ICOT Technical Report: TR-712

TR-712

並列3次元ダイナミックプログラミング法によるタンパクの配列解析 —KL1による並列実装とその性能特性に関する考察—

戸谷　智之、星田　昌紀、石川　幹人、
新田　克己

November, 1991

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

並列３次元ダイナミックプログラミング法によるタンパクの配列解析

Protein Sequence Analysis by Parallel 3-dimensional Dynamic Programming

— ＫＬ１による並列実装とその性能特性に関する考察 —

— Implementation with KL1 and Study of its Performance —

戸谷智之　星田昌紀　石川幹人　新田克己

Tomoyuki Toya, Masaki Hoshida, Masato Ishikawa, Katsumi Nitta

（財）新世代コンピュータ技術開発機構

Institute for New Generation Computer Technology (ICOT)

要旨

タンパクのアミノ酸配列の解析手法の一つであるマルチプルアライメントには、ダイナミックプログラミング法がよく使用される。だが、それに伴う計算量は極めて多いため、我々は、３次元ダイナミックプログラミング法を並列実装し、並列処理によるマルチプルアライメントの計算時間削減を実現した。本論文では、並列言語 KL1 を用いたシステムの実装方式を述べ、システムの性能特性を、処理モデルに基づいて考察した結果を解説する。

1　はじめに

本章では、我々が構築したシステムの背景について簡単に述べる。まず、解決すべき問題であるマルチプルアライメントと、その典型的解法であるダイナミックプログラミング法を説明し、３次元ダイナミックプログラミング法の並列化の意義を説く。また、次の第２章では、並列言語 KL1 を用いたシステムの実装方式を述べ、続く第３章では、並列処理の性能特性を詳細に考察する。

1.1　マルチプルアライメント

近年、タンパクのアミノ酸配列の自動分析法が開発され、それらのデータを蓄えたデータベースが急速に膨らんでいる [Bilofsky 88]。それに比べて配列情報がいかなる意味を持つかを探る遺伝子情報処理技術は未熟であるため、データベースに蓄えられたデータが十分に利用されてはいない。遺伝子情報処理のうちで最も基本的な技術は、配列間の類似性を調べる解析処理である。我々は、そのなかでも、マルチプルアライメントという、非常に計算量の多い解析処理に注目し、並列システムの構築を試みた [Ishikawa1 91]。まず、マルチプルアライメントについて解説する。

２つの配列を、類似する部分を同じ列に揃えて並べ合わせる操作をアライメントといい、３つ以上の配列を並べる操作をマルチプルアライメントという。たとえば、次のような性質の類似したタンパクのアミノ酸配列が３つあったとする。

```
LLDFLEQLTELSFSKMKALLERSESPYYMLERDRTLKEITETCKACAQ
VLQLSPAELESFTECGQTALTLQGATTTEASEILRSCEACRG
AYPLREAKDLETALEIGPRALSKACEISMQQAREVVQTCPECES
```

これらの文字列がアミノ酸配列である。１つの文字が１つのアミノ酸に対応する。アミノ酸は２０種類あり，２０種のアルファベットで識別される。例えば、最下段左からA,Y,P,L,Rは、それぞれアラニン、チロシン、プロリン、ロイシン、アルギニンを意味している。

これをマルチプルアライメントすると、次のようになる。

```
---LLDF--LEQLTELSFSKMKALLERSESPYYMLERDRTLKEITETCKACAQ
VLQLSPA-ELESFTECG---QTALTLQGATT-----TE--ASEILRSCEACRG
AYPLREAKDLETALEIG---PRALSKACEIS-----MQ-QAREVVQTCPECES
```

配列のところどころにギャップ“-”を入れているが、これは、その部分に対応するアミノ酸がないことを示す。この例で、C**C （＊は任意のアミノ酸を表す）などの共通文字が同じ列に並んでいるのがわかる。C**C のように複数の文字が、複数の配列で共通になっている文字の組を配列モチーフと呼び、タンパクのうちの重要な部分を指し示していると判断される。この背景には、タンパクの配列のうち、機能的、構造的に重要である部分には遺伝的変異が起きにくいという進化論的考え方 [Kimura 86] がある。

マルチプルアライメントされた結果は次のように利用できる。第一に、先に述べたように、重要な配列の部分であるモチーフを見い出して、データベースから新たな類似配列を検索する助けとできる。第二に、類似した構造を持つ配列をマルチプルアライメントした結果から、共通の部分構造に対応するアミノ酸の性質の並びがわかり、その部分構造の形態を予測する助けとできる。第三に、マルチプルアライメントから進化系統樹を描くことができ、類似配列の各々が、どのような遺伝的過程を経てきたかを推測することができる。このようにマルチプルアライメントは、遺伝子情報処理の基本技術と位置づけることができる。

