GPU TECHNOLOGY CONFERENCE

거대 과학 실험 분야에 적합한 진단 데이터의 고속 병렬처리

KSTAR tokamak 장치에 적용한 사례들

오동근/국가핵융합연구소

spinhalf@nfri.re.kr

GPU TECHNOLOGY CONFERENCE

대화형 실험분석과 데이터 병렬처리

- 핵융합 연구용 자기가둠 플라즈마 실험 ➔ 거대과학 장치를 이용한 실험과학 분야
  - 다양한 진단장치의 데이터를 유기적으로 다루는 작업 ➔ 대화형 계산도구의 성능확보가 중요
  - 특성상 다채널 ➔ Data parallelism을 기반의 CUDA programming 모델에 대부분 적합
  - 대화형 분석도구로서의 python 언어 ➔ pyCUDA를 이용한 GPU연산의 구현과 통합이 용이

Development tools

- Python 2.6+
- Scientific libraries as python package for interactive computing (Numpy, Scipy ..)
- CUDA compiler interface by weaving technique : pyCUDA
- Ipython framework

For instance…

IP[y]: Notebook @cstone2

Logout

some examples - pyCUBLAS, pyCUDA, Cython and Weaving Last Checkpoint: Sep 11 16:51 (autosaved)

File Edit View Insert Cell Kernel Help

Code Cell Toolbar: None

A simple pyCUDA example

- element-wise multiplication of two vectors, a and b

In [2]:
```
import pycuda.autoinit
import pycuda.driver as drv
import numpy

from pycuda.compiler import SourceModule
mod = SourceModule("""
__global__ void multiply_them(float *dest, float *a, float *b)
{
  const int i = threadIdx.x;
  dest[i] = a[i] * b[i];
}
""")

multiply_them = mod.get_function("multiply_them")

a = numpy.random.randn(100).astype(numpy.float32)
b = numpy.random.randn(100).astype(numpy.float32)

dest = numpy.zeros_like(a)
multiply_them(drv.Out(dest), drv.In(a), drv.In(b),block=(400,1,1), grid=(1,1))

print dest - a*b
```

```
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
```

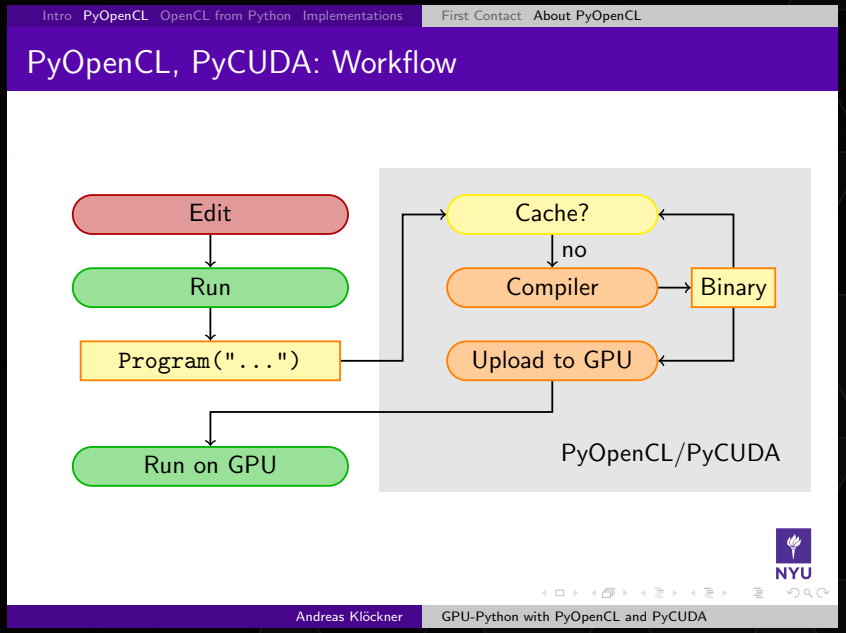
GPU TECHNOLOGY CONFERENCE

Why script language (ex> python) with GPU?

According to the developer, Andreas Klönker →

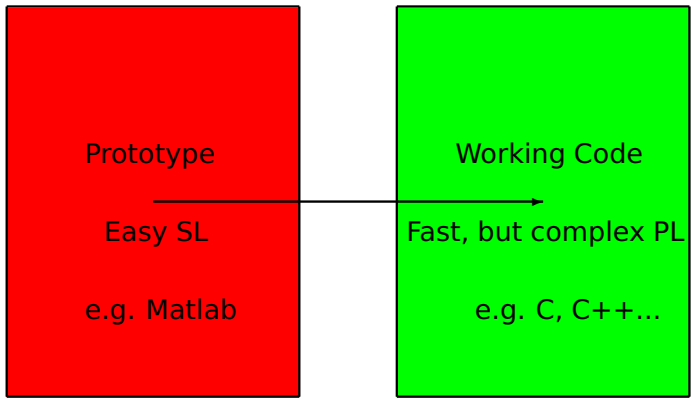
python + CUDA = pyCUDA ←
python + OpenCL = pyOpenCL

- GPU : everything that the scripting language are not!
  - Highly Parallel
  - Architecture sensitive
  - Built for maximum FP/ Memory throughput
- CPU : largely restricted to control tasks (~1000/s)
  - Scripting fast enough

