name
82.8	149.60	149.60			udft1_
9.3	16.74	166.34			grad_
0.9	1.66	168.00			getrx_
0.7	1.19	169.19			ftopx_
0.7	1.18	170.38			fv_vamxva
0.5	0.94	171.32			smtrx_
0.5	0.84	172.16			vcpyr_

以下略

サブルーチンudft1は1次元FFTルーチンである。予想通り1次元FFTルーチン部分で、82.8％もの計算時間を消費していることがわかる。

そこで前章で説明したベクトル化多次元離散フーリエ変換ルーチンを本プログラムに適用した。

ベクトル化版のプログラム性能情報（Zn-Mg-Y）

****** Program Information ******		
Real Time (sec)	:	41.506693
User Time (sec)	:	40.412630
Sys Time (sec)	:	0.400989
Vector Time (sec)	:	19.500172
Inst. Count	:	2222469294.
V. Inst. Count	:	249130568.
V. Element Count	:	30097745109.
FLOP Count	:	15444250748.
MOPS	:	793.590604
MFLOPS	:	382.163956
VLEN	:	120.811129
V. Op. Ratio (%)	:	93.846985
Memory Size (MB)	:	334.031250
MIPS	:	54.994423
I-Cache (sec)	:	2.677392
O-Cache (sec)	:	5.051325
Bank (sec)	:	0.107515

プログラム経過時間を比較するとオリジナル版の182.11secに対し、ベクトル化版は41.50secと約4.4倍の高速化がなされている。またベクトル化率も約94％と格段に向上していることがわかる。なおベクトル化版浮動小数点演算数がオリジナル版に比べ約4倍に増加している。これはオリジナル版は多次元離散フーリエ変換にFFTを用いて計算量を減少させているのに対し、ベクトル化版ではベクトル長は長くなっているが計算量自体は変化していないことによる。

本性能測定は手早くデータを得るために、比較的少ない格子点数とし、さらに反復数を極端に少なくした(反復数3回)。そのため相対的にファイル入出力等のコストが大きくなっており、ベクトル化版の性能が過

小評価されている。実用計算ではベクトル化率がより大きな値になり、さらなる性能向上が期待できる。

このことを確認するため大規模なデータを用い、反復数を多くしてプログラムを実行してみた。解析データはアルミニウム-パラジウム-マンガン (Al-Pd-Mn) 準結晶のX線回折データを用い ([11])、格子点数は $14^6 (\approx 7530000)$ とした。また反復数は 100 回とした。

オリジナル版のプログラム性能情報 (Al-Pd-Mn)

```
******  Program Information  ******
Real Time (sec)       :       9824.226591
User Time (sec)       :       9745.178566
Sys  Time (sec)       :         56.103899
Vector Time (sec)     :        122.783050
Inst. Count           :     936223790655.
V. Inst. Count        :       1125211953.
V. Element Count      :     232212687390.
FLOP Count            :     309367555995.
MOPS                  :        119.783466
MFLOPS                :         31.745704
VLEN                  :        206.372397
V. Op. Ratio (%)      :         19.892954
Memory Size (MB)      :        277.031250
MIPS                  :         96.070460
I-Cache (sec)         :         16.640904
O-Cache (sec)         :       3235.461565
Bank (sec)            :          0.000001
```

ベクトル化版のプログラム性能情報 (Al-Pd-Mn)

```
******  Program Information  ******
Real Time (sec)       :        815.409558
User Time (sec)       :        790.928373
Sys  Time (sec)       :          7.920612
Vector Time (sec)     :        728.219594
Inst. Count           :     21188040786.
V. Inst. Count        :      9153818386.
V. Element Count      :   2287414711426.
FLOP Count            :   1238917084618.
MOPS                  :       2907.278348
MFLOPS                :       1566.408700
VLEN                  :        249.886399
V. Op. Ratio (%)      :         99.476648
Memory Size (MB)      :        334.031250
MIPS                  :         26.788824
I-Cache (sec)         :         13.899729
O-Cache (sec)         :          5.000188
Bank (sec)            :          0.008945
```

