社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2003－HPC－96 (9)
2003／10／16

計算機アーキテクチャに対する素朴な疑問

福井 義成
理化学研究所 情報基盤センター

現在のパーソナルコンピュータは，数年前のスーパーコンピュータより高い性能を持っている．ノート PC でさえ，かなりの高性能である．手元にあるＰＣで，どの程度の大きさの連立一次方程式を解くことが可能か実験したところ，6,000 弱の密行列の連立一次方程式を解くことができた．しかし，行列の大きさを変えて，計算時間を測定したところ，非常に「興味深い」データとなった．これを踏まえ，現在の計算機アーキテクチャに対する素朴な疑問を述べ，問題提起をする．

A simple question for computer architecture

Yoshinari Fukui
Advanced Center for Computing and Communication, RIKEN

The present personal computer has a performance higher than the supercomputer of several years ago. With my notebook PC, it experimented in whether it is possible to solve what size of simultaneous linear equations. The simultaneous linear equations of a little less than 6,000 dense matrix were able to be solved. However, when the size of a matrix was changed and calculation time was measured, it became data very "interesting." Based on this, the simple question over the present computer architecture is expressed, and problem institution is carried out.

1．はじめに

現在のパーソナルコンピュータ（PC）は，数年前のスーパーコンピュータより高い性能を持っている．ノート PC でさえ，かなりの高性能である．普通の人が，手近な所にあるノートＰＣで，どの程度の大きさの連立一次方程式を解くことが可能か実験を行った．コンパイラ等も特別に用意するのではなく，有り合わせの環境での可能性を調べた．結果的には，使用した環境でも，6,000 弱の密行列の連立一次方程式を解くことができた．しかし，開始する前は，キャッシュメモリー等の影響で，連立一次方程式の演算量の推定と少し異なる結果が得られるであろうと推定してい

たが，行列の大きさを変えて，計算時間を測定したところ，非常に「興味深い」データが得られた．その結果とその結果から考えついた現在の計算機アーキテクチャに対する素朴な疑問を述べ，問題提起を行う．

２．解いた問題

テスト問題は，密行列で対角優位（対称）で，必ず解けるデータを用意した．

対角要素	３ｎ＋１
非対角要素	1
解	1

使用した手法は，部分軸選択のガウスの消去法である．ＰＣで計算することを考慮し，データの大きさと配列の大きさは一致させている．浮動少数点表現は６４ビットを使用した．

３．テスト環境

使用したＰＣと環境は有り合わせのもので，以下の通りである．

ノートＰＣ

ハードウェア

８６６ＭＨｚ超低電圧モバイルインテル Pentium Ⅲ

Intel 830MG

Ｌ１キャッシュ	１６ＫＢ／１６ＫＢ
Ｌ２キャッシュ	２５６ＫＢ
メモリー	５１２Ｍバイト SDRAM

ソフトウェア

Windows２０００

Fortran Power Station 4.0 (/Ox)

この環境であるとデータを８バイト浮動少数点数とするとＬ１キャッシュ(データ)は１６ＫＢであるので，これに入る行列の次元は４５まである．また，Ｌ２キャッシュは２５６ＫＢであるので，これに入る行列の次元は１８１まである．しかし，この程度の大きさであると，時間計測の精度が十分でないので，正確な測定ができなかった．十分な精度で計測できたのは，行列の次元数が５００あたりからであった．

４．計算結果

計算時間の結果を表１，図１，図２に示す．予想としては，キャッシュ等の影響で，ガウスの消去法の計算量のおおまかな見積もりである　$n^3/3$　とは異なると考えていたが，結果はそれ以上に面白いものとなった．

特に N＝３２００，３９００，４０００，４４００，４８００，５２００，５６００では各点の近くの値に比べて時間が掛かっている．念のため，各点の N－１，N＋１の点も計算したが，これは大きな値ではなかった．配列 A の大きさを N＝３２００の時は１だけ大きい３２０１とし

てみたが，同じ状況であった（表２）．また，使用したＰＣ固有の問題であるかもしれないので，別のチップセットのＰＣでもＮ＝３２００を確認してみた（表３）．結果は，予想以上に「面白い」ものとなった．