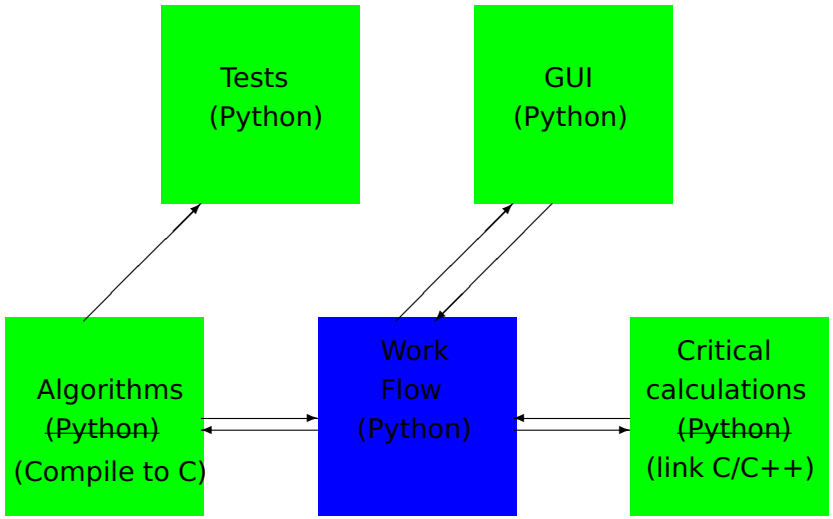


Paradigm shift in code development

Classical prototyping process



Prototyping with Python

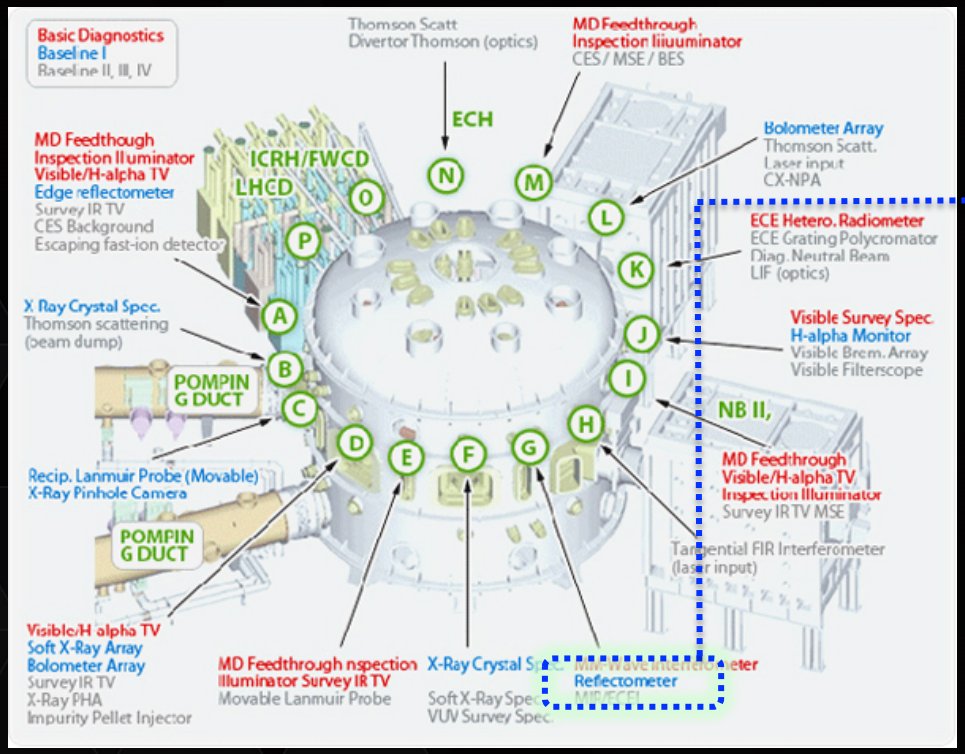


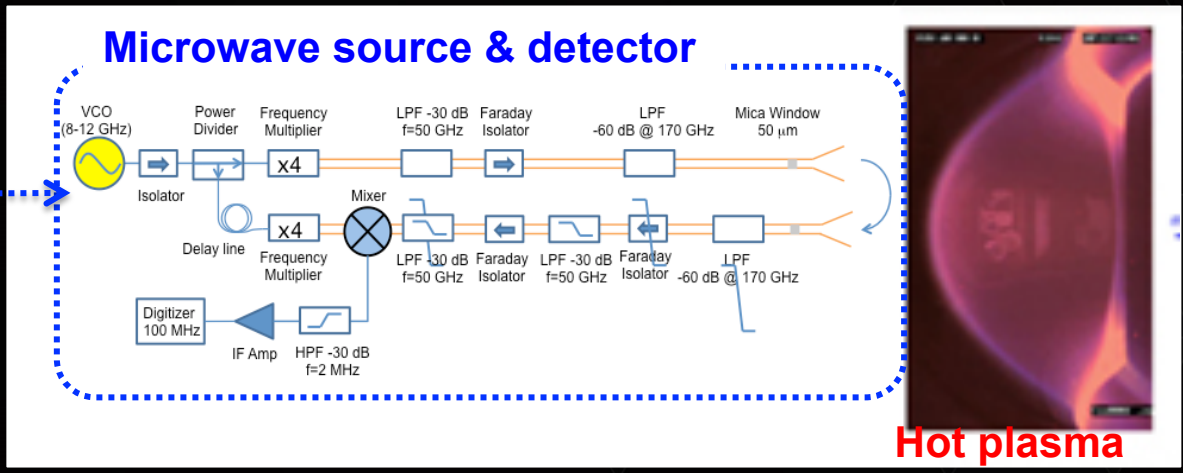
Case studies in KSTAR project (2014)

- CASE #1: Microwave 반사계를 이용한 KSTAR 플라즈마 전자밀도분포
- CASE #2: KSTAR 운전 열부하와 초전도 자석 온도 예측을 위한 0-D 모형

CASE #1. Diagnostics of electron density profile using microwave reflectometer

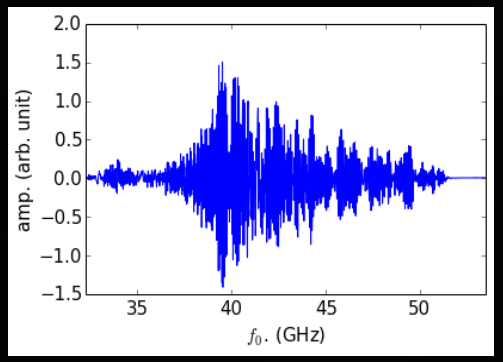
- KSTAR the tokamak – a plasma device of magnetic confinement



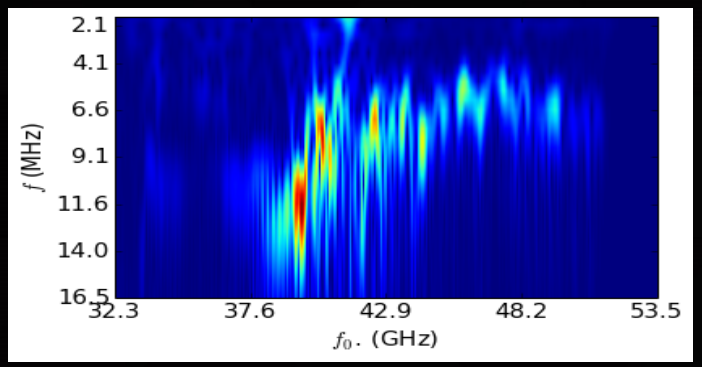


GPU TECHNOLOGY CONFERENCE

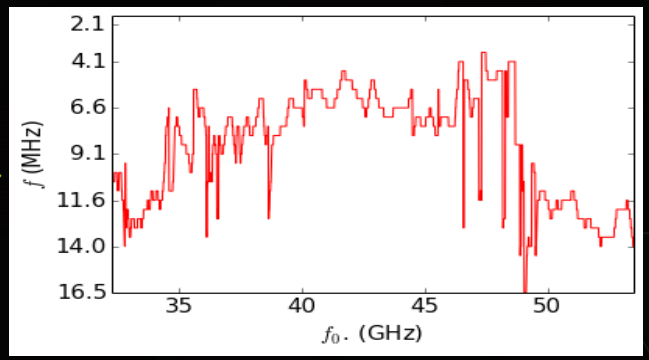
Tool-path in reflectometry (~1000 signals in parallel)