プログラム経過時間を比較すると、ベクトル化版はオリジナル版の 12.0 倍の高速化が達成されている。またベクトル化率は 99 %を超え、フロップス値は 1566MFLOPS に達している。このフロップス値はＳＸ－４ 1CPU のピーク性能 2GFLOPS の約 80 %に相当する。ベクトル化版では最内側 DO ループは単純な積和計算なので、チェイニング等によりベクトルパイプラインが非常に効率よく動作しているものと思われる。

一方オリジナル版では、ベクトル化率が相変わらず小さい (19.9 %)。そのためベクトル長が長くなっても (206.4)、フロップス値は約 32MFLOPS に留まっている。またスカラ計算部においても、データキャッシュミス時間が 3235 秒と全経過時間の 1/3 も費やしている。これは FFT でのメモリアクセス間隔は大きくしかも不規則なので、キャッシュミスが頻繁におこることを示している。

6 並列化による性能向上

前章でベクトル化した多次元離散フーリエ変換ルーチンをさらに高速化するため、ＳＸ－４のマイクロタスク機能による並列化を行った。

ＳＸ－４には次のような特徴がある ([13],[14],[15])。

- SX-4は共有／分散メモリ型ベクトル並列スーパーコンピュータである。1ノードは32個までのCPUと16GBまでの共有メモリからなる。1個のCPU速度は2GFLOPSであり、ノード内の並列化は共有メモリ型である。
- 16ノードまでをHIPPIスイッチまたはIXS(ノード間クロスバスイッチ)で接続することにより、分散メモリ型並列マシンとして利用できる。
- ノード内の並列化はマイクロタスク機能もしくはマクロタスク機能によりなされる。マイクロタスク機能はコンパイラにより自動的に、または必要に応じて並列化指示行を用いることにより利用できる。一方マクロタスク機能は、並列化ライブラリルーチンをコールすることにより利用できる。本研究では並列化が容易で性能のでやすくメモリ効率のよい、マイクロタスクによる並列化を試みる。
- ノード間にまたがる並列化にはMPI(Message Passing Interface)またはHPF(High Performance Fortran)を使用する。なお、MPIとHPFはノード内の並列化でも使用できる。
- マイクロタスクにおいて並列実行するのは、実際に並列化DOループ等を含むルーチン単位である。つまり、多次元離散フーリエ変換ルーチンは並列実行(冗長実行部も含む)するが、それ以外のルーチン(メインルーチン等)は逐次実行(1CPUのみによる実行)する。(注意 冗長実行ではない!!)
- 変数はstatic領域にとられる変数(common文やsave文に現れる変数と並列化ルーチンの仮引数の変数)とstack領域にとられる変数(上記以外)に大別される。static領域変数は並列化プロセス数に関係なく1個しかないが、stack領域変数は並列化プロセス数分あり、プロセス毎に異なる値を取り得る。

ここでメモリ効率について一言付言しておく。X線結晶構造解析プログラムでは非常に大きな配列を使用しており、そのためメモリ容量は膨大なものとなる。例えば前章で解析したAl-Pd-Mnの場合には格子点数を$14^6(\approx 7530000)$としたが、各格子点のデータを8BYTEの単精度複素数としても(フーリエ変換を行うので複素数で扱う必要がある)、これだけで60MBのメモリを消費する。ワーク領域も含めると300MB程度はすぐに消費される。これに分散メモリ型の並列化を行う場合、特別な指定をしないと各配列は重複して確保されるのでメモリの無駄が生じやすい。一方共有メモリ型の並列化では、大きな配列はstatic領域に確保し各並列化プロセスで共有すれば良いので、効率的にメモリを使用できる。

ベクトル化多次元離散フーリエ変換ルーチンの並列化は容易である。$k_3 = 0, 1, 2, ...$の各平面に対して順次行っていた加算計算は独立に実行できるので、並列化が可能である(図6参照)。

```
○並列化したj3軸方向計算
*pdir pardo
do k3 = 0,n-1
   do j3 = 0,n-1
      do j12 = 0,n**2-1
         fhat(j12,0,k3) = fhat(j12,0,k3) + f(j12,0,j3)*OMEGA**(-k3*j3)
      end do
   end do
end do
(*pdir pardoは直後のdo k3 = 0,n-1を並列実行することを表す)
```

