数理解析研究所講究録
第483巻 1983年 164-180

FFTのM-200H/280H IAP向けコーディング

筑波大物質工学系　吉野さやか (Sayaka Yoshino)

一つ前に唐木氏により、東大大型計算機センターのM200H/280Hアレイプロセッサ（以下IAPと略す）についての概略が話されたが、本講演では使う立場からみたIAPについて、次の2つの例をとりあげて報告した。一つは高速フーリエ変換（以下FFTと略す）、もう一つは大次元行列の固有値問題である。実際の計算ではほとんど現われないような場合（ループ長が非常に大きい）のアレイプロセッサによる高速化ではなく、現実的な問題での高速化の例としてとりあげたものである。

唐木氏によりIAPを有効に活用するための注意がいくつか与えられたが、そのコツは次の3つにつきるといってよい。

(i) ループ長を大きくする。

(ii) 連続アクセス（逆方向でも可）をするようにする。

(iii) 一度に乗算、加減算両方を一緒にやってしまうベクトル

命令を活用する。具体的には次の4種類の命令を活用することである。

(a) いわゆる積和： FPR ← FPR ± $x_i * y_i$

(b) いわゆるはきだし(?)： z_i ← z_i ± $x * y_i$

(x, x_i, y_i, z_i はストレージ上の領域、FPRは浮動小数点レジスタの意味。)

前の2つはどのような種類のアレイプロセッサにも共通の要請であろうが、3番目のものはM200H/280Hに特有のものでこれの活用が一つのポイントとなる。特に(b)のはきだしはループ内で毎回ストアを伴なうことから、かつては高速計算機では嫌われたことなので注意を要する。この3つのことに注意してコーディングしていけばIAP向きのプログラムができあがるわけであるが、ちょっとうっかりするとIAPを有効に働かせられない場合がでてくる。本稿でとりあげた2例はそれに該当すると思う。前者(FFT)はアルゴリズム自身に問題がある(FFTそのものが悪いのでなく、IAPにとって好ましくない)例で、後者(大次元固有値問題)はI/OがからんできてそれによってIAPの高速性がそこなわれる例である。表題がFFTにのみ触れているのは、最初の予定では前者についてのみ話しをするつもりであったためである。

さてM200H/280H IAPの性能であるが、ループ長が10^4

とか10^5とかいう場合はいざ知らず、普通に使って（ループ長~100）いればベクトル命令で1個のベクトル要素を処理するのに要する時間は大体次の通りである。

M200H　約80 ns.（単精度）、約140 ns.（倍精度）

M280H　約60 ns.（単精度）、約100 ns.（倍精度）

これは1ベクトル命令に対する時間であり、前述した積和、はきだしの場合には、1個の積または加減算あたりの時間はこれの半分になるわけである。ある計算手順に必要な演算量を勘定し、この数字を使えばその処理に要する時間を見積ることができる。この見積りと実際の計算に要した時間とを比較してIAPを有効に活用したか否かの判断材料とする。IAP使用時と不使用時（以下不使用時のことをNOIAPと略す）との計算時間の比を測って判断したのでは、思いがけないところで無駄な計算をやっていたということにもなりかねないからである。

[1] 高速フーリエ変換（FFT）

ここでいうFFTとは$N=2^n$個の実データに対するFFTのことである。FFTのアルゴリズムはよく知られているように、原データをf_k（$k=0,1,2,\cdots,N-1$）、そのフーリエ変換：$\tilde{f}_l$（$l=0,1,2,\cdots,N-1$）を、

$$\tilde{f}_l = \sum_{k=0}^{N-1} f_k \exp\left(\frac{2\pi i}{N} kl\right)$$

と定義するとき、

$$g^{(m)}(k,l)=\sum_{j=0}^{2^{m-1}} f_{k+2^{n-m}j}\exp\left(\frac{2\pi i}{2^m}jl\right)\quad\begin{pmatrix}k=0,1,\cdots,2^{n-m}-1\\ l=0,1,\cdots,2^m-1\end{pmatrix}$$