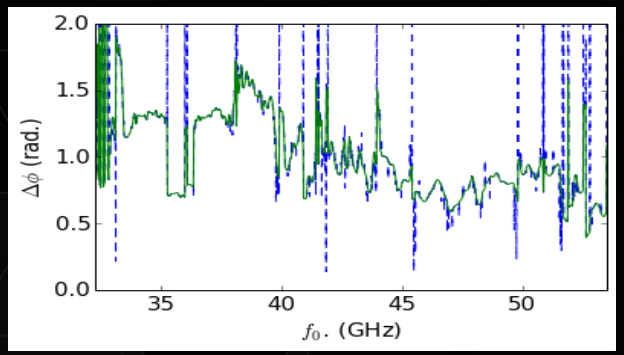
Detected signal



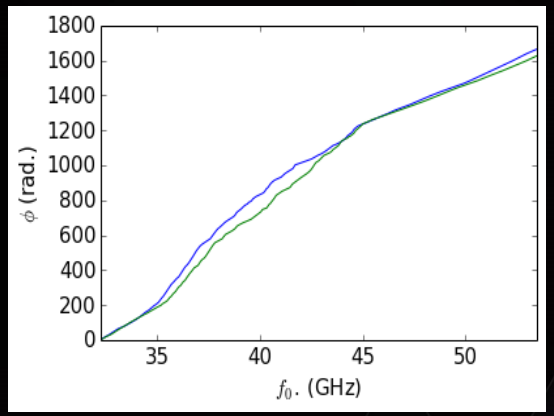
Spectrogram (Morlet wavelet)



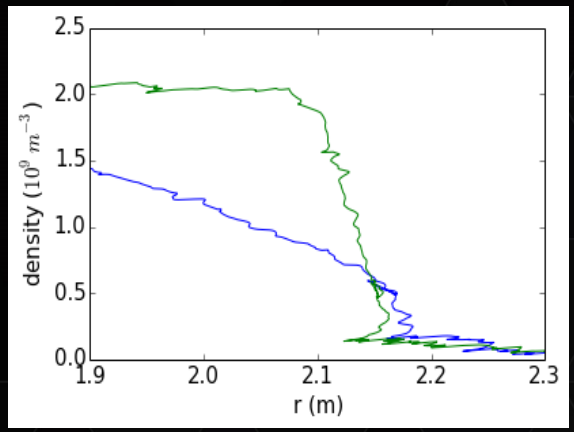
Tracing the reflected wave at the peak



Phase recovery for reflected wave

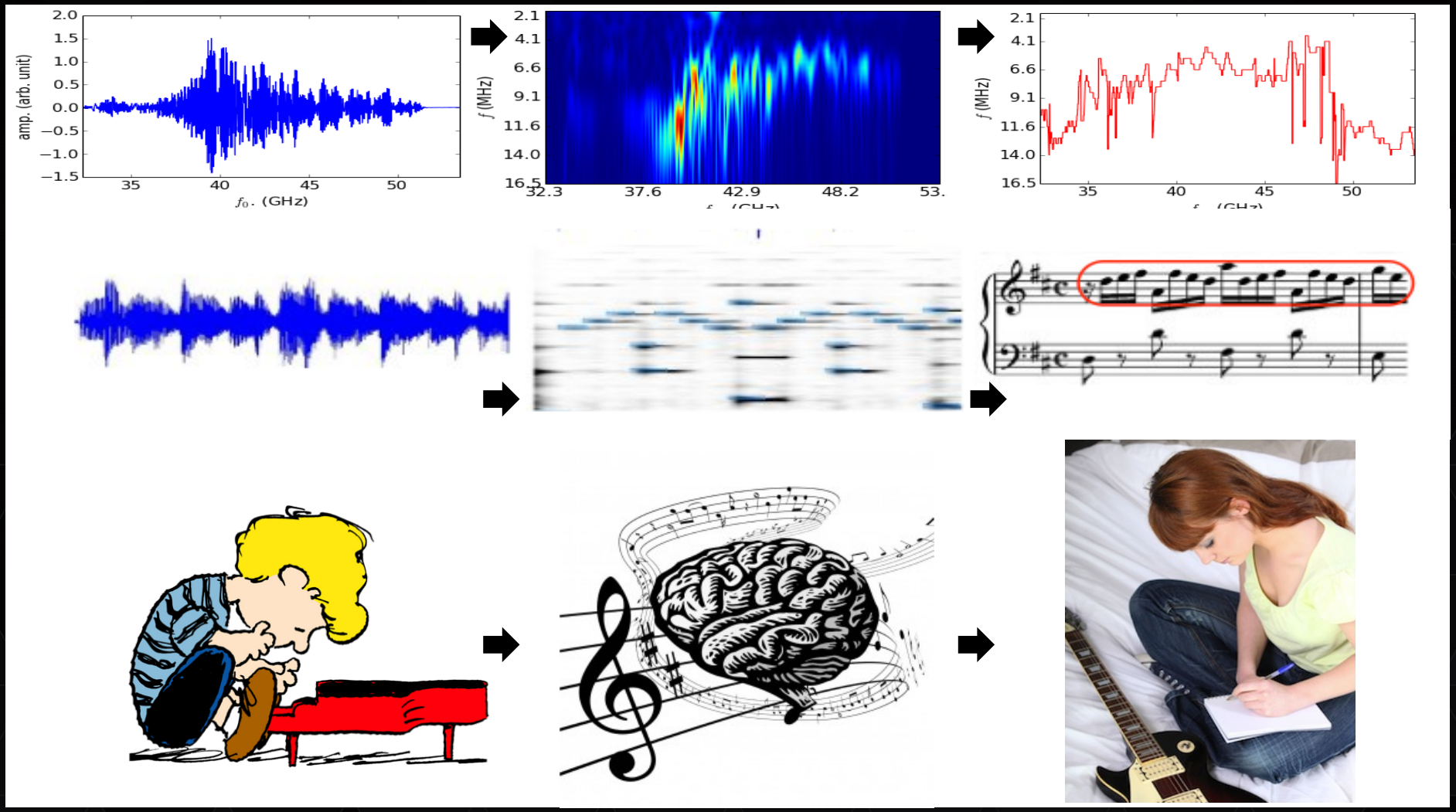


Cummulative phase

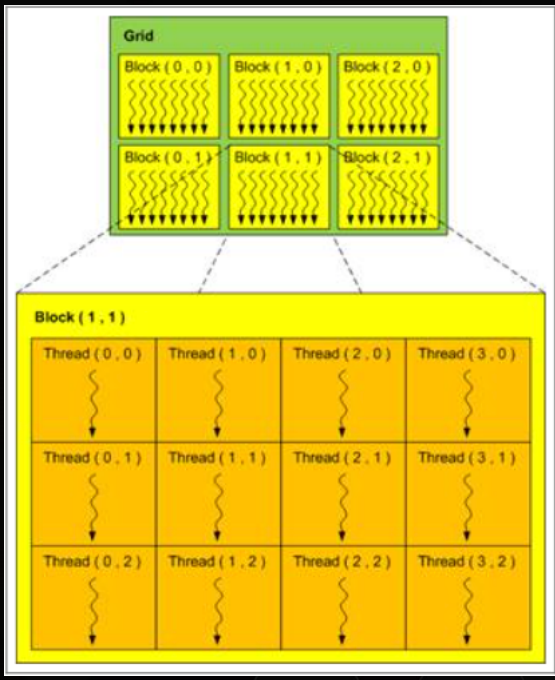


Profile reconstruction
(Bottolier-Curttet algorithm)

