Reduction of Runtime Memory in Geant4 Using Compressed Sensing

Jonathan R. Madsen

Department of Nuclear Engineering
Texas A&M University
College Station, TX, USA 77843

jonathanrmadsen@tamu.edu

NUCLEAR ENGINEERING
TEXAS A&M UNIVERSITY

Outline

What is compressed sensing?

Definition

- Compressed sensing is a signal processing technique for efficiently acquiring and reconstructing a signal by finding solutions to under-determined linear systems
- It is based on the principle that, through optimization, the sparsity of the signal can be exploited from fewer samples required by the Shannon-Nyquist sampling theorem

- Compressed sensing is a well-established field with a growing multitude of papers written of the last couple decades
- There is nothing particularly unique about the compressed sensing algorithms we are using — the novelty is the domain of the application and the fact that there is no existing C++ library to do these routines other than ours
- For an analogy, consider compressed sensing the JPEG format of storing scoring tallies

Signal Recovery

- Compressed sensing has two conditions under which recovery is possible from far fewer samples required by the Shannon-Nyquist theorem
  1. Sparsity — the signal must be sparse in some basis
  2. Incoherence
- The incoherence condition is the basis for our methodology
  - Two bases are said to be coherent when they have a large value when integrated against each other [2]
  - Incoherence can be almost guaranteed with any basis where the sampling procedure is random [1]

What is the benefit of using compressed sensing?

Memory savings

- Memory usage is reduced by storing data is a lossy compression format
- Memory savings are drastically reduced as number of threads and number of scoring quantities are increased

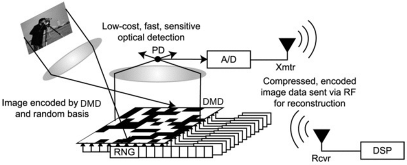
Statistical de-noising

- The reason the compression is "lossy"
- Same concept/techniques used in denoising CT/MRI scans
- Reconstruction from compression format is a peak-preserving, statistical de-noising algorithm that accelerates the computation — requiring less primary particles to be tracked (see note)

- Note: we have evidence of reduced error in reconstructing noisy solutions from comparison with higher-resolved solutions, however, this process currently requires experimentation with certain parameters

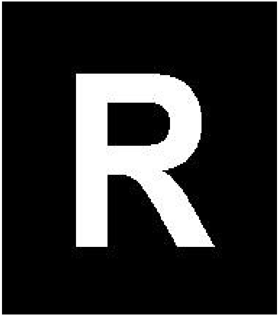
Applications

- Concept — Single Pixel Camera
  - Take one sample of the image and project onto a random linear combination of basis functions resulting in a single scalar value
  - Two values: the single value result and an identifier for the random linear combination are the only data that needs to be transferred
  - As the number of samples increase, the image can be reconstructed with an increasing amount of accuracy



http://dsp.rice.edu/cscamera

Applications — Single Pixel Camera Example



Original Image
(256x256 pixels)

After 1300 measurements
(∼2% of pixels sampled)

After 3300 measurements
(∼5% of pixels sampled)

http://dsp.rice.edu/cscamera

Disjoint Tallies

- As an example, in the case of a single pixel camera gray-scale measurement projected onto a random basis set represented as a 3x3 grid, a sample and projection would look like the following ($.*$ denotes element-wise multiplication):

5	2	5
9	6	7
1	2	3

Single Pixel Camera Sample

$.*$

1	1	0
1	0	1
0	0	1

Random Basis Set

$\Rightarrow$

5	2	0
9	0	7
0	0	3

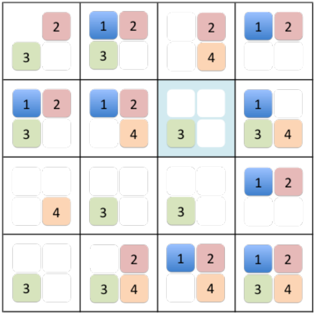
Projection Result $\sum = 26$

The stored data would be (1) the $\sum = 26$ and (2) the random number seed that generated the random basis set

Disjoint Tallies

- The same concept of random linear combinations can be applied to scoring quantities in Monte Carlo transport
- For a scoring mesh of dimensions MxN:
  - We define a parameter r_s — the subrate — which represents the fraction of the data we want to store
  - We define n_ℓ random basis sets where $n_\ell = M * N * r_s$ (n_ℓ "disjoint tallies")
  - Each basis set has a size of $M * N$
    - If these were double precision values, we would use n_ℓ times as much memory
    - Luckily, the basis set values do not need to be a floating-point random from [0, 1), but instead can be cast to a boolean: 0 or 1
- In the disjoint tally system, individual voxels can have multiple disjoint tallies they score into and different voxels scoring into the same disjoint tally have a reference to the same value

Disjoint Tallies (cont.)



