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

2005－HPC－104（6）
2005／10／7

高速直交関数変換ルーチン FXTPACK の球面調和関数変換における高性能実装と性能評価

須　田　礼　仁[†]

本稿では我々の新しい高速直交関数変換アルゴリズムの実装である FXTPACK について，球面調和関数変換を題材に性能を報告する．FXTPACK は従来の FMM (Fast Multipole Method) による高速変換アルゴリズムで必要であった分離ルジャンドル関数を必要としない一般化 FMM による新しいアルゴリズムを用いている．このアルゴリズムによる演算量の減少は球面調和関数変換では若干であったが，ルジャンドル多項式変換ではその威力を発揮した．また変換の実装による所要時間の違いを評価することにより，高性能実装に向けてのさまざまな知見が得られた．

High Performance Implementation and Evaluation of the Spherical Harmonic Transforms in the Fast Orthogonal Function Transform Routine FXTPACK

Reiji SUDA[†]

This paper discusses the performance of FXTPACK, an implementation of fast orthogonal function transform algorithm, evaluated mainly for spherical harmonic transforms. FXTPACK uses the new algorithm with the generalized Fast Multipole Method (FMM) to eliminate split Legendre functions, which are needed for the fast transform algorithm with the conventional FMM. The new algorithm reduces floating-point operation counts for spherical harmonic transforms slightly, but the effects are clearer in Legendre polynomial transforms. The computational times are evaluated for several different implementations of the transforms, and several implications useful for high performance implementations are obtained.

1. はじめに

球面調和関数変換はフーリエ変換についで応用上重要な直交関数変換であり，気象シミュレーションや信号処理のほか，一部の数値アルゴリズムの基礎としても用いられている．よく知られているようにフーリエ変換は FFT と呼ばれる $O(N \log N)$ の高速アルゴリズムがあるが，球面調和関数変換に対しては FFT のようなシンプルな高速アルゴリズムは存在しない．これに対して著者ら[1),2)] は高速多重極子展開法 (Fast Multipole Method, FMM) を利用した高速アルゴリズムを提案し，FLTSS という名前でプログラムを公開してきた[3),4),12)]．我々のもの以外にも球面調和関数変換の高速アルゴリズムの提案はあるが，最新の研究[5)] では我々のアルゴリズムが最も高く評価されている．

上記のアルゴリズムは高速変換を多項式の計算に帰着させているが，そのために「分離ルジャンドル関数」なる関数を導入しており，これが計算量をほぼ倍増させる結果となっている．そこで我々は数値的な行列圧縮を用いる一般化 FMM[6),7)] を用い，分離ルジャンドル関数を経由しないで高速変換を行うことを試みた[8),9)]．その結果，従来手法に比べて最大では約 1.5 倍の（球面調和関数全体としては若干の）高速化が達成された．これらの結果は，高速直交関数変換ルーチン集 FXTPACK として公開[13)] を予定している．

本稿では，従来の発表では簡単な報告に留まっていた新しいアルゴリズムの性能について議論を行う．また，従来は性能を浮動小数点数演算数 (flop) のみで評価していたが，変換の所要時間による評価と，高性能実装についても論じる．

2. 高速直交関数変換アルゴリズム

本節では高速変換アルゴリズムを簡単に紹介する．新アルゴリズムの詳細は別途発表したいと考えている．我々のアルゴリズムは直交関数変換であれば一般に適

† 東京大学 情報理工学系研究科/JST CREST
Grad. Schl. of Information Science and Technology, the University of Tokyo/CREST of JST

用できるが，以下では例としてルジャンドル陪関数を取り上げる．

ルジャンドル陪関数展開の**評価計算**は

$$g^m(\mu_k) = \sum_{n=m}^{N} g_n^m P_n^m(\mu_k) \qquad (1)$$

と表すことができる．ここで $P_n^m(\mu)$ はルジャンドル陪関数，g_n^m が展開係数である．関数値から展開係数を求める**展開計算**もガウス積分を用いることにより行列転置に相当する変更を行えば同じ計算量で実現できるので，評価点 μ_k は**ガウス点**（ルジャンドル多項式の零点）に取っておく．展開ができるためには評価点の数 P は $P > N$ を満たさなければならないが，以下では $P = O(N)$ であるとする．

