Northeastern UNIVERSITY

Sparse Matrix Algorithms on GPUs and their Integration into SCIRun

Devon Yablonski
yablonski.d@husky.neu.edu

&

Miriam Leeser
mel@coe.neu.edu

Dana Brooks
brooks@ece.neu.edu

This work is supported by:

- NSF CenSSIS - The Center for Subsurface Sensing and Imaging
- BIAT – A Biomedical Imaging Acceleration Testbed: NSF Award Number 0946463
- CIBC, the Center for Integrative Biomedical Computing, which develops and maintains SCIRun

Outline

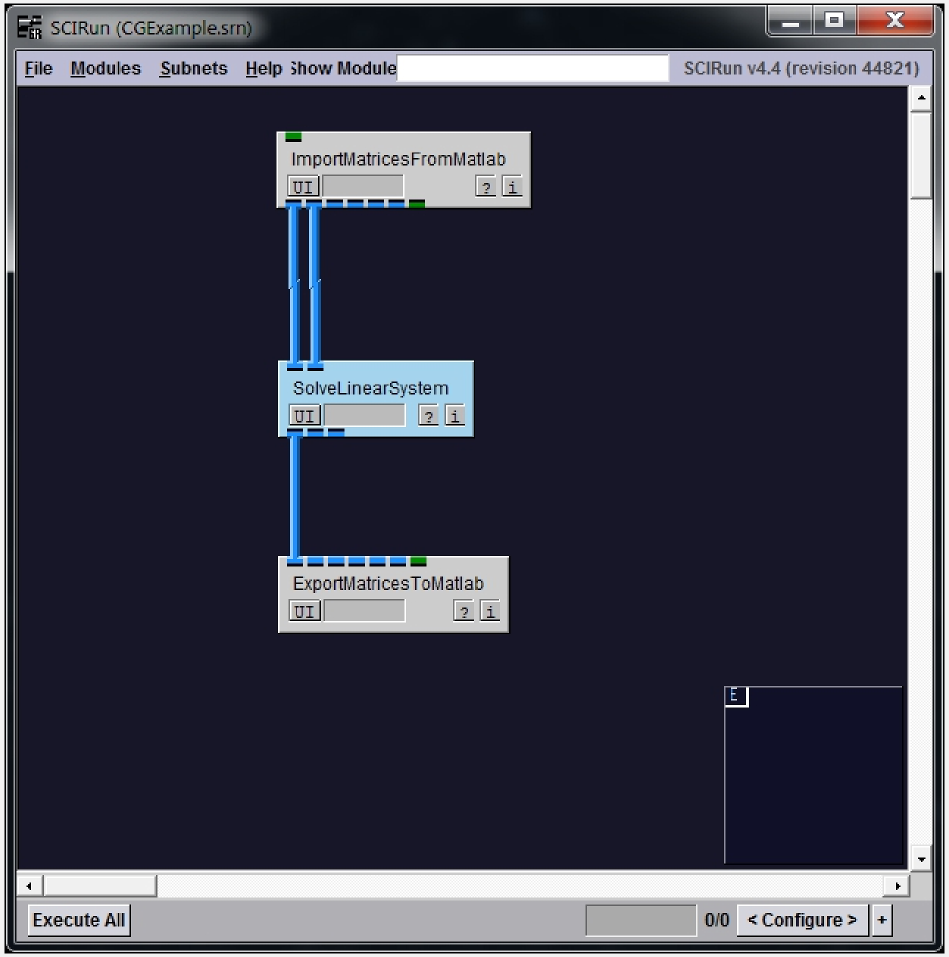
1. Introduction
   1. Goals
   2. SCIRun Biomedical Problem Solving Environment
   3. GPU Architecture
2. Theory and Calculations
   1. Linear Solvers
3. Design
   1. GPU implementation
   2. User Interface Modifications
4. Results
5. Discussion and Conclusions

