地球流体データ解析・数値計算ワークショップ2016年2月11日

スペクトル法による高速数値計算ライブラリ

石岡 圭一 (京大理・地惑)

E-mail: ishioka@gfd-dennou.org

はじめに

地球流体の研究のために球面のスペクトル法をずっと使い続けてきた関係で，それに必要な球面調和関数変換のための数値ライブラリ(ISPACK)の開発を個人的に細々と趣味的に続けてきました.

今日は，これまでの開発の経緯と，最近の開発状況(ISPACK2)について簡単にお話します.

「趣味でヒーローをやっている者だ」(by サイタマ)

ISPACKの球面調和関数変換ルーチンのちょっと前までのCHANGELOG

stpack(1995年) 最も素朴な実装

smpack(1998年) ベクトル化を追求

snpack(1999年) ベクトル化を追求しつつ省メモリ化

sjpack(2009年) IA-32なCPUでの高速化

sjpack-cuda(2010年) GPU上での計算

A brief review of the spherical spectral method

Dependent variables are expanded in spherical harmonics.

$$f(\lambda, \mu, t) = \sum_{n=0}^{M} \sum_{m=-n}^{n} a_n^m(t) Y_n^m(\lambda, \mu).$$

λ: longitude, $\mu = \sin\theta$, θ: latitude, t: time,

M: truncation wavenumber.

$$Y_n^m(\lambda, \mu) = P_n^{|m|}(\mu) e^{im\lambda}.$$

$P_n^m(\mu)$ is the associated Legendre function.

$$P_n^m(\mu) \equiv \sqrt{(2n+1)\frac{(n-m)!}{(n+m)!}\frac{1}{2^n n!}}$$
$$\times (1-\mu^2)^{m/2} \frac{d^{n+m}}{d\mu^{n+m}} (\mu^2 - 1)^n \quad (0 \le m \le n).$$

Expansion in the associated Legendre functions

In SHT, most computational time is spent in the following transforms with the associated Legendre functions.

Backward transform:

$$g_j^m = \sum_{n=m}^{M} s_n^m P_n^m(\mu_j) \quad (m = 0, \cdots, M;\ j = 1, \cdots, J).$$

Forward transform:

$$s_n^m = \sum_{j=1}^{J} w_j g_j^m P_n^m(\mu_j) \quad (m = 0, \cdots, M;\ n = m, \cdots, M).$$

J: the number of Gaussian latitudes, μ_j: Gaussian Latitude, w_j: Gaussian weight.

Computational cost

Considering that both real and imaginary parts must be computed for $m \geq 1$ and that the loop length for the suffix j can be reduced to $J/2$ because of the symmetry of $P_n^m(\mu)$ on μ, the number of ADDs and MULs required by each transform is estimated as,

$$N = \frac{J}{2} \cdot (M+1)^2 \cdot 2 = J(M+1)^2.$$

Furthermore, we must set as $J > \frac{3}{2}M$ to eliminate the alias error in the forward transform. Then,

$$N \sim \frac{3}{2}M^3.$$

When M is large, we must tune up the transform subroutines to speed up the global spectral model.

Optimization for IA-32 CPUs

Nowadays, IA-32 CPUs are ubiquitous. It is advantageous to optimize the code for IA-32 CPUs.

IA-32 CPUs have relatively narrow memory bandwidth. Then, it is important to use the cache memory efficiently.

However, efficient use of the cache memory is difficult in a simple coding of SHT because the transform is matrix-times-vector type.

$\rightarrow$ breakthrough: compute the associated Legendre functions on the fly with the transform.

However, the cost for computing the Legendre functions is of the same order as that for the transform itself.

Then, any effort should be done to reduce the cost for computing the Legendre functions if possible.

Normally, the Legendre functions are computed using the following recurrence formula.

$$P_{n+1}^m(\mu) = (\mu P_n^m(\mu) - \epsilon_n^m P_{n-1}^m(\mu))/\epsilon_{n+1}^m.$$

$$\epsilon_n^m = \sqrt{(n^2 - m^2)/(4n^2 - 1)}.$$

To compute the recurrence formula one step, 3 MULs and 1 ADD are required (ϵ_n^m and $1/\epsilon_n^m$ are computed in advance).

It is possible to reduce 1 MUL with a trick. If we introduce α_n^m and $p_n^m(\mu)$ as

$$P_n^m(\mu) = \alpha_n^m p_n^m(\mu),$$

and set α_n^m properly, we can obtain the following recurrence formula for $p_n^m(\mu)$.

$$p_{n+1}^m(\mu) = \beta_n^m \mu p_n^m(\mu) + p_{n-1}^m(\mu).$$

This recurrence formula requires 2 MULs and 1 ADD for one step.