表１．　計算結果

N	t(秒)	N	t(秒)	N	t(秒)	N	t(秒)
100	0.010	2,100	267.165	3,450	1,111.238	4,800	12,140.948
200	0.030	2,200	318.097	3,500	1,386.730	4,801	4,582.580
300	0.170	2,300	396.380	3,550	1,185.635	4,900	5,100.154
400	0.831	2,399	240.265	3,600	2,361.917	5,000	7,965.193
500	2.190	2,400	788.434	3,700	1,706.765	5,100	5,491.877
600	4.196	2,401	245.413	3,800	2,279.518	5,199	5,772.360
700	7.250	2,500	474.610	3,899	1,302.132	5,200	14,066.607
800	11.948	2,600	1,021.649	3,900	5,179.007	5,201	6,679.695
900	16.824	2,700	607.684	3,901	2,686.282	5,300	6,990.522
1,000	23.320	2,800	877.332	3,999	1,683.641	5,400	10,323.795
1,100	32.907	2,900	769.496	4,000	5,617.660	5,500	9,665.478
1,200	43.532	3,000	880.690	4,001	1,434.632	5,599	8,322.817
1,300	57.362	3,050	743.600	4,100	2,922.763	5,600	18,098.742
1,400	71.472	3,100	943.216	4,200	3,382.133	5,601	11,897.107
1,500	89.740	3,150	818.397	4,300	4,029.364	5,700	10,344.204
1,599	72.845	3,199	748.837	4,399	2,132.016	5,750	10,594.303
1,600	271.261	3,200	3,422.741	4,400	5,751.631	5,788	11,107.111
1,601	70.852	3,201	652.218	4,401	2,601.090	5,789	11,159.477
1,700	133.131	3,250	967.120	4,500	3,381.692		
1,800	162.704	3,300	1,152.167	4,600	5,077.801		
1,900	191.996	3,350	996.393	4,700	4,895.319		
2,000	248.800	3,400	1,391.781	4,799	3,309.699		

表２．Ｎ＝３２００で配列のサイズを３２０１とした場合

N	t(秒)
3,200	3,554.060

表３．異なるＰＣでの結果（700MHz 440MX）

N	t(秒)
3,199	828.862
3,200	3,953.204
3,201	622.605

図２の階層型メモリーによる転送速度の違いは一般のユーザにも理解しやすいが，図１の結果は一般のユーザには理解し難い．これでは計算時間の見積りは不可能である．

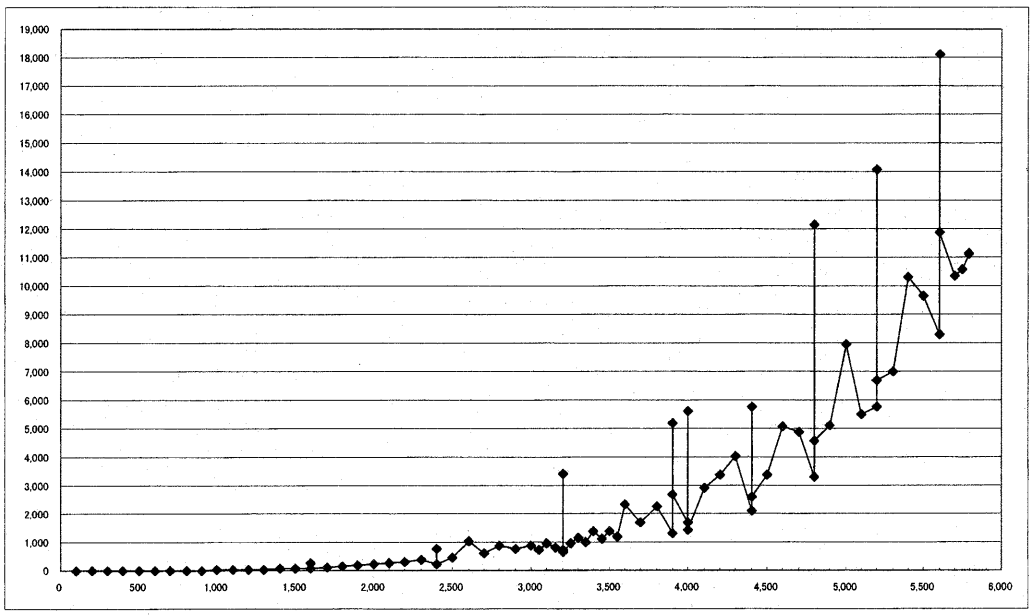

図１．計算結果(行列サイズと計算時間（秒））

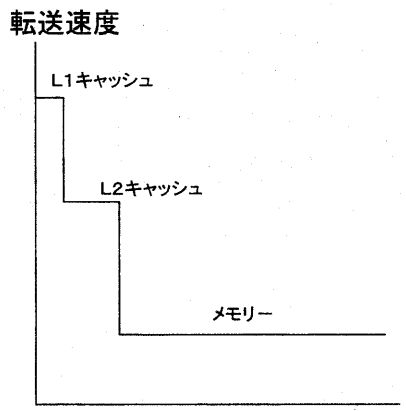

図2. 階層型メモリー

５．高速化について

現在ある計算機の性能を引き出すための高速化（並列化を含み）は，現状のリソースでより大きな計算を可能とする仕事であり，重要な仕事である．筆者も多くの計算機に対して，高速化を行ってきた．場合によっては，機械語まで使用し，オリジナルのアプリケーションの約１０００倍の高速化をしたこともある．これまで高速化を行ってきた体験と，今回のテスト結果から，１