It's similar to picking up the melody in transcription



GPU coding strategy



- ✓ **Data parallel in multiple thread blocks = basic parallelism**
  : processing many ($O(10^2)$ ~ $O(10^3)$) signals in parallel

- ✓ **Fine-graining depending on the numerical process in each stage**
  : Element-wise operations → easy to fine-graining
  (wavelet multiplication in Fourier space, picking up maximum amplitude, phase difference etc…)
  : Reduction
  (cut-off finding by bitwise-operation, integral in reconstruction)
  : Scan
  (getting cumulative phase by work efficient scan)

- ✓ **Multi-pass implementation in the stage**
  : No concurrency is guaranteed for inter-block synchronization
  → unavoidable overhead of threads kernel loading in each pass

GPU TECHNOLOGY CONFERENCE

Parallelized wavelet transform

It's straightforward : FT → multiplication of wavelet → IFT

```
import pycuda.driver as cudrv
import pycuda.gpuarray as gparray
import pycuda.autoinit
from pycuda.compiler import SourceModule
from scikits.cuda import fft as cufft

source = """
__device__ __constant__ float tpi =6.2831853071795862;
__device__ __constant__ float s_omega0 = 6.0;
__device__ __constant__ int Nscale;
__device__ __constant__ int NscaleQ;
__device__ __constant__ int NscaleV;
__device__ __constant__ int Nn;
__device__ __constant__ int Nsig;
__device__ __constant__ int Ncol;

__global__ void set_psihat(float *scale, float *psihat) {
    int ix = threadIdx.x + blockIdx.x*blockDim.x;
    int iy = threadIdx.y + blockIdx.y*blockDim.y;
    float s_omega, x;

    if (iy < Nscale && ix < Ncol) {
        s_omega = .5*tpi*ix/(Ncol-1)*scale[iy];
        x = s_omega - s_omega0;
        psihat[ix+Ncol*iy] = 0.75112554*exp(-.5*x*x) *sqrt(tpi*scale[iy]);
    }
}

__global__ void conv_rc( float *c_data, float *r_wf, float *c_ret) {
    int ix = threadIdx.x + blockIdx.x*blockDim.x;
    int iy = threadIdx.y + blockIdx.y*blockDim.y;
    int iscale = iy % Nscale;
    int isig = iy / Nscale;
    int idata;

    if (iscale < NscaleQ)
        idata = 2*isig;
    else if (iscale < NscaleQ+NscaleV)
        idata = 2*isig+1;

    if (ix < Ncol && iscale < Nscale && isig < Nsig) {
        c_ret[2*ix+2*Nn*iy] = c_data[2*ix+2*Ncol*idata]*r_wf[ix+Ncol*iscale];
        c_ret[1+2*ix+2*Nn*iy] = c_data[1+2*ix+2*Ncol*idata]*r_wf[ix+Ncol*iscale];
    }
}
"""

mod = SourceModule( source)
```

```
set_psihat = mod.get_function("set_psihat")
conv_rc = mod.get_function("conv_rc")

set_psihat( gpScale, gpPsihat, block=((Nn/2+1)/32,Nscale/4,1),  grid=(64,4))

plan = cufft.Plan( Nn, np.float32, np.complex64, batch=Nsig)
cufft.fft( gpData, gpDatahat, plan)
conv_rc( gpDatahat, gpPsihat, gpConvhat, block=((Nn/2+1)/32, Nscale/4,1),  grid=(64,Nsig*4))
plan = cufft.Plan( Nn, np.complex64, np.complex64, batch=Nscale*Nsig)
cufft.ifft( gpConvhat, gpCWT, plan)
```

100 ms
to calculate
100 spectrograms in parallel …

Cut-off finding - a kind of reduction algorithm

: Finding the position of negative segment longer than given tolerance
→ reduction of indices by bitwise operation is designed to get the distance and leading index between rising and lowering edge

: Reduction process

- Present the positions as 2 bits data with the left and right indices
  : + to - edge as 01, - to + edge as 10, otherwise 00.

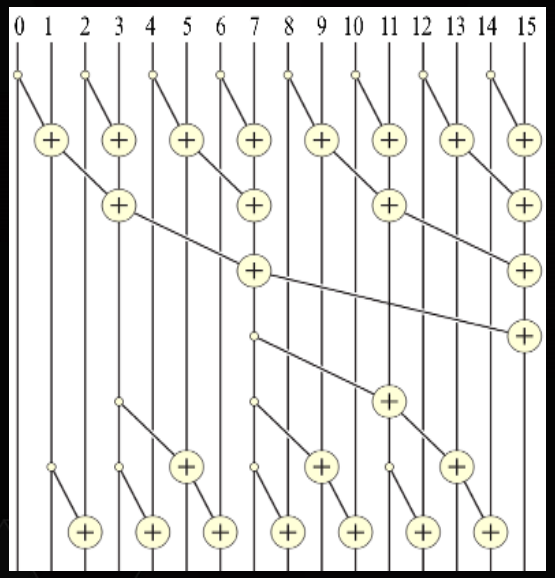
- Apply the bitwise operation for reduction :

$x \otimes 00 = x$ and $00 \otimes x = x$	(rejecting non-zerocrossing indices)
$\square 1 \otimes 1 \square = \square\square$ & updating cutoff	(reduction and update)
$10 \otimes 01 = 11$	(keeping the positive segment)

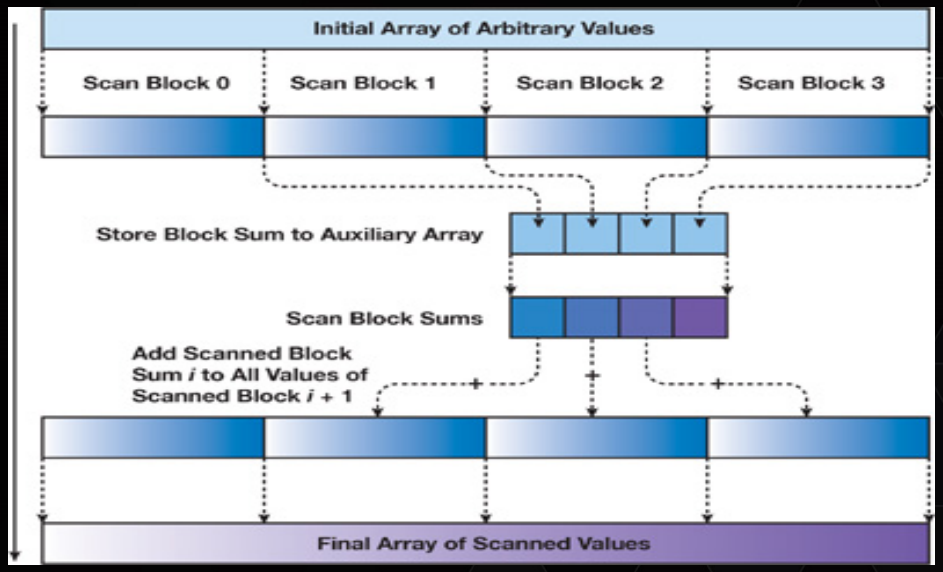
Cumulative phase $[a_1,\ a_2\ ,a_3\ \ldots,\ a_n] \rightarrow [a_1,\ a_1 \oplus a_2\ ,a_1 \oplus a_2 \oplus a_3\ \ldots,\ a_1 \oplus a_2 \oplus a_3 \cdots \oplus a_n]$