Furthermore, for optimization on IA-32 CPUs, it is important to make the best use of AVX instructions.

Although compilers have become smarter these days, hot spots in computing SHT should be coded in an assembly language to make the best use of AVX instructions.

Optimization with AVX

Main advantage of AVX:

- 256bit-length SIMD instructions are available. Then, four DP data can be treated at once. This yields double speed compared to SSE2 instructions theoretically.
- Three-operand instructions are available. Using them, operations like `C = A + B` can be simply coded.

Let us consider the same example of a hot spot.

```
      SUBROUTINE LJLSWG(JH,S,R,Y,QA,QB,W)

      REAL*8 W(JH,2),Y(JH),QA(JH),QB(JH),S(2),R

      DO J=1,JH
        W(J,1)=W(J,1)+S(1)*QA(J) ! coeff. to grid (real  part)
        W(J,2)=W(J,2)+S(2)*QA(J) ! coeff. to grid (imag. part)
        QB(J)=QB(J)+R*Y(J)*QA(J) ! recurrence formula
      END DO

      END
```

The corresponding assembly code with AVX instructions.

```
.globl ljlswg_
ljlswg_:
  movl (%rdi), %edi
  vbroadcastsd  (%rsi),%ymm0
  vbroadcastsd 8(%rsi),%ymm5
  vbroadcastsd  (%rdx),%ymm1
  movq  8(%rsp),%r10
  shlq $3,%rdi
  xorq %rdx,%rdx
  subq %rdi,%rdx
  movq %r10,%r11
  addq %rdi,%r11
  subq %rdx,%r8
  subq %rdx,%r9
  subq %rdx,%r10
  subq %rdx,%r11
  subq %rdx,%rcx
.L0:
  vmulpd (%rcx,%rdx),%ymm1,%ymm4
  vmovapd (%r8,%rdx),%ymm2
  vmulpd %ymm2,%ymm4,%ymm4
  vmulpd %ymm0,%ymm2,%ymm3
  vmulpd %ymm5,%ymm2,%ymm2
  vaddpd  (%r9,%rdx),%ymm4,%ymm4
  vaddpd (%r10,%rdx),%ymm3,%ymm3
  vaddpd (%r11,%rdx),%ymm2,%ymm2
  vmovapd %ymm4,(%r9,%rdx)
  vmovapd %ymm3,(%r10,%rdx)
  vmovapd %ymm2,(%r11,%rdx)
  addq $32,%rdx
  jnz .L0
  ret
```

ISPACK2の開発の動機

ISPACK に含めていた球面調和関数変換ルーチン sjpackはこれまでのIntelのCPUアーキテクチャのもとではそれなりに速かった(多分, 一層用としては少なくとも2011年頃までは世界最速であったと思われる)ので, sjpack開発後のここ数年間はそんなに新しいものを作ろうという動機が湧かなかったが, ここにきて, いくつか課題が出てきた.

2013年頃から以降, 数理研の竹広さんが ES2上で spmodel を用いた球殻対流計算を始めた. sjpack はベクトル化のことを考慮していない(正しくは, sjpack内の Legendre 陪関数変換パッケージ ljpack はベクトル化しやすく設計されているが, FFTの部分の FFTJ が全くベクトル化のことを考慮していない)ので, sjpack は ES2上では全く性能が出ない.

問題が ES2上の話だけなら，もはや滅びゆくベクトル機のために新たにコードを書く気にもならないのだが，Intel の CPUにおいてもAVX で4つの倍精度変数が同時に扱えるようになるなど，ある意味「ベクトル化」への対応が求められてきている．

特に Intel のXeon Phi や，Skylake な Xeon(2016年末の E5以降?)では，AVX512 といわれる 8つの倍精度変数が同時に扱える命令セットが使えるので，これらも睨みつつ，キャッシュの効率利用だけでなく，ベクトル化を意識したコードを書いていくことが(IntelのCPUを前提とするならば)時代の要請となってきた．

ISPACK2の方向性

以上の動機付けによって，現在，新しい球面調和関数変換ルーチンsvpack を開発した．基本的な設計思想は以下の通り．

svpack の中の FFT部分 fvpack は，FFTJとは異なり，複数のデータの同時処理によってベクトル化しやすい設計にする．AVX命令などを使ったチューンは当然行う．

svpack の中の Legendre陪関数変換部分 lvpack は，sjpackでの設計を基本的には踏襲するが，Intel の Haswell 以降に搭載されたFMA(積和算)命令の利用も睨んでアルゴリズムの見直しを行う．