Goals

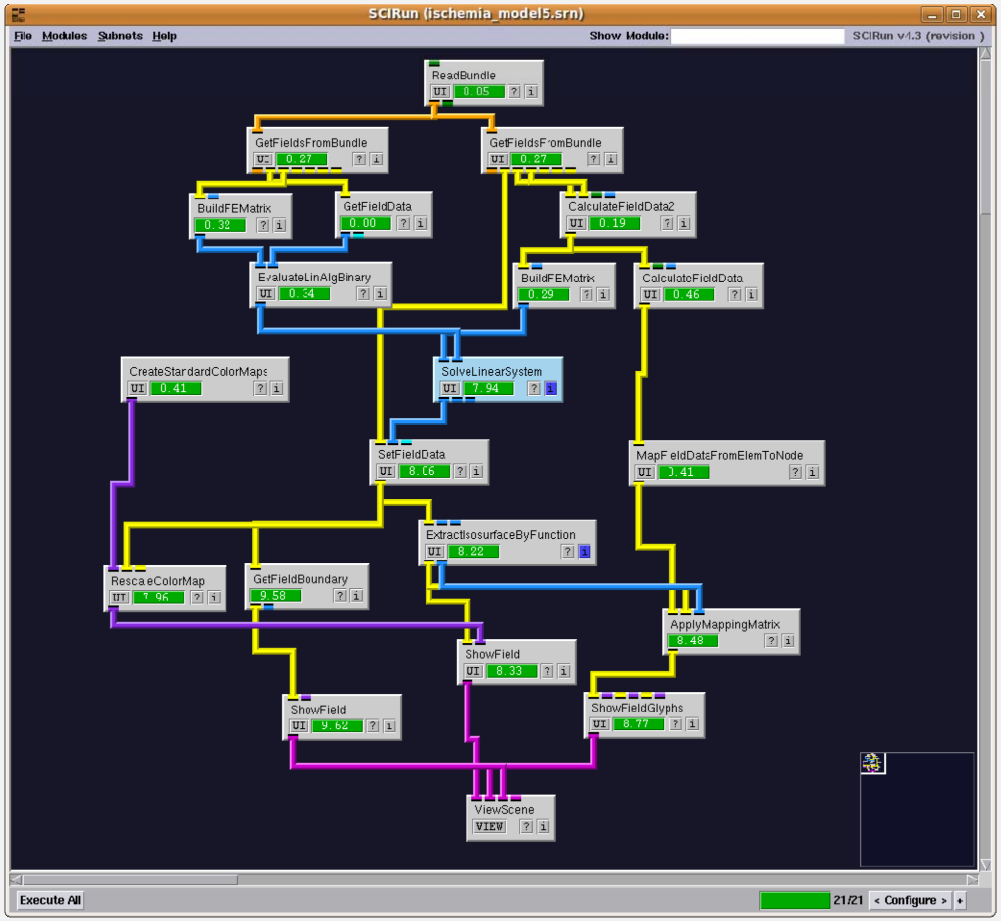
- Accelerate SCIRun Problem Solving
    - To create an implementation of double precision sparse linear solvers in a problem solving environment for the GPU including:
        - Conjugate Gradient Method (CG)
        - Minimal Residual Method (MinRes)
        - Jacobi Method
    - To provide a mechanism to accelerate many SCIRun algorithms while remaining transparent to the scientist
        - Retaining in-progress algorithm visualizations
        - Allowing for future GPU algorithm development within the environment

University of Utah's SCIRun

- SCIRun is a biomedical problem solving environment (BioPSE)
  - Center for Integrative Biomedical Computing (CIBC)
  - Designed to be **extensible and scalable.**
  - Supports interaction among the modeling, computation and visualization phases of biomedical imaging
  - Uses include:
    - Cardiac electro-mechanical simulation
    - ECG & EEG forward and inverse calculations
    - Deep brain stimulation modeling
  - Available for Windows, Mac/OSX and Linux