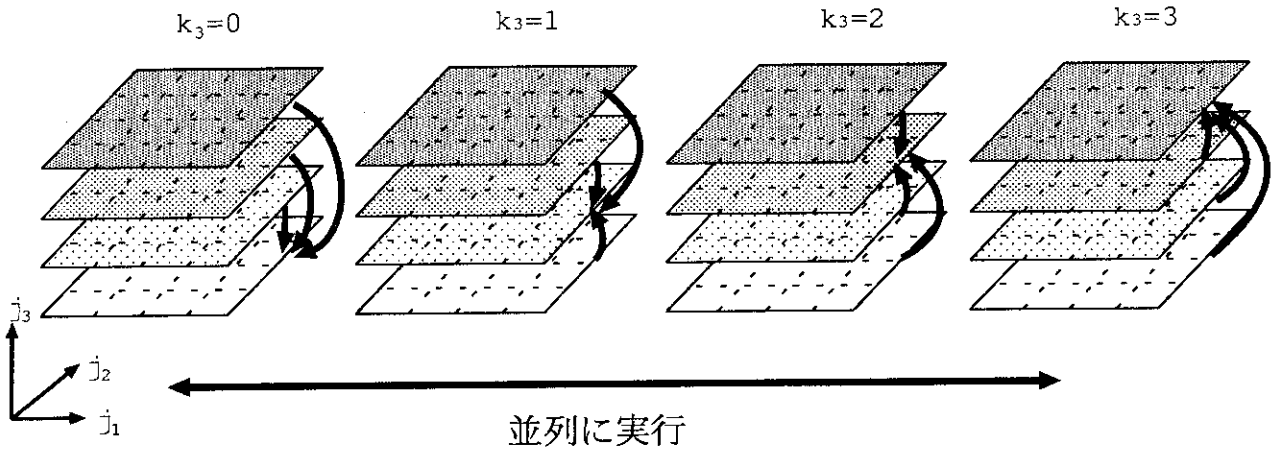

図 6: 並列化多次元離散フーリエ変換ルーチンの計算

並列化多次元離散フーリエ変換ルーチンのソースリストを付録に示すが、この場合の並列化作業は非常に容易であり、以下のソースプログラム変更の後、リコンパイルするだけであった。

- プログラム実行開始と同時に並列化プロセスを確保するため、メインルーチンの最初に*pdir reserve=(CPU 数) という並列化指示行を挿入した。
- 並列化対象ルーチンである fftcr1(多次元離散フーリエ変換の本体) と dmrot1(計算方向を変更するためのルーチン) の最外側 DO ループを並列実行するため、*pdir pardo nobarr という並列化指示行を挿入した。なお nobarr は DO ループの前後で同期をとらないことを指定するオプションであり、同期をとる必要がある場合はこれをつけない。
- ベクトル化版では dmzero2 というサブルーチンで、計算結果を格納する配列を一括してゼロクリアしていたが、ロードバランスを公平にするため、並列化ルーチン (fftcr1) 内部で各並列化プロセスごとにゼロクリアするようにした。
- 並列化部分の経過時間を測定するため、時間計測システムルーチンを挿入した。

この並列化多次元離散フーリエ変換ルーチンを用いて、再度アルミニウム-パラジウム-マンガン (Al-Pd-Mn) 準結晶の解析を行った。結果は表 1 及び図 7 の通りである。測定は専有状態で行った。

表 1: 並列化効果 (Al-Pd-Mn)　　sec(ratio)

	1CPU	2CPU	4CPU	7CPU	8CPU	14CPU
実測値	835.0(1.00)	532.5(1.57)	402.7(2.07)	316.5(2.64)	316.4(2.64)	273.7(3.05)
理想値	-	531.8(1.57)	381.3(2.19)	316.3(2.64)	304.7(2.74)	272.9(3.06)

