GPUを使ったNon-voigt吸収線形計算の高速化

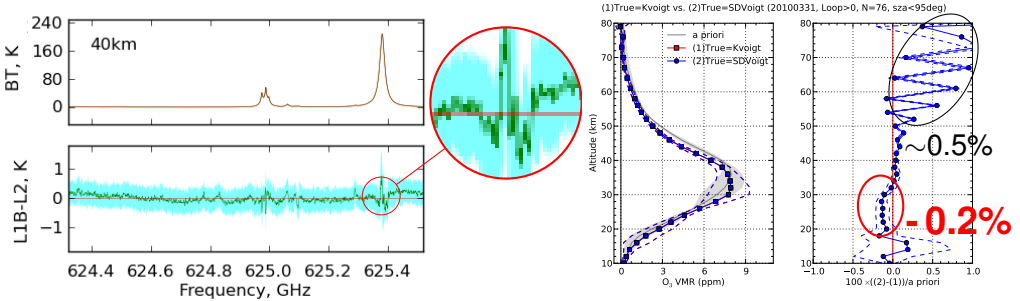
眞子直弘[1]， 鈴木睦[2]， 佐野琢己[2]， 今井弘二[2]，
尾関博之[3]， 光田千紘[4]， 塩谷雅人[5]

[1]千葉大・CEReS , [2]ISAS/JAXA , [3]東邦大, [4]富士通FIP , [5]京大・RISH

2014年2月14日
平成25年度 宇宙科学情報解析シンポジウム

This document is provided by JAXA.

SMILESにおける吸収線形状の問題



- SMILESは高感度超伝導センサにより従来にない低ノイズを実現
- 他の衛星では問題にならないNon-voigt吸収線形の影響が見える可能性
- SMILES Band Aにある強いO_3, HClのラインについてVoigt→SD-Voigtに置き換えるとリトリーバル結果が有意に異なることが分かった
- しかしながら, 計算時間の問題によりSD-Voigt関数の実装ができなかった
- 今回, 吸収線形計算にGPUを用いることで実装の目処が立った

This document is provided by JAXA.

吸収線形状モデル

- Voigt関数（Gauss関数とLorentz関数の畳み込み）

$$f(x;\sigma,\gamma) = \int_{-\infty}^{\infty} G(x';\sigma)L(x-x';\gamma)\,dx'$$

$$f(x,y) = \mathrm{Re}\,[w(x,y)] \quad w \cdots \text{複素誤差関数}$$

- Galatry関数（分子衝突によるDoppler幅の変化）

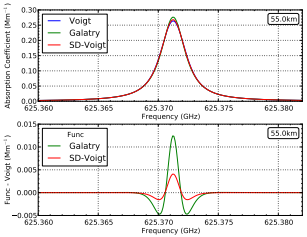
$$f(x,y) = \frac{1}{\sqrt{\pi}}\mathrm{Re}\left[\frac{1}{\frac{1}{2z}+y-ix}\,M\left(1;1+\frac{1}{2z^2}+\frac{y-ix}{z};\frac{1}{2z^2}\right)\right]$$

$M \cdots$ 合流型超幾何関数

- SD-Voigt関数（吸収線幅の速度依存性）

$$f(x,y,B_w) = \frac{2}{\pi^{1.5}}\int_{-\infty}^{\infty} e^{-v^2} v\left\{\tan^{-1}\left[\frac{x+v}{yB_w(v)}\right] + \frac{i}{2}\ln\left[1+\left(\frac{x+v}{yB_w(v)}\right)^2\right]\right\} dv$$

$$B_w(v;\alpha,q) = (1+\alpha)^{-(q-3)/(2q-2)} M\left(-\frac{q-3}{2q-2},\frac{3}{2},-\alpha v^2\right)$$

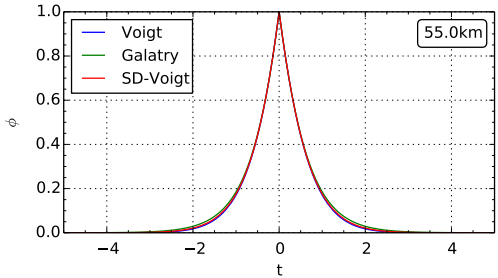


This document is provided by JAXA.