University of Utah's SCIRun

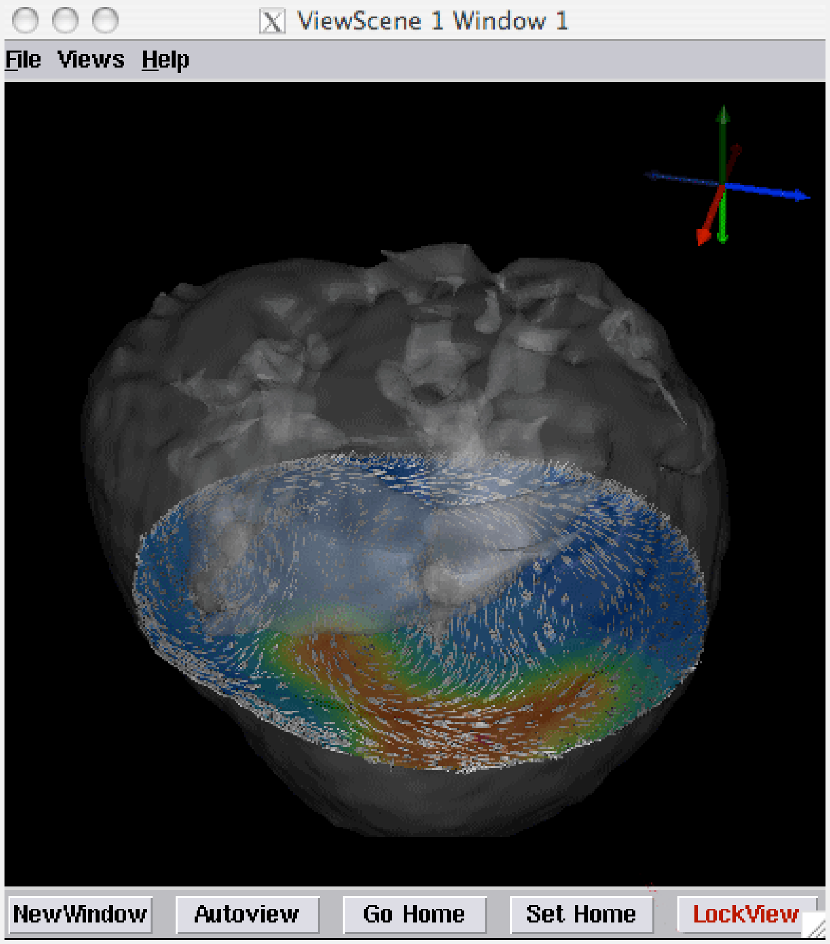


- Allows scientists to create a network of mathematical functions
- The network visualizes a simulation from start to finish
- Many of these algorithms are time consuming

… and display parallelism!

Heart Ischemia Model

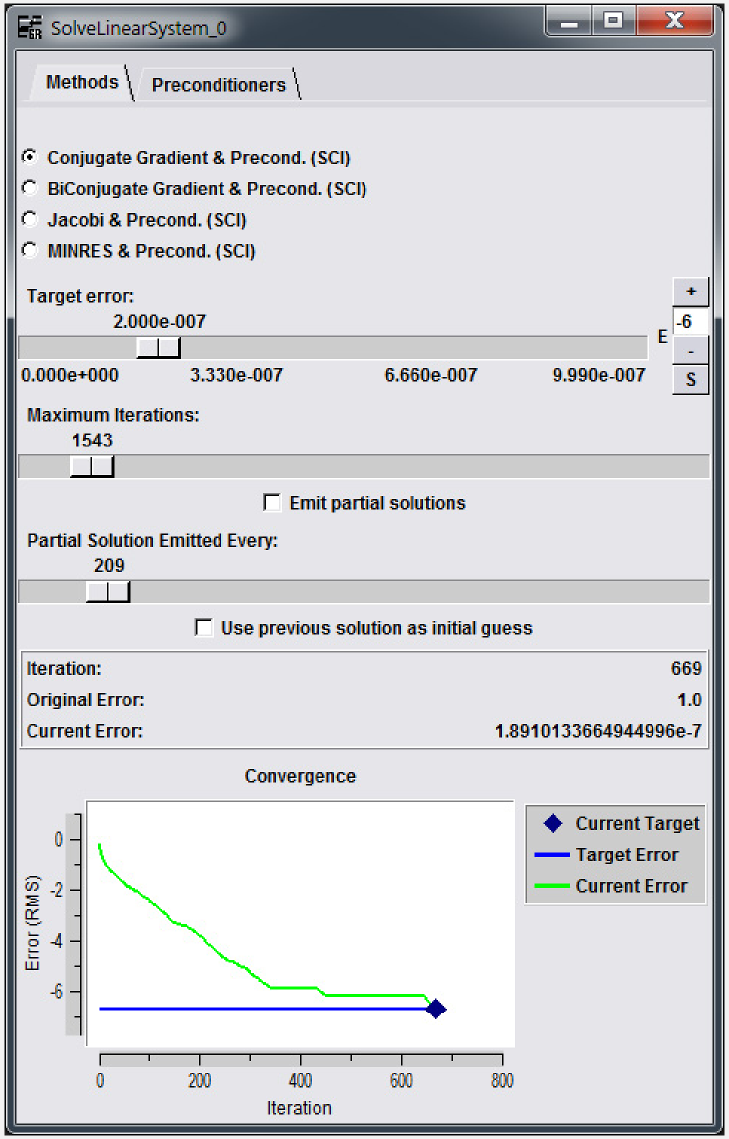
- Ischemia: Tissue damaged by a lack of blood flow
- The model is a 3D interactive model based on a scan of an ischemic dog heart
  - For measuring and predicting extracellular cardiac potentials
- The network on the previous slide generates this image
- The sparse data in this model is 107,000 x 107,000 with 1.2 million nonzeros
  - Compressed Row Storage Format