ここで速度向上率 (SPEEDUP ratio P) の理想値は、多次元離散フーリエ変換本体の経過時間と (605.6sec)、プログラム全体の経過時間 (835.0sec) から並列化率 ($Rp = 0.725 = 605.6/835.0$) を計算し、次のアムダールの法則により推定した ([16])。(Np は CPU 数)

$$P = \frac{1}{1 - Rp + Rp/Np} \tag{36}$$

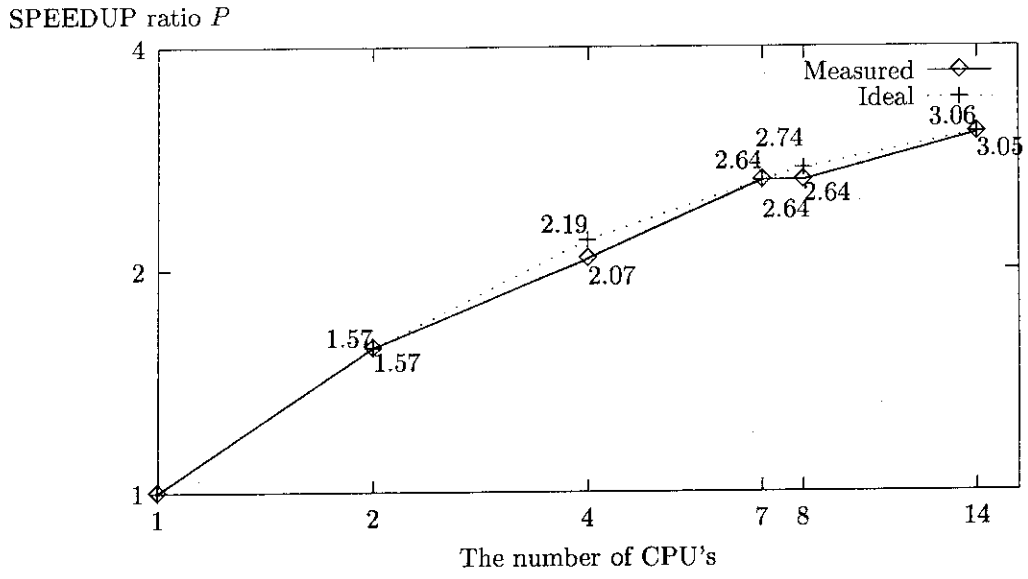

図 7: CPU 数と加速率の関係 (実測値と理想値)

まず、ベクトル化版と並列化版 (1CPU) の経過時間の違いについて説明する。ベクトル化版ではプログラム経過時間が 815.4 秒であるのに対し、並列化版では 835.0 秒と約 20 秒程遅くなっている。原因としては、配列 0 クリア方法の変更、時間計測システムルーチンの挿入、さらに並列化プロセスの生成が考えられる。本来ならば並列化版の性能については、ベクトル化版の全経過時間を基準に議論すべきである。しかし、ベクトル化版と並列化版 (1CPU) の経過時間の差異が全経過時間の 2.5 %程度と微少であること、並列化版 (1CPU) を基準にした方が並列化性能に関する考察が簡明になることから、ここでは並列化版 (1CPU) の経過時間を基準に議論を行う。

表 1 及び図 7 を見ると、2CPU、7CPU、14CPU の場合はほぼ理想的な性能が得られている。14CPU の場合は並列版 (1CPU) の 3.05 倍、ベクトル化版に比べても 2.98 倍、オリジナル版に比べると実に 35.9 倍の高速化が達成されている。

一方、4CPU と 8CPU の場合には若干の性能劣化が見られる。このことは、実際に並列化した部分の経過時間を見ると、より明らかになる (表 2 参照)。

表 2: 並列化部分の経過時間　　sec(ratio)

	1CPU	2CPU	4CPU	7CPU	8CPU	14CPU
実測値	605.6(1.00)	303.1(2.00)	173.3(3.49)	87.1(6.95)	87.0(6.96)	44.3(13.67)
理想値	-	302.8(2.00)	151.4(4.00)	86.5(7.00)	75.7(8.00)	43.3(14.00)

この表から明らかに 4CPU と 8CPU での性能劣化が観察される。その原因は各並列化プロセスに負荷が公平に分散されていないことにある。

本プログラムで並列化される DO ループのループ長は、単位胞の 1 方向分割数 14 である。そのため 14 の約数である 2CPU、7CPU、14CPU の場合は公平な負荷分散になる。一方、4CPU の場合はループ長 4 の並列化プロセスが 2 プロセス、ループ長 3 の並列化プロセスが 2 プロセス生成される。各並列化プロセスの中で最も長い経過時間が測定されるので、性能向上は $14/4 = 3.5$ 倍を超えない。8CPU の場合も同様である。このように考えると、並列化部分は最大限の性能向上が達成されていると考えられる。