つの疑問が生じてきた．

計算機の歴史を見ると，計算機が生まれたころはすべてのプログラムが機械語で書かれていた．その後，ソフトウェア作成の能率化のために，アセンブラやコンパイラー言語が開発されたはずである．すなわち，人間の負担を軽くする方向に向かっていたはずである．しかし，最近の傾向は，それに逆行しているように思えてならない．

ユーザから見れば，一度行った高速化はソフトウェア資産であり，その効果が続いて欲しい．ベクトル型計算機の場合は，一度行った高速化の効果は，持続されていた．キャッシュ・メモリーの場合は，同じシリーズの計算機でも，モデルにより，キャッシュ・メモリーの大きさが異なると，高速化の効果が一定しなかった経験がある．

計算機の高速化の手段について考えてみると，

（1）素子の高速化（プロセッサ／メモリー）
（2）メモリーの大容量化（メモリーが大きいと速く計算できる場合がある）
（3）キャッシュメモリー（データ／命令）の使用
（4）メモリーのインタリーブを行う
（5）ループを効果的に行う命令を用意する
（6）ベクトル命令を用意する
（7）プロセッサの投機的実行

（1）もすべてに効果があるか怪しげになっているが，（1）以外が完全にある条件を満足した場合に，高速化の効果があるものばかりである．そのため，現在では，「投機的」高速化が主流になっていると思われる．

一般のユーザから見た場合，投機的な高速化の方法は，計算機の情報（アーキテクチャ？）に熟知していないと，計算機の性能を十分引き出すことは不可能である．しかも，投機的手法が変化すると，高速化を再度行わなければならなくなる．ソフトの資産性の消滅である．ユーザにとって，高速化，一度行ったら，計算機が進歩しても，継続性を維持して欲しい．

6．ユーザからの要望

現在の計算機の性能を十分引き出すための，高速化を否定するつもりつもりはない．現在の計算機の「投機性」は容認するとして，将来の計算機は，特別な高速化が必要なくなることが，理想である．少なくとも，ユーザが行った高速化の努力が，新世代の計算機においても，再コンパイル程度で継続される方向に向かうことが，重要であると考える．

そのためには，ユーザが何もしなくても，高速性が得られる「完全な」計算機である必要はない，ユーザが使える計算機であって欲しい．ユーザが使える計算機とは，ユーザの思考を歪めない形の制限（高速化の工夫）を持つ計算機である．ユーザの思考を歪めないと制限は，ユーザは受け入れるはずである．たとえば，以前のＩＡＰを付加した計算機のコンパイラーは「Ｓ＝Ｓ＋Ａ（Ｉ）＊Ｂ（Ｉ)」はベクトル化するが，「Ｓ＝Ａ（Ｉ）＊Ｂ（Ｉ）＋Ｓ」はベクトル化してくれないというようなことがあったが，これはユーザの思考を歪めない制限である．

図２にシミュレーションの流れを示す．現在の高速化は主に計算部分で行われることが多いが，計算の前段階の部分での高速化は，ユーザの思考を歪めないで，なおかつ効果が大きい．計算機アーキテクチャを考える人と，モデルを考える人，データ構造とアルゴリズムを考える人が協調することが，重要と考える．

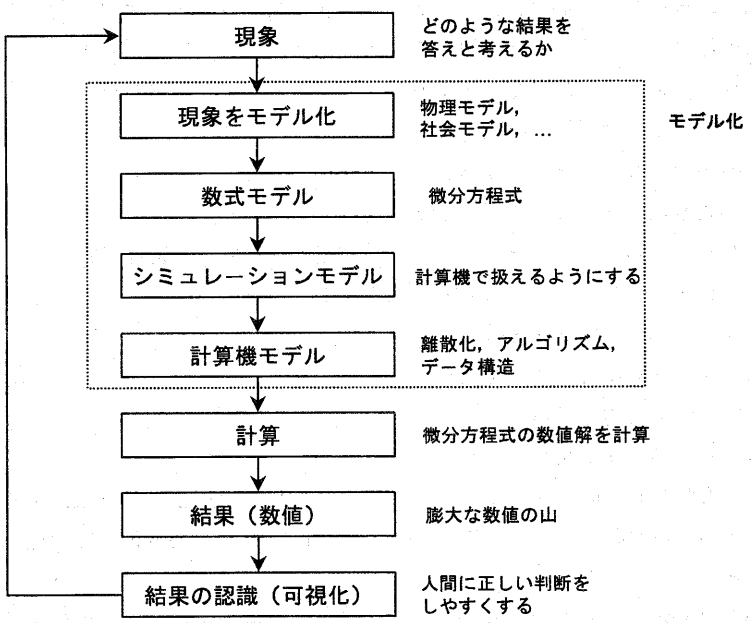

図２．数値シミュレーションの手順