

トランスピュータ 64 台上での 並列 DCT アルゴリズムの性能評価

安原浩春

梅尾博司

大阪電気通信大学

Abstract

DCT(Discrete Cosine Transform) とは, 画像処理等によく用いられる信号処理の一つである. 我々は 3 種類のトランスピュータネットワーク上でこの DCT を実行した. 各アルゴリズムの計算量と実行時間からアルゴリズムの性能評価を行う. そして更に, 並列アルゴリズムの問題点と将来性について考える.

Performance Evaluation of Parallel DCT Algorithms on Transputer-Based Parallel Computer

Hiroharu Yasuhara

(haru@umeolab.osakac.ac.jp)

Hiroshi Umeo

(umeo@umeolab.osakac.ac.jp)

Osaka Electro-Communication University

Abstract

DCT(Discrete Cosine Transform) is one of the signal processings which are often used for an image processing and so on.

We executed this DCT on three kinds of Transputer networks.

It does the performance evaluation of the algorithm from quantity of the computation of each algorithm and the execution time.

Moreover, and then, it thinks about the problem and the possibility of the parallel algorithm.

1 はじめに

本稿では、画像処理によく用いられる 2 次元 DCT (Discrete Cosine Transform) を 64 台のコンピュータ上で行ない、アルゴリズムの評価を行なう。

本稿は以下のような構成をとる。

- 2 節 … DCT の概要
- 3 節 … 本実験で用いた並列 DCT アルゴリズムの簡単な説明と理論上の時間計算量を考察する。
- 4 節 … 前節で考察した時間計算量に含まれていない通信量について考察を行なう。
- 5 節 … 実験結果と理論値との比較検討をおこなう。
- 6 節 … まとめ

2 DCT について

本節では、この実験で用いた DCT について説明する。入力画像はサイズ 256×256 の白黒 256 階調で、これを 8×8 のサブブロックに分割し 2 次元 DCT を行なう。

一般的に $N \times N$ の正方行列の 2 次元 DCT は、入力データを $f(x, y)$ とし出力データを $F(x, y)$ とすると、以下のように定義される。

$$F(u, v) = \frac{2}{N} \alpha(u) \alpha(v) \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i, j) \cdot \cos \frac{(2i+1)u\pi}{2N} \cos \frac{(2j+1)v\pi}{2N}$$

但し、

$$\alpha(w) = \begin{cases} \frac{2}{\sqrt{2}} & w = 0 \\ 1 & \text{otherwise} \end{cases}$$

この式から、出力データ 1 点に対する逐次的な 2 次元 DCT の時間計算量は $O(N^2)$ となるのは自明である。入力データのサイズを $n \times n$ とすると時間計算量の合計は $O(n^2 N^2)$ となる。

3 並列 DCT アルゴリズムの時間計算量

ここでは、通信に関しては全く考慮しない、すなわち通信時間は 0 という理想的なモデルで考える。通信時間に関しては次節で述べる。

3.1 1 次元メッシュ上での DCT

P 個のプロセッサを 1 列に並べて接続したのが 1 次元メッシュ結合である (図 1 参照)。このネットワーク上で DCT のパイプライン処理を行なう。2 次元 DCT では 1 点に対する計算量は N^2 となる。パイプライン処理ではこの N^2 の処理を各プロセッサに均等に分配する。すなわち 1 点に対する各プロセッサ内での計算量は $O(\frac{N^2}{P})$ となる。よって n^2 個のデータ全てが処理されるには $O(N^2 + \frac{N^2}{P}(n^2 - 1)) = O(N^2 + \frac{N^2 n^2}{P})$ の時間が必要である。但し通信に関しては一切考慮していない。

今回の実験のように $P = N^2$ の場合には $O(N^2 + n^2)$ となる。しかし一般的に N^2 は n^2 と比べて充分小さい¹ので $O(n^2)$ と考えられる。

3.2 2 次元 8 段パイプライン上での DCT

64 台のプロセッサを 8 台ずつ並べて 8 本の 1 次元メッシュを作り (図 2 参照) 8 本同時にパイプライン処理をおこなう。ここでは 1 回の DCT を $N (= 8)$ 個の工程に分けて