ルジャンドル陪関数は超球多項式 $q_{n-m}^m(\mu)$ と

$P_n^m(\mu) = c_n^m P_m^m(\mu) q_{n-m}^m(\mu)$

(c_n^m は定数) なる関係があるから，ルジャンドル陪関数で展開された関数 $g^m(\mu)$ は

$$g^m(\mu) = P_m^m(\mu)\omega(\mu) \sum_{i=1}^{M} \frac{1}{\mu - \mu_i} \frac{g^m(\mu_i)}{P_m^m(\mu_i)\omega_i(\mu_i)}$$

のようにすれば多項式と同様に点 $\{\mu_i\}$ での関数値から他の点 μ での関数値を「内挿」で求めることができる．ここで

$$\omega(\mu) = \prod_{i=1}^{M} (\mu - \mu_i)$$
$$\omega_i(\mu) = \omega(\mu)/(\mu - \mu_i)$$

であり，標本点数 M は次数にあわせて $M = N - m + 1$ ととる．$\omega(\mu)$ と $\omega_i(\mu_i)$ の値をあらかじめ計算しておくとすると，内挿計算は FMM を用いて $O(N)$ の計算量で計算することができる．従って，

- 最初に M 点の関数値を計算し，
- 次にそれを $O(N)$ の計算量で高速に内挿する

という二段アルゴリズムが効率的である．

このままでは前半に $O(M^2)$ の計算量がかかるが，それについて式 (1) の総和を半分ずつに分割し，それぞれに再び上記と同じ「最小限の点の上で評価して，高速に内挿」という二段アルゴリズムを適用すると，さらに高速化が可能である．これを再帰的に行うと，再帰の各深さでの計算量が $O(M)$ になり，$O(\log M)$ 段の分割の後には $O(M)$ 個の定数サイズの問題に帰着される．従って，$O(N + M \log M)$ の計算量で式 (1) が計算できる．これをすべての $0 \le m \le N$ に対して適用すれば，球面調和関数変換が計算量 $O(N^2 \log N)$ で実現できる．

しかし，上述の FMM による高速内挿公式は，再帰の 2 段目からは部分問題のうちひとつを除いて成立しなくなる．従来は高速内挿公式を適用するために**分離ルジャンドル関数**というものを導入していた[2)]が，これが計算量を約 2 倍に膨らませていた．今回の FXTPACK の実装では一般化 FMM を用いることにより分離ルジャンドル関数を用いずに直接内挿行列を圧縮している．以下では分離ルジャンドル関数を用いた旧来の実装を**旧アルゴリズム**，分離ルジャンドル関数を用いない FXTPACK の実装を**新アルゴリズム**と呼ぶ．新アルゴリズムにより計算量は最大半分まで減らせるが，逆に悪くなる可能性もあり，その場合悪化率には上限がない．しかし実験的には従来手法よりも計算量は少なく済んでいる．この結果の理論的な裏づけは現時点では何も得られていない．

3. 球面調和関数変換の演算量の評価

本節では，FXTPACK に実装される新しいアルゴリズムによる演算量の削減状況について報告を行う．

3.1 球面調和関数変換による評価

まず，球面調和関数変換を想定したルジャンドル陪関数変換で評価を行う．評価点はガウス点であり，最高次数 N は**エイリアス誤差**を取り除くために $N = \lfloor (2P-1)/3 \rfloor$ と選んである．次数 n と位数 m は（式 (1) にあるように）$m \le n \le N$ の関係にあり，**三角切断**と呼ばれる方式になっている．

表 1 新旧アルゴリズムによる球面調和関数変換の高速化率の比較
Table 1 Comparison of speedup rates of spherical harmonic transform by old and new algorithms

切断波数 N	旧高速化率	新高速化率	改善率
255	1.46	1.46	1.00
511	1.76	1.78	1.01
1023	2.14	2.32	1.08
2047	2.76	3.17	1.15