以上の並列化は指示行挿入等による" 手動の" 並列化であった。一方、ＳＸ－４にはマイクロタスクによる自動並列化機能がサポートされている。これはコンパイラが (正確にはプリプロセッサが) ソースプログラム中の DO ループレベルの並列性を抽出し、自動的に並列化するという機能である。この機能を用いて再度並列化を試みたところ、表 1 とほぼ同等の性能が得られた。本研究では並列化対象部分が限定されており、当初から作業量が少ないと予想されたので、手動による並列化を行ったが、並列化対象が広範囲に渡る場合には、この自動並列化機能は非常に有効であると思われる。

本研究における並列化率 R_pは 72.5 % と中程度であり、そのため並列加速率は 14CPU で 3.05 倍の高速化という結果になっている。これは見ようによっては、やや物足りない結果といえるかもしれない。今回並列化しなかった残りの 27.5 % (229.4sec) は、主にファイル入出力部分及び結果出力のためのデータ加工部分であり、原理的に並列化が困難な部分である。これらの部分の経過時間は、オリジナル版においては全経過時間 (9824.2sec) の 2.3 % を占めるに過ぎなかったが、ベクトル化により多次元離散フーリエ変換部分が大幅に高速化されたため、相対的に比重が大きくなったといえる。本研究では、ファイル入出力部分及びデータ加工部分は反復数によらず常に一回限りしか行われないこと、解析規模が大きくなるに従い再度多次元離散フーリエ変換部分の比重が大きくなることから、あえて非並列化部分の高速化を図ることはしなかった。

並列化版 (1CPU) のプログラム性能情報 (Al-Pd-Mn)

```
   ******  Program Information  ******
Real Time (sec)        :          834.998801
User Time (sec)        :          834.762973
Sys  Time (sec)        :            0.215835
Vector Time (sec)      :          728.311825
Inst. Count            :        22114476940.
V. Inst. Count         :         9153939897.
V. Element Count       :      2287416674327.
FLOP Count             :      1238917090381.
MOPS                   :         2755.725021
MFLOPS                 :         1484.154341
MOPS   (concurrent)    :         2755.629184
MFLOPS (concurrent)    :         1484.102726
VLEN                   :          249.883296
V. Op. Ratio (%)       :           99.436591
Memory Size (MB)       :          309.000000
Max Concurrent Proc.   :                  1.
 Conc. Time(>= 1)(sec):          834.792005
Event Busy Count       :                  0.
Event Wait (sec)       :            0.000000
Lock Busy Count        :                  0.
Lock Wait (sec)        :            0.000000
Barrier Busy Count     :                  0.
Barrier Wait (sec)     :            0.000000
MIPS                   :           26.491924
MIPS (concurrent)      :           26.491002
I-Cache (sec)          :           18.237131
O-Cache (sec)          :            9.960453
Bank (sec)             :            0.009775
```

並列化版（14CPU）のプログラム性能情報（Al-Pd-Mn）

```
   ******  Program Information  ******
Real Time (sec)        :        273.697143
User Time (sec)        :       1015.437963
Sys  Time (sec)        :          1.906827
Vector Time (sec)      :        729.490062
Inst. Count            :      24620157699.
V. Inst. Count         :       9153939991.
V. Element Count       :    2287416697067.
FLOP Count             :    1238917545538.
MOPS                   :       2267.871597
MFLOPS                 :       1220.081965
MOPS   (concurrent)    :       8421.416817
MFLOPS (concurrent)    :       4530.599878
VLEN                   :        249.883296
V. Op. Ratio (%)       :         99.328398
Memory Size (MB)       :        328.000000
Max Concurrent Proc.   :                14.
 Conc. Time(>= 1)(sec):        273.455520
 Conc. Time(>= 2)(sec):         57.414720
 Conc. Time(>= 3)(sec):         57.364172
 Conc. Time(>= 4)(sec):         57.317569
 Conc. Time(>= 5)(sec):         57.268131
 Conc. Time(>= 6)(sec):         57.220772
 Conc. Time(>= 7)(sec):         57.173664
 Conc. Time(>= 8)(sec):         57.125759
 Conc. Time(>= 9)(sec):         57.072468
 Conc. Time(>=10)(sec):         57.008169
 Conc. Time(>=11)(sec):         56.941685
 Conc. Time(>=12)(sec):         56.880591
 Conc. Time(>=13)(sec):         56.828039
 Conc. Time(>=14)(sec):         56.562737
Event Busy Count       :                 0.
Event Wait (sec)       :          0.000000
Lock Busy Count        :                 0.
Lock Wait (sec)        :          0.000000
Barrier Busy Count     :                 0.
Barrier Wait (sec)     :          0.000000
MIPS                   :        24.245851
MIPS (concurrent)      :        90.033501
I-Cache (sec)          :        18.478904
O-Cache (sec)          :        14.167184
Bank (sec)             :        13.703724
```