とおけば、次の漸化式、

$$\begin{cases} g^{(m)}(k,l)=g^{(m-1)}(k,l)+\exp\left(\frac{2\pi i}{2^m}l\right)g^{(m-1)}(k+2^{n-m},l)\\ g^{(m)}(k,l+2^{m-1})= \quad 〃 \quad - \quad 〃 \end{cases}$$

が成り立つから、$g^{(0)}(k,0)\equiv f_k$ から出発し、この漸化式を n ステップ繰り返して $g^{(n)}(0,l)\equiv \tilde{f}_l$ を得ようというものである。もちろん実際の計算では実数を使って計算し、結果はフーリエ余弦、正弦変換として与える。この変換に必要な計算量は、実数の乗算、加減算を各々1回と数えて、$\frac{5}{2}nN$ 回となることは容易に示せる。しかし以下では簡単のため、複素数の $g^{(m)}(k,l)$ の形のまま考えてみることにする。

上の漸化式からわかるようにFFTの計算手順はFORTRANのdoループの形を使えば次の2通りのやり方がある。

(a)
```
do 20 m=1,n
do 10 l=0,2**(m-1)-1
do 10 k=0,2**(n-m)-1
---
10 continue
20 continue
```

(b)
```
do 20 m=1,n
do 10 k=0,2**(n-m)-1
do 10 l=0,2**(m-1)-1
---
10 continue
20 continue
```

(a)では最内側ループのループ長は $N/2, N/4, \cdots, 2, 1$ と変

化し、1つ外側のループでそのループを 1, 2, ⋯, N/4, N/2 回繰り返すという形をしており、(b) ではその逆である。余りループ長が短いと IAP は NOIAP よりも時間がかかるのでループ長がある大きさ以上では IAP それ未満では NOIAP で計算するようなプログラムを作り、その所要時間を測ってみると図1のようになった。(縦軸は大きさをそろえるために N で割っている。) 例えば横軸16でのプロット値は、ループ長16以上 IAP、8以下 NOIAP で計算してみたときの CPU