表 1 は新旧のアルゴリズムによるルジャンドル陪関数変換における高速化率の比較である．ここで「高速化率」とは，所要時間で評価したものではなく，行列・ベクトル積による単純な関数変換の実装に対する相対的な演算量により定義された高速化率である．$N = 255$ 程度の小さい問題でも 1.5 倍近い高速化率が出ているが，これは誤差を許したことによりルジャンドル陪関数の小さい値がドロップされたことによる[11)]もので，FMM による計算量の軽減はこの段階では小さい．この表の変換の変換誤差は 10^{-10} である．

表 1 では，大規模な問題に対しては若干の演算量

の削減が見られるが，小規模な問題ではほとんど改善していないことがわかる．実は別の発表[8),9)]ではもう数パーセントよい結果を報告しているが，これは今回FMMのアルゴリズムを変更したために旧来の実装に比べてFMMの性能が数パーセント低下していることが原因である．その代わりに前処理時間は劇的に改善されており，以前の発表の際には $N = 2047$ の前処理には推定で4ヶ月もかかるほどであったが，現在の実装では5日で終了する（参考までに，前処理の計算量は $O(N^4)$ である）．今後はFMMの実装を改良して，本来の性能に戻したいと考えている．

3.2 ルジャンドル多項式変換による評価

次に，ルジャンドル陪関数変換で位数 m が0の場合であるルジャンドル多項式変換について評価を行う．ルジャンドル多項式はチェビシェフ多項式についでよく用いられており，非常に重要性が高い．ここではエイリアス除去は考慮にいれず，評価点数は $P = N+1$ としている．

図1 旧アルゴリズムによるルジャンドル多項式変換の相対計算量
Fig. 1 Relative computational costs of Legendre polynomial transforms by the old algorithm

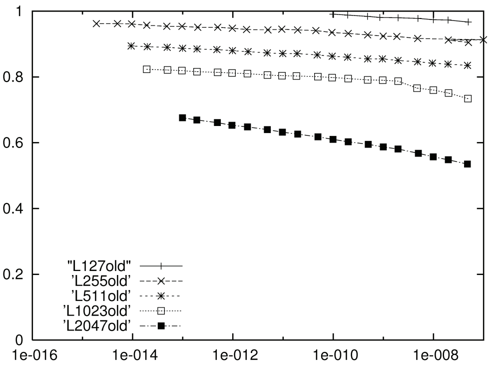
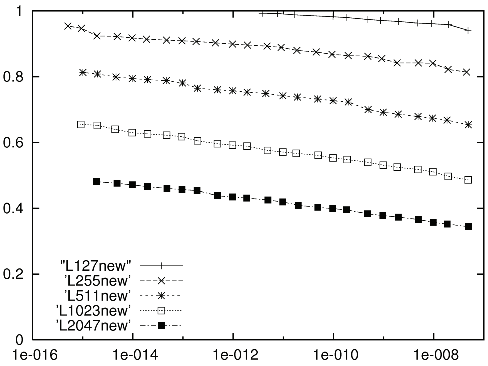


図1および図2は，それぞれ旧アルゴリズム，新アルゴリズムによるルジャンドル多項式の高速変換の計算量を，単純な行列・ベクトル積で実装した場合の相対計算量で評価したもので，横軸は達成精度である．問題サイズは凡例で示されており，'L127old' および 'L127new' は127次までのルジャンドル多項式変換のデータである．

これらの図を比較すると，$N = 127$ では両者の違いも単純計算に対する改良もわずかであるが，$N = 2047$ では新アルゴリズムの方は旧アルゴリズム1.5倍程度少ない演算量を達成していることがわかる．旧アルゴリズムでは $N = 1023$ の低精度の場合と $N = 2047$ では他の場合とは異なる傾きになっているが，これは

図2 新アルゴリズムによるルジャンドル多項式変換の相対計算量
Fig. 2 Relative computational costs of Legendre polynomial transforms by the new algorithm



これらの範囲でのみ分離ルジャンドル関数が使用されているからであると思われる．それ以外では，計算コストが大きいために分離ルジャンドル関数を使わないほうが有利なのである．新アルゴリズムではそのような現象はほとんどみられず，精度と計算量が安定した関係にあることがわかる．