7 まとめ

本論文では、X線結晶構造解析プログラムのベクトル化及び並列化について報告した。ベクトル化ではベクトル長の長い多次元離散フーリエ変換ルーチンを開発導入し、プログラム全体でＳＸ－４ 1CPU のピーク性能の約 80 %にあたる 1566MFLOPS というフロップス値が得られた。プログラム経過時間で見ると、オリジナル版の 12.0 倍の高速化がなされている。

また上記ルーチンをＳＸ－４のマイクロタスク機能を用いて、共有メモリ型の並列化を行った。その結果、14 並列でさらに 3.0 倍の高速化が図られた。オリジナル版の経過時間と 14 並列版の経過時間を比べると、35.9 倍の高速化が達成されている。

共有メモリ型の並列化は比較的作業が容易であるという特徴がある。本プログラムもその例にもれず、非常に容易に並列化がなされた。一方、今後の方向性として、マルチノードによる分散メモリ型の並列化が考えられる。テラフロップス級のスーパーコンピュータは分散メモリ型にならざるを得ないからである。現時点で分散メモリ型プログラミングを行うには、MPI を用いるのが最も現実的であろう。そこでX線結晶構造解析プログラムを MPI で並列化することが、今後の課題として考えられる。しかし数万行もあるプログラムを、手動で MPI プログラムに書き換えるのは非常に困難である。既存プログラムを MPI 化する際に、ユーザを支援する環境もしくはツールの開発が強く望まれる。

謝辞

本研究において作業の一部は金属材料技術研究所のＳＸ－４／２０上で行いました。

並列版 (4 , 7 , 8 , 14CPU) の測定にはＮＥＣ府中事業所のＳＸ－４を利用させて頂きました。

ＮＥＣの左近彰一課長殿、月田逸郎主任殿、三浦聡殿、天羽宏嘉殿にはＳＸ－４利用技術について多くのご教示を頂きました。ここに感謝致します。

7 まとめ

本論文では、X線結晶構造解析プログラムのベクトル化及び並列化について報告した。ベクトル化ではベクトル長の長い多次元離散フーリエ変換ルーチンを開発導入し、プログラム全体でＳＸ－４ 1CPU のピーク性能の約 80 %にあたる 1566MFLOPS というフロップス値が得られた。プログラム経過時間で見ると、オリジナル版の 12.0 倍の高速化がなされている。

また上記ルーチンをＳＸ－４のマイクロタスク機能を用いて、共有メモリ型の並列化を行った。その結果、14 並列でさらに 3.0 倍の高速化が図られた。オリジナル版の経過時間と 14 並列版の経過時間を比べると、35.9 倍の高速化が達成されている。

共有メモリ型の並列化は比較的作業が容易であるという特徴がある。本プログラムもその例にもれず、非常に容易に並列化がなされた。一方、今後の方向性として、マルチノードによる分散メモリ型の並列化が考えられる。テラフロップス級のスーパーコンピュータは分散メモリ型にならざるを得ないからである。現時点で分散メモリ型プログラミングを行うには、MPI を用いるのが最も現実的であろう。そこでX線結晶構造解析プログラムを MPI で並列化することが、今後の課題として考えられる。しかし数万行もあるプログラムを、手動で MPI プログラムに書き換えるのは非常に困難である。既存プログラムを MPI 化する際に、ユーザを支援する環境もしくはツールの開発が強く望まれる。