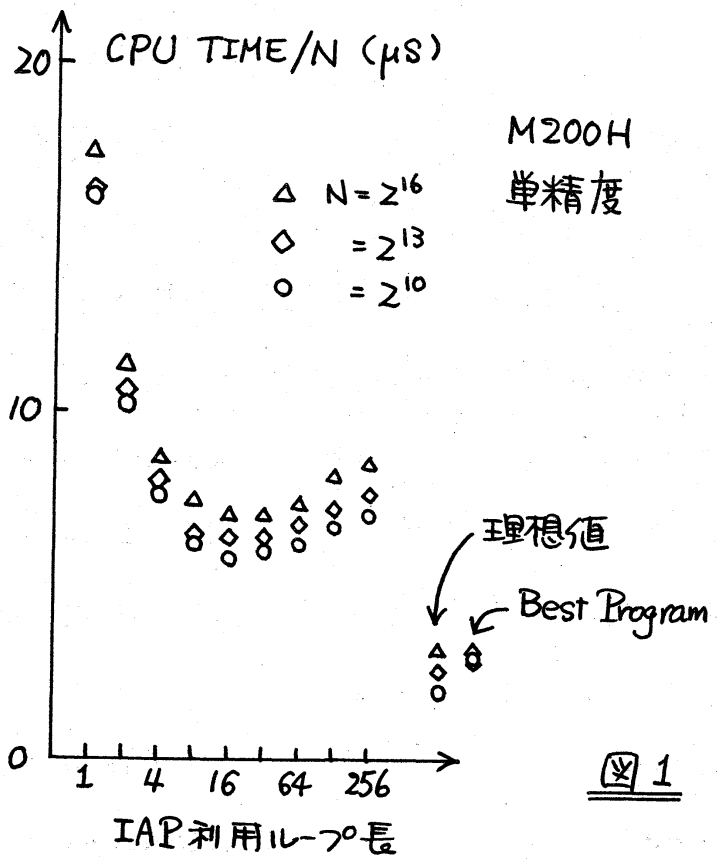

図1

時間を示している。これをみるとループ長16を境にしてIAP、NOIAPを使いわけるのが計算時間最小であることがわかる。これは唐木氏が前の講義で述べられたように、IAP、NOIAPでの計算時間が交叉するループ長~10であることから考えてみても妥当なものである。しかしながら、これでは決してIAPを有効に活用したことにはなっていない。全計算量~$\frac{5}{2}nN$から見積った値を図1の中に理想値として示しているが、この値からのズレは余りに大きすぎる。$N=2^{16}$の場合でも2倍以上時間がかかっているわけである。この結果は実は後述する解析をしてみると当然のことであるが、ここではまず、最内側ループの平均回数という観点からFFTのIAPへの適合性をみてみよう。

$$\text{最内側ループの平均回数} = \frac{\text{最内側ループの全実行回数}}{\text{1つ外側のループの全実行回数}}$$

であるから、前述の(a)、(b)どちらのやり方をしても、この平均回数は~$nN/\sum_{m=1}^{n} 2^{m-1} \fallingdotseq \log_2 N \sim 10$となる。この値は複素数の$g^{(m)}(k,l)$の形のままで考えたのでこの程度の大きさになったが、実数で計算してみた場合にはその1/4になるので、結局最内側ループの平均回数は2~3回となり、全部IAPでやってみようというのは全く不適切であることがわかる。こう考えてくると、IAP向きにFFTを実行するにはどう

すればいいかは自然に思いつく。要するに最初の方のステップでは(a)の形の繰り返しをやり、途中($m \fallingdotseq n/2$)で(b)の形の繰り返しに切り換えればよいわけである。連続アクセスという観点から、当然$g^{(m)}(k,l)$を1次元の形に並べるときの並べ方もそこで切り換わり、並べ換えが必要になるが、1度やるだけのことだからたいした負担にはならない。このようにすれば最内側ループの回数は$\sqrt{N}$以上になり、その平均回数は$\sim nN/2\sum_{m=1}^{n/2}2^{m-1} \fallingdotseq (\sqrt{N}/2)\log_2 N$となって(その1/4をとってみても)十分大きくなる。このやり方でコーディングしたプログラムによる計算時間を図1にBest Programによる時間として示してある。理想スピードに充分近い速さが得られていると言ってよいであろう。蛇足かもしれないが、最初にのべたはきだしの型のベクトル命令が一部に使用できて、ベクトル命令としての全演算回数は少し減って$\sim\frac{9}{4}nN$回となる。

最後にもう少し定量的に計算時間の見積りを行なってみることにする。長さlのループを実行したときの所要時間は、NOIAPではlにほぼ比例してclと書けるが、IAPでは準備のための時間が付け加わって$a+bl$の形になる。ここでM200H単精度では$a\sim1.5\mu s$, $b\sim0.06\mu s$, $c\sim0.25\mu s$である。この値を使ってIAP、NOIAPをループ回数によって使いわけ

たとき（図1）の全計算時間を見積ってみよう。最内側ループの回数が $N/2, N/4, \cdots, 2^p$（要するに $n-p$ ステップ分）ではIAP、残りpステップはNOIAPで計算したとすると、全実行時間Tは、第1項IAP、第2項NOIAPによる部分として、

$$T = \sum_{k=1}^{n-p} (a + b\frac{N}{2^k}) \cdot 2^k + cpN$$

と書ける。したがって全部IAPでやると（$p=0$）、

$$T_{IAP}/bnN = 1 + 2a/bn \sim 1 + 50/n。$$

最適のpをとってみる、すなわち $\frac{\partial T}{\partial p}=0$ なるpでの値を T_{min} とすると（pは整数なのだが、ラフな見積りとして気にしない。）

$$T_{min}/bnN \sim 1 + 15/n$$

となる。理想的な場合（ループ長∞）では $T/bnN=1$ であるので、常識的なNの値では T_{min} でも理想的な場合よりも2倍以上時間がかかる。途中で繰り返しのやり方を切り換える（最適の）方式では、

$$T_{best}/bnN = 1 + 4a/bn\sqrt{N} \sim 1 + 100/n\sqrt{N}$$

となり、T_{min} よりもはるかに改善されていることがわかる。

以上述べたように、FFTのアルゴリズムは気軽にコーディングしただけではIAPに全く適合しない。しかしちょっと工夫をするだけで充分にIAPの活用が可能となる。FFTは特別な一例かもしれないが、他のアルゴリズムでもこのような工