FFTを用いた吸収線形状の計算

- Voigt, Galatry, SD-Voigtの各関数はFourier変換を使って計算できる

$$f(x) = \mathcal{F}(\phi(t))$$



- Voigt関数

$$\phi = \exp\left(ix_0 t - \gamma|t| - \frac{\sigma^2 t^2}{2}\right)$$

- Galatry関数

$$\phi = \exp\left(ix_0 t - \gamma|t| - \frac{\sigma^2\left(1 - \beta|t| - \exp(-\beta|t|)\right)}{\beta^2}\right)$$

- SD-Voigt関数

$$\phi = \frac{\exp\left(ix_0 t - (\gamma - 1.5\gamma_2)|t| - \frac{\sigma^2 t^2}{2(1+\gamma_2|t|)}\right)}{(1+\gamma_2|t|)^{1.5}}$$

This document is provided by JAXA.

GPUとは

- Graphics Processing Unitの略
- PC等の画像処理を行う半導体チップ
- 内部に演算器やメモリを持つ
- CPUに比べてコア数が多く（数百〜数千），
  並列計算に適している
- GPGPU（General-Purpose computing on GPU）
  画像処理以外に一般計算にも使われる
  - **NVIDIA CUDA**
  - AMD ATI Stream
  - OpenCL
  - OpenMP

This document is provided by JAXA.

GPU 性能比較

	GTX 680	GTX TITAN	Tesla K20	Tesla K20X
コア数	1536	**2688**	2496	**2688**
コアクロック（MHz）	1006	837	706	732
メモリ容量（MB）	2048	**6144**	5120	**6144**
メモリクロック（GHz）	3	3	2.6	2.6
メモリバス幅（bit）	256	384	320	384
メモリ帯域（GB/s）	192.2	288	208	250
PCIe	**Gen3**	**Gen3**	Gen2	Gen2
単精度計算速度（Tflops）	3.09	4.5	3.52	3.95
倍精度計算速度（Tflops）	0.13	1.27	1.17	**1.32**
Compute Capability	3.0	3.5	3.5	3.5
Dynamic Parallelism	×	○	○	○
Hyper-Q	×	○	○	○
GPUDirect	×	×	○	○
ECC	×	×	○	○
価格（￥10,000）	**5**	10	35	90

This document is provided by JAXA.

CUDAを使ったGPUプログラミング

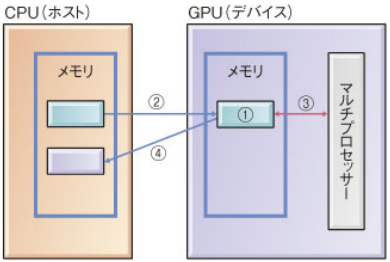
- CUDAを使うとC++を使ってGPUのプログラミングができる
- forループをスレッドに分割して並列化
- nvccを使ってコンパイル

CPUコード

```
VectorAdd(int *a,int *b,int *c,int n)
{
  for(int i=0; i < n; i++)
    c[i] = a[i] + b[i];
}
```

GPUコード

```
VectorAdd(int *a,int *b,int *c,int n)
{
  int i = threadIdx.x;
  if (i < n) c[i] = a[i] + b[i];
}
```



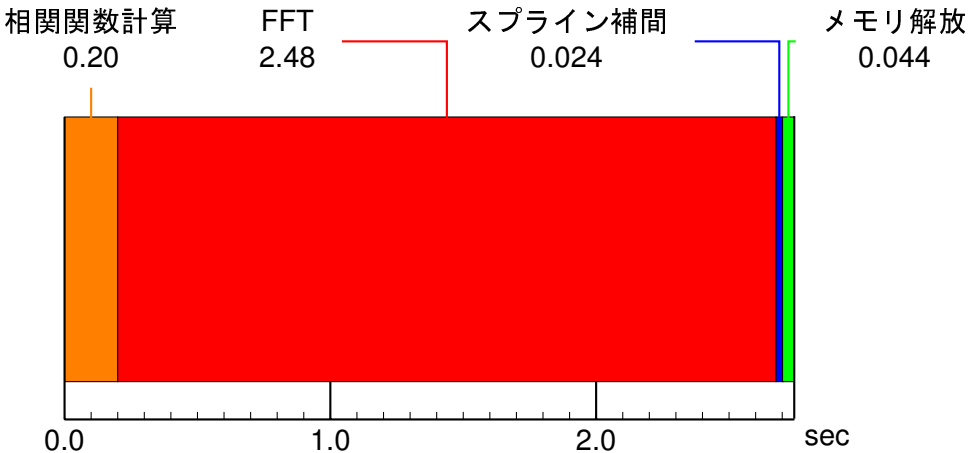
http://ascii.jp/elem/000/000/503/503572/

1. GPU上にメモリを確保する
2. CPU→GPUにデータを転送する
3. GPUで並列計算を行う
4. GPU→CPUにデータを転送する

- CPU↔GPU間の通信（PCIe）は遅いので，データ転送は少なくする

This document is provided by JAXA.

