FFTX for Micromechanical Stress-Strain Analysis

Anuva Kulkarni, Daniele G. Spampinato, Franz Franchetti
Dept. of Electrical and Computer Engineering,
Carnegie Mellon University

Porting Scientific codes to GPUs

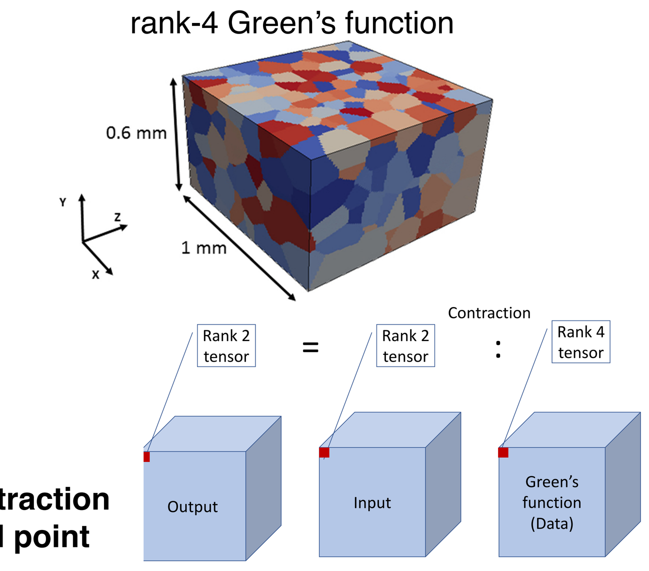
Common characteristics of scientific codes:
- Usually in Fortran
- FFT-based simulations involve all-to-all communication
- High memory requirement

Incompatibility with GPUs:
- GPUs haves small on-chip memory (~16GB max)
- Communication latencies in data movement

Solutions for porting code:
- Domain decomposition (regular or irregular)
- Exploit properties of data and convolution kernel
- Sampling/ pruning used so that domain results fit on GPU memory

Combining performance with scaling scientific codes requires algorithm restructuring.

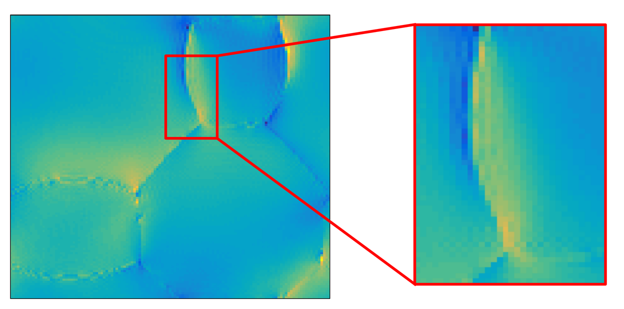
Case study: MASSIF
- Hooke's law simulation
- Partial Differential Equation solved by Green's function method
- FFT-based convolution and tensor contraction between rank-2 tensors and rank-4 Green's function