Northeastern

SolveLinearSystem Module

- Solves sparse linear systems with a variety of algorithms
- Allows the user to modify parameters such as preconditioners, target error, and iteration limit
- Displays current error, iteration count and convergence graph
    - This helps the scientist visualize results


SolveLinearSystem_0

Methods | Preconditioners

[✓ Conjugate Gradient & Precond. (SCI)]
BiConjugate Gradient & Precond. (SCI)
Jacobi & Precond. (SCI)
MINRES & Precond. (SCI)

Target error: 2.000e-007

0.000e+000 3.330e-007 6.660e-007 9.990e-007 E -6

Maximum Iterations: 1543

Emit partial solutions

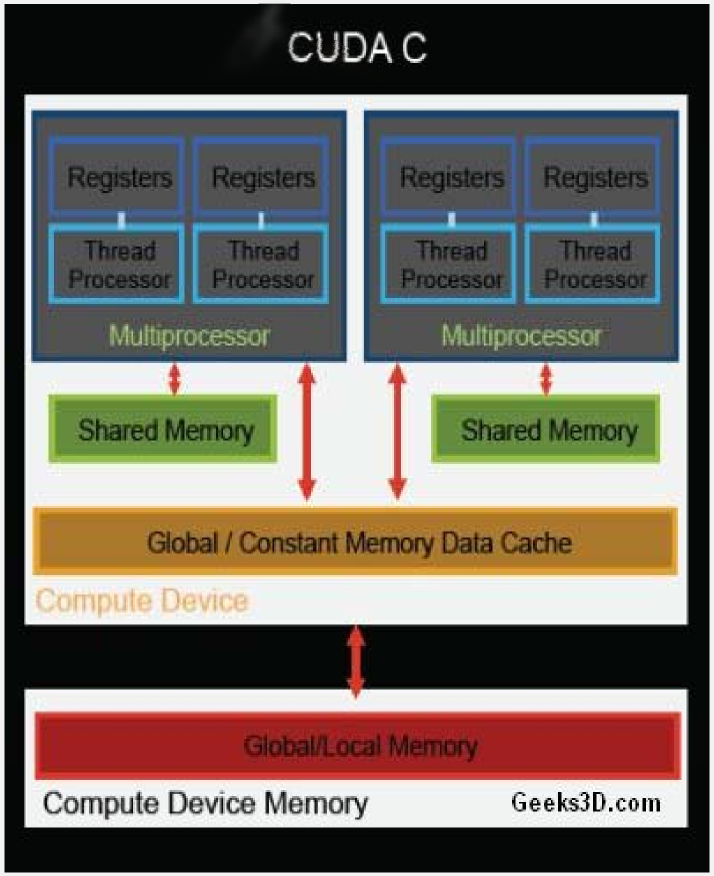
Partial Solution Emitted Every: 209

Use previous solution as initial guess

Iteration:	669
Original Error:	1.0
Current Error:	1.8910133664944996e-7

GPU Architecture – NVIDIA GeForce GTX 280

- Graphics processing units are **S**ingle **I**nstruction **M**ultiple **D**ata (SIMD)
- 240 cores (32 multiprocessors with 8 processors in each)
- Multi-tiered memory layout
  - 1GB **global** memory
  - 16kB per-core **shared** memory
  - 64kB total read-only **constant** memory
  - 16384 registers per multiprocessor
- 32 warp threads perform the same instruction on a set of data
- Programmable using NVIDIA CUDA C or OpenCL