¹本実験では $n = 256, N = 8$ である。

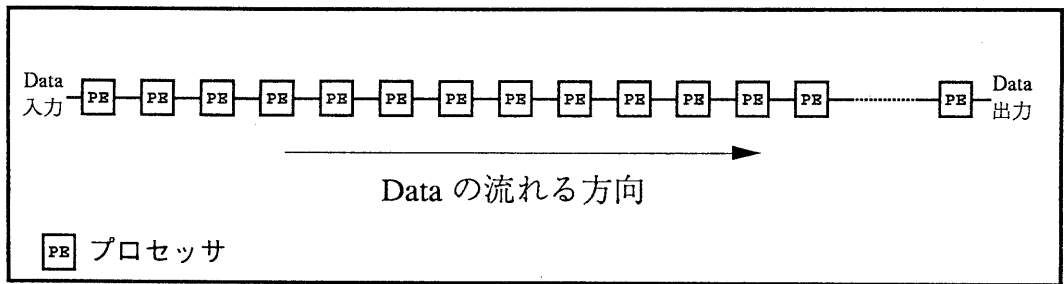


図 1: 1次元メッシュ結合

いるので各プロセッサの1点に対する計算量は $O(N)$ にかかる, しかし各パイプラインが処理するデータは $\frac{n^2}{8}$ になる. よって全体では $O(N^2 + \frac{n^2}{8} \times N)$ となり, $N = 8$ なので1次元メッシュ上での DCT と理論上は同じ $O(N^2 + n^2) \simeq O(n^2)$ になる.

3.3 2次元メッシュ上での DCT

前節と同じ結合を用いて各プロセッサに均等にデータを与え各々のプロセッサは与えられたデータ数分の DCT の計算を行なう. プロセッサ数を P とするとプロセッサに与えられるデータ数は $\frac{n^2}{P}$ となり計算量は $O(\frac{n^2}{P} \times N^2)$, ここでは $P = N^2$ なので $O(n^2)$ となる.

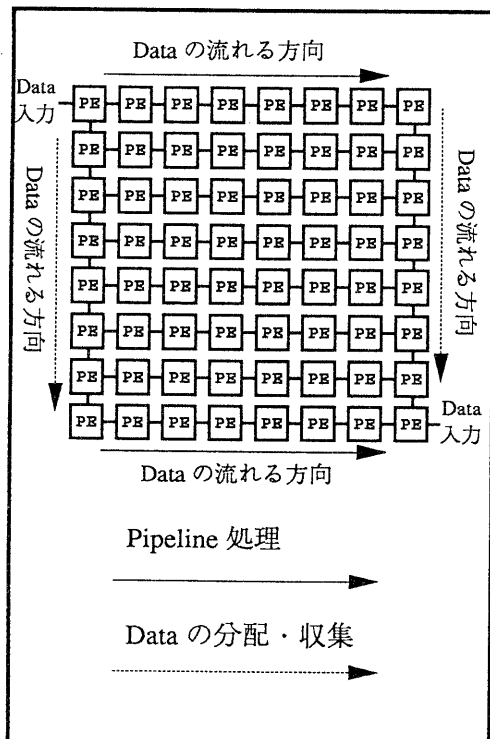


図 2: 2次元8段パイプライン

4 通信量

前節で考えたアルゴリズムの評価法では, プロセッサ間の通信時間については全く考慮をしなかった. しかし実際に並列計算機を用いて並列プログラムを動かす場合に, この通信時間を無視する訳にはいかない. すなわち時間計算量が同じならば, より通信量の少ないアルゴリズムがよいアルゴリズムとなる.

4.1 1回の通信時間

1回の通信にかかる時間は大きく分けると次の3つに分けることができる.

転送オーバーヘッド時間 データを転送するには, 転送用の初期設定や終了処理が必要なのはである. この時間はデータの転送を行っていないため完全なオーバーヘッドとなる. ここでは, この時間を転送オーバーヘッド時間と呼ぶ

ことにする。この転送オーバーヘッド時間は転送を行なう度に必ず1回必要となりこれにかかる時間は転送データのサイズにかかわらず1定になると考えられる。つまりアルゴリズム内で行なわれるデータ転送の回数に比例する。

転送時間 データの転送が始まってから転送が終了するまでの時間を転送時間と呼ぶ。この転送時間は転送するデータのサイズに比例する。

同期待ち時間 データ転送を行なってもデータを受け取る側のプロセッサが受け取る準備ができていない場合には受け取り状態になるまで待たなければならない。この時間を同期待ち時間と呼び、アルゴリズムの最適化によって無くすことができるはずである。

4.2 アルゴリズム全体での通信時間

上に述べた3つからアルゴリズム内の全通信時間は以下ようになる。

転送オーバーヘッド時間を T_o 、転送時間を T_r 、同期待ち時間の合計を T_i とし、サイズ k のデータを x 回転送するのにかかる時間は、

$$xT_o + xnT_r + T_i = x(T_o + kT_r) + T_i$$

となる。

データ量 k と転送回数 x は一般的に独立ではない。例えば、転送回数 x を減らすと一度に転送するデータ量 k が増える。次に、先に示したアルゴリズムの通信量について考察する。