Tensor contraction at each grid point

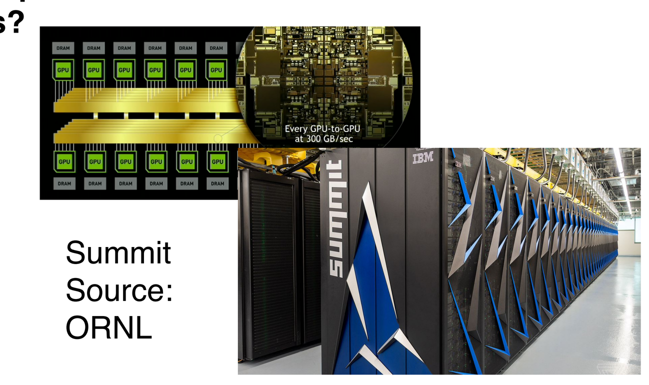
Algorithm 1 MASSIF Inner loop

1: **Initialize:**
$\epsilon^0 \leftarrow E, \quad \sigma^0_{mn}(\mathbf{x}) \leftarrow C_{mnk\ell}(\mathbf{x}) : \epsilon^0_{k\ell}(\mathbf{x})$
2: **while** $e_s > e_{\text{tol}}$ **do**
3: $\hat{\sigma}^{(i)}_{mn}(\xi) \leftarrow \text{FFT}(\sigma^{(i)}_{mn}(\mathbf{x}))$
4: Check convergence
5: $\Delta\hat{\epsilon}^{(i+1)}_{k\ell}(\xi) \leftarrow \hat{\Gamma}_{k\ell mn}(\xi) : \hat{\sigma}^{(i)}_{mn}(\xi)$
6: Update strain: $\hat{\epsilon}^{(i+1)}_{k\ell}(\xi) \leftarrow \hat{\epsilon}^{(i)}_{k\ell}(\xi) - \Delta\hat{\epsilon}^{(i+1)}_{k\ell}(\xi)$
7: $\epsilon^{(i+1)}_{k\ell}(\mathbf{x}) \leftarrow \text{iFFT}(\hat{\epsilon}^{(i+1)}_{k\ell}(\xi))$
8: Update stress: $\sigma^{(i+1)}_{mn}(\mathbf{x}) \leftarrow C_{mnk\ell}(\mathbf{x}) : \epsilon^{(i+1)}_{k\ell}(\mathbf{x})$



Proposed algorithmic solution:
- Domain decomposition with grains are domains
- Domain-local FFT followed by convolution and tensor contraction
- Green's function computed on-the-fly to avoid storage
- Adaptive sampling of dense convolution result to fit on GPU memory

Complex data mappings! How to get maximum performance on various platforms?



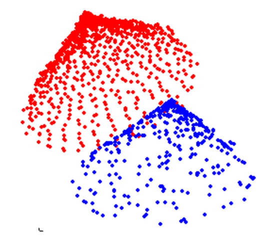
DGX-2
Source: NVIDIA

Summit
Source: ORNL

Challenges FFTW

FFTW is de-facto standard interface for FFT
- **Vendor libraries support the FFTW 3.X interface:** Intel MKL, IBM ESSL, AMD ACML (end-of-life), Nvidia cuFFT, Cray LibSci/CRAFFT

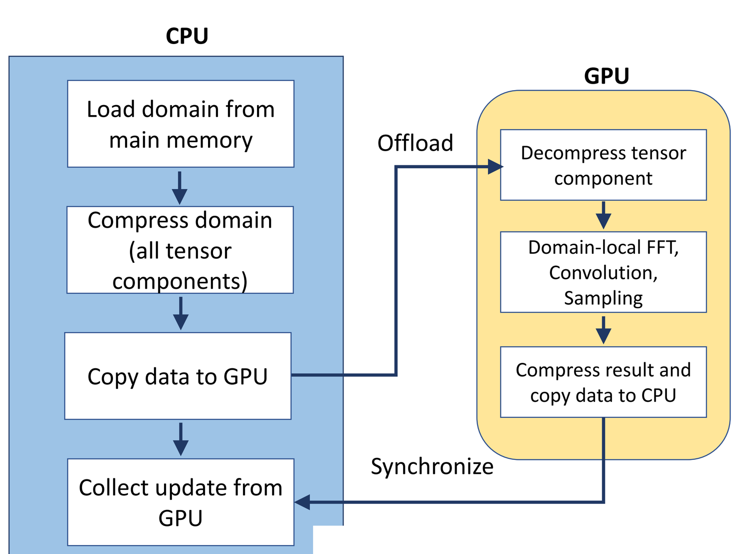
Some Issues:
- No native support for accelerators (GPUs, Xeon PHI, FPGAs) and SIMT
- Parallel/MPI version does not scale beyond 32 nodes
- No analogue to LAPACK for spectral method