謝辞

本研究において作業の一部は金属材料技術研究所のＳＸ－４／２０上で行いました。

並列版 (4 , 7 , 8 , 14CPU) の測定にはＮＥＣ府中事業所のＳＸ－４を利用させて頂きました。

ＮＥＣの左近彰一課長殿、月田逸郎主任殿、三浦聡殿、天羽宏嘉殿にはＳＸ－４利用技術について多くのご教示を頂きました。ここに感謝致します。

参考文献

[1] 小出昭一郎 '物理現象のフーリエ解析' 東京大学出版会 (1981)

[2] 坂田誠 '精密結晶構造解析に対する一考察' 日本結晶学会誌 30,135(1988)

[3] G.Bricogne, et al. Acta Cryst.(1990).A46,284-297

[4] M.Sakata, M.Sato Acta Cryst.(1990).A46,263-270

[5] W.Steurer 'Methods of Structural Analysis of Modulated Structures and Quasicrystarls' Singapore World Scientific(1991) 344-349

[6] 日野幹夫 'スペクトル解析' 浅倉書店 (1977)

[7] E.T.Jayncs IEEE Trans. Syst. Sci. Cybern. SSC-4,227(1968)

[8] D.M.Collins Nature 298,49(1982)

[9] D.Shechtman, et al. Phys. Rev. Lett.(1984) 53, 1951-1954

[10] A. Yamamoto 'Crystallography of Quasiperiodic Crystals' Acta Cryst.(1996).A52,50-560

[11] A.Yamamoto, Y.Matsuo, et al. Aperiodic '94 Singapore World Scientific(1995) 393-398

[12] 物理学辞典編集委員会編 '改訂版 物理学辞典' 培風館 (1994)

[13] 'SX-4 シリーズ システム概説書'

[14] 'Super-UX FORTRAN90/SX プログラミングの手引き'

[15] 'Super-UX FORTRAN90/SX 並列処理機能利用の手引き'

[16] 村田健郎 'スーパーコンピュータ' 丸善 (1985)