状況の変化

というあたりまでが 2014年夏頃の話．その段階でおおむね svpack の実装ができつつあり，sjpack の2倍速くらいでそうというあたりまで進んでいて，あとは夏休みを利用してコードとマニュアルの整備をして 2014年夏中にリリースしようと思っていた．

しかし，2014年7月になって，ライバル(SHTns)の存在を知らされる(Thanks to 榎本さん)．

SHTnsについて

作者: Nathanaël Schaeffer (ISTerre, Université de Grenoble)
(ジオダイナモ屋さんらしい)

URL: http://users.isterre.fr/nschaeff/SHTns/

論文: Schaeffer(2013): Efficient spherical harmonic transforms aimed at pseudospectral numerical simulations, *Geochemistry, Geophysics, Geosystems*, vol. 14, pp.751–758.

開発履歴: Changelog を見ると, 2010年に v1.0 がリリースされていたようだ. AVXへの対応は 2012年になされていたよう. 現在は v2.6.5-r530.

概要:

- ISPACKの sjpack と同じように，基本，1層用の球面調和関数変換ライブラリである．OpenMPによる並列化はなされているが，MPIは考えていない．
- キャッシュを効率よく利用するすために on-the-fly でルジャンドル陪関数を計算している．
- Intel-x86なCPU上での高速化を意識して，SSE2 および AVX命令(および FMA)にも対応している．

感想: 方向性が私と一緒で驚く(Intel の石の上で速く走らそうと究極に突き詰めて考えていけば必然的に辿りつく方向性ではあるが)． しかも完成度が高くて速いし(後述)．

「このレスラー，おれと頭の構造が同じ!?」(by 炎尾 燃)

開発の現状

ということで, svpack はちゃんと SHTns を凌駕できていることを確認した上でリリースしなければということでかなり長いことソースのブラッシュアップをやっていた.

特に, SHTns は TL1023を超える切断波数でも精度があまり落ちないことを一つのウリにしているので, svpackもそれに対抗するために手を加えた(後述).

また, FFT(fvpack)についても, TL1023くらいでもそれなりにFFTの計算が占める割合はあるので, アルゴリズムから抜本的に見直して高速化した(結局, アルゴリズムを Cooley-Tukey型の時間間引き型のFFTに FMAが効くようになる変更を施したものにした).

変換精度の向上(というか破綻の防止)について

Legendre陪関数を，漸化式

$$P_{n+1}^m(\mu) = (\mu P_n^m(\mu) - \epsilon_n^m P_{n-1}^m(\mu))/\epsilon_{n+1}^m,$$

$\epsilon_n^m = \sqrt{(n^2-m^2)/(4n^2-1)}$ (または lvpack で採用しているような変形版)で計算していく際，出発点には，

$$P_m^m(\mu) = \frac{\sqrt{(2m+1)!}}{2^m m!}(1-\mu^2)^{m/2}, \quad P_{m+1}^m = \sqrt{2m+3}\mu P_m^m(\mu)$$

を用いる．高緯度においては $(1-\mu^2)^{1/2} = \cos\phi$ が小さいため，切断波数が非常に大きい場合，大きなmについて$(1-\mu^2)^{m/2}$が非常に小さくなり，倍精度の範囲で表現できず，underflow で 0になってしまう．その場合，上記の漸化式を使っている限り，その mについては高緯度域で$P_n^m(\mu)$が全て 0になってしまう．しかし，本来，$P_n^m(\mu)$は，そのような緯度でもnが十分大きいと，$O(1)$ の値をとりうる．

従って，この問題が顕在化するような切断波数を扱う場合には，何らかの工夫が必要になる．

ちなみに，IEEE754倍精度で表現できる絶対値最小の正規化数は，2.2251×10^{-308} である(非正規化数まで含めると4.9407×10^{-324} までいけるが，精度が低下する)．

TL1023くらいまでなら倍精度で問題無いが，それを超えると精度低下さらには計算の破綻に至る．4倍精度にすれば，3.3621×10^{-4932} までいけるが，それでも TL8191とかのレベルになると精度低下を起しうるし，そもそも4倍精度の計算は遅いので，on-the-fly での漸化式計算には採用できない．

もう一つのよくある回避法としては，別の漸化式(m方向も変える) を使うというものもあるが，それも on-the-fly 計算では計算量が増えてしまう(し，m方向の並列化もできなくなる)．

計算量を増やさない解決策:

前もって必要な Legendre陪関数を一旦計算しておき，あるnでどのガウス緯度までの$P_n^m(\mu)$を非零として扱わなければいけないかを記憶する．漸化式計算においては，そのnに到達した段階で，それまで零として扱われていた範囲で新たに非零の範囲に含まれたμの区間の$P_n^m(\mu)$を前もって計算されていたものに置き換える．

