多倍長精度浮動小数点演算の並列化

八木 武尊[1], 菱沼 利彰[2], 石渡 恵美子[1], 長谷川 秀彦[2]
[1]東京理科大学, [2]筑波大学
e-mail: hasegawa@slis.tsukuba.ac.jp

1 概要

高精度演算が必要かどうかは，対象とする問題，使用するアルゴリズム，解に要求する精度によって異なる．状況によっては高精度演算の使用が計算時間の浪費になりうるため，まずは高精度演算の有効性をテストすることが重要であろう．GMP のような可変精度演算プログラムは汎用的であるが，プログラム作成の労力が大きいことと，ある精度に限定したプログラムを高速に実行できないという問題がある．

多倍長精度浮動小数点演算は，演算精度が基本となる精度の２倍，４倍などに限定されるが，ハードウェアで実装された高速な浮動小数点数演算が利用できるため，高速に実行できることが多い．基本となる演算が単精度，倍精度，４倍精度と変化すれば，それに応じた多倍長精度演算が容易に実装できる．

われわれは，倍精度浮動小数点数２つを組み合わせた Double-double (DD)精度[1]，4 つを組み合わせた Quad-double (QD)精度[2]を Scilab と Matlab 上に Multiple Precision Arithmetic Toolbox (MuPAT)として実装した[3]．MuPAT は，異なる演算精度を容易に組み合わせて使える環境になっている．本報告では，「DD/QD 演算をどのように高速化するか」について述べる．

2 DD 演算と QD 演算の高速化

Matlab のような対話的環境では，素朴な実装(Matlab 関数の組み合わせ)はインタプリタ形式での実行となる．ベクトルや行列のような複数要素に同一の演算を施す場合は，その演算を実行する外部関数を作成し，外部関数の実行環境での高速化が現実的である．一方，単一要素に対する演算は，外部関数を呼び出す際のオーバヘッドのため高速化は困難である．

DD/QD 演算には，約 10～600 倍の倍精度演算が必要になる．対話的な利用を想定し，一般的なパソコンでどれだけ高速化できるかに興味がある．4 コアのパソコンでは，FMA で 2 倍，AVX2 で 4 倍，OpenMP で 4 倍，全体で最大 32 倍の高速化の可能性がある．

2.1 FMA

FMA(Fused Multiply and Add)[4]は，倍精度の積演算と和演算の組み合わせ x=a*b+c を 1 演算として実行する演算であり，内積，行列ベクトル積の基本演算などは FMA 演算 1 回で計算できる．メモリアクセスは変わらないが，演算回数が半減し，最大で 2 倍の性能向上が見込める．

DD 加算は倍精度加算 11 回，DD 乗算は倍精度加算 15 回，倍精度乗算 9 回なので，DD 精度の積和演算には 35 倍精度演算が必要である．そのうち FMA が利用できるのは split 関数内に 1 カ所，twoProd 関数内に 4 カ所あり，FMA を使った DD 精度の積和演算は加算 18 回，乗算 1 回，FMA 8 回の 27 倍精度演算となる．35/27 で約 1.3 倍の高速化が期待できる．QD 精度の積和演算では，加算 91+171 回，乗算 46 回の 308 倍精度演算が，加算 222 回，乗算 6 回，FMA 40 回の 268 倍精度演算となり，約 1.1 倍の高速化が期待できる．

2.2 AVX2

Intel AVX2 (Advanced Vector Extensions 2)[4]は，4 つの倍精度浮動小数点数演算を同時に実行する命令である．メモリアクセス数は変わらず，4 倍の性能向上が見込めるが，常に 4 つの要素を用意する必要があり，入力データ数が 4 の倍数でないときに付加的な操作が必要になる．FMA を併用できる．

2.3 OpenMP

OpenMP は共有メモリ上での並列化を可能にする API であり，コアの数だけの性能向上が見込める．実験では 1 コアに 1 スレッドを割り当て，スケジューリングタイプを guided とした．

2.4 性能改善

FMA の効果は，DD/QD 演算で約 1.1 倍だった．AVX2 の効果は，DD/QD とも行列-ベクトル積と行列積が約 4 倍，QD の内積が約 3.2 倍，その他が 2 倍以下だった．OpenMP の効果は，行列-ベクトル積は DD が 2.6 倍，QD が 3.1 倍，行列積は DD が 3.7 倍，QD が 3.4 倍，行列和が DD/QD