Images from NVIDIA and Geeks3D.com

Conjugate Gradient Method

- The most commonly used iterative solver of linear systems, Ax=b
- Matrix *A* must be square, symmetric and positive definite
- Benefits include:
  - Ease of use
  - Minimal storage requirement
  - Good convergence rate if there is sufficient numerical precision

Iterate:

$$\alpha_{(i)} = \frac{r_{(i)}^T r_{(i)}}{d_{(i)}^T A d_{(i)}}$$

$$x_{(i+1)} = x_{(i)} + \alpha_{(i)} d_{(i)},$$

$$r_{(i+1)} = r_{(i)} - \alpha_{(i)} A d_{(i)},$$

$$\beta_{(i+1)} = \frac{r_{(i+1)}^T r_{(i+1)}}{r_{(i)}^T r_{(i)}},$$

$$d_{(i+1)} = r_{(i+1)} + \beta_{(i+1)} d_{(i)}$$

Iterate:

$$\alpha_{(i)} = \frac{\boxed{r_{(i)}^T r_{(i)}}}{\boxed{d_{(i)}^T A d_{(i)}}}$$

$$x_{(i+1)} = \boxed{x_{(i)} + \alpha_{(i)} d_{(i)},}$$

$$r_{(i+1)} = \boxed{r_{(i)} - \alpha_{(i)} A d_{(i)},}$$

$$\beta_{(i+1)} = \frac{\boxed{r_{(i+1)}^T r_{(i+1)}}}{\boxed{r_{(i)}^T r_{(i)}}},$$

$$d_{(i+1)} = \boxed{r_{(i+1)} + \beta_{(i+1)} d_{(i)}.}$$

Algorithm descriptions from Wolfram MathWorld

Other Methods

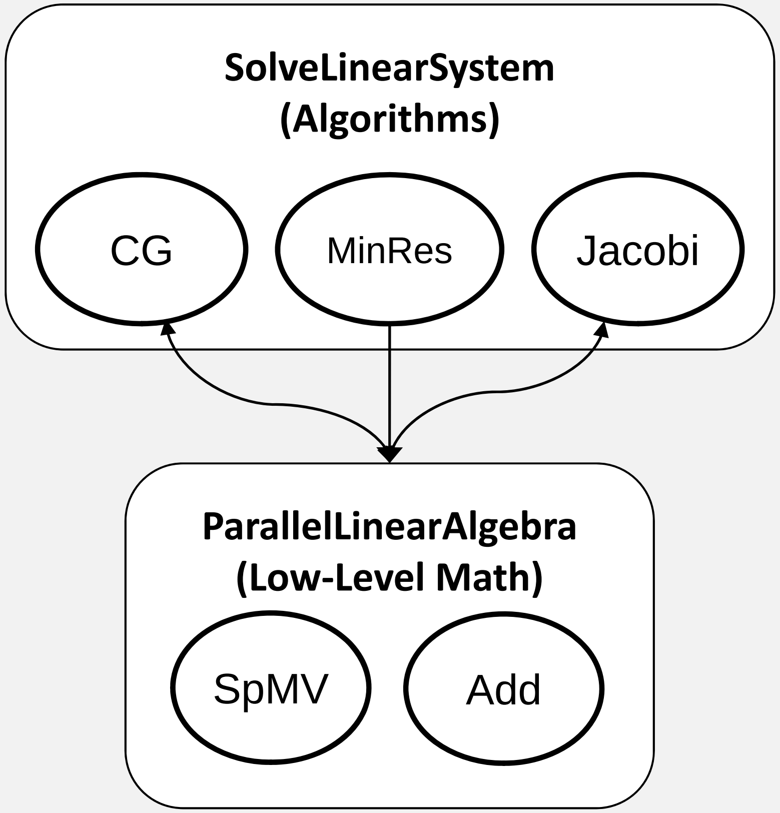
- **Jacobi**
  - Simplest algorithm for linear solvers
  - Matrix *A* must be *diagonal* – the absolute value of each diagonal element must be:
    - Non-zero
    - Greater than the absolute value of each element in that row.