タンパク中のアミノ酸はしばしば似た性質の別のアミノ

酸に置き換わるので、異なる文字でもそれらが表すアミノ酸の性質が似ていれば同じ列に置くことを許容して、マルチプルアライメントの処理は行われる。しかし、アミノ酸の性質には親水性、疎水性、極性、酸性、塩基性、大きさなど多数あり、その類似性評価も多数の方法がある。現在最も広く使われている類似性評価尺度は、Dayhoff マトリックス [Dayhoff 78] である。この尺度は、当時知られていたアライメントをもれなく調べ、同じ列に該当アミノ酸対が並んでいることが偶然に対して何如に少ないかを数値化したものである。数値は確率の対数値になっているため、それらの足し算は複合事象の共起確率を算出したことに相当する。我々もこの評価尺度を使用している。

Dayhoff マトリックスのような評価尺度が与えられていれば、それを用いたアライメントの評価基準により決まる「最適なマルチプルアライメント」を求めるダイナミックプログラミング法が知られている [Needleman 70]。この方法では、アライメントの評価基準をアミノ酸の全組み合わせに対して、Dayhoff 値および Dayhoff 値をもとに設定したギャップコストの総和をとったものとして定義している。

1.2 ダイナミックプログラミングによるアライメント

ダイナミックプログラミング（以下 DP と略す）は段階的に決定を行う特徴を持つ最適化問題を解くためのアルゴリズムのひとつである。これを用いて「最適なアライメント」を求めることができる。なお、以下に出てくるコストとは、アライメントにおいては先に述べた評価基準に対応するものである。

最適化問題が、複数の段階で決定を行い、その各段階における決定が直前の段階の決定にのみ依存するという形式に定式化できるとき、DP を用いると非常に効率良く最適解（コスト最小の決定経路）を求めることができる。もし、この種の問題を最初に全ての場合を尽くして、そのうちの最小のものを選ぶという解法で解いた場合、指数オーダーの計算量がかかる。しかし DP を用いると多項式オーダーで解を得ることができる。

アミノ酸配列のアライメントは、この DP を用いて行うことができる。簡単のために配列 2 本のアライメントについて DP の概念的説明を行う。説明のために図 1 を用いる。例えば、ADHE、AHIE という 2 つの配列をアライメントする場合、この 2 つの配列を図 1 のような 2 次元のネットワークの辺に対応させる。各アーク（矢）には、コストが割り振られる。斜め方向のアークは、そのアークの位置に対応する 2 つのアミノ酸の類似度がコストとして割り振られる。この類似度には前述の Dayhoff マトリクスを用いている。また縦および横方向のアークはギャップに対応し、ギャップを挿入するときのコストが割り振られる。こうして全てのアークにコストが割り当てられる。

このように問題を定式化すると、最適なアライメントを求めることは、このネットワーク上のコスト最小の経路を求めることに対応する。図 1 の例では、太いアークで表された経路がコスト最小となる。この太いアークで表された経路を順に見ていくと、A と A が対応し、D に対応するもう片方のアミノ酸はなく（つまりギャップが対応し）、H には H が対応し ... という具合に解釈することができる。結果として図 1 の右側にあるアライメントが得られる。

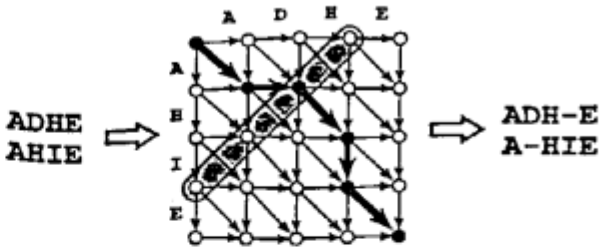


図 1 ：2 次元ＤＰマッチングによるアライメント

コスト最小の経路は左上の端から右下の端に向かって（逆でも可能）各ノードに至る最短経路を段階的に決定していくことにより求めることができる。各段階は図 1 の右上がり斜め線上に存在するノード群に対応する。段階的に各ノードへ至る最短経路を求めていくと、いったん求まった部分的最短経路はもはや変更されることがない。それゆえ、この部分的最短経路を用いて次の段階の計算を行うことができる。具体的には、ある段階の各ノードの計算を行うためには、直前の段階の各ノード（2 次元 DP では 3 つある）で求まった部分的最短経路のコストを参照して、今求めたいノードに至るコストをそれぞれ計算し、このうちの最小値を求めてそれをそのノードに至るコストとすればよい。直前のどのノードを選択したかという情報も記憶しておく。この操作を最後まで繰り返せば、ネットワーク全体の最短経路を求めることができる。

1.3 並列３次元ＤＰの意義

ダイナミックプログラミング法を用いれば最適なアライメントを得ることができる。しかしこの手法が必要とする計算量は多く、配列が 3 本以上では、これまで近似的にしか使用されていなかった [Murata1 85][Murata2 90] [Carrilo 88]。通常は配列 2 本のアライメントを繰り返してマルチプルアライメントを行う方法が試みられている [Waterman 86]。ところが、それでは精度が十分でなく、難しいところは、もっぱら熟練した生物学者の勘に頼っている。