また，ふたつの図を比較すると，新アルゴリズムの方が高精度まで達成していることがわかる．これは実装が改良されたことも理由の一部ではあるが，アルゴリズムの安定性が根本的に改善されたことが本質的である．分離ルジャンドル関数を用いる旧アルゴリズムの数値的な安定性はルジャンドル陪関数の漸化式の安定性に依存する[10)]．一方，新アルゴリズムでは漸化式を利用しておらず，数値的な安定性のほとんどは内挿計算の安定性で決まるが，内挿計算は適切な標本点を選択すれば安定性は保証できる[10)]．新アルゴリズムでは枢軸選択付き Gram-Schmidt 法という単純な方法で標本点選択を行っているが，旧来の方法に比べて格段に安定性が改善されて，ほとんど丸め誤差に近い精度まで達成可能であることが観測されている（データは本稿では省略）．若干のチューニングにより，さらに丸め誤差レベルに近い精度まで実現できる可能性がある．

4. 球面調和関数変換の計算時間の評価

前節では演算量に着目して評価を行ったが，実機上での計算時間は実装により異なる．本節では演算内容は固定しておいて，その高性能な実装について議論をおこなう．

変換は $N = 170, P = 256$ の三角切断によるルジャンドル陪関数変換で，精度は 10^{-8} である．サイズがこの倍の問題についても評価をしたが類似の結果が得

られたので省略する．使用した計算機は IBM xSeries 335 で，CPU は Xeon 2.8 GHz, 主記憶は DDR2 1GB である．OS は Redhat kernel 2.4.20-19.8smp, コンパイラは Intel C Compiler 7.1 で，オプションとして `-O3 -fno-alias` をつけた．測定の際には 10 回以上計算を反復し，最初の 1 回を除く所要時間の平均を求めている．これを 3 回繰り返し，各位数 m に対して所要時間の最小値を結果とした．

4.1 フラットな構造への展開

まず，我々のアルゴリズムは 2 節で説明したように分割統治法を用いており，さらに内部で使用している一般化 FMM も分割統治法によっている．この再帰的な構造をそのままプログラムに反映させると種々のオーバーヘッドのために高い性能が出ない．そこで FXTPACK ではアルゴリズムを乗加算のあつまりにばらして，フラットな表現に変換している．

この変換の結果，我々のアルゴリズムによる高速直交関数変換は疎行列とベクトルの積に近い構造となる．行列・ベクトル積と異なるのは中間変数があることで，入力変数と中間変数を参照しながら中間変数を計算してゆき，最後に出力変数を計算するという処理となる．これらの中間変数や出力変数の一つ一つの値を決める計算は疎ベクトルと密ベクトルの積であり，これが疎行列・ベクトル積との類似点である．今回は基準となるデータ構造として，疎行列の CRS (Compressed Row Storage) に相当する形式を用いた．以下ではこの基本データ構造を SAR (Simple Array Representation) と呼ぶ．評価計算は CRS では転置行列とベクトルの積に相当し，展開計算が CRS での単純な行列・ベクトル積に対応する．

4.2 ブロック化データ構造

我々のアルゴリズムは疎行列と計算の構造が類似しているので，疎行列で用いられている高性能化技術を参照することが可能である．疎行列演算の高性能化のためのデータ構造は非常に多数のものが提案されているが，今回はそれらを実装検討するだけの時間的余裕がなかったので，比較的単純なもの一種類を試行的に実装した．

今回用いたものは，一定以上非零パターンが共通な 2 行についてそれらを一緒に扱うというもので，以下では B2AR (Block-2 Array Representation) と呼ぶ．非零パターンの相違部分にはゼロをつめるので，演算数を増やすことになる．今回の場合 2 行しか共通化しないので，格納された要素が非零である割合（以下，**非零率**という）は非零パターンが完全に同一なら 1, 非零パターンが完全に異なれば 1/2 となる．今回は与えられた一定の値（以下，**非零率下限**という）よりも非零率が大きくなるようなペアしか共通化しないことにした．なお，以下では非零率下限としてそれを 2 倍した値を表示してある．