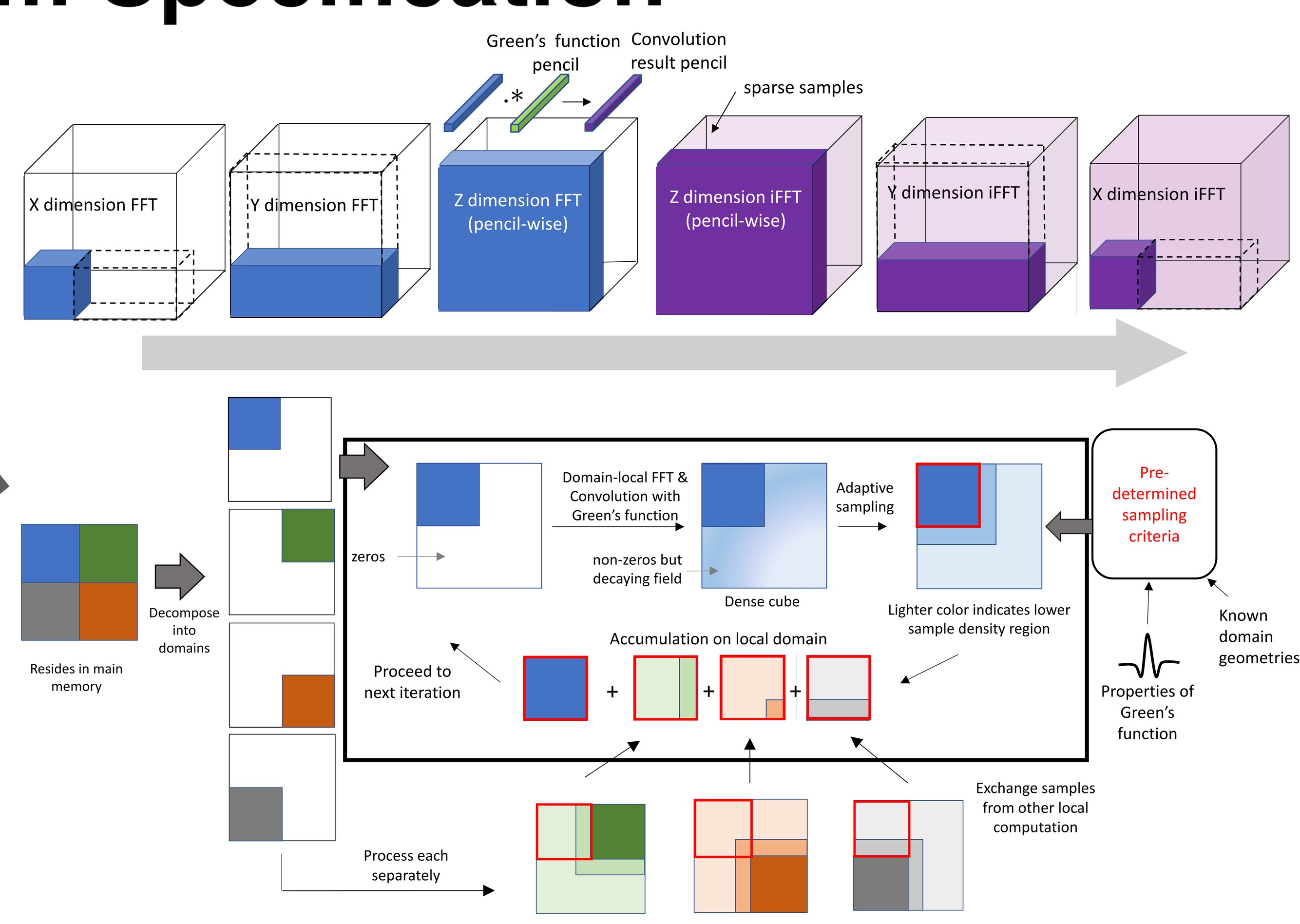


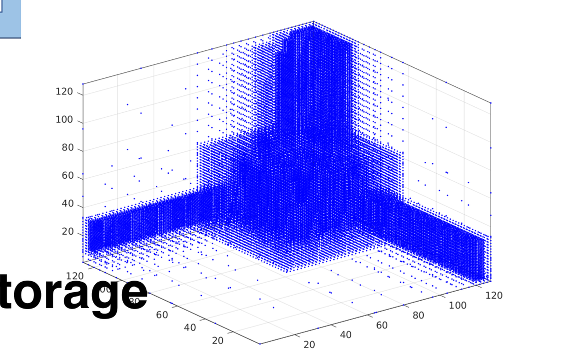
Complex data patterns may need to be expressed, FFTW currently falls short. But, extensions like FFTX could add new descriptors.