我々はこの計算量の多い 3 次元 DP を、マルチ PSI 上で並列に動作させるプログラムを開発した [Ishikawa1 91]。プロセッサ６４台で並列実行させたところ、パイプライン効果を含め、1 台に比べ約３７倍の高速化が実現できた。すなわち、計算量が多く、今まで本格的に扱われてこなかった 3 本の最適なアライメントを、並列処理により、現実的な時間内で実行可能にしたのである。

3本の最適なアライメントができても4本以上は無理なのだから、本質的に2本のときと変わらないと考えられがちであるが、そうではない。我々は(A,B,C)と(A,B,D)という2本の配列が重複する2つのアライメント結果をつなぎ合わせて、(A,B,C,D)というマルチプルアライメントを導くプログラムを研究、開発している。このプログラムにより、配列2本のアライメントをつなぎ合わせる方法に比べ、3次元DPによるマルチプルアライメントの精度が飛躍的に向上することが確認されている[Ishikawa2 91]。

2 並列3次元DPの実装

2.1 実装環境

我々は、本システムの実装において、ICOTで開発された並列言語KL1および並列マシンマルチPSIを使用した。これらのシステムについて簡単に説明する。

KL1は、並列論理型言語であるが、そのプログラミングにおいては、一般に、プロセス指向（オブジェクト指向）のスタイルをとる。このスタイルにおいては、なんらかのメッセージを外部から受けとってアクティブに動作するプロセスと、そのメッセージを流すためのストリーム（通信路）とでプログラムが構成される。結果としてプログラムは、多数のプロセスがストリームで相互に結合されたネットワーク構造をとることになる。

マルチPSIは疎結合の並列マシンであり、現在、最大構成で64台のプロセッサを使用することができる。ネットワークのトポロジーは2次元メッシュであるが、どの2つのプロセッサ間でも、ほぼ同一のコストで通信が行える。疎結合マシンは密結合のマシンに比べて、より大きなプロセッサ構成をとることができる。しかし、その反面、通信の負荷が大きいという欠点を持つ。そのため、各応用プログラムについて、通信のオーバヘッドを抑えながら負荷の分散を行う方策を考えねばならない。

一般にKL1におけるプロセス指向プログラミングでは、プロセッサ台数よりかなり多くのプロセスを発生させるので、これらのプロセスを全体の処理時間が小さくなるように、それぞれのプロセッサに分散させる必要がある。この作業のことを負荷の分散と呼んでいる。

KL1は、その特長としてプログラムの意味とは独立した負荷分散メカニズムを持っている。これは、どのプロセスをどのプロセッサで実行するかを、そのプロセスネットワークの形状とは無関係に指定できるものである。この機能を用いて、望ましい負荷分散方法がまだよく分かっていないプログラムに関して、実験および改善を簡単に行うことが可能である。

2.2 実装方式

並列化の基本的アイデア及び実装の方法を説明する。3次元DPは図2のような3次元のネットワークとして表現できる。3次元DPによるアライメントでは各段階が図2の点線で囲まれた面の領域に存在するノード群に対応する。また2次元DPによるアライメントでは各ノードは直前の3つのノードと連結されていたが、3次元DPでは、図3のように、7つのノードと連結されている。

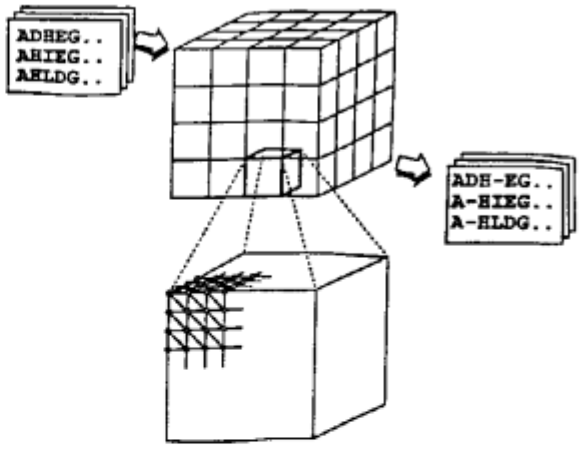


図2：3次元DPによるアライメントとプロセッサへの割り当て

並列アルゴリズムを考えるとき、並列性の有無を考えるのは当然であるが、もうひとつ注意しなければならないのは、通信のオーバヘッドである。いくら並列性が高くても、プロセッサ間の通信のコストが並列性の効果を打ち消すほど大きければ、全体として無意味なものになってしまう。幸いなことに、ここで議論しているDPは、ある段階から次の段階へ遷移するときのノード間の依存関係に局所性が存在するため、適切にプロセッサへの割り当てを行えば通信オーバヘッドを実際の計算に比べ相対的に低く抑えることができる。