付録　並列化多次元離散フーリエ変換ルーチン

```
c***********************************************************************
c This program was made by Minami and modified by Watanabe.
c                                                              96/5/17
c***********************************************************************
      subroutine cft1(a,b,n,m,isn,icon)
c
      implicit real*4 (a-h,o-z)
c
      parameter (n1 = 14,ndim = 6)
c
      dimension a(*),b(*),n(*)
      dimension xrd(n1*ndim),xid(n1*ndim),nxd(n1*n1*ndim)
      dimension wkr(n1**ndim),wki(n1**ndim)
c
      data init/0/
      save init,xrd,xid,nxd
c                                                          check
      if (m.ne.ndim) stop 'Error in fftpara. (1)'
c
      iflg = 0
      do i = 1,ndim
         if (n(i).gt.n1) then
            write(*,*) 'n(i) = ',i,n(i)
            iflg = 1
         endif
      enddo
      if (iflg.eq.1) stop 'Error in fftpara. (2)'
c
      init = init + 1
      if (init.eq.1) call fftcri(ndim,n,n1,xrd,xid,nxd)
c
      call fftcr(ndim,n,n1,a,b,wkr,wki,xrd,xid,nxd,isn)
c
      return
      e n d
cmin++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++cmin
cmin  initial routine
cmin                                    1995.11   created by m.minami
cmin++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++cmin
      subroutine fftcri(ndim,n,nmax,xrd,xid,nxd)
      dimension xrd(nmax,ndim),xid(nmax,ndim)
      dimension nxd(nmax*nmax,ndim),n(ndim)
cmin
      do nd = 1 , ndim
cmin
         nnn = n(nd)
         dpos = 4.0d0*acos(0.0d0)/nnn
         do i = 1 , nnn
            xrd(i,nd) = cos( dpos*(i-1) )
            xid(i,nd) = sin( dpos*(i-1) )
         enddo
         do k = 1 , nnn
            do j = 1 , nnn
               nxd(j+(k-1)*nnn,nd) = mod((j-1)*(k-1),nnn) + 1
            enddo
         enddo
cmin
      enddo
cmin
      return
      e n d
cmin++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++cmin
cmin  ndim-dft main routine
cmin                                    1995.11   created by m.minami
```

```
cmin+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++cmin
      subroutine fftcr
     &     (ndim,n,nmax,wkr1,wki1,wkr2,wki2,xrd,xid,nxd,nsgn)
      dimension wkr1(*),wki1(*)
      dimension wkr2(*),wki2(*)
      dimension xrd(nmax,ndim),xid(nmax,ndim)
      dimension nxd(nmax*nmax,ndim),n(ndim)
      real*8 d0,d1,dsum
      data dsum/0.d0/
      save dsum
cmin
      nnn = 1
      do i = 1,ndim
         nnn = nnn*n(i)
      enddo
cmin
cmin << 1d-dft >>
cmin
      norm = 0
      call etime(d0)
      do i =  ndim,1,-1
         n1 = n(i)
         n2 = nnn/n1
cmiu
cmiu     call dmzero2(nnn,wkr2,wki2)
cmiu
         call fftcr1(n1,n2,wkr1,wki1,wkr2,wki2,
     &        xrd(1,i),xid(1,i),nxd(1,i),nsgn)
         if ( i.eq.1 .and. nsgn.eq.1 ) norm = 1
         call dmrot1(n1,n2,wkr2,wki2,wkr1,wki1,norm)
      enddo
      call etime(d1)
      dsum = dsum + (d1 - d0)
      write(*,*) dsum
cmin
      return
      e n d
cmin+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++cmin
cmin  rearrange routine
cmin                                    1995.11   created by m.minami
cmin+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++cmin
      subroutine dmrot1(n1,n2,wkir,wkii,wkor,wkoi,norm)
c
      dimension wkir(n2,n1),wkii(n2,n1)
      dimension wkor(n1,n2),wkoi(n1,n2)
c
      nnn = n1*n2
*pdir pardo nobarr
*vdir noloopchg
      do ii = 1 , n1
         do ij = 1 , n2
            wkor(ii,ij) = wkir(ij,ii)
            wkoi(ii,ij) = wkii(ij,ii)
         enddo
      enddo
c
      return
      e n d
cmin+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++cmin
cmin  zero clear routine
cmin                                    1995.11   created by m.minami
cmin+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++cmin
      subroutine dmzero2(n,work1,work2)
cmin
      dimension work1(n),work2(n)
cmin
```

```
      do i = 1,n
         work1(i)=0.d0
         work2(i)=0.d0
      enddo
cmin
      return
      e n d
cmin++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++cmin
cmin  1d dft subroutine
cmin                                    1995.11   created by m.minami
cmin++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++cmin
      subroutine fftcr1(n1,n2,wkr1,wki1,wkr2,wki2,xrd,xid,nxd,nsgn)
cmin
      dimension wkr1(n2,n1),wki1(n2,n1)
      dimension wkr2(n2,n1),wki2(n2,n1)
      dimension xrd(n1),xid(n1),nxd(n1,n1)
cmin
      if ( nsgn.eq.1 ) then
cmin
cmin << plas-side >>
cmin
*pdir pardo nobarr
         do k = 1 , n1
cmiu
            do ii = 1 , n2
               wkr2(ii,k) = 0.0d0
               wki2(ii,k) = 0.0d0
            enddo
cmiu
            do j = 1 , n1
               itmp = nxd(j,k)
               xxrd = xrd(itmp)
               xxid = xid(itmp)
               do i = 1 , n2
                  cirx = wkr1(i,j)
                  ciix = wki1(i,j)
                  wkr2(i,k) = wkr2(i,k) + cirx*xxrd + ciix*xxid
                  wki2(i,k) = wki2(i,k) + ciix*xxrd - cirx*xxid
               enddo
            enddo
         enddo
      else
cmin
cmin << minus-side >>
cmin
*pdir pardo nobarr
         do k = 1 , n1
cmiu
            do ii = 1 , n2
               wkr2(ii,k) = 0.0d0
               wki2(ii,k) = 0.0d0
            enddo
cmiu
            do j = 1 , n1
               itmp = nxd(j,k)
               xxrd = xrd(itmp)
               xxid = xid(itmp)
               do i = 1 , n2
                  cirx = wkr1(i,j)
                  ciix = wki1(i,j)
                  wkr2(i,k) = wkr2(i,k) + cirx*xxrd - ciix*xxid
                  wki2(i,k) = wki2(i,k) + ciix*xxrd + cirx*xxid
               enddo
            enddo
         enddo
      endif
```

```
cmin
      return
      e n d
```