$$\begin{tikzcd}[column sep=small] & \underline{A}x = b \arrow[dl] \arrow[dr] & \\ (Diag \arrow[r, phantom, "+"] & & Rem)x = b \arrow[d] \\ & & Diag * x_i = b - Rem * x_{i-1} \end{tikzcd}$$

Solve for x_i
Use previous iteration x as x_{i-1}

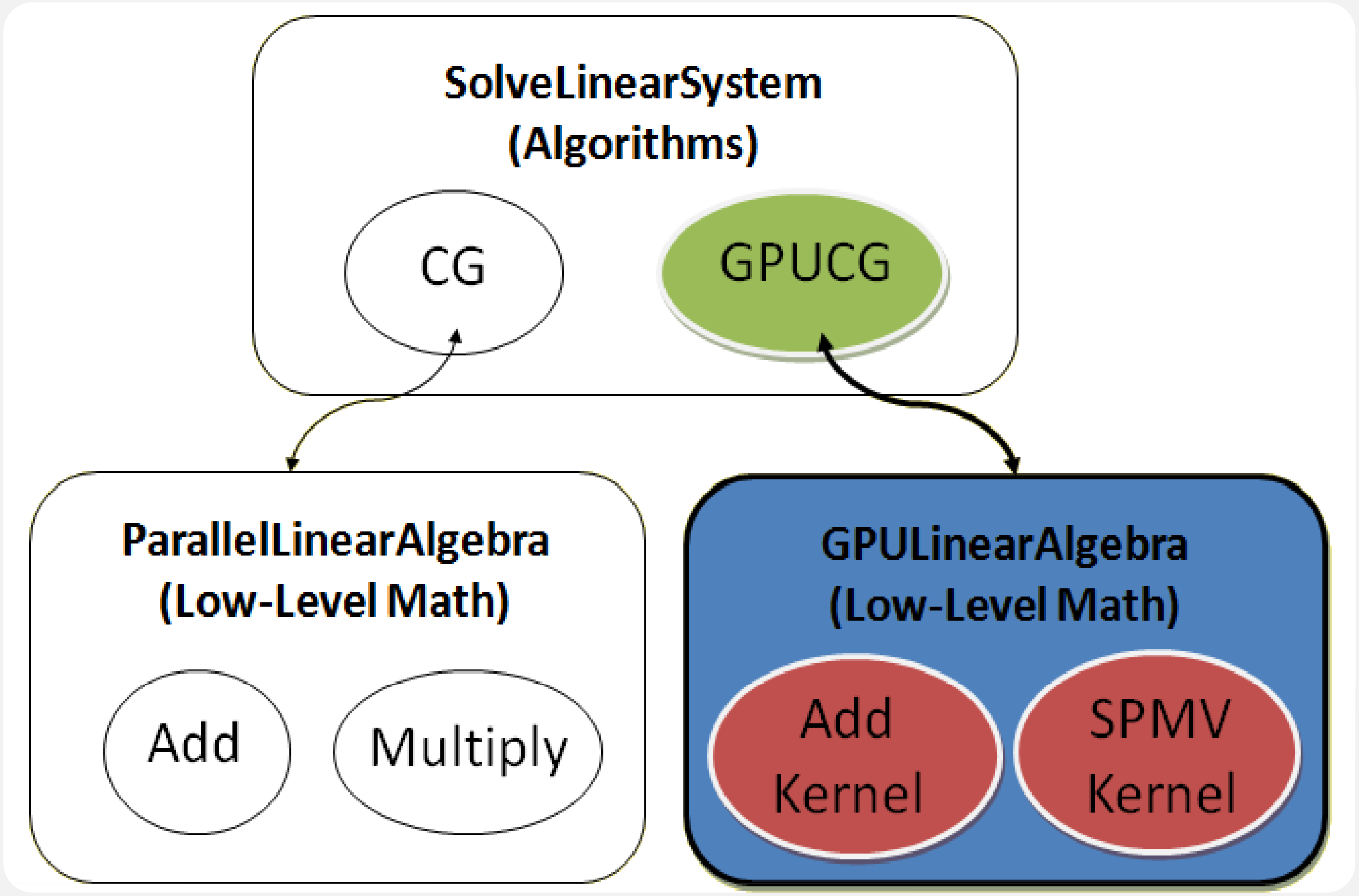
- **Minimal Residual**
  - More complicated than CG
  - Can also solve symmetric indefinite systems
  - Stronger convergence behavior with infinite precision
    - Guaranteed to have non-decreasing residual errors each iteration