Matrix **A** for 4 disjoint tallies @ subrate of 0.25 (M*N*r --> 4 x 4 x 0.25 --> 4 disjoint tallies)

@ [2,3] : A(3) = 1; A(1) = A(2) = A(4) = 0

Memory: stored in bitset array for each disjoint tally (4 total) with 16 values each = 64 bits = 8 bytes (size of one double precision)

.*

Element-wise multiplication

1	1	1	1	= Σ A(1) .* x	8
1	1	0	1	= Σ A(2) .* x	10
1	1	1	1	= Σ A(3) .* x	9
1	1	1	1	= Σ A(4) .* x	7

Matrix **x** - Example set of values scored into 4x4 voxel array during runtime

Scoring results stored in memory

Temporary variables -- not stored in memory

4 doubles (32B) vs. 16 doubles (128B)

(32+8)/128 = 0.3125 **compression to 31.25% original memory size**

Disjoint Tallies (cont.)

- In theory, memory savings in the scoring tally alone (right column on previous slide) are defined by the subrate

$$Savings = (1 - r_s) * (M * N) * (size\ of\ double) \quad (1)$$

- Savings are possibly reduced based on the method of storing A
  - Pre-allocation of A: $\uparrow$ memory, computation time $\downarrow$ (preferred)
  - Storing random seed: $\downarrow$ memory, computation time $\Uparrow$
- Due to the nature of A, as more threads are added, the memory savings continue to grow
- Beyond the scoring of 1 quantity (*i.e.* scoring 2+ quantities), the memory savings follow Eqn. 1

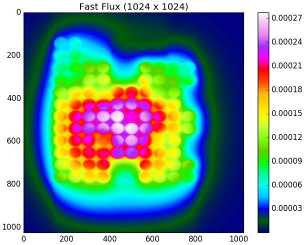
Algorithm

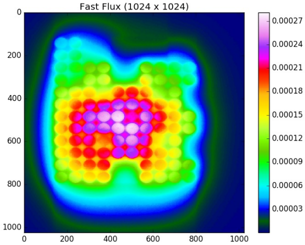
- We utilize a technique known as total variation minimization (TVM) and impose quadratic constraints with a log-barrier algorithm (interior point method)
- We find **a** solution to $Ax = b$ where A is the disjoint tally matrix, x is the reconstructed solution, and b is the solution stored in the compressed format using the concept of *disjoint tallies*
- The reconstruction is divided into sub-meshes, which allow for localized reconstruction and keep compute time at a negligible increase

Summary

- Given an estimate of x on the interior of $Ax = b$, we start a series of log-barrier iterations that seek out a solution with the minimal variation of the gradient in x
- Each iteration of the log-barrier algorithm increases the proximity to the boundary of the feasible region of x by a series of Newton steps

Proof of Concept

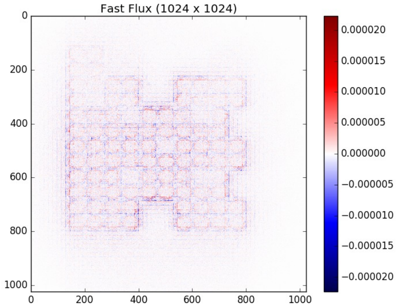




- MCNP6 Calculation of fast flux in TRIGA reactor at Texas A&M
- 1024 x 1024 mesh; 200,000 particles/cycle; 2,500 cycles

- Reconstruction of fast flux of TRIGA reactor at Texas A&M
- $r_s = 0.20$
- 16x16 blocks

Proof of Concept (cont.)