実がないためにIAPの性能を殺している場合がないとは言えないであろう。これからの科学技術計算の高速化は、アレイプロセッサの活用によるものとなることはほぼ確実であるだけに、この方面の配慮は充分になされるべきであろう。

[2] 大次元行列の固有値問題

2番目の例は行列の固有値問題であるが、ここで行列は帯行列になっている場合を考えている。これは7年程前から当時はHITAC 8700/8800を使ってやっていた計算を、IAPを使ってやってみようというものである。対象となる行列Hは実対称で、その次元をN、半帯幅をmとする。どの程度の大きさの行列を扱いたいかというと、$m=100$、$N=10^4$ 程度はやりたい。ただし、全部の固有値、固有ベクトルが欲しいというわけではなく、ある値近傍の10個程度（最大あるいは最小固有値からいくつかというのではなく、例えば$N=10^4$のときに小さい方から数えて4000番目付近の固有値の近くで、近い方から10個程度）の固有値および固有ベクトルを、固有ベクトルの精度も十分に良く求めたいというものである。大次元になると当然倍精度計算が必要だから、$N=10^4$, $m=100$だと行列要素だけで8MB必要になるが、HITAC 8700/8800システムでは最大1MB程度の主記憶（仮想記憶としての、

の意味）が通常は上限(*)であったので、当然磁気ディスク等の二次記憶へのI/Oが不可欠となる。M200H/280Hシステムでも最大限(*)7MB程度なので、やはりI/Oは必要である。このI/Oのおうさ(・・)とIAPのスピードとのからみを調べてみようというのが主たる興味の対象である。

さてここで、大次元行列を扱うときに、その行列の性質の判定について一言コメントしておきたい。大次元行列を扱う場合、その行列というのはの次元空間での演算子を近似として有限次元空間に射影した表現であることが、ほとんどすべてであろうと思われる。したがって問題としている大次元行列の性質とは、もともとのの次元空間での演算子の性質、特にその固有値のスペクトル分布の様子で分類すべきものであろう。もちろんスペクトル分布が完全にわかっているはずはないが、少なくとも今求めたいとしている固有値近傍で

---

*)上限というのは東大計算機センターの運用上の制限であって、ハードウェア上の制限ではない。HITAC 8700/8800では31ビットアドレスであったのでハードウェア上の制限はないに等しかった。これに対し、M200H/280Hは24ビットアドレスでOS空間を考えると、7MBというのはハードウェア上の制限といってよい。

のスペクトルがだいたいどうなっているかは、物理的な考察からわかるはずである。特に次の3つの分類のうちどれに属するかぐらいは。その3つというのはリゾルバント、

$$\lim_{N\to\infty}\left[\text{const.}\times \mathrm{Tr}\frac{1}{z-H}\right],\quad (\text{const. は適当な規格化定数})$$

を考えたとき、z平面上の求めたい固有値の値付近で、①固有値が極になっている、②固有値が密集してブランチ・カット（あるいは自然境界？）を形成している、③固有値に対応する極のすぐ近くに集積特異点がある、の3通りである。もちろん性質としては①、②、③の順に悪くなっていく。0の近傍を考えて、①の例として、

$$a_{ij}=\begin{cases} i & \text{for } i=j \\ 1 & \text{for } |i-j|=1 \\ 0 & \text{otherwise} \end{cases},$$

②の例として、

$$a_{ij}=\begin{cases} 0 & \text{for } i=j \\ 1 & \text{for } |i-j|=1 \\ 0 & \text{otherwise} \end{cases},$$