さて、KL1プログラミングにおける並列の実行単位はプロセスである。したがって並列実行の主体である各ノードをKL1のプロセスで実現することにした。この各段階において計算を行うプロセスを段階が1つ進むごとに生成するのは効率が悪い。そこで3次元DPにおけるノードをKL1プロセスに、アークをKL1プロセスの通信路に、それぞれ対応させることを考えた。このように対応づけを行うと、3次元DPのネットワークをそのまま反映したKL1のプロセスネットワークを、あらかじめ生成してしまうことができる。このようにプロセスネットワークを構築すると、各プロセスが隣接するプロセスとメッセージの授受を行うことによって全体の計算が進んでいくことになる。

次に3次元DPのプロセッサへの割り当てに関して述べる。上記の方法を用いると、3次元のプロセスネットワークを3次元メッシュに分割してプロセッサに割り当てることにより、並列実行を行うことが可能である。各段階の処理が、波面状にネットワークの中を進んでいくのである。

図2では縦、横、高さ、各方向にそれぞれ4分割されて64個のプロセッサにネットワークが割り当てられた例が示

されている。メッシュに分割された各領域は各プロセッサが対応している。各プロセッサにはノードに対応するプロセスが多数割つけられている。前にも述べたが、プロセスネットワーク自身もメッシュ構造を成しており、各ノードは近接するノードとのみ通信を行うので通信に局所性が存在する。そのため、もともとメッシュ状のプロセスネットワークを、このように相対的により大きいメッシュ状に切断しプロセッサに割り当てる方法を用いれば、通信のオーバヘッドを実際に意味のある計算に比べ相対的に小さくすることが可能である。また、この割り当て方法は、縦、横、高さ方向の切断数を容易に変えられる。

ただこの方法では、ある時点で波面を含まない部分のプロセッサは稼働しなくなってしまう。しかし、解くべき問題が複数ある場合には、それらを次々とネットワーク中に投入することが可能である。こうしたパイプライン的な並列性を導入すると、（最初と最後を除いて）プロセッサが稼働しなくなることはほとんどなくなり、プロセッサ全体の稼働率をさらに上昇させることができる。

整理すると、本並列処理方式には２種類の並列性が存在する。１つはDPの各段階におけるノード間の並列性で、各アライメントに対応する波面内の並列性ということができる。もう１つは複数のアライメントをパイプライン的に実行するときの並列性で波面間の並列性ということができる。

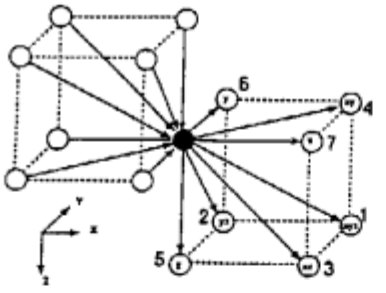


図３：プロセス間のメッセージ通信

3　並列実装の性能特性解析

並列処理システムが構築できた後には、実際に並列処理が理論どおりに行われているか、行われていないとしたらどこに問題があるかを検討するために、実験を行って性能特性解析をすることが重要である。それによって得られた処理モデルは、システムの並列処理性能の向上の手掛りになるだけでなく、他の類似の並列システムを実装するうえでの貴重な資料となる。我々も、この処理系を分析して、処理モデルを構築した結果、得られたモデルは実際のパフォーマンスの特徴をよく表現しているということが確認された。

3.1　モデルの構築

３次元ＤＰ法によるマルチプルアライメントは、その分野の要求としてパイプライン的に実行を行うことの意味は大きい。しかしここでは、より基本となる、プロセスネットワークをどのように分割すれば、ひとつの問題を解くのに最適なパフォーマンスが得られるかについてのみ解析を行った。すでに最適経路問題についてこうしたモデル構築による解析[Wada 90]がなされている。以下に具体的に行ったモデル作成について述べる。

まず、並列処理に要する時間を把握するには、全体の処理時間を決定する最も遅い処理に注目する必要がある。このような処理を全体の実行時間を律速する処理と呼ぶ。その処理の流れに着目できれば、処理時間は実際に処理を行っている時間とプロセス間の通信にかかっている時間とに分けることが可能である。ここで通信について仮定を設けた。プロセッサ内の通信は非常に小さいのに対して、プロセッサ間を渡るメッセージ通信は大きいので、プロセッサ間の通信のみを考慮するものとした。全体の実行を律速する処理のうち、プロセッサ間通信は、送り手側プロセスがメッセージをパケットに詰めて送る時間、データがプロセッサ間のネットワーク上を移動する時間、および受け手がパケットを解いてデータを読みとる時間から成る。ネットワーク上の移動時間は他の二つに比べて非常に小さいので無視したうえ、残りの二つが通信に要する時間は等しいと近似した。そして、この問題においては、すべてのプロセスから送られるメッセージはすべて、ほぼ均一なデータ型から成るので、この時間は一定の値（通信単位時間C）になるものと仮定した。