CPUを使ったVoigt関数の計算



This document is provided by JAXA.

吸収線形計算の高速化

- FFTの計算をGPUで行う
  - CUFFTライブラリを利用
  - FFT（D2Z）の入力データは2^{25}程度の倍精度実数（256 MB）
  - ➥ CPUで入力データを作成して転送すると時間が掛かる
  - ➥ GPU内部でFFTの入力データを作成
  - FFT（D2Z）の出力データは2^{24}程度の倍精度複素数（256 MB）
  - ➥ GPU内部で必要なグリッドにおける値を計算
- スプライン補間をGPUで行う
  - 必要なグリッドの値（約10000点, 100 kB）をGPUへ転送
  - 各グリッド点について, 近隣32点を使ってスプライン補間する
  - 補間で得られた値（約10000点, 100 kB）をCPUへ転送
- 計算量は膨大でも, 最終的に必要なデータは少ない
  - GPUのアプリケーションとしては理想的

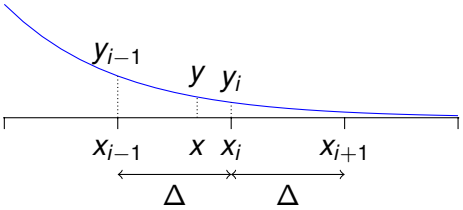
This document is provided by JAXA.

CUFFTライブラリ

- NVIDIAが開発したGPU用FFTライブラリ
- 倍精度要素数の最大値2^{26}
  メモリ容量 16バイト$\times$ 64M = 1024 MB
  （他にワークスペースが必要）
- FFTWを参考にして作られている
- 単精度実数, 複素数(R, C),倍精度実数, 複素数(D, Z)
  R2C, C2R, C2C, D2Z, Z2D, Z2Z
  （R2R, D2Dには対応していない）
- 1D, 2D, 3Dに対応
- 2, 3, 5, 7の倍数に対して最適化

This document is provided by JAXA.

32点スプライン補間



- $x_{i-1} \geq x < x_i$となるiを探す
- x周辺の32点を使って$\mathbf{v}$を求める
- $\mathbf{A}^{-1}$はあらかじめ計算してGPUメモリに格納しておく
- k_i, k_{i-1}を求める
- a_i, b_i, t, u, yを求める
- 1つのxに1スレッドを割り当てて並列計算する

$$
\begin{aligned}
\mathbf{v} &= 3 \begin{pmatrix} y_1 - y_0 \\ y_2 - y_0 \\ y_3 - y_1 \\ \vdots \\ y_{n-2} - y_{n-4} \\ y_{n-1} - y_{n-3} \\ y_{n-1} - y_{n-2} \end{pmatrix} \\
\mathbf{A} &= \begin{pmatrix} 2 & 1 & & & & & & \\ 1 & 4 & 1 & & & & \mathrm{O} & \\ & 1 & 4 & 1 & & & & \\ & & & \ddots & & & & \\ & & & & 1 & 4 & 1 & \\ & \mathrm{O} & & & & 1 & 4 & 1 \\ & & & & & & 1 & 2 \end{pmatrix} \\
\mathbf{k} &= \mathbf{A}^{-1}\mathbf{v} \quad \text{（微分係数）} \\
a_i &= k_{i-1} - (y_i - y_{i-1}) \\
b_i &= -k_i + (y_i - y_{i-1}) \\
t &= \frac{x - x_{i-1}}{\Delta}, \; u = 1 - t \\
y &= uy_{i-1} + ty_i + tu(a_i u + b_i t)
\end{aligned}
$$

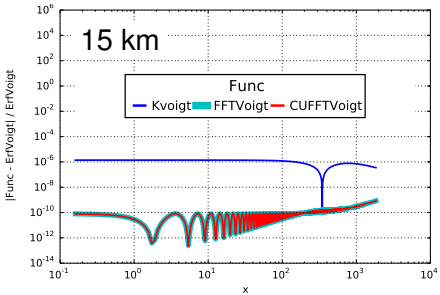
This document is provided by JAXA.