表 2 B2AR のブロック化率，非零率，変換時間
Table 2 Block rates, non-zero rates, and transform times of B2AR

非零率下限	ブロック化率	非零要素率	評価時間	展開時間
1.10	97.7 %	97.1 %	9.67 ms	6.89 ms
1.30	96.7 %	97.5 %	9.63 ms	6.86 ms
1.50	95.5 %	97.9 %	9.61 ms	6.86 ms
1.70	92.8 %	98.3 %	9.64 ms	6.91 ms
1.90	82.0 %	98.9 %	9.54 ms	6.87 ms

表 2 は，非零率下限を変えて得られたデータ構造に関するデータを示している．ブロック化率は，すべての行のうちペアにされたものの割合，非零要素率は演算のうち非零要素によるものの割合である．非零率下限を上げてゆくとブロック化率が下がり，非零率が上がるという単純な傾向が見られる．注目に値するのは非零率下限を 1.10 にまで落としても非零率は 97 % もあり（あるいは 1.90 まで上げても 82 % がペアリングされており），ほとんどすべての行がほとんど同じ非零パターンの相手を見つけていることである．このことから，さらに組にする行の数を増やしたデータ構造を利用することも可能であると思われる．

所要時間は 1 % 程度しか違わず，有意な差があるようには見えないが，ブロック化率や非零率がほとんど変わらないので当然である．なお，基本データ構造 SAR での所要時間は，評価計算で 10.35 ms, 展開計算で 6.96 ms である．まとめると，評価計算では B2AR-1.90 が，展開計算では B2AR-1.30 が最短時間であった．展開計算が評価計算よりも速いことについては，評価計算では最内ループに書き込みがあるのに対して評価計算にはないことによると思われる．B2AR と SAR の差が評価計算の方が大きいことも同じで，書き込み回数が B2AR で削減される効果によるものと思われる．

4.3 複数同時変換

気象シミュレーションなどの球面調和関数変換の応用では，一般に多数の関数に対して球面調和関数変換を適用する．これには，(1) 位数 m による対称性を利用して変換数が 2 倍になる，(2) シミュレーションにおけるさまざまな物理量に対応する関数がある，(3) 関数の（緯度・経度方向の）微分に対しても球面調和関数変換をほどこす必要がある，(4) 球面だけではな

く高さもある現実的な気象シミュレーションの場合，高さ方向の格子点ごとに球面調和関数変換が必要になる，などのさまざまな理由がある．

これに対して今回は 4 通りの方法で複数同時変換を実装した．

- **serial**: 単発の球面調和関数変換を行うライブラリを想定したもので，すべての位数に関するルジャンドル陪関数変換を一組のデータに適用する．これをすべてのデータに対して繰り返すものである．
- **exloop**: ひとつの位数 m を固定して，すべてのデータに対して位数 m のルジャンドル陪関数変換を適用する．これをすべての位数 m に対して繰り返すものである．
- **inloop**: 上記 exloop の実装において，データに関するループを最内ループに移したものである．CRS の連想で説明すると，疎行列と多数のベクトルの積を，疎行列と行優先で格納された密行列の積で実装したことになる．
- **unroll**: 上記 inloop の実装において，最内ループをアンローリングしたものである．同時変換数ごとに異なるプログラムを用意することになる．

このうち serial と exloop は単一の変換と同じプログラムを呼び出している．また，inloop と unroll は一時配列が同時変換数倍だけ必要になるが，すべての位数 m に対して一時配列が使いまわしできるので，同時変換数がそれほど大きくなければメモリに対する負担は大きくない．また inloop と unroll はデータの再配置を必要とするが，serial と exloop でも赤道に関する対称性を利用するためにデータの再配置を行っており，不利にならないという点が，疎行列・ベクトル積とは大きく異なり特徴的である．