③の例として、

$$a_{ij}=1/(i+j-1),$$

が考えられる。実際の問題として、③の場合が現われるのはごくまれではないかと思う。

今、解こうとしている行列は、②の範疇に属するものである。この場合、欲しい固有値を0の近傍だとして、∞次元で

のスペクトルの密度 $\rho(a)$ が定義され、N次元の行列Hはaの近傍で $\sim 1/N\rho(a)$ 間隔で固有値が並ぶことになる。∞次元で連続スペクトルになっているのだから、欲しいのは固有値でなく固有ベクトルの方である。

次にこの固有値問題をどう解くかであるが、スペクトルの端の方から求めるというわけではないので、$(a-H)^{-1}$ を扱うことは恐らく必須であろう。筆者はいろいろと試みた結果、$(a-H)^{-1}$ にLanczos法を使い、かつその不安定性から逃れるため再直交化を毎回全部の基底との間にやるのが最適であろうと考えている。再直交化を完全にやることは、必要な記憶域を増大させ、また計算の局所性を壊すとして嫌われることが多いが、後で示すように今のような場合には大した負担にはならない。$(a-H)^{-1}$ を使うというのは、実際にはあらかじめ $(a-H)$ をSQRTなしのコレスキー分解しておくというやり方をするので、全計算に必要な演算量は次のようになる。また同時に必要なI/Oの回数も示しておく。

① コレスキー分解	(演算) $\frac{1}{2}m^2N$	(I/O) $3mN$
② Lanczos法による基底生成	(演算) $2pmN$	(I/O) $2pmN$
③ 再直交化	(演算) p^2N	(I/O) case by case $p \to$ 大では $\frac{1}{2}p^2N$

ただしpは生成された基底ベクトルの数である。演算回数の単位は、積和1回（乗算1回と加算1回）を演算1回と数え、I/O回数の単位は、変数1個分（8バイト）のI/Oを1回と数えたもの（したがってブロックのI/O回数はこれより3ケタ以上小さい）のことである。

実際に計算を実行してみた結果はすこぶる満足すべきものであった。まず、必要な基底ベクトルの数pについては、10個程度の固有ベクトルを求めるのにたかだか30程度であり、またpの値は次元数Nにほとんど依存しない。これはN次元行列 $(\alpha-H)^{-1}$ のスペクトルの端の方では離散的（前述の分類で①のタイプ）でかつその離散の仕方はNに余り依存しないはずであるから当然のことであろう。したがって毎回再直交化を行なっても大きな負担にはならない。むしろ精度およびpの値を大きくしないという点から得るところの方が大である。前記の演算量、I/O量の評価からわかるように、半帯幅 $m\sim 100$ のときには、$p\sim 30$ 程度だから再直交化の部分は演算量、I/O量双方の点で全体に占める割合は小さい。

次に固有ベクトルの精度についてである。次の不等式はすぐに得られる。

$$1-|(w_c, w_t)|^2 \leqq \|(E_c-H)w_c\|^2/\Delta^2.$$

ここで E_c は計算された固有値、w_c, w_t は各々計算されたおよ

び真の固有ベクトル、Δはすぐ隣の固有値との差である。この不等式の右辺は計算結果からすぐに評価できるから、左辺すなわち他の固有ベクトルの誤差としての混入量の上限を見積ることができる。例えば、他の固有ベクトルの混ざりをε以下にするには、$\|(E_c-H)\psi_c\| < \sqrt{\varepsilon}\Delta$であれば十分である。以下に計算結果の一例をあげておく。これは$N=10^4$, $m=100$, $\rho(a)\sim 0.12$（すなわち平均固有値間隔$\sim 8\times10^{-4}$）の場合で、E_c-aの相対誤差の最大が10^{-8}以下になったところで計算を打ち切り、このとき基底ベクトルの数pは28になっていた。