: Scan (parallel prefix sum) algorithm = work efficient parallel scan

In the lecture of W. Hwu (https://gist.github.com/wh5a/4500706#file-mp5-c)



Basic idea of scan block for work efficient parallel scan
(Lecture 14 in applied parallel programming
ECE408/CS483/ECE498al, University of Illinois, 2007-2012
by Wen-mei W. Hwu)



GPU Gems 3 (a free ebook)
Chapter 39. Parallel Prefix Sum (Scan) with CUDA

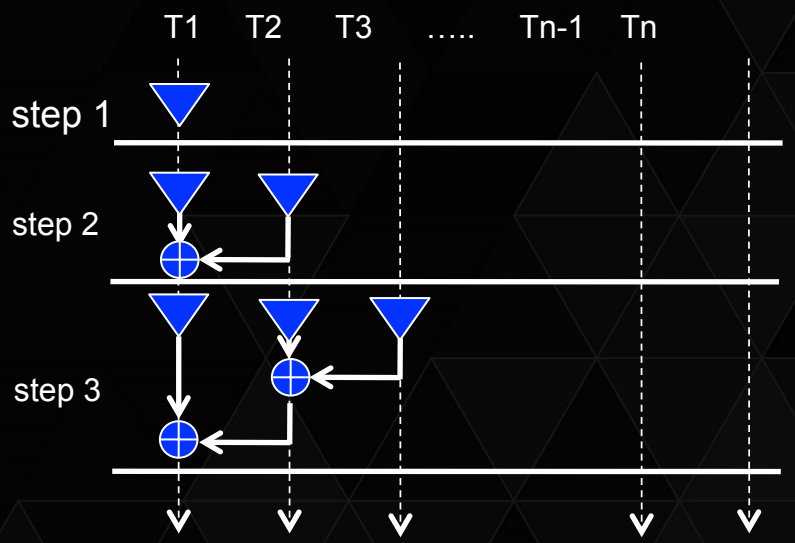
Bottolier-Curtet reconstruction

- **Parallel code is NOT possible for the main steps of (r, n_e) pair reconstruction**
  → We apply data parallelism for the bunch of signals !

- **We launch thread-kernel *N* times - *N* is the number of pairs (data points)**
  : Overheads are not avoidable for the loading time

- **Fine-graining for the integral in each loop**
  : for n^{th} step, $n(n-1)/2$ loops ⟶ ~$n\log(n)$ loops

- **Parallelism :**
  : multiple data x threads for integral



Step 1: Import density_profile class - specify the shot number of the data set when importing..

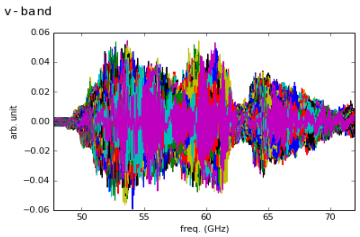
```
In [1]: from KSTARreflect import density_profile

In [2]: eprof = density_profile( shotnumber=10206, params="refQVW11383.sav")
```

Step 2: Check the structure of reflectometer data

```
In [3]: time, ndata, Ip = eprof.getTimeInfo( band='v')
        - number of segments = 3

        - time stamps (start and end of the segments):
        [['0.490' '0.540']
         ['0.990' '1.040']
         ['1.490' '1.540']]

        - number of data in the segments:
        [4999999 4999999 4999999]

        - plasma current of the segments:
        ['222 kA', '374 kA', '506 kA']
```

Step 3: Read a segment data (+ time slices information) of specified plasma shot

```
In [4]: eprof.setSignal( bands = 'qv', segidx = 1, sigstep = 4)
        .. reading time domain information (band = q) .....
        .. reading rf signal (band = q, segment index = 1) .....

        .. reading time domain information (band = v) .....
        .. reading rf signal (band = v, segment index = 1) .....
```

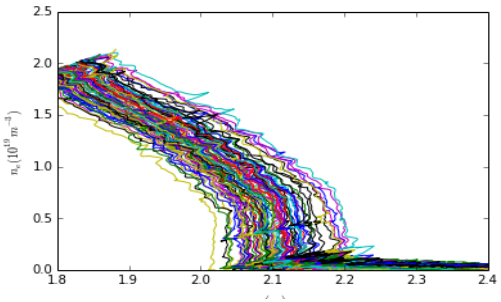


Step 4: Calculate density profiles (Signals --> Wavelet spectrogram --> Phase recovery --> Botollier-Curtet reconstruction)

```
In [8]: import time
        t0 = time.time()

        l_prof = eprof.calcProfile()

        print "- total computation time =", (time.time() - t0)*1000., 'ms'
        nsig = len(l_prof)
        print "- number of profiles =", nsig
```

```
- total computation time = 474.127054214 ms
- number of profiles = 250
```

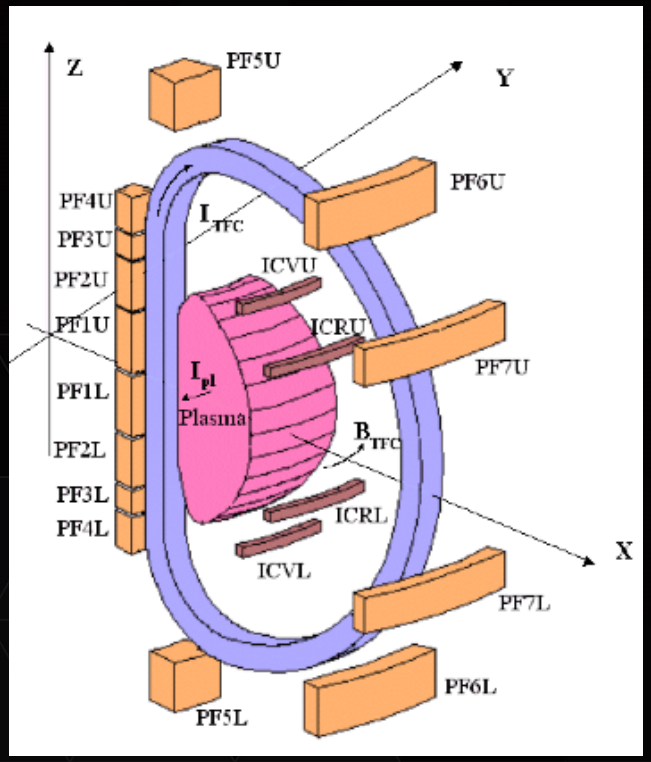
```
In [10]: for epf in l_prof:
             plot(epf[0], epf[1]*1E-19)
         xlim([1.8, 2.4])
         xlabel('$r (m)$')
         ylabel('$n_e (10^{19} m^{-3})$')
         show()
```