次に、プロセスネットワーク上のプロセスの実行がどのように伝わっていくかについて検討する。この実行にＫＬ１処理系による実行の指向性が反映する。図３のような、プロセスネットワーク上に仮想的に座標軸を考えることにする。各プロセスは一般に、自分の直前の７つのプロセスからの情報を全て受けとった時点で自分の計算を行い、次の７つのプロセスに情報を送る。送り先のプロセスを図のように、x、y、z、xy、xz、yz、xyz、とすると、プログラム上では普通、この順にユニフィケーションが書かれる。すると、送り先のプロセスは実行可能ならば、ゴールスタックにこの順に積まれるため、実際の実行は逆順で行われる。図に書き込まれた数字はその順番を表している。

しかし、xyz、yz、xz、xyのプロセスはさらにx、y、zのプロセスからのメッセージを待っているので、本プログラムでは、zのプロセスが最初に実行可能となる。つまり、zのプロセスに実行が移ってしまう。zは自分の処理を行った後、同様にメッセージ送信を行い、その後も同じ理由から、Z軸方向に隣接するプロセスが順次実行される。このように、１プロセッサ内では、まず、Z軸方向のプロセスが優先的に処理を行うこととなる。Z軸方向の最後のプロセスまで行き着くと、同様にメッセージを送るが、そこから先のZ軸方向は、他のプロセッサに処理がまかされる。このとき、処理を始めるためのメッセージが揃っているプロセスはxとyだけである。xとyのうち、スタックに後から積まれたyに実行

が移る。yにおいても同様に、まずZ軸方向に処理は進む。Z軸方向の最後のプロセスまで行き着くと、yに対してY軸方向に隣り合うプロセスが、次に起動される。このようにZ軸方向への一連の処理の流れがY軸方向に進んでいくことになる。Y軸方向にプロセッサの最後までZ軸方向の一連の処理が終了して初めて、X軸方向に処理が移動する。つまり、これまでに述べたZ-Y平面に平行な処理の流れが、最後にX軸方向へ進むのである。各方向に3つずつプロセスがあるネットワークが1プロセッサに割り当てられる場合、プロセッサ内の実行順序を図4に例示しておく。丸がプロセスを表し、数字は実行順序を表している。

プロセスネットワーク全体でも処理の流れには指向性があり、並列実行可能な波面が各軸に対して均等には進まないことが分かる。図5の矢印は1プロセッサ内でのZ軸方向への一連の処理を表している。添えられている数字は、矢印の予想される実行時刻である。同じ数字を持つ矢印は同時刻に並列に実行されると考えられる。ただし、プロセッサ間の通信遅延はこの時点では考慮していない。以下、この矢印1本に相当する処理を単位時間として、モデル化を進める。つまり、実行時間を「この矢印を実行する時間の何倍に相当するか」という観点で表現する。

ここで、Nx, Ny, NzをX、Y、Z方向の分割数、lx, ly, lzを各プロセッサに割り付けられるプロセスネットワークのX、Y、Z方向のプロセス数とする。図6に律速となる処理を矢印で示したが、その処理に要する時間、すなわち並列実行に要する時間Tpは次式で表せる。

$$Tp = (lx \times ly) + (lx - 1) \times ly \times (Nz - 1) + (ly - 1) \times (Ny - 1) + (Nz - 1)$$
(単位時間)

一方、全処理を逐次実行する時間Tsは次のようになる。

$$Ts = (lx \times Nx) \times (ly \times Ny) \times (lz \times Nz)$$
(単位時間)

