The University of Osaka
Institutional Knowledge Archive

Title	固有値計算のベクトル化と並列化
Author(s)	平井, 國友
Citation	大阪大学大型計算機センターニュース. 1998, 108, p. 84-86
Version Type	VoR
URL	https://hdl.handle.net/11094/66271
rights	
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

The University of Osaka

固有値計算のベクトル化と並列化

奈良県立医科大学　平井國友
khirai@nmu-gw.naramed-u.ac.jp

私の専門は物性理論であり，主に電子構造の計算にスーパーコンピュータを用いている．この際，数十から数百個の行列の固有値および固有ベクトルを繰り返し計算しなければならない．行列の次元はそれほど大きくないが (1000 程度)，ワークステーション等では計算はかなり苦しい状況にある．このため，スーパーコンピュータでの固有値計算のベクトル化と並列化，すなわち，大きな次元の行列の固有値と固有ベクトルが SX-4 でどの程度能率良く計算できるかという点に興味がある．そこで，SX-4 での固有値計算のベクトル化と並列化について少し調べてみた．まだ系統的に十分調べたわけではないので，少し初歩的な事柄について報告する．

まず最初に，行列の次元とベクトル化について述べる．用いた計算プログラムは行列要素を計算しその行列のすべての固有値と固有ベクトルを求めるものであり，ベクトル化についてはほとんど配慮していない．計算は CPU 数が 1 の場合であり，f77sx コマンドでコンパイルしている (オプションはデフォルトのまま)．以下はその結果であり，実行時オプションとして "`setenv F_PROGINF DETAIL`" を指定して得られた情報の一部を表にしたものである．

Table 1: 行列の次元とベクトル化

次元	MFLOPS 値	平均ベクトル長	ベクトル演算率	バンクコンフリクト率
100	104.7	51.9	90.8	1.6
200	205.8	77.2	94.2	8.4
300	290.1	98.1	95.6	4.8
400	309.2	114.8	96.4	26.2
500	407.7	131.1	96.9	6.8
600	429.4	142.8	97.2	17.0
800	309.1	165.5	97.7	53.7
1000	543.3	184.1	98.0	21.8
1200	465.4	199.3	98.2	40.2
1600	340.4	194.8	98.4	61.7
2000	516.0	208.9	98.5	44.5
800	566.9	165.5	97.7	0.5
1200	700.8	199.3	98.2	0.4
1600	788.3	194.8	98.4	0.4
2000	881.5	208.9	98.5	0.4

ここで，バンクコンフリクト率はバンクコンフリクト時間のベクトル命令実行時間に対する割合 (%) である．表の上半分では，このバンクコンフリクト率がかなり大きい，特に次元が

800や1600では非常に大きい．この理由は，これらの計算では行列要素を格納する配列Aを以下のように宣言していたためである．

```
parameter(n=200)
dimension a(n,n)
```

このため次元nが8や16，32，… の倍数になると，バンクコンフリクトが頻繁に起こってしまう．そこで，

```
parameter(n=200,n1=n+1)
dimension a(n1,n)
```

と変更した結果が表の下半分である(プログラムも少し修正しなければならない)．バンクコンフリクトがほとんどなくなっているのが分かる．バンクコンフリクトがなくなると実行時間はその分少なくなるので，MFLOPS値は当然大きくなる．バンクコンフリクトをなくせば，次元が増すとMFLOPS値は徐々に増加してゆく傾向が見える(いずれは飽和すると思われる)．一方，平均ベクトル長やベクトル演算率はバンクコンフリクトには全く関係しない．バンクコンフリクトに十分注意する必要があることを再認識した．

次に，並列化について述べる．ただし，ここで述べる並列化は数個の行列の固有値を並列して計算するという意味ではない．並列化によってベクトル長が1個のCPUのベクトル長である256の4倍や8倍に実質上長くなることを期待している．そこで，これまでと同じプログラムをCPU数を4として，すなわち，f77sxコマンドでオプション“`-multi -reserve=4 -fopp par for=4`”を指定してコンパイル，実行時オプションとして“`setenv F_RSVTASK 4`”を指定して実行し，CPU数が1の場合との違いを調べる．結果を以下の表に示す．