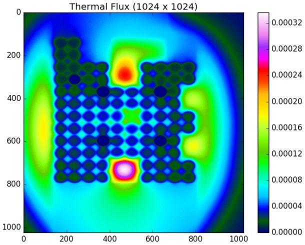
- Difference between MCNP6 calculation and reconstruction

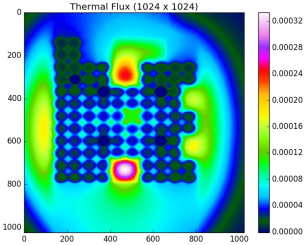
$$\sum_{i=1}^{M}\sum_{j=1}^{N}\|x_{ij,recons} - x_{ij,mcnp}\|$$

$$= 0.6263$$

- $\bar{\epsilon} = 5.97286x10^{-7}$

Proof of Concept

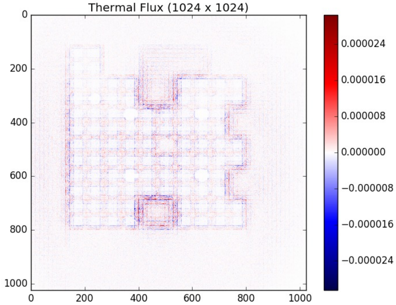




- MCNP6 Calculation of thermal flux in TRIGA reactor at Texas A&M
- 1024 x 1024 mesh; 200,000 particles/cycle; 2,500 cycles

- Reconstruction of thermal flux of TRIGA reactor at Texas A&M
- $r_s = 0.20$
- 16x16 blocks

Proof of Concept (cont.)



- Difference between MCNP6 calculation and reconstruction

$$\sum_{i=1}^{M}\sum_{j=1}^{N} ||x_{ij,recons} - x_{ij,mcnp}||$$

$$= 0.77698$$

- $\bar{\epsilon} = 7.4099x10^{-7}$

Implementation — Reconstruction Library

- Implemented reconstruction algorithm(s) as a separate C++11 library from Geant4
  - Handles both proof-of-concept case (full solution with post-compression and reconstruction) and reconstruction from runtime compression of the solution
    - Allowing for exploitation of statistical de-noising if runtime compression is not used/desired
  - Handles serialization for long-term storage
  - Produces bitmap images of solutions
- Matrix calculations use Armadillo linear algebra library [3]
  - Very easily allows for offloading matrix calculations to GPU by simply linking in GPU-optimized library (*e.g.* nvblas)
- Python interface available (created via SWIG)
  - Very easy to implement, should possibly be considered for G4Py?

Geant4 Implementation — Geometry

- Results for reactor bundle-type geometry
  - 24 bundles (5x5 array minus 1 bundle)
  - 145 pins/bundle arranged in hexagon
  - Each fuel pin is UO_2 with 15% enrichment and Zr cladding
  - One bundle of control rods with 80% Ag, 15% In, and 5% Cd composition
- 512 x 512 voxel scoring mesh in Geant4 parallel world encompassing entire "world" geometry
- Moderating material is water
- Random e^-, e^+, p, α, n, γ
- Random fuel pin, random location within pin, isotropically emitted

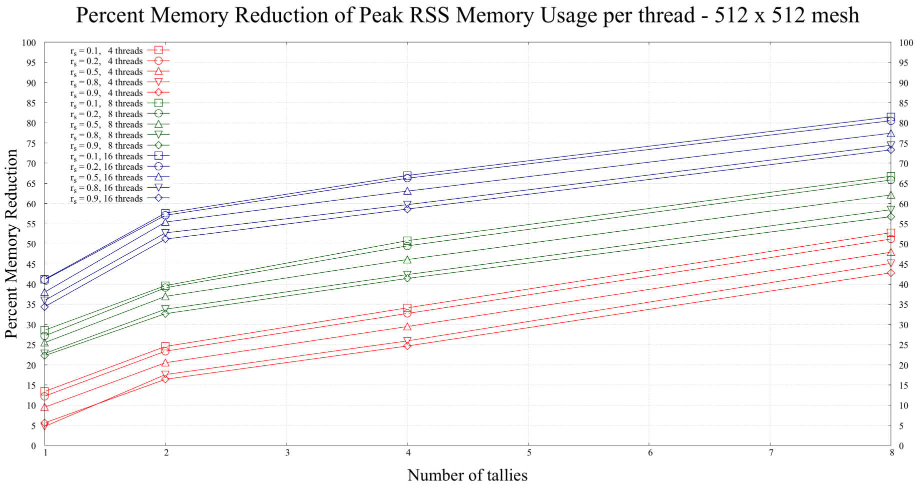
Geant4 Implementation — Physics

- Physics Lists
  - EM Standard Physics (option 4)
  - Decay Physics
  - Radioactive Decay Physics
  - Hadron Elastic Physics
  - Hadron Physics QGSP BERT HP
    - QGSP - Quark Gluon String (fragmentation) + Precompound (de-excitation)
    - BERT - Bertini Cascade for inelastic scattering
    - HP - High Precision
  - Ion Elastic Physics
  - Ion Binary Cascade Physics
  - Step Limiter Physics

