

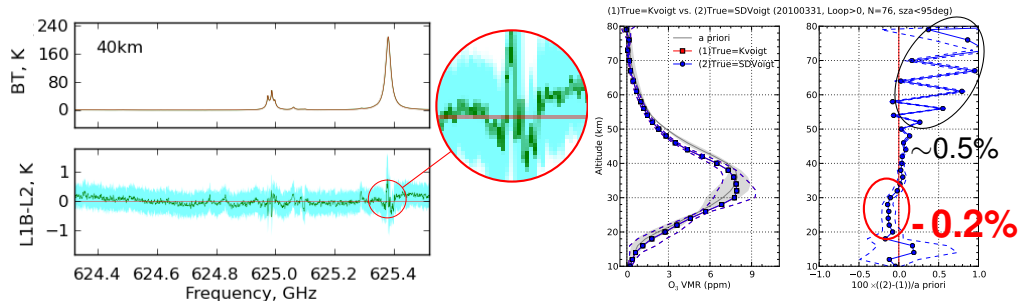
GPUを使ったNon-voigt吸収線形計算の高速化

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，²ISAS/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

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O₃, HClのラインについてVoigt→SD-Voigtに置き換えるとリトリ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

吸収線形状モデル

- **Voigt関数** (Gauss関数とLorentz関数の畳み込み)

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

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

- **Galatry関数** (分子衝突によるDoppler幅の変化)

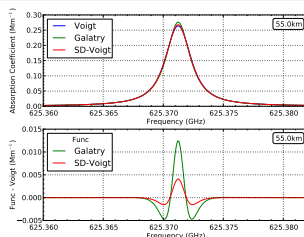
$$f(x, y) = \frac{1}{\sqrt{\pi}} \text{Re} \left[\frac{1}{\frac{1}{2z} + y - ix} M \left(1; 1 + \frac{1}{2z^2} + \frac{y - ix}{z}; \frac{1}{2z^2} \right) \right]$$

$M \cdots$ 合流型超幾何関数

- **SD-Voigt関数** (吸収線幅の速度依存性)

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

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



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**

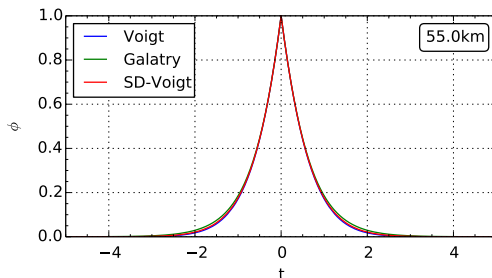
$$\phi = \exp\left(ix_0 t - \gamma|t| - \frac{\sigma^2 t^2}{2}\right)$$

- Galatry関数**

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

- SD-Voigt関数**

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



GPUとは

- **G**raphics **P**rocessing **U**nit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



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

CUDAを使ったGPUプログラミング

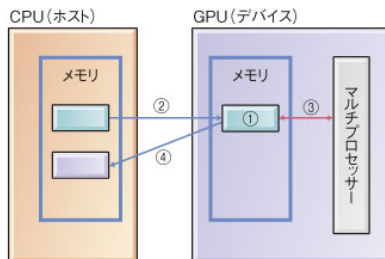
- CUDAを使うとC++を使ってGPUのプログラミングができる
- forループをスレッドに分割して並列化
- nvccを使ってコンパイル

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

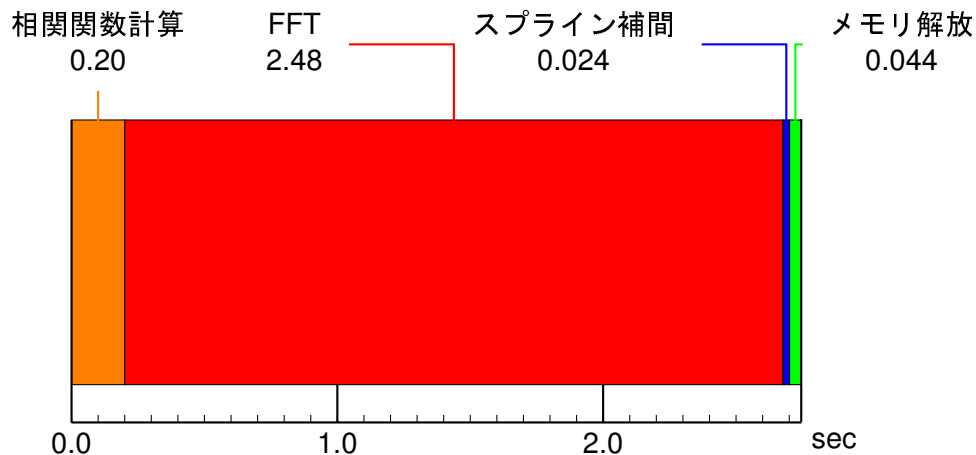
```
VectorAdd(int *a,int *b,int *c,int n)
{
    int i = threadIdx.x;
    if (i < n) c[i] = a[i] + b[i];
}
```