すると、このモデルから並列実行の実計算時間Trは1プロセッサでの実行時間$T1$のTp/Ts倍となることがわかる。

$$Tr = T1 \times \left(\frac{Tp}{Ts}\right)$$

実際の実行時間Tには、通信オーバヘッドToが含まれるので、

$$T = Tr + To$$

である。

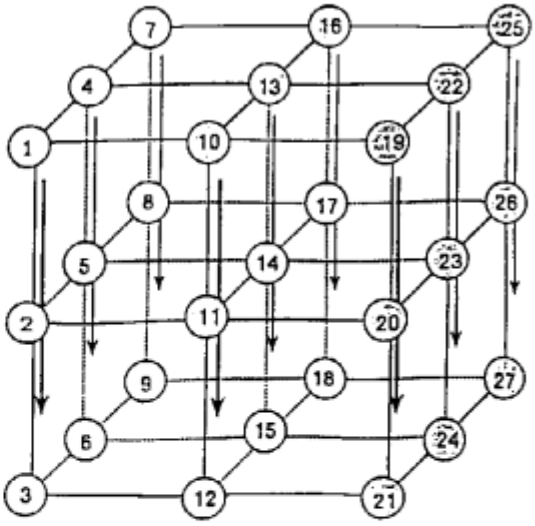


図4：プロセッサ内の実行順序

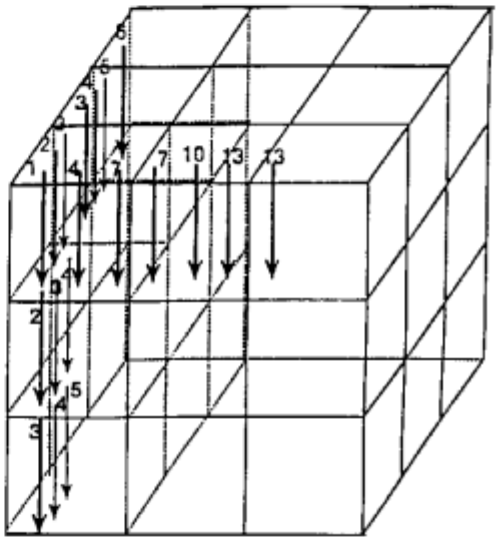


図5：全体の実行の流れ

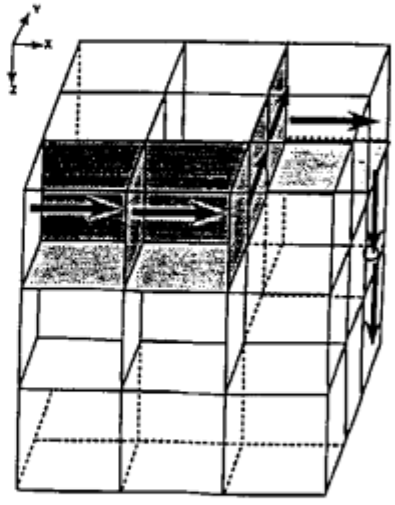


図6：律速処理と通信オーバヘッド

我々のモデルでは、図6に示したハッチングの部分に相当するプロセッサ間通信だけが最終的な実行時間に寄与する。なぜならば、本モデルにおいては、全体の実行時間を決める処理が、その処理の指向性によって、図6に示す矢印の処理に等しくなるからである。それゆえ、この律速となる処理を行うプロセッサの界面にあるプロセスの数とそれらが行うプロセッサ渡りの通信の回数により通信量は決まる。このことにより、先ほど仮定したメッセージ送信および受信に掛かる時間を通信単位時間Cとして、通信オーバヘッドは計算可能となる。前と同様に、Nx, Ny, NzをX、Y、Z方向の分割数、lx, ly, lzを各プロセッサに割り付けられるプロセスネットワークのX、Y、Z方向のプロセス数とすると、通信オーバヘッドToは、

$$\begin{aligned} To = 4C \times [\, & [lz \times (lx-1) + lz \times (ly-1) + (3/2)lz] + \\ & (Nx-2) \times [lz \times (lx-2) + lz \times (ly-1) + \\ & (3/2)lz] + (Ny-2) \times [lz \times (ly-2) + \\ & (3/2)lz] + lz \times (ly-1) + 2 \times [(Nz-2) + \\ & Nx \times lx \times ly + (Ny-2) \times ly]] \end{aligned}$$

と表される。

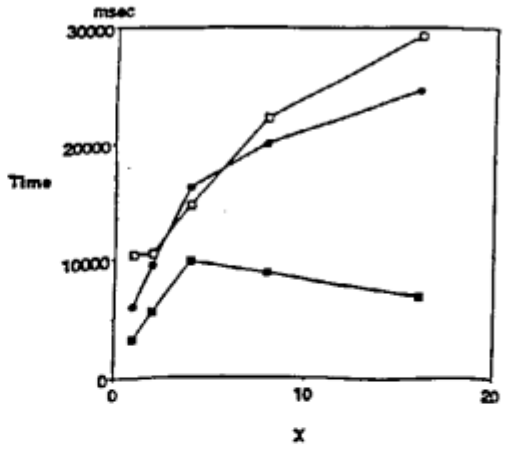
この式について、少し説明を加えると、最初にかかっている$4C$は、各プロセスの4つのメッセージ通信がプロセッサ渡りになっていることを意味している。また、それにかかる部分は図6のハッチングの部分のプロセス数、つまりプロセス間通信を行うプロセス数を表している。ただし、$(3/2)lz$というのは、6つのメッセージをプロセッサ渡りにさせるプロセス群の通信である。また、Z軸方向には、通信も並列性で相殺されるため、図5の矢印の両端の2回の通信のみ一層分だけが累積される。

3.2 実験結果

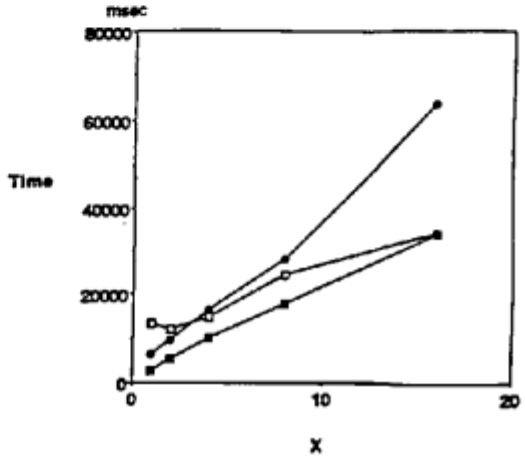
得られたモデルに従って、実験結果との比較を行い、ある程度定量的なプログラムの特徴を捕らえることができた。その結果を以下に述べる。