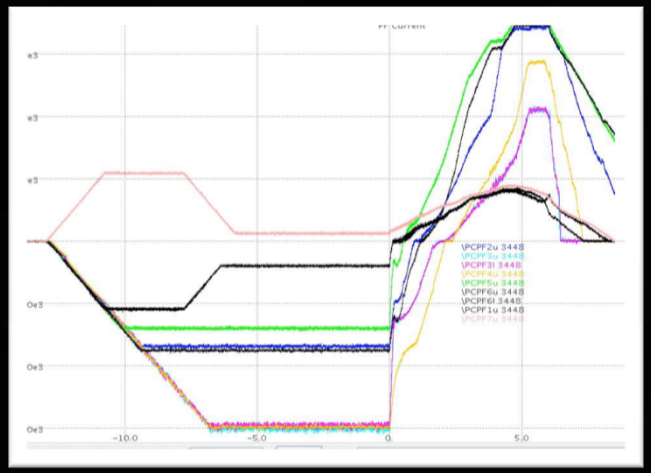


- 2.3 sec. for 1000 signals by Tesla® K20m (including initial loading time of code into GPU)
- 500 times faster than optimized IDL routine !
- Very easy to integrate with the data processing platform as a python class

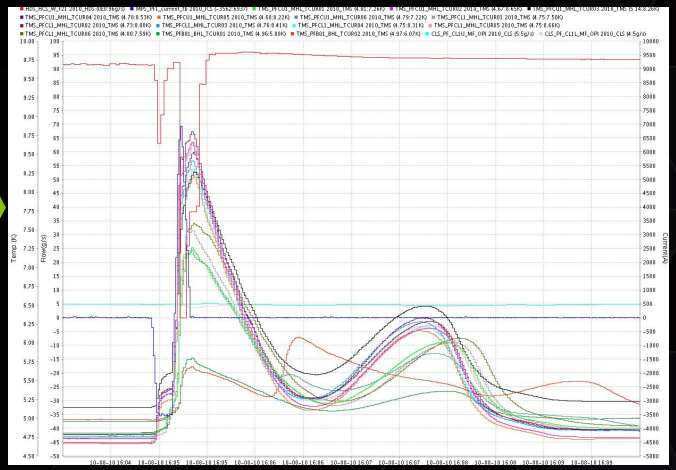
CASE #2. heat load of superconducting magnet by AC loss

- Superconducting magnet system of KSTAR tokamak





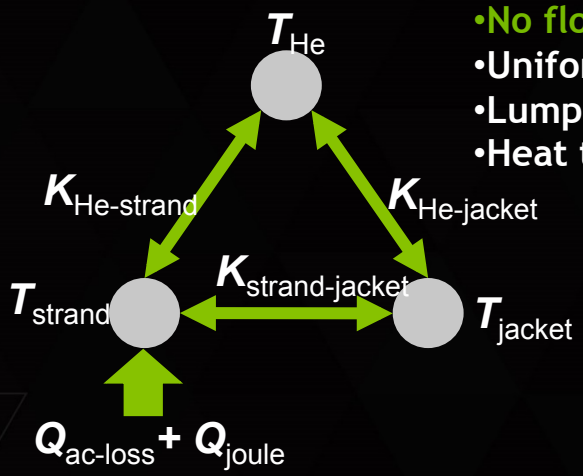
Magnet current profile (~kA)



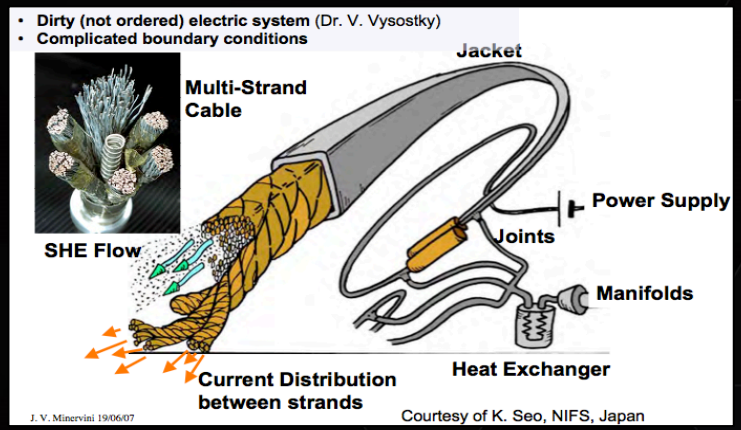
Magnet temperature (4~10K)

Zerodee (0-D) model of the superconducting PF CICCs

✓ The minimalist white-box (physics-based) model of superconducting CICC …



- No flow
- Uniform (current etc..)
- Lumped heat capacities
- Heat transfer by contacts



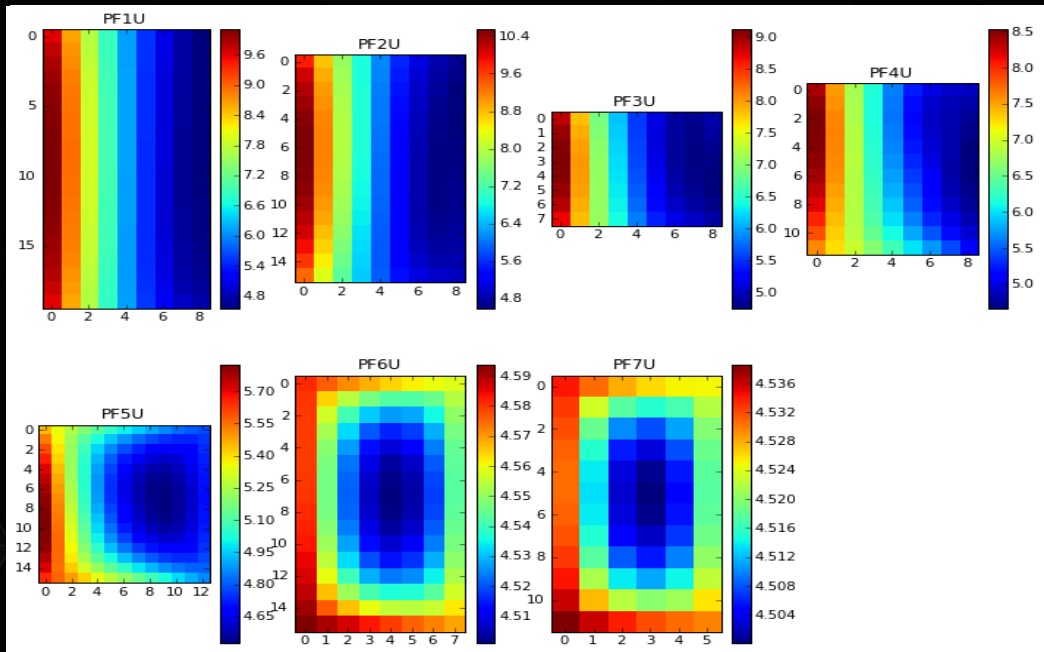
Sure, real-world is complex!