とも約3倍で，その他は2倍以下だった．

FMA，AVX2，OpenMP のすべてを適用すると，DD 行列和 2.8 倍，DD 行列-ベクトル積 9.9 倍，DD 行列積 16.7 倍，QD 行列和 5.2 倍，QD 内積 5.1 倍，QD 行列-ベクトル積 14.9 倍，QD 行列積 16.1 倍である．高速化された場合は，理論ピーク性能の 4〜5 割を達成している．

3 DD 演算のさらなる高速化

ベクトルに対する演算が高速化されないのは，メモリからのデータ供給が演算に追いついていないためである．CPU の並列化・高速化はこれからも続くと考えられるが，メモリアクセスの高速化はあまり期待できない．繰り返し使用するようなプログラムを作成する場合は，多倍長浮動小数点演算をより高速化することが望ましい．

3.1 係数行列の倍精度化

メモリアクセスが高速化のネックとなっているため，係数となる行列を倍精度のままとすることで高速化を図る．現時点では「計算の高精度高速化」が目標であり，入力の高精度を先送りする．多くの問題には影響がないが，必要なメモリ量を半分にできる．Bytes per Flops 値は 2.26 から 2.09 に改善できる．

3.2 疎行列格納形式と BCRS

行列計算の高速化では，演算回数の削減が定石であり，ゼロ要素は格納せず，演算もしないのが一般的である．格納を非ゼロ要素だけにすると，要素へのアクセスにはインデックスを用いたメモリに対する間接参照となる．行列の疎度にもよるが，多くの場合，要素へのアクセスが複雑になるデメリットより，データ量・演算量の削減によるメリットのほうが大きい．

行列を標準的な CRS (Compressed Row Storage) 形式で格納すると，AVX2 用に，不規則に配置された 4 要素をまとめることが困難となる．AVX2 向けに，4*1， 2*2， 1*4 の小ブロックからなる BCRS (Blocked CRS)形式を導入する．4 要素からなるブロックを構成することで，ゼロ要素の部分も格納・演算することになり，最大 4 倍のメモリ容量と演算が必要になる．

3.3 性能

AVX2 と OpenMP による並列化に効果的だったのは，4*1 のブロックを用いた BCRS(4,1)である．転置行列-ベクトル積の場合は，列方向の並列化を行う必要がある．ゼロ要素の増加による演算量の増加と AVX2 と OpenMP による高速化のトレードオフは問題による．テスト行列コレクションによる実験結果では，要素数は 3 倍程度まで増えることもあるが，その場合でも倍精度行列-DD 精度ベクトル積と倍精度転置行列-DD 精度ベクトル積との性能は約 2〜4 倍高速化できた．

4 まとめ

DD/QD 演算は，データ量に比べて演算量が多く，計算順序を変えることはできないが，複数の要素は独立に計算できる．DD/QD を用いた多倍長精度演算に対して，個々の演算に対する高速化，ベクトルと行列に対する共有メモリ上の並列化を行った．分散メモリ向けの並列化では，使用するノード数とその際のデータ転送が問題になるはずだが，Bytes per Flops の小さい計算なので問題になりにくいと思われる．

ここでの成果を疎行列に対する BiCG 法に適用すると，1 反復の所要時間が倍精度演算の約 3 倍となるが，アルゴリズムの高精度化によって反復回数 1/3 になれば計算時間は同等になる．少ない反復回数なりのメリットもある．また，混合精度演算として，DD/QD 演算の使用を減らすような利用法も現実的である．

謝辞　本研究は JSPS 科研費 JP17K00164 の助成を受けた．

参考文献

[1] D.H. Bailey, High-Precision Floating-point arithmetic in scientific computation, Computing in Science and Engineering, Vol.7 (2005), 54-61.

[2] Y. Hida, X. S. Li and D.H. Baily, Quad-double arithmetic: algorithms, implementation and application, Tech. Report LBNL-46996, LBNL, Berkeley (2000).

[3] S. Kikkawa, T. Saito, E. Ishiwata and H. Hasegawa, Development and acceleration of multiple precision arithmetic toolbox MuPAT for Scilab, JSIAM Letters, Vol.5 (2013), 9-12.

[4] Intel: Intrinsics Guide, https://software.intel.com/sites/landingpage/IntrinsicsGuide/.