図 3　6 ベクトル同時の評価計算の実行時間
Fig. 3　Times of six-vector evaluation transforms

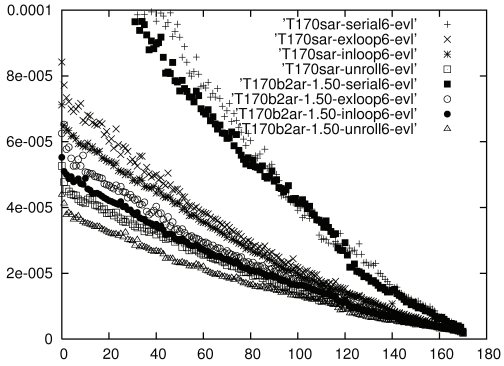
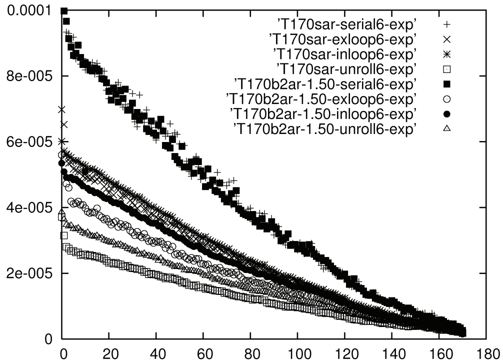


図 3 と図 4 は 6 組同時変換の場合の変換所要時間を位数 m に対してプロットしたものである．簡単のため非零率下限は 1.50 のみ表示した．全体として，各手法の順位が位数 m にあまり依存しないことが注目に値する．

図 4　6 ベクトル同時の展開計算の実行時間
Fig. 4　Times of six-vector expansion transforms



評価計算では下から b2ar-unroll, sar-unroll, b2ar-inloop, b2ar-exloop, sar-inloop, sar-exloop, b2ar-serial, sar-serial となっている．全体として B2AR が SAR よりも速く，同じデータ構造では unroll – inloop – exloop – serial の順序になっている．変換数を 2 から 10 まで変えても unroll が最も速く serial が最も遅いことに変わりはない．inloop は同時変換数が 5 までは最内ループが短すぎるらしく exloop より遅いが，6 以上では exloop よりも速い．なお，serial は単一変換のときとほとんど同じグラフになっている．

展開計算では下から sar-unroll, b2ar-unroll, b2ar-exloop, b2ar-inloop, sar-exloop, sar-inloop, b2ar-serial, sar-serial となっているが，最後の 2 つはほとんど重なっている．展開計算ではまだ inloop よりも exloop が速く，同時変換数が 10 になってようやく inloop が exloop より速くなる．SAR と B2AR の順序が評価計算とは逆になるが，これはコンパイラの最適化能力の問題である可能性がある．すなわち，最内ループの演算量が多い B2AR の最適化がうまく行かなかったかもしれない．

以上のような実験を同時変換数を 1 から 10 まで変えて，最も速かった実装方式を表にしたものが表 3 である．全体として評価計算では B2AR が，展開計算では SAR が高速である．所要時間では，展開計算はベクトル数が 6 の時が最短で，それ以上では徐々に所要時間が延びている点が興味深い．このため，例えば 9 本のベクトルの場合には 4 本 + 5 本などに分けたほうが速いという計算になる．また，評価計算は展開計算よりもかなり時間がかかっている．評価計算を展

表 3 各同時変換数に対する最速実装方式と所要時間
Table 3 Fastest implementation and transform times for each number of simultaneous transforms

同時変換数	評価計算方式	評価時間	展開計算方式	展開時間
1	b2ar-1.90-single	9.54 ms	b2ar-1.30-single	6.86 ms
2	b2ar-1.90-unroll	5.65 ms	b2ar-1.30-unroll	3.86 ms
3	b2ar-1.90-unroll	4.30 ms	sar-unroll	2.89 ms
4	b2ar-1.30-unroll	3.60 ms	sar-unroll	2.47 ms
5	b2ar-1.50-unroll	3.42 ms	sar-unroll	2.68 ms
6	b2ar-1.30-unroll	2.95 ms	sar-unroll	2.20 ms
7	b2ar-1.30-unroll	2.77 ms	sar-unroll	2.40 ms
8	b2ar-1.30-unroll	2.64 ms	sar-unroll	2.29 ms
9	b2ar-1.30-unroll	2.70 ms	sar-unroll	2.75 ms
10	b2ar-1.50-unroll	2.56 ms	sar-inloop	2.37 ms