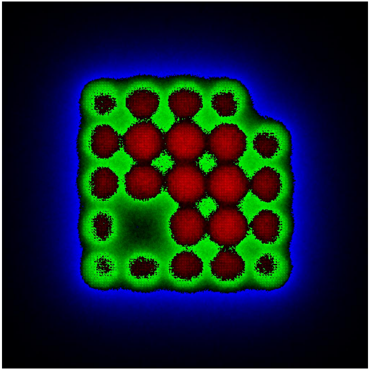
Geant4 Setup — Reconstruction

- Unit-testing verification that reconstruction using proof-of-concept technique (scoring on full mesh and reconstruction from post-compression of full results) and Geant4 runtime compression of scoring and reconstruction yield the same reconstruction result
- Split 512x512 mesh into 32x32 sub-units (i.e. 16x16 divisions)
- Geant4 application uses several preprocessor definitions to permit compilation of individual scoring processes or combinations of scoring processes among: Geant4 standard scoring, Geant4 standard scoring with post-compression, and runtime compressed scoring
- Memory is measured by reading /proc/self/statm
- Runtime compressed scoring is stored at conclusion of run to reconstruct at a later time or on a different machine
- Reconstruction is exceptionally fast when a GPU is available to off-load matrix calculations

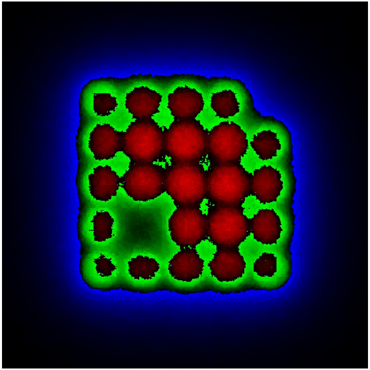
Memory Reduction Results per thread



Cell Flux Results - $r_s = 0.2$

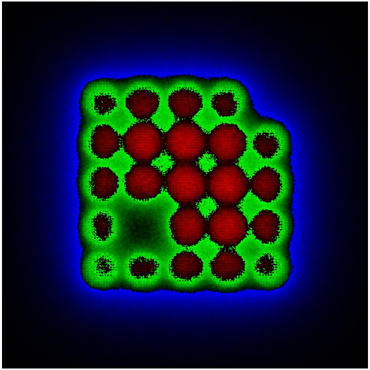


- Tally of Cell Flux using standard Geant4

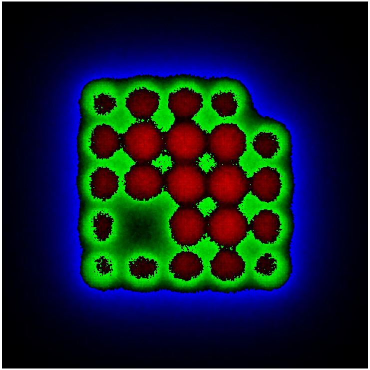


- Tally of Cell Flux using compressed sensing ($r_s = 0.2$)

Cell Flux Results - $r_s = 0.5$

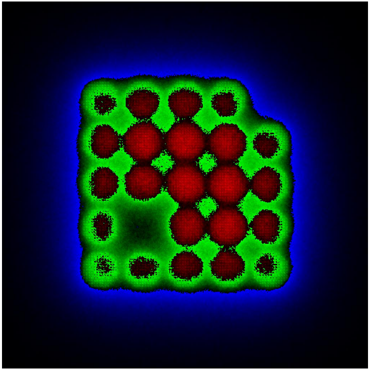


- Tally of Cell Flux using standard Geant4

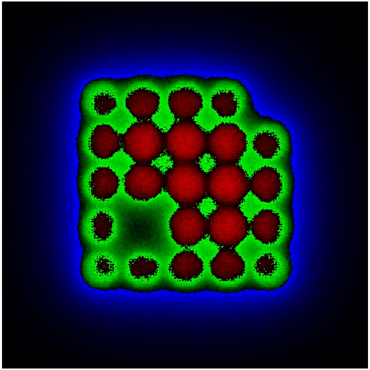


- Tally of Cell Flux using compressed sensing ($r_s = 0.5$)