Solution: Emerging interfaces like FFTX, extension of FFTW, enables algorithm specification as composition of sub-plans

Front end: Algorithm Specification







Octree Sampling reduces storage on GPU

FFT, tensor contraction and sampling

```
//GPU side, compute on individual domain
#define NUMSUBPLANS 5
plan subplans[NUMSUBPLANS];
plan p; // top-level plan
//... Initialize ....

// create zero-initialized temporary
// n x n x n array with 3 x 3 tensor at each point
tmp1 = create_zero_temp(cube_size, tensor_size);

// copy k x k x k input domain into n x n x n tmp1
subplans[0] = copy_plan(domain, tmp1);// (from, to)

// DFT on the input
tmp2 = create_complex_temp(size_tmp1);
subplans[1] = dft_plan(tmp1);

//Tensor contraction
//In this case we know that output size is the same as tmp2
tmp3 = create_zero_temp(size_tmp2);
subplans[2] = tensor_contraction_plan(tmp2, data, tmp3,
      dimensions_to_contract);//(in,data,out,info)

// iDFT on the contracted output
tmp4 = create_complex_temp(size_tmp3);
subplans[3] = inverse_dft_plan(tmp3, tmp4);

//The next plans apply adaptive sampling
subplans[4] = plan_sample(tmp4, final_output, Octree_S); // (from, to ,
      Octree_descriptor)

// create the top level plan. this copies the sub-plan pointers
p =  plan_compose(NUMSUBPLANS, subplans);

// plan to be used with execute()
return p;
```

Accumulation

```
//CPU side, accumulate over all domains

#define NUMSUBPLANS 3
plan subplans[NUMSUBPLANS];
plan accum; // top-level accumulate plan

// n x n x n array with 3 x 3 tensor at each point
temp = create_zero_temp(cube_size, tensor_size);

//smaller temp arrays
output_cube = create_zero_temp( domain_d_size, tensor_size);
net_output_cube = create_zero_temp( domain_d_size, tensor_size);

for j in [1,...,D] except d:

 subplans[0] = plan_decode_octree(S[j], data_array, temp); //decode octree.
       copy into temp

 subplans[1]= plan_multires_interpolate( S[j], temp, domain_d, output_cube
       ,output_size);
 //descriptor, input cube (samples missing), filter (only interpolate that
       region), outputcube, outputsize

 subplan[2] = plan_sum(output_cube, net_output_cube);

 // create the top level plan
 accum = plan_compose(NUMSUBPLANS, subplans);

 // plan to be used with execute()
 return accum;
```

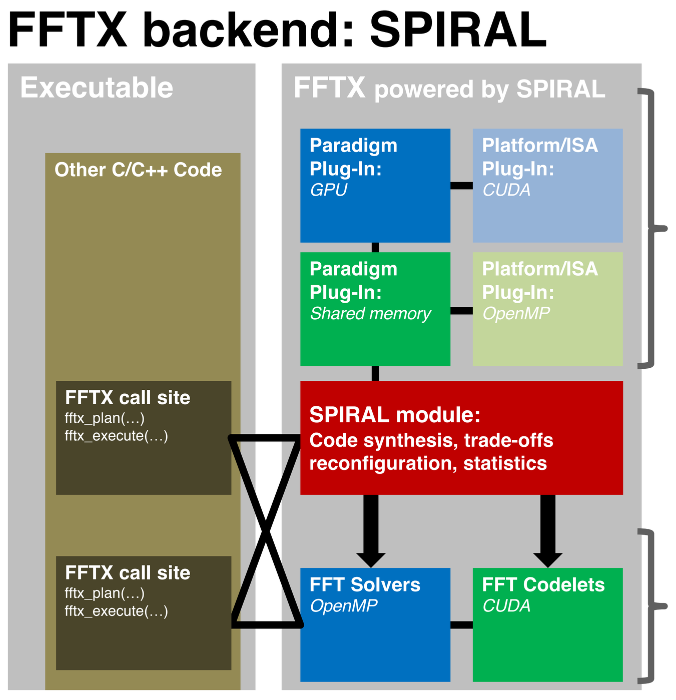
Back end: Code Optimization

FFTX is..