開計算のように記述するデータ構造を別途準備するほうが速度的には有利であると考えられる．

5. まとめと今後の課題

本稿では，分離ルジャンドル関数を用いずに一般化 FMM で高速化する高速球面調和関数変換アルゴリズムの実装である FXTPACK について，その性能を演算数と実行時間において評価を行った．演算数では従来手法に比べてわずかな改良に留まったが，ルジャンドル多項式では大きな効果を見せた．実行時間ではいくつかの実験により高性能実装の方向性が示唆された．

演算数に関してはさらなる削減の可能性はほとんど FMM に依存する．一般化 FMM についてはまだ若干の性能改善の余地が残っているので，これについては取り組みたいと考えている．

実行時間については，密行列ベクトル積でルジャンドル陪関数変換を実装する場合の演算量は 170 の場合およそ 3.7 Mflop である．従ってもっとも速かった 6 変換同時の展開計算での処理性能は 1.7 Gflops に相当する．CPU が Xeon 2.8 GHz であることを考慮すると，これは必ずしも十分な性能とは言えない．さらなる高性能化およびマシンやコンパイラなどの計算環境の変化に対して自動的に最適な実装を選択することを今後の課題としたい．

謝　辞

本研究の一部は JST CREST SSI プロジェクト，文部科学省 21 世紀 COE プログラムおよび科学研究費の補助を受けています．

参 考 文 献

1) 須田礼仁,「高速球面調和関数変換」, 情報処理学会研究報告, 98-HPC-73, pp. 37–42 (1998).
2) R. Suda and M. Takami: A Fast Spherical Harmonics Transform Algorithm, *Math. Comp.*, Vol. 71, No. 238, pp. 703–715 (2002).
3) R. Suda: Fast spherical harmonics transform of FLTSS and its evaluation, *The 2002 Workshop on the Solution of Partial Differential Equations on the Sphere*, Fields Inst., U. Toronto (2002).
4) R. Suda: Fast spherical harmonic transform routine FLTSS applied to the shallow water test set, *Mon. Wea. Rev.*, Vol. 133, No. 3, pp. 634–648 (2005).
5) V. Rokhlin and M. Tygert: Fast algorithms for spherical harmonic expansions, to appear in *SIAM J. Sci. Comp.*
6) N. Yarvin and V. Rokhlin: A Generalized One-Dimensional Fast Multipole Method with Application to Filtering of Spherical Harmonics, *J. Comp. Phys.*, Vol. 147, pp. 594–609 (1998).
7) R. Suda and S. Kuriyama: Another Preprocessing Algorithm for Generalized One-Dimensional Fast Multipole Method, *J. Comp. Phys.*, Vol.195, pp. 790–803 (2004).
8) 須田礼仁,「高速球面調和関数法：アルゴリズム、応用、展開」, 日本応用数理学会 2004 年度年会, pp. 26–27 (2004).
9) R. Suda: Fast Spherical Harmonic Transform with the Generalized Fast Multipole Method, *2005 International Conference on Scientific Computing and Differential Equations (SciCADE05)*, p. 86 (2005)
10) R. Suda: Stability analysis of the fast Legendre transform algorithm based on the fast multipole method, *Proc. Estonian Acad. Sci. Phys. Math.*, Vol. 53, No. 2, pp. 107–115 (2004).
11) 須田礼仁, 高見雅保: 高速球面調和関数変換法の誤差の解析と制御, 情報処理学会論文誌：ハイパフォーマンス コンピューティングシステム, Vol. 42, No. SIG12 (HPS4), pp. 49–59 (2001).
12) FLTSS ホームページ：`http://www.na.cse.nagoya-u.ac.jp/~reiji/fltss/`
13) SSI プロジェクトホームページ：`http://ssi.is.s.u-tokyo.ac.jp`