4.3 各アルゴリズムの通信量

同期待ち時間に関しては実際に動かさなければ判りにくいので通信の量だけを考える。

4.3.1 1次元メッシュ

ここでは、パイプライン処理を行なっているので転送回数 x はプロセッサ数を P データ数を n^2 とすると $x = P + n^2 - 1$ となる。1度に送られるデータ数は1つなので全体での通信時間は

$$(P + n^2 - 1)(T_o + T_r)$$

となる。また一般的には $n^2 \gg P$ が成立するので $n^2(T_o + T_r)$ と近似できる。

4.3.2 8段パイプライン

8個のパイプラインを並列に動作させるために1つのパイプライン内のプロセッサ数は $\frac{P}{8}$ 個、データ数は $\frac{n^2}{8}$ 個なので $x = \frac{P}{8} + \frac{n^2}{8} - 2$ となる。

図3にデータの流れと1度に転送されるデータ量を示した。矢印がデータの流れを表し数字が転送されるデータ数を表す。各パイプラインの先頭ではデータを分配し、終端ではデータを集める、そのための通信回数は、16回で、各通信ごとに送られるデータ数は、 $\frac{i}{8}n^2 (i = 1, 2, \dots, 8)$ となる。よって1度に送られるデータ数の平均値は

$$\frac{1}{16} \times 2 \sum_{k=1}^8 \frac{k}{8} n^2 = \frac{9}{16} n^2$$

となる。

全体では、

$$\left(\frac{P}{8} + \frac{n^2}{8} - 1\right)(T_o + T_r) + 16\left(T_o + \frac{9}{16}n^2T_r\right)$$

となりここでも $n^2 \gg P$ から $\frac{n^2}{8}(73T_o + T_r)$ となる。

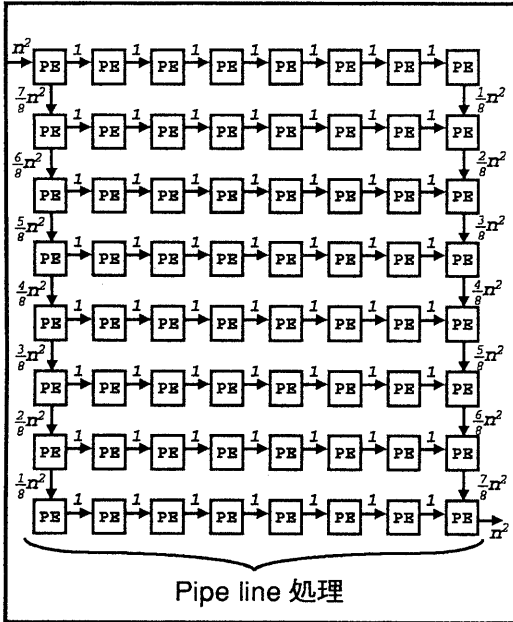


図 3: 2次元8段パイプラインの通信量

4.3.3 2次元メッシュ

このアルゴリズムでは最初に各プロセッサへデータを分配しその後に演算、そして、結果を集めて出力する。このために1度に出力するデータ数は多くなるがデータの転送回数は少なくなる、つまり転送オーバーヘッド時間が小さくなる。データの転送回数は $2(2\sqrt{P}-1)$ とデータ数に依存しない。また、データの転送量であるがデータの転送経路は図3と同じで、まず縦方向にデータを $\frac{n^2}{8}$ づつ分配する。よってこの部分の転送データ数は $\sum_{k=1}^8(\frac{k}{8}n^2)$ である。次にそのデータを受け取ったプロセッサはその受け取ったデータを更に $\frac{1}{8}$ (即ち $\frac{n^2}{64}$) づつ横方向に分配する (図4参照)。このための転送データ数は $\sum_{k=1}^8(\frac{k}{64}n^2)$ となり、データの分配と収集の両方を考えると全体での転送データ数は

$$\sum_{k=1}^8 (\frac{k}{8}n^2 + \frac{k}{64}n^2) = \frac{9}{64}n^2 \sum_{k=1}^8 k$$