Modernized FFTW-style interface
- **Backwards compatible to FFTW 2.X and 3.X**
- **Small number of new features, familiar interface**

Code generation backend using SPIRAL
- **Library/application kernels are interpreted as specifications in DSL**
  extract semantics from source code and known library semantics
- **Compilation and advanced performance optimization**
  cross-call and cross library optimization, accelerator off-loading,…
- **Reference library implementation and bindings to vendor libraries**

FFTX backend: SPIRAL



Extensible platform and programming model definitions

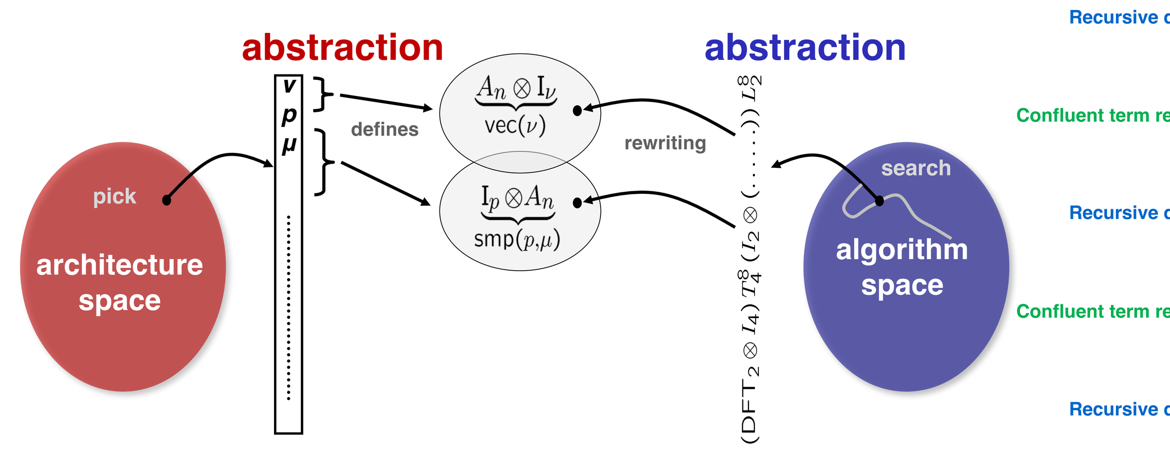
Core system: SPIRAL engine

Automatically generated FFTW-like library components

DARPA BRASS

Platform-aware formal program synthesis

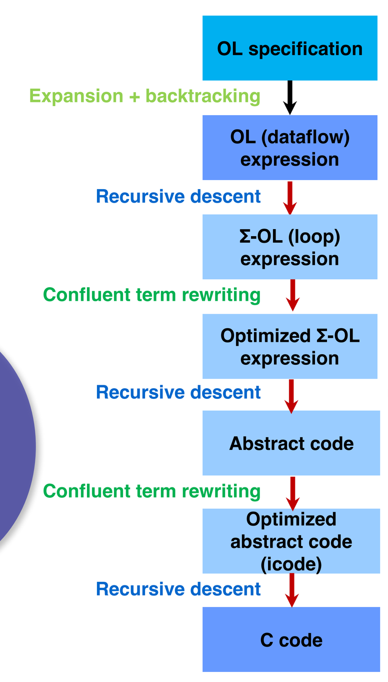
Model: common abstraction = spaces of matching formulas



Architectural parameters: vector length, #processors, …

Kernel: problem size, algorithm choice

Translating an OL expression into code



Constraint Solver Input: $\underbrace{\mathrm{DFT}_8}_{\mathrm{AVX(2\text{-}way}\ \mathbb{C})}$

Output = Ruletree, expanded into

OL expression:

$$\left((F_2 \otimes I_2)T^4_2(I_2 \otimes F_2)L^4_2 \vec{\otimes} I_2\right) \underbrace{T^8_2}_{\mathrm{vec}(2)} \left(I_2 \otimes \underbrace{L^4_2}_{\mathrm{vec}(2)} (F_2 \vec{\otimes} I_2)\right)\left(L^4_2 \vec{\otimes} I_2\right)$$

Σ-OL:

$$\left(\sum_{j=0}^{1}\left(S_{\imath_2 \otimes (j)_2} F_2 \mathrm{Map}^2_{x \mapsto \omega_4^{2i+j} x} G_{\imath_2 \otimes (j)_2}\right) \sum_{j=0}^{1}\left(S_{(j)_2 \otimes \imath_2} F_2 G_{\imath_2 \otimes (j)_2}\right)\right) \vec{\otimes} I_2 \ldots$$