64プロセッサで実行するとして、X軸、Y軸、Z軸の分割の違いによるパフォーマンスの違いについて検討した。22の異なった分割に対して実験を行い、実行時間Tの計測を行った。分割しない実行時間$T1$も計測した。それらをもとにモデルからCを求め、集計したところ、1.032±0.364 msecとなり、ほぼ一定とみられる値が得られた。以下、Cの値は1.0として、解析を行った。一つの軸方向の分割を4として固定した時に、他の軸方向の分割を変化させたものの実測値とモデルから得た値のグラフを図7に示す。モデルの$C = 1$は、Cを1.0msecとして、通信オーバヘッドを考慮したグラフ、$C = 0$は通信オーバヘッドを考えない場合のグラフである。

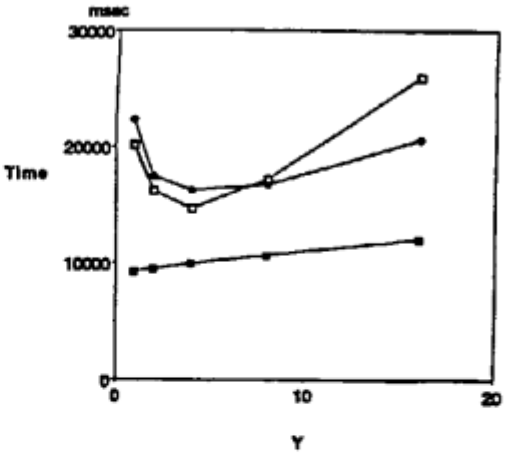
図7から分かるように、モデルから得られる値と実測値は、ある程度定量的に一致していると言える。それぞれのグ



Zの分割を4に固定したとき



Yの分割を4に固定したとき



Xの分割を4に固定したとき

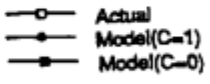


図7：X、Y、Zの分割を4に固定した時のグラフ

ラフに対して解説を加える。Y軸方向の分割を固定した場合、X軸方向を細かく分割するほど実行時間は実測、モデルともに悪くなっている。Z軸方向の分割を固定した場合も、同じく、X軸方向の分割が好ましくないことで一致を得ている。そして、X軸方向の分割を固定した場合であるが、この場合は残りのYとZ軸についてトレードオフ点が存在し、かつ、それはYとZに均等に分割したときで、実測とモデルが一致している。これらの傾向は、固定する値を変えた場合にも同様な結果が得られている。

このようなモデルの特徴から、全体のプロセッサ数を一定にしたとき、最も効果的な分割はX軸方向には分割をせずに、Y、Z軸方向を均等に分割した時だと推測できる。

そして、実験においても、64プロセッサを使用して最も速く解を得られたのは、1＊8＊8の分割でマッピングを行った場合で、まさにモデルから得られた推測に一致している。ちなみに、X軸方向の分割を1にした時のグラフが図8である。また、他のデータに対しても、上記の傾向は一致している。

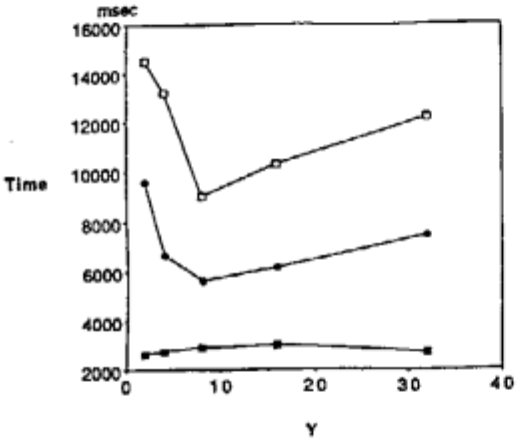


図8：Xの分割を1に固定したときのグラフ

3.3 考察

我々がプログラム設計、開発を行っていた当初は、実行は波面状で、かつ波面はX,Y,Z の各軸方向に時間的にほぼ均等に進むと予想していた。ところが、実際の実行結果には強い指向性があり、波面の進行は軸によって大きく偏りが生じた。そのため本モデルでは、実際の実行がどのように行われ、どの部分の実行が律速段階になっているかについて、踏み込んだ分析を行う必要が生じた。結果としてモデルが若干複雑になったが、プログラムの挙動をより正確に把握することができた。

本プログラムでは、1つのアライメントの実行において各ノードは必ず1回だけ実行され、無駄計算は一切生じない。それゆえ、並列化を行ったことによって生じるオーバヘッドは、通信オーバヘッドのみと考えて差し支えない。

一般に、プロセッサ稼働率と通信オーバヘッドはトレードオフの関係にある。つまりプロセッサ稼働率を上げることのみを考えれば、通信オーバヘッドの影響が並列効果を打ち消すほど強く出てしまうし、一方、通信オーバヘッドをなくすことのみに専念すると稼働率が下がってしまって、結局、実行効率は全体として悪くなってしまう。そのため、その間で最も実行効率のよいバランスのとれた点を見つけることが重要になる。