となる。これを通信回数で割った値が1回あたりの転送データ数になる。よって全体での通信量は

$$2(2\sqrt{P}-1)(T_o + (\frac{9}{64}n^2 \sum_{k=1}^8 k)T_r)$$

$$= 2(2\sqrt{P}-1)T_o + (\frac{9}{64}n^2 \times 36)T_r$$

となる。

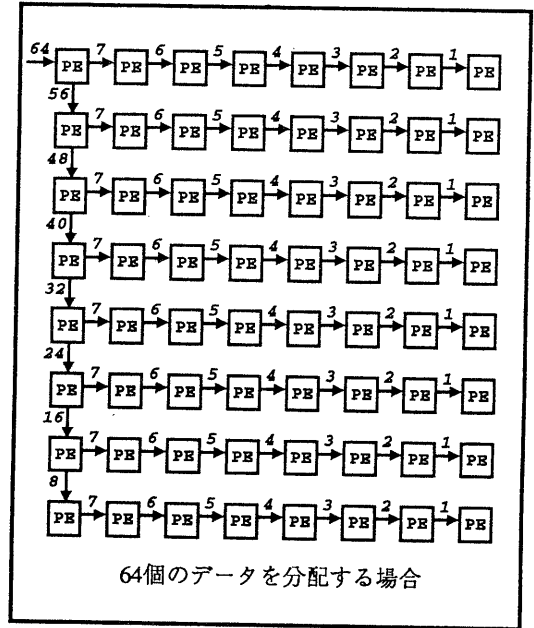


図 4: データの分配

5 実験の結果と考察

以上のアルゴリズムを実際のトランスペュータ上で実行した結果が次の表である。

ネットワーク	処理時間
1次元メッシュ	10.93 sec
8段パイプライン	4.96 sec
2次元メッシュ	3.58 sec

表 1: 実際の処理時間

1次元メッシュでの結果と2次元8段パイプラインでの結果を比べると約6秒の差がある。計算の時間自体は同じはずなので²この6秒は通信時間の差であると考えられる。通信時間は1次元メッシュの方が8倍かかるので1次元メッシュが $\frac{48}{7} \approx 7$, 2次元8段パイプラインが $\frac{6}{7} \approx 1$ となり実際の計算時間は4秒程度であると考えられる。つまり1次元メッシュでの処理時間は60%以上が通信のための時間となる。逆に2次元メッシュの方はほとんど通信時間はかかっていないことになる。

そこで実際に同じプログラムで計算の部分コメント・アウトして時間を測定した。以下がその結果である。

ネットワーク	通信時間
1次元メッシュ	10.87 sec
8段パイプライン	1.55 sec
2次元メッシュ	0.25 sec

表 2: 実際の通信時間

この結果から実際は1次元メッシュでは処理時間の99.5%が通信時間であることが判った。他の2つはほぼ期待通りの結果である。

1次元メッシュでのアルゴリズムでは1つのプロセッサではデータを受け取ってからそ

²正確にはループのための処理などのために2次元8段パイプラインの方が時間がかかる

のデータに対して行なう処理が1回のコサインの計算だけなので処理時間のほとんどが通信の時間となる。そこで1度に転送するデータ数を増やしてデータの転送回数を減らしデータを受け取ってから演算処理を多くしてみた以下にその結果を示す。

データ数	処理時間	通信時間
1	10.93 sec	10.87 sec
2	5.98 sec	5.97 sec
4	5.71 sec	3.48 sec
8	5.70 sec	2.31 sec
16	6.20 sec	1.64 sec
32	7.20 sec	1.38 sec
64	8.14 sec	1.44 sec

表 3: 転送データ数の変化に対する処理時間の変化

この結果から判ることは、トランスピュータでは転送オーバーヘッド時間 T_c を無視できないということである。つまりデータ転送は転送回数を少なくして転送データ数を大きくした方が良いということである。

6 終りに

本稿で行なった実験から、並列処理を行なうにはできる限り通信量を少なくしてできるだけプロセッサごとに独立な処理を増やすことが必要であることが判った。すなわち、ここで用いたDCTの様に完全に独立なプロセスに分解できる問題なら初めにすべてのデータを分配し各プロセッサの処理が終了後にすべての結果を集めるといった方法がもっとも効果的であるということになる。このことは通信を主とするアルゴリズムに対して非常に

不利である。また数万個或いは数十万個というプロセッサを集めて作られる。超並列計算機を考えるうえで非常に障害となるものである。しかし将来的にはウェーハ・スケール・インテグレーション等の開発によりプロセッサ内のデータ転送速度とプロセッサ間のデータ転送速度が同じくらいになり通信を基本としたアルゴリズムが主流となるだろう。

謝辞

本研究の一部は文部省科研費 (No.03680035) より補助を受けている。

参考文献

- [1] 大町隆夫, 大野文孝, カラー静止画像符号化国際標準方式 (JPEG) の概説 (その1), 画像電子学会誌, 第 20 巻, 第 1 号, 1991
- [2] R. S. Cok, Parallel Programs for the Transputer, Prentice Hall, 1992