<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

CPUを使ったVoigt関数の計算



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

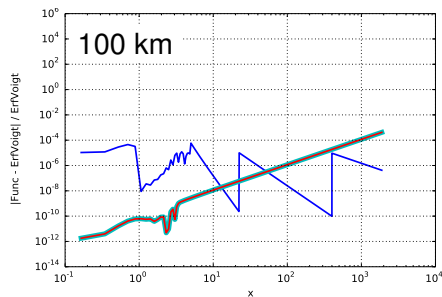
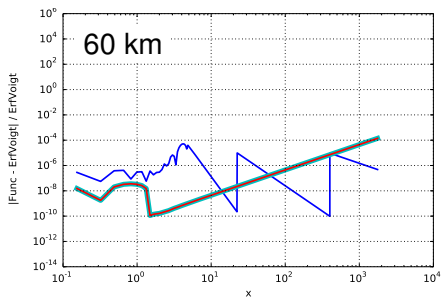
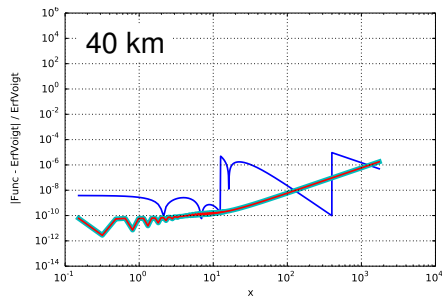
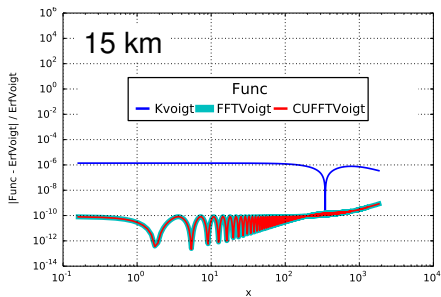
CUFFTライブラリ

- 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×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

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

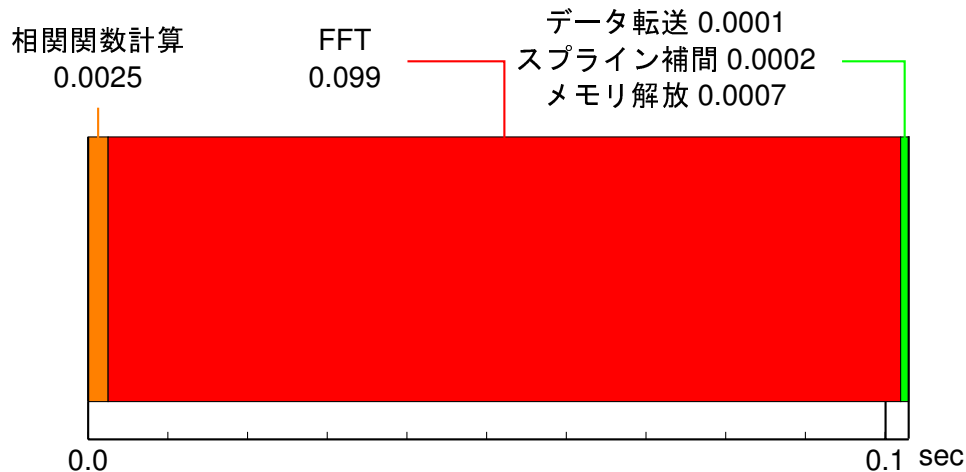
テスト結果（計算誤差）



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

GPUを使ったVoigt関数の計算



- GPUの初期化に0.13秒必要

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