Original Design - ParallelLinearAlgebra (CPU)



- All algorithms exist at SCIRun's module level - SolveLinearSystem
- All low-level parallel computations exist at SCIRun's algebra level – ParallelLinearAlgebra
  - CPU matrix and vector computations with optimizations
- Algorithms call these low level math functions as an abstraction
  - This structure lends itself to a convenient GPULinearAlgebra sibling

Modified Design - GPULinearAlgebra



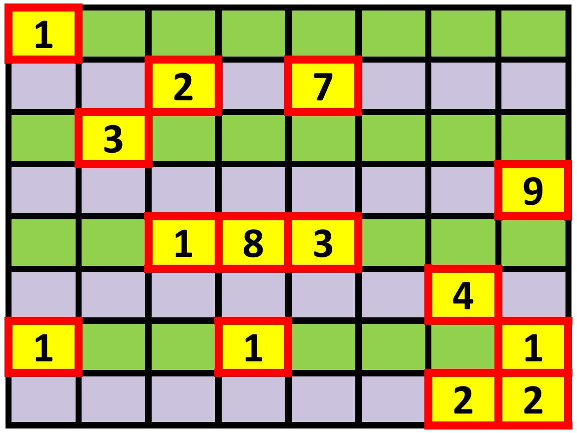
Computation Details

- Operations Accelerated:
    - GPULinearAlgebra now contains accelerated versions of all functions in the ParallelLinearAlgebra CPU library provided with SCIRun
    - Sparse Matrix-Vector multiplication (SpMV)
    - Vector addition, simultaneously adding and scaling, subtraction, copy, dot product, normalization, maximum, and minimum, threshold invert and more…
    - Operations implemented using NVIDIA CUBLAS libraries and direct coding in CUDA
        - CUBLAS does not handle sparse data

Computation Details

- Numerical Precision
  - Double precision floating point is necessary for accurate convergence of these algorithms
  - The GPU version is performed in double precision in order to achieve convergence in all examples, as in SCIRun's double precision CPU implementation
- Data Storage
  - The problems are large in size, with sparsely populated matrices
  - Sparse data formats are required, adding complexity and decreasing parallelism

Compressed Sparse Row Storage



Non-zero Values	1	2	7	3	9	1	8	3	4	1	1	1	2	2	
Column Index	0	2	4	1	7	2	3	4	6	0	3	7	6	7	
Row Pointer	0	1		3	4	5			8	9			12		14

14 Non-zeros requires 59 memory fetches in one SpMV
Filling ratio in memory = 100%

- Rows may have few nonzero entries
  - Lots of wasted memory and calculations if stored in a dense format
- Instead, store only relevant points A[i] and two location vectors
  - Location vectors
    - Column index C[i] gives column number of element A[i]
    - Row pointer R[i]=j gives location A[j] of a row change

L.Buatois, G.Caumon & B.Lévy, Concurrent Number Cruncher: An Efficient Sparse Linear Solver on the GPU, High Performance Computing and Communications, Third International Conference, HPCC, 2007.