Table 2: 行列の次元と並列化

次元	MFLOPS値	平均ベクトル長	ベクトル演算率	CPU時間	
800	566.9	165.5	97.7	10.1	
1200	700.8	199.3	98.2	26.7	
1600	788.3	194.8	98.4	54.0	
2000	881.5	208.9	98.5	91.5	
800	948.7	185.0	97.9	6.0	(1.5)
1200	1181.5	219.9	98.4	15.8	(3.3)
1600	1295.6	210.0	98.5	32.9	(6.2)
2000	1407.1	223.8	98.7	57.3	(9.8)

ここで，表の上半分は比較のためのCPU数が1の場合の結果(Table 1の下半分と同じ計算)であり，下半分がCPU数が4の場合である．CPU数が4の場合のCPU時間は課金対象CPU時間，すなわち，1台以上で実行した時間である．CPU時間の括弧内の数字は2台以上で実

行した時間であり，3台以上で実行した時間と4台以上で実行した時間は2台以上で実行した時間とほぼ同程度である．これらの数値も“setenv F_PROGINF DETAIL”を指定すれば得られる．また，MFLOPS値は実行時間換算した値であり，CPU数が1の場合の1.6～1.7倍程度になっている(この倍率は次元にはあまり依らない)．したがって，並列化は十分有効であるといえる．

ここで注意しなければならないのは，この結果から直接固有値計算の並列化の有効性について速断してはならないということである．以上の結果は用いた計算プログラムにかなり依存している．実は，この計算プログラムでは行列要素の計算があまりベクトル化されていない．このため，固有値計算それ自体は並列化によって非常に有利になっているにもかかわらず，全体的には並列化がそれほどは有利になっていないのである．このことは1台以上で実行した時間と2台以上で実行した時間から判断できる．おそらく，2台以上で実行した時間が概ね固有値計算の時間であり，1台以上で実行した時間から2台以上で実行した時間を差し引いた時間が行列要素計算等のループ長の短い部分の計算時間と推測される．この点をより明確にするために，CPU数を変えた結果(行列の次元は1600)を以下の表に示す．ただし，その数値はASLライブラリの実対称行列の全固有値・全固有ベクトルを求めるDCSMAAサブルーチンを用いたもので，Table 2の数値よりも優っている．DCSMAAはベクトル化および並列化に対する配慮がある程度なされているようである．

Table 3: CPU数と並列化の効率

CPU数	MFLOPS値	平均ベクトル長	ベクトル演算率	CPU時間	
1	1040.9	192.4	99.0	48.4	
4	1898.7	208.3	99.2	26.5	(6.2)
8	2132.5	208.3	99.2	23.6	(3.3)
16	2260.2	208.3	99.2	22.3	(1.9)

表より，CPU数を2倍にすると2台以上で実行した時間がほぼ半減すること，また，1台以上で実行した時間から2台以上で実行した時間を差し引いた時間はほぼ20程度で変化しないことが分かる．これらのことから，上で述べた推測がほぼ正しいことが分かる．したがって，行列要素計算等のループ長の短い部分をできるだけベクトル化できるように改良すれば，並列化がさらに有効になってくると予想される．また，上の例では，CPU数を増してもCPU時間はそれほど小さくならず，CPU時間の換算係数を考えると，CPU数は4が最適である．しかし，この最適なCPU数もベクトル化への改良の度合によって変わるはずである．

結論として，固有値計算それ自体の並列化は十分有効である．実際問題としては，行列要素計算等の部分のベクトル化がかえって重要になり，各自がプログラムを改良する必要があるといえる．また，CPU数には最適な数値があり，その数値を調べることも重要である．