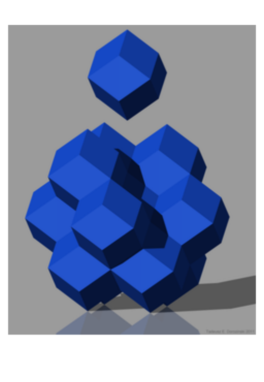
C code:

```
void dft8(_Complex double *Y, _Complex double *X) {
   __m256d s38, s39, s40, s41,...
   __m256d  *a17, *a18;
   a17 = ((__m256d  *) X);
   s38 = *(a17);
   s39 = *((a17 + 2));
   t38 = _mm256_add_pd(s38, s39);
   t39 = _mm256_sub_pd(s38, s39);
   ...
   s52 = _mm256_sub_pd(s45, s50);
   *((a18 + 3)) = s52;
}
```

Future Plans + Other Applications

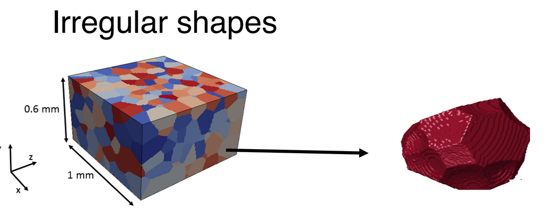
Future work: MASSIF

- Irregular domain decomposition
- Extension of adaptive sampling for irregular domains

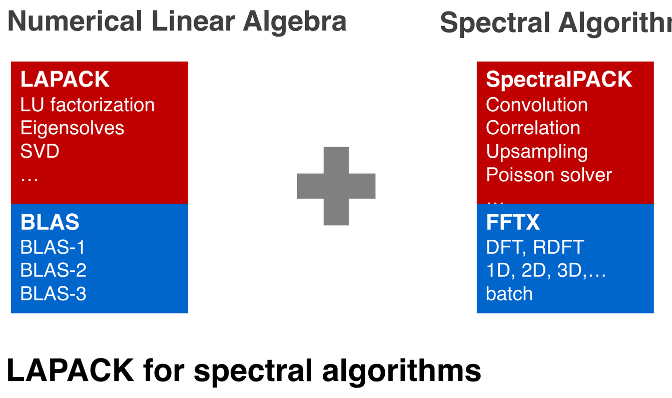


Tesselated polytopes

Irregular shapes



Future work: FFTX and SpectralPACK



LAPACK for spectral algorithms
- **Define FFTX as the analogue to BLAS**
  Define class of numerical algorithms to be supported by SpectralPACK
  PDE solver classes (Green's function, sparse in normal/k space,…), signal processing
- **Define SpectralPACK functions**
  circular convolutions, NUFFT, Poisson solvers, free space convolution

Other FFT-based simulations

Poisson's equation in free space

Partial differential equation (PDE) Solution

$$\Delta(\Phi) = \rho \qquad \Phi : \mathbb{R}^3 \to \mathbb{R}$$
$$\rho : \mathbb{R}^3 \to \mathbb{R}$$
$$D = \mathrm{supp}(\rho) \subset \mathbb{R}^3$$

Poisson's equation. Δ is the Laplace operator

$$\Phi(\vec{x}) = \frac{Q}{4\pi||\vec{x}||} + o\left(\frac{1}{||\vec{x}||}\right) \text{ as } ||\vec{x}|| \to \infty$$
$$Q = \int_D \rho d\vec{x}$$

Approach: Green's function

$$\Phi(\vec{x}) = \int_D G(\vec{x} - \vec{y})\rho(\vec{y})d\vec{y} \equiv (G * \rho)(\vec{x}), \quad G(\vec{x}) = \frac{1}{4\pi||\vec{x}||_2}$$

Solution: ϕ(.) = convolution of RHS *ρ*(.) with Green's function *G*(.). Efficient through FFTs (frequency domain)