E_c-a	$\|(E_c-H)\psi_c\|$
3.6×10^{-4}	3×10^{-12}
5.3×10^{-4}	1.7×10^{-11}
-7.3×10^{-4}	1.0×10^{-11}
-1.16×10^{-3}	7×10^{-12}
1.63×10^{-3}	4×10^{-12}
-2.11×10^{-3}	7×10^{-12}
2.34×10^{-3}	1.5×10^{-10}
2.37×10^{-3}	8×10^{-10}
-2.83×10^{-3}	3×10^{-9}
4.02×10^{-3}	3×10^{-6}

この一例からもわかるように、固有ベクトルの精度も充分良

く得られている。

演算量、I/O回数を前述の評価式から見積ってみると、演算量～1.2×10^8（単位は1回の積和）、I/O回数～7×10^7（単位は8バイト分のI/O）となる。I/Oは1トラック1ブロックで行なうとすると、HITAC8800では積和1回に0.8～1μS，8バイト分のI/OにCTIMEで～0.3μS、ETIMEで～30μSを要したが、M280H(IAP)ではこれらの値が各々、0.1μS，0.1μS，20μSになっている。算術演算のスピードアップに比べてI/Oの方は少ししか速くなっていない。このため計算所要時間は、HITAC8800でCTIME～130S.（うち算術演算に110S.）、ETIME～85min，M280H(IAP)ではCTIME～18S.（うち算術演算に12S.）、ETIME～23min.と見積もられ、また実際に好環境下ではこの程度の時間で計算できたし、あるいはできる。この計算ではCTIMEのETIMEに占める割合が非常に小さく、計算機にとって非常に好ましくないシロ物であろうが、M280Hでもまだ何とかIAPを活用しているという体裁はとっている。CTIME～18S.，演算量～2.4×10^8（1回の積和を2回の加減乗算と数えて）だけから、（I/Oに要したCTIMEも算術演算にかかったとしてしまっても）13MFLOPSという値が得られる。この値はなかなかのものと言ってよいであろう。

最後に、この計算での検討すべき（してきた）問題点を3つほどあげておきたい。第1は、$\alpha - H$という行列は正定値でも負定値でもない。このような行列の逆行列を大次元でとってみても大丈夫かということである。このことは気にする必要はほとんどないと言ってよかった。これは行列の性質に強く依存することで、前述の3つのタイプにわけたとき、③のタイプ（集積特異点）のときは問題となろうが、①、②のタイプでは大した問題ではないようである。2番目は、I/Oの回数を減らすために、サブスペース法のような同時にいくつかのベクトルに$(\alpha - H)^{-1}$をかけるという方法をとってみた方がいいのではないかということである。Lanczos法では当然のことながらこういう芸当はできないので、前述のように相対的に大きなI/O量になったのであるから。この点は反復の回数の大小がからんでくることであるが、現在の程度のI/O比ではまだLanczos法の方が有利なようである。3番目の検討点というのは、実は問題としている行列は帯であると同時に疎でもあるので、この性質を生かして逆行列をかけるのにCG法のようなものを使ってみてはということである。CG法を使ってみた結果から言うと、収束の速さ（実は途中の計算での精度の悪さからくるのであろうが）の点から、特に大次元では全く駄目であると言ってよいと思う。今の問題で、30個

程度の基底ベクトルで収束してくれたのは、毎回きちんと$(\alpha-H)^{-1}$をかけ、かつきちんとこれまでの基底全部と直交化させていたからこそできたことで、このどちらかを(CG法では前者)をいいかげんにやると、とたんに収束が悪くなるようである。

この問題では、I/O量が非常に大きな割合を占めているにもかかわらず、IAPもある程度活用するということが可能であった。しかし、算術演算処理はアレイ・プロセッサによりこれからも高速化されていくであろうが、ディスク等、可動部分のある補助記憶へのI/Oは現状から飛躍的な速度向上は不可能であろう。したがってこの種の計算をこれから更に大次元なものへと拡張していくには、可動部分のない大容量の(したがって安価な)高速補助記憶の存在が不可欠である。このような記憶装置の研究も進んでいる様であるが、CPUの高速化への進歩と比べると遅々としているようで、もちろん技術的な困難度も大きいためであろうが、残念でならない。