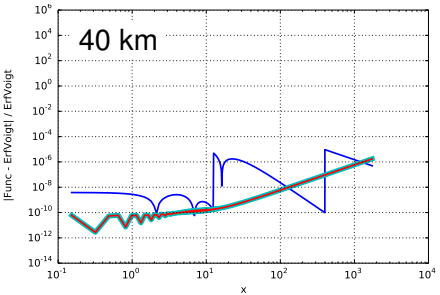
テスト方法

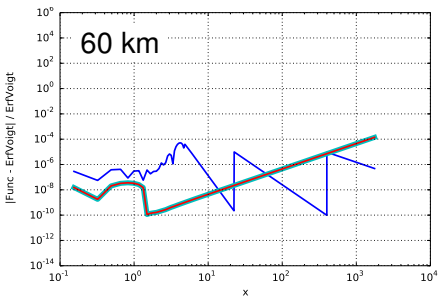
- 4つのアルゴリズムでVoigt関数を計算
  - ErfVoigt（複素誤差関数を使うアルゴリズム）
  - Kvoigt（SMILESの現行アルゴリズム）
  - FFTVoigt（FFTW3を使うアルゴリズム）
  - CUFFTVoigt（CUFFTを使うアルゴリズム）
- 周波数：624.32 〜625.52 GHz （SMILES Band A）
- サンプリング間隔：0.1 MHz
- FFTのサンプル数2^{25}
- SMILES Band Aで一番強いO_3の線形状を計算
- ErfVoigtの結果を真値として他の3つのアルゴリズムの誤差を計算

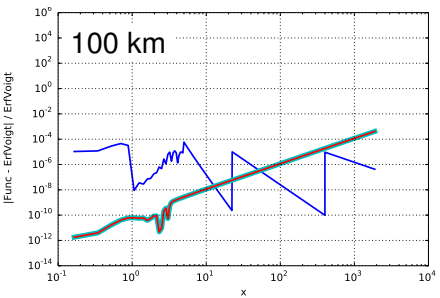
This document is provided by JAXA.

テスト結果（計算誤差）









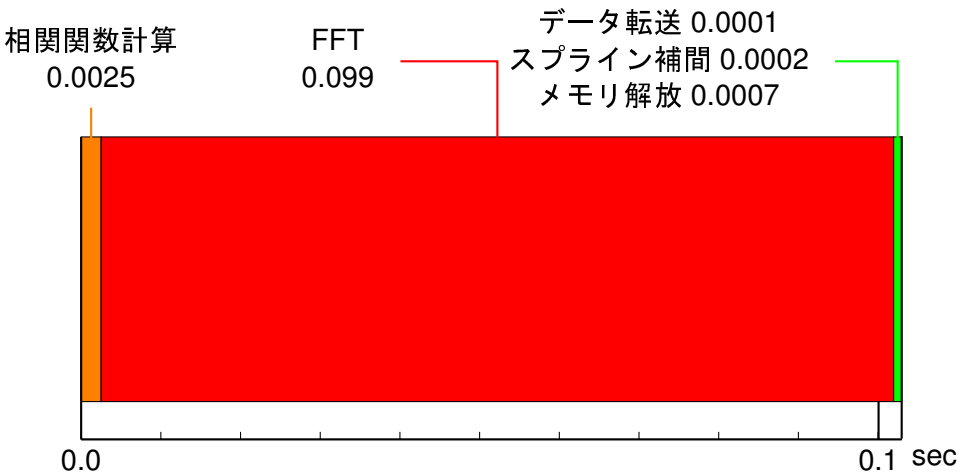
This document is provided by JAXA.

テスト結果（計算時間）

アルゴリズム	計算時間
ErfVoigt	0.0029 sec
Kvoigt	0.0001 sec
FFTVoigt	2.75 sec
CUFFTVoigt	0.1 sec

This document is provided by JAXA.

GPUを使ったVoigt関数の計算



- GPUの初期化に0.13秒必要

This document is provided by JAXA.

まとめと今後の課題

- ✓ FFTを使ったVoigt関数の計算で, 入力データの作成, FFT, 出力データの補間を全てGPU内部で行うことができた
- ✓ GPUを使うことでCPUより約30倍高速に計算することが可能になった
- 今後, SD-Voigt関数の計算をSMILES L2処理プログラムに実装する
- FFTの要素数は処理時間と計算精度から決定する

This document is provided by JAXA.