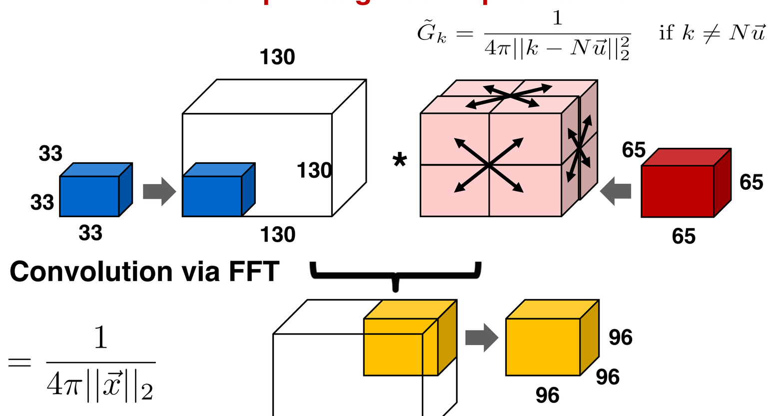
$$\tilde{G}_k = \frac{1}{4\pi||k - N\vec{u}||_2^2} \quad \text{if } k \neq N\vec{u}$$

Green's function kernel in frequency domain

Hockney free-space convolution

Hockney: Convolution + problem-specific zero padding and output subset

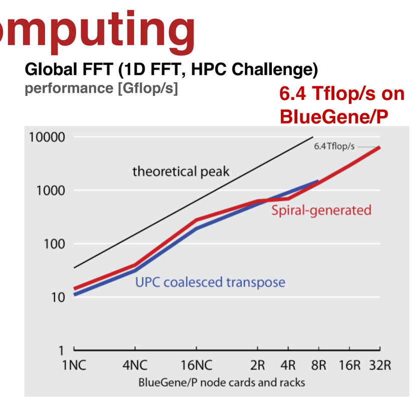
$$\tilde{G}_k = \frac{1}{4\pi||k - N\vec{u}||_2^2} \quad \text{if } k \neq N\vec{u}$$



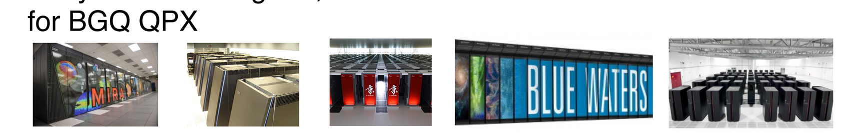
$$\Phi(\vec{x}) = (G * \rho)(\vec{x})$$

SPIRAL: success in HPC/supercomputing

- **NCSA Blue Waters**
  PAID Program, FFTs for Blue Waters
- **RIKEN K computer**
  FFTs for the HPC-ACE ISA
- **LANL RoadRunner**
  FFTs for the Cell processor
- **PSC/XSEDE Bridges**
  Large size FFTs
- **LLNL BlueGene/L and P**
  FFTW for BlueGene/L's Double FPU
- **ANL BlueGene/Q Mira**
  Early Science Program, FFTW for BGQ QPX



BlueGene/P at Argonne National Laboratory
128k cores (quad-core CPUs) at 850 MHz



2006 Gordon Bell Prize (Peak Performance Award) with LLNL and IBM

2010 HPC Challenge Class II Award (Most Productive System) with ANL and IBM

References

[1]V. Tari,R. A. Lebensohn, R. Pokharel,T. J.Turner,P. A.Shade, J. V. Bernier, and A. D. Rollett. 2018. Validation of micro-mechanical FFT-based simulations using High Energy Di raction Microscopy on Ti-7Al. Acta Materialia 154 (8 2018). https://doi.org/10.1016/j.actamat.2018.05.036
[2] M. Frigo and S. G. Johnson. 2005. The Design and Implementation of FFTW3. Proc. IEEE 93, 2 (Feb 2005), 216–231. https://doi.org/10.1109/JPROC.2004.840301
[3] A. Kulkarni, F. Franchetti, and J. Kovačević. 2018. Large-Scale Algorithm Design for Parallel FFT-based Simulations on GPUs. In 2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP). 301–305. https://doi.org/10.1109/ GlobalSIP.2018.8646675
[4]F. Franchetti, et al., 2018. FFTX and SpectralPack: A First Look. In IEEE International Conference on High Performance Computing, Data, and Analytics (HiPC).
[5]P. McCorquodale,P. Colella,G. T.Balls,and S.B.Baden.2006.A Local Corrections Algorithm for Solving Poisson's Equation inThree Dimensions. 2 (10 2006).

Carnegie Mellon Electrical & Computer ENGINEERING

SpiralGen Accelerating Innovation in Computing

ECP EXASCALE COMPUTING PROJECT