Cell Flux Results - $r_s = 0.9$

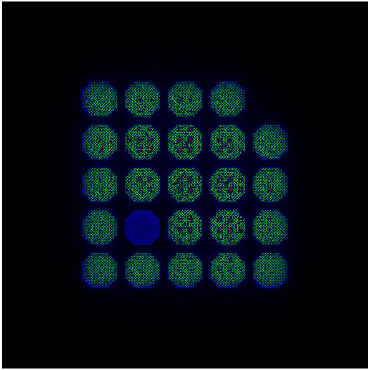


- Tally of Cell Flux using standard Geant4

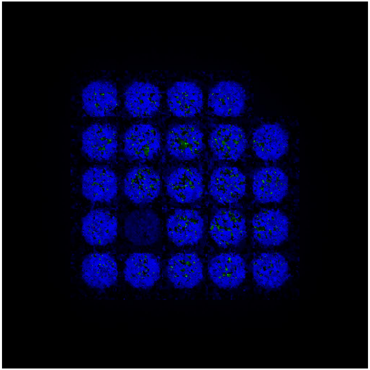


- Tally of Cell Flux using compressed sensing ($r_s = 0.9$)

Number of Collision Results - $r_s = 0.2$

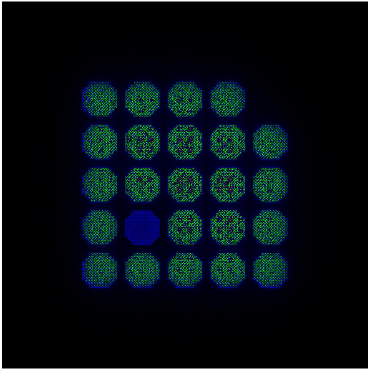


- Tally of Number Of Collisions using standard Geant4

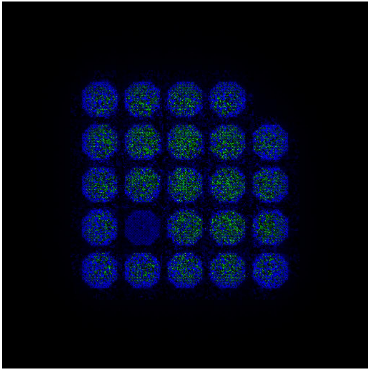


- Tally of Number Of Collisions using compressed sensing ($r_s = 0.2$)

Number of Collision Results - $r_s = 0.5$

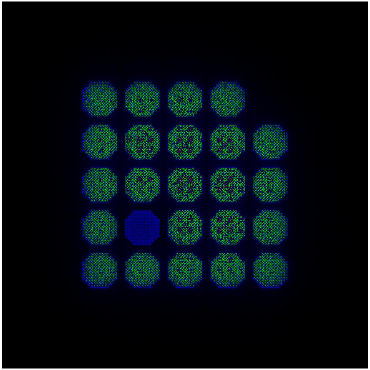


- Tally of Number Of Collisions using standard Geant4

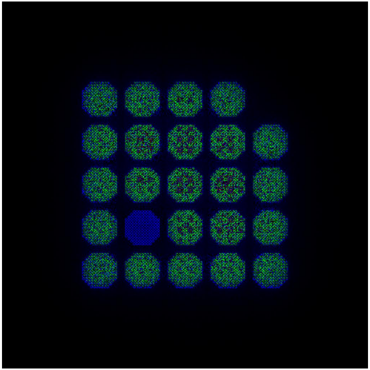


- Tally of Number Of Collisions using compressed sensing ($r_s = 0.5$)

Number of Collision Results - $r_s = 0.9$

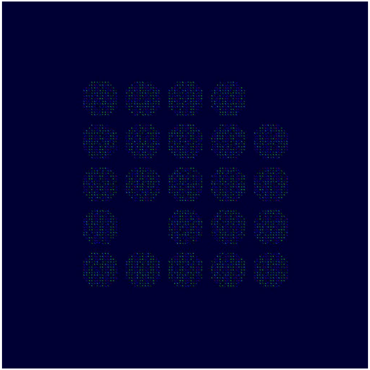


- Tally of Number Of Collisions using standard Geant4

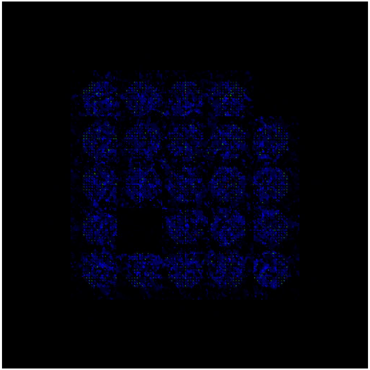


- Tally of Number Of Collisions using compressed sensing ($r_s = 0.9$)