この手法により，特に計算量を増やすことなく精度低下の問題が回避できるだけでなく，零として扱われる$P_n^m(\mu)$に関する計算を端折って計算速度を上げることができる．

では，そもそもその「前もって」の$P_n^m(\mu)$の計算で精度低下をどう防ぐ?

$$P_m^m(\mu) = \frac{\sqrt{(2m+1)!}}{2^m m!}(1-\mu^2)^{m/2}, \quad P_{m+1}^m = \sqrt{2m+3}\mu P_m^m(\mu)$$

の代わりに，

$$\tilde{P}_m^m(\mu) = \frac{\sqrt{(2m+1)!}}{2^m m!}, \quad \tilde{P}_{m+1}^m = \sqrt{2m+3}\mu \tilde{P}_m^m(\mu)$$

に対して漸化式を計算していき，後から$(1-\mu^2)^{m/2}$のファクターを掛ける．その計算はもちろんそのままやるとoverflow/underflow が起こるが，対数および指数関数をうまく利用すれば，問題を回避できる．

このようなスケールファクターを使う手法は，SHTns でも組込まれているらしい(on-the-fly でやっているようだ．ただ，私の手法の方が効率が良い筈だが)．

並列化の新たな工夫

OpenMP並列化において，Legendre陪関数変換部分は各 mに関する変換を適宜スレッドに割り当てれば良い.

sjpackの頃の実装では，スレッド毎の計算量のバランスをとるためにmと$M-m$の変換を組にして規則的にスレッドにばらまいていたが，各 mでの変換の計算時間はスレッド間のメモリ競合の影響などもあって必ずしも机上の想定通りにはならないので，動的スケジューリングを採用して，各スレッドが手が空き次第どんどん動的に残りの作業に着手していくように変更した．これでかなり並列化効率が上がった.

この動的スケジューリングは，各緯度毎の FFTの計算の部分の並列化にも採用した.

ベンチマーク

Platform:
CPU: Haswell Xeon E5-2699v3 (2.3GHz Turbo boost時は 3.6GHz)
(18コア)×2
Compiler: ifort 16.0.0, option -xHost -openmp

TL1023 の設定(2048× 1024 グリッド)

スペクトル⇔格子点値 1回あたりの変換にかかる時間(sec)

	1スレッド		36スレッド	
	逆変換(sec)	正変換(sec)	逆変換(sec)	正変換(sec)
SHTns	0.0636	0.0618	0.00351	0.00354
svpack	0.0530	0.0433	0.00224	0.00180

より高解像度にした場合の速度．36スレッドの場合のみを示す．

スペクトル⇔格子点値 1回あたりの変換にかかる時間(sec)

	TL1023		TL2047		TL4095	
	逆変換	正変換	逆変換	正変換	逆変換	正変換
SHTns	0.00351	0.00354	0.0214	0.0218	0.167	0.169
svpack	0.00224	0.00180	0.0165	0.0131	0.107	0.087

	TL8191		TL16383	
	逆変換	正変換	逆変換	正変換
SHTns	1.187	1.245	9.25	9.79
svpack	0.825	0.581	5.76	4.49

より高解像度にした場合の最大誤差(各スペクトル展開係数の実部と虚部に $(-1,1)$の範囲の一様乱数を与えて，逆変換＋正変換で戻した場合の誤差の絶対値の最大値)．　これは乱数の種に依存するので変動しうるが，変わっても数割程度)．

	TL1023	TL2047	TL4095	TL8191	TL16383
SHTns	1.79E-12	5.97E-12	1.07E-11	3.77E-11	7.35E-11
svpack	6.82E-13	1.19E-12	5.54E-12	1.61E-11	3.92E-11

svpack での実効的な GFlops値(36スレッドの場合のみを示す).

	TL1023		TL2047		TL4095	
	逆変換	正変換	逆変換	正変換	逆変換	正変換
svpack	505	630	536	672	655	803

	TL8191		TL16383	
	逆変換	正変換	逆変換	正変換
svpack	672	955	767	983

ちなみに, このマシンのピーク性能は(AVX命令使用時の定格 1.9GHzでの見積もりでは), 1.094TFlops

まとめ

ということで, ISPACK2 中の svpack について, 速度・精度ともライバルのSHTns を十分に凌駕したものができたと考えている.

なお, 今日は話さなかったが, 既に MPI + OpenMP なハイブリッド並列なパッケージ(supack)も作成してある.

とりあえず, 現状を ispack-2.0.0alpha4 として開発者向けαリリースしてある.

「And the rivalry has helped the game. 」(by Roger Federer)