✓ Traditionally, this model has been used in the stage of magnet design…

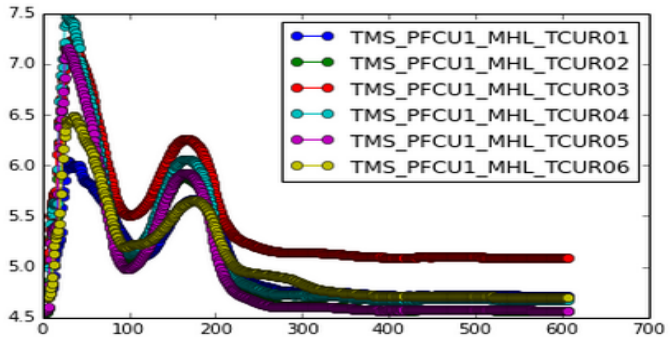
- Just for quick estimation, for instance, of stability margin

: It's simplicity allows extensive parametric study ← essential in the design phase ..

KSTAR hotspot code based on the zerodee (0-D) model




```
In [7]: from KSTARTHData import readPFarch

        time, data, unit, chs = readPFarch(5947, 'PF1U')

        . . .

In [8]: plot(time, data[:,5:], '-o')
        legend(chs[5:])
        show()
```



- Conservative (overestimated) estimation
- Independent calculation for each cross-section of the conductors
  → easy to be parallelized…

How to accelerate for real-time TH model (I)

✓ Step 1 : Speed optimization of the main ODE routine

~~implicit scheme with linearization~~

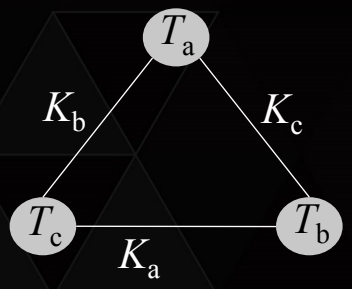
RK4 with the time step of stable condition

$$\frac{d}{dt}\begin{pmatrix} T_{st} \\ T_{he} \\ T_{jk} \end{pmatrix} = \begin{pmatrix} -\frac{K_{st-he}+K_{st-jk}}{C_{st}} & \frac{K_{st-he}}{C_{st}} & \frac{K_{st-jk}}{C_{st}} \\ \frac{K_{st-he}}{C_{he}} & -\frac{K_{st-he}+K_{he-jk}}{C_{he}} & \frac{K_{he-jk}}{C_{he}} \\ \frac{K_{st-jk}}{C_{jk}} & \frac{K_{he-jk}}{C_{jk}} & -\frac{K_{st-jk}+K_{he-jk}}{C_{jk}} \end{pmatrix} \begin{pmatrix} T_{st} \\ T_{he} \\ T_{jk} \end{pmatrix} + \begin{pmatrix} \dot{q}_{st}/C_{st} \\ 0 \\ 0 \end{pmatrix}$$

von Neumann stability condition should be satisfied for each row…

: Our system has an analogy of cylindrical lattice.
So, the Fourier component of the solution is ..
For (a,b,c)=(1,2,3) (2,3,1) (3,1,2)

$$T_n^{(i)} = a^{(i)} e^{im\theta_n}$$
$$(\theta_a = 0,\ \theta_b = +\tfrac{2\pi}{3},\ \theta_c = -\tfrac{2\pi}{3})$$



$$|G|^2 = \left|\frac{T_a^{(i+1)}}{T_a^{(i)}}\right|^2 = \left\{1 - \Delta t\left(\frac{K_b + K_c}{C_a}\right)(1 - \cos\tfrac{2\pi}{3}m)\right\}^2 + \Delta t^2\left(\frac{K_b + K_c}{C_a}\right)^2 \sin^2\tfrac{2\pi}{3}m$$

Then, the stability condition is…

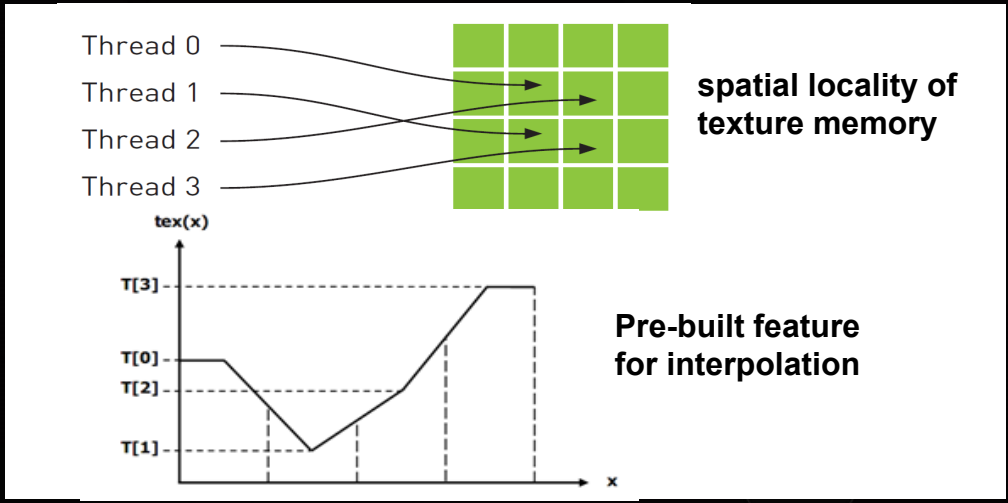
$$\forall a \quad |G|^2_{\max} < 1 \quad \Rightarrow \quad \Delta t < \min\left(C_a\left(\frac{K_b + K_c}{K_b^2 + K_c^2}\right)\right)$$

How to accelerate for real-time TH model (II)