Cell Charge Results - $r_s = 0.2$

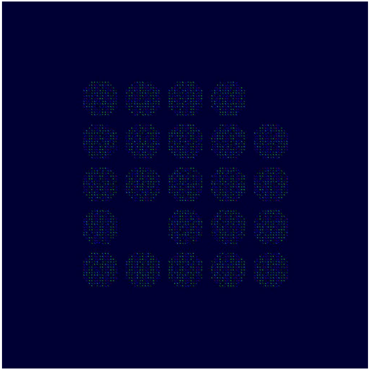


- Tally of Cell Charge using standard Geant4

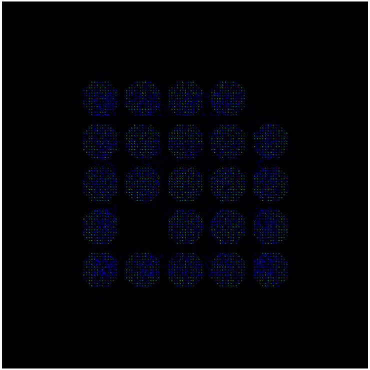


- Tally of Cell Charge using compressed sensing ($r_s = 0.2$)

Cell Charge Results - $r_s = 0.5$

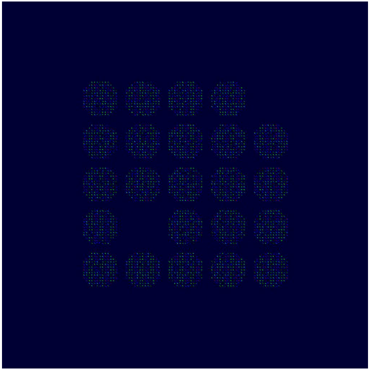


- Tally of Cell Charge using standard Geant4

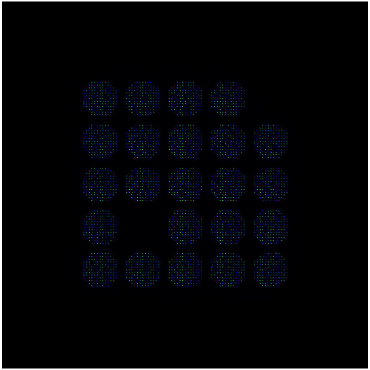


- Tally of Cell Charge using compressed sensing ($r_s = 0.5$)

Cell Charge Results - $r_s = 0.9$



- Tally of Cell Charge using standard Geant4



- Tally of Cell Charge using compressed sensing ($r_s = 0.9$)

Conclusions

- **Memory reduction can be achieved with high subrate settings that minimize the statistical de-noising, producing nearly identical reconstructions**
- The proof of concept shows that reconstruction works extremely well for smooth, highly-resolved quantities of interest with a low subrate setting
- The reduction in runtime memory is shown in a real runtime environment, not theoretical memory reduction!
- Multi-dimensional memory reduction (# of threads, # of scoring quantities)
- Combination with G4atomic (examples/extended/parallel/ThreadsafeScorers) will produce even larger reductions in runtime memory
- Reconstruction can be applied as a post-processing technique for statistical de-noising

NUCLEAR ENGINEERING
TEXAS A&M UNIVERSITY

Review

- Significant reduction in runtime memory with or without significant statistical de-noising
- Negligible compute time increase
- Statistical de-noising can be applied automatically or as a post-processing technique
- Potentially reduced compute time via fewer primary particles once statistical de-noising if parameter settings are able to be adaptive and automated
- Other optimizations are available (*e.g.* ℓ_1 minimization)

Future Work

- Three-dimensional reconstruction
- Apply to shielding problem
- Apply to DICOM problem
- Implement other optimization methods (*e.g.* ℓ_1 minimization)
- Investigate re-application of reconstruction on sub-mesh boundaries to resolve sub-mesh boundary error peaks
- Attempt to reconstruct the variance by the compressed storage format

NUCLEAR ENGINEERING TEXAS A&M UNIVERSITY

Thank you for your attention

Questions?

- E. Candes and T. Tao. The dantzig selector: statistical estimation when p is much larger than n. 2006.
- D. L. Donoho and X. Huo. Uncertainty principles and ideal atomic decomposition. 2001.
- C. Sanderson and R. Curtin. Armadillo: C++ linear algebra library. http://arma.sourceforge.net, 2016. Accessed: 2016-07-27.