本プログラムの解析で用いたモデル、および、それから導かれる式は、プログラムの特性をよく表現しており、トレードオフ点を予測するという点でも大きな意味を持つものであった。特に通信オーバヘッドがないときの理想的理論値と、通信オーバヘッドの理論値が、分離された形で表現できたことは興味深い。

実際、図7のグラフを見ると、通信オーバヘッドまで含めて定量的にモデルを考えないと、プログラムの特性を表現するのに不十分であることがよく分かる。特に、Xの値を固定した場合とZの値を固定した場合は、通信オーバヘッドがないときの理想的理論値のグラフが、実測値のグラフと大きく異なっている。しかし、通信オーバヘッドまで含めた理論値のグラフは実測値のグラフと、かなり近いものになっていることが分かる。

並列処理を考察するうえでは、通信のない理想的なモデルだけを考えて、理論値と実測値の差を単に通信オーバヘッドの影響であると言うだけでは十分ではない。今後は、通信オーバヘッドまで含めたモデル構築を行うことが、ますます重要になると考えられる。

また本解析において、1つの式でモデルを表現できたことも興味深い。図7、8の理論値のグラフは、ただ1つの式から求められたものである。本プログラムの実行の進み方は強い指向性を持つと先に述べた。この実行の進み方の指向性はX軸方向に対して最も弱い。そのためX軸方向に細かく刻んでプロセッサマッピングを行うのは好ましくないということは、ある程度直観的に理解できる。しかしX軸方向のプロセッサマッピングを固定したときに、Y軸とZ軸の間にトレードオフの関係があるということは、実際にモデルを作って式を立てた後、理論値のグラフを書いてみて初めて分かったことである。このように本モデルを用いて、プログラムの挙動を多角的に把握することができ、プロセッサマッピングの指針を得ることができた。

4 おわりに

以上のように、適切な仮定のもとにモデルを考えれば、KL1のように高度に並列動作するプログラムでさえも、その解析は十分に可能である。今回仮定したことは、処理系などの多少の知識があれば妥当であろうと考えられるもののみである。そして、それをもとに解析した結果、モデル化が現実のシステムに即したものであることが確められた。こうしたモデルは、プロセッサへの並列マッピング方法の検討や、

並列プログラムの挙動の理解に、極めて有用なものである。

また、今回の解析によって、並列3次元DPの処理効率の向上を検討する基盤が整った。しかし、実際のシステムはパイプライン的にマルチプルアライメントの処理を行う。今回の解析では、パイプライン処理を考慮していないため、今回得られた結果をパイプライン効果をも含めた範囲に拡張することが、次の課題となっている。

参考文献

[Bilofsky 88] Bilofsky, H. S. and Burks, C. *"The GenBank Genetic Sequence Data Bank"* in *Nucleic Acids Research 16:5,* 1988, pp.1861-1863.

[Kimura 86] 木村：分子進化の中立説 1986, 紀伊国屋書店

[Dayhoff 78] Dayhoff, Hunt and Hurst-Calderone *"Composition of Proteins"* in *Atlas of Protein Sequence and Structure 5:3,* Nat. Biomed. Res. Found., Washington, D. C., 1978, pp.363-373.

[Needleman 70] Needleman, S. B. and Wunsch, C. D. *"A General Method Applicable to the Search for Similarities in the Amino Acid Sequences of Two Proteins"*, in *Journal of Molecular Biology 48,* 1970, pp.443-453.

[Waterman 86] Waterman, M. S. *"Multiple Sequence Alignment by Consensus"* in *Nucleic Acids Research 14:22,* 1986, pp.9095-9102.

[Murata 1 85] Mitsuo Murata *"Simultaneous Comparison of Three Protein Sequences"* in *Proc. Natl. Acad. Sci. USA Vol. 82,* 1985, pp.3073–3077

[Murata 2 90] Mitsuo Murata *"Three-Way Needleman - Wunsch Algorithm "* in *Methods in Enzymology Volume 183,* Academic Press, 1990, pp.365–375

[Carrillo 88] Himberto Carrillo and David Lipman *"The Multiple Sequence Alignment Problem in Biology"* in *J. Appl. Math. 48,* 1988, pp.1073–1082

[Wada 90] 和田、市吉：“マルチPSI上の最短経路問題の実現と評価”, 1990, Proceedings of KL1 Programming Workshop '90, pp.10–17

[Ishikawa1 91] 石川、星田、広沢、戸谷、鬼塚、新田、金久：“並列推論マシンを用いたタンパク質の配列解析”, 情報処理学会 情報処理基礎研究会報告 23-2,1991

[Ishikawa2 91] 石川、星田、広沢、戸谷、新田：“3次元ダイナミックプログラミングを用いたタンパク質の配列解析”, 情報処理学会 第43回全国大会論文集 3S-7, 1991