✓ Step 2 : Speed optimization for computation of material properties

~~Calling functions in the Cryosoft™ library~~

Accessing tables in texture memory

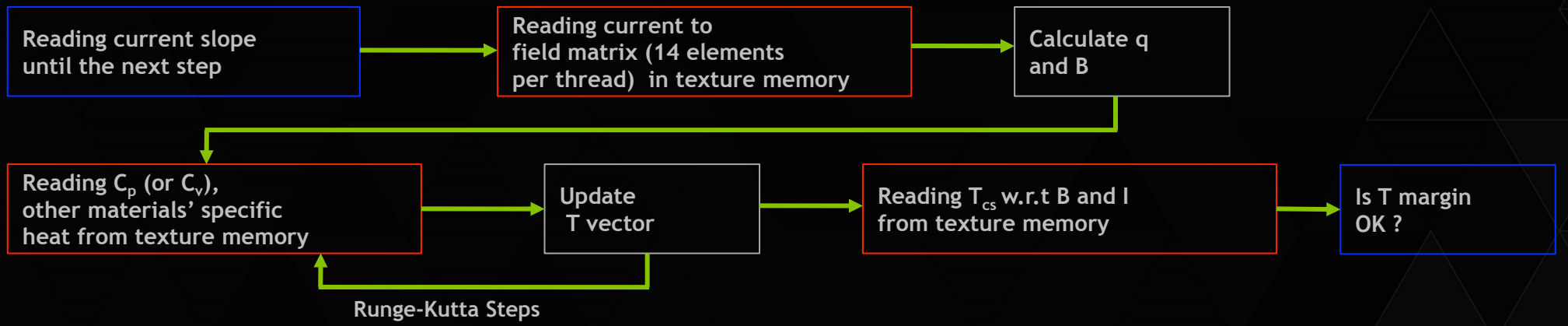


Why texture memory

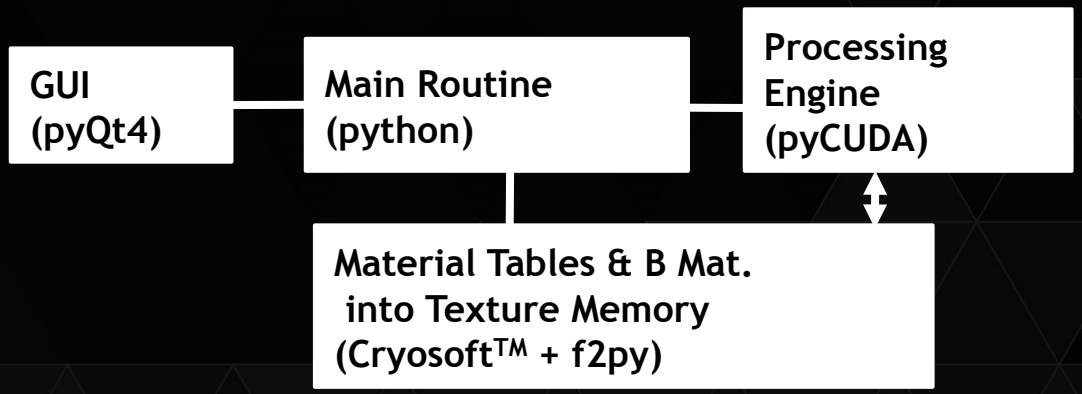
- Texture memory is optimized for 2D spatial locality (where it gets its name from).
- The addressing calculations can be calculated outside of the kernel in the hardware
- Data can be accessed by different variables in a single operation 8 bit and 16 bit data can be automatically converted to floating point numbers between 0 and 1.0

Plan to code in CUDA - data parallel model with SIMD cores

✓ For each thread (1824 threads for all PF cross-sections) :

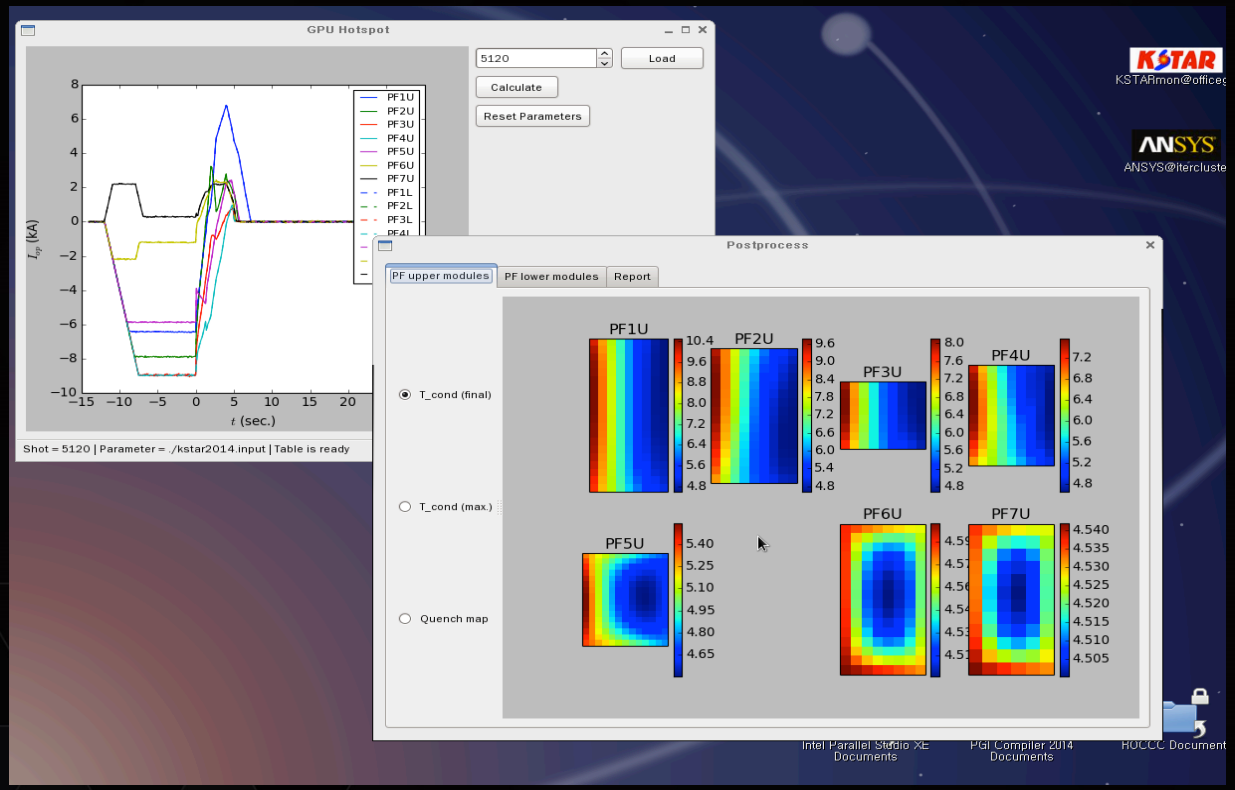


✓ pyQt + pyCUDA + f2py + cryosoft™ libray :



GPU TECHNOLOGY CONFERENCE

So, what we have is …



- ✓ 1.6 sec for 45 sec current scenario.
- ✓ Existing code in Fortran spends ~10 min for the same computation - 300 times faster
- ✓ Written as python class - easy to be integrated into the interactive plat-form

- Instantaneous analysis is possible for designed scenarios to check operation safety.
- Within 13 ms to estimate 0.1 sec later → feasible to real-time applications in speed

마치며...

- Nvidia® GPU을 이용한 (Tesla® K20m) 핵융합 플라즈마 진단데이터 가속처리기법을 소개하였다

  : 다채널 대용량인 진단의 특성상 매우 효과적인 성능개선이 가능하다.

- 특히 대화형 분석도구 구현을 위해서는, pyCUDA을 이용하면

  효율적인 코딩이 가능하며, 쉽게 통합 분석 환경에 이식할 수 있다.

- 향후 KSTAR 대화형 실험분석 platform 구현에 있어서 GPU를 기본 HW로

  활용할 수 있는 라이브러리를 개발하고 있다.

- 대부분 진단은 Plasma 제어와 접목 가능하므로 (이 분야 중요 기술적의제), 핵융합에서 GPU의 RTOS 응용는 향후 흥미로운 연구가 될 것이다.


GPU TECHNOLOGY CONFERENCE
THANK YOU
JOIN THE CONVERSATION
#GTC15