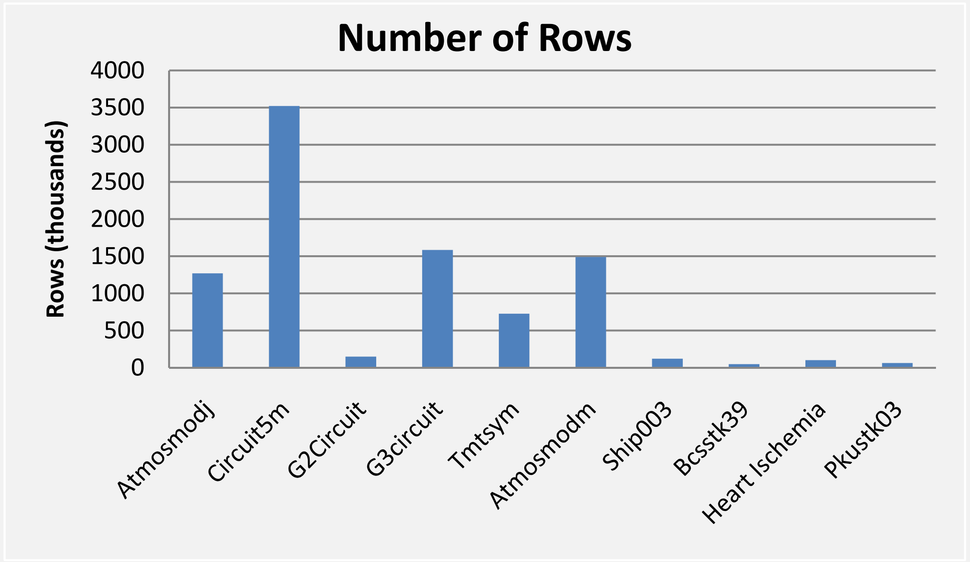
Experiment Details

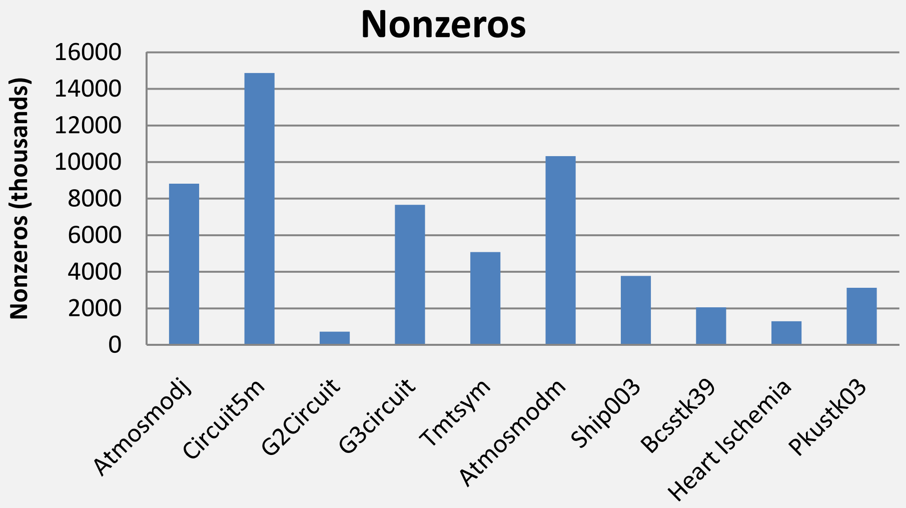
- Test Machine
  - CPU – Intel Core 2 E6300 1.86GHz, 2Mb L2 cache, 1066MHz FSB
  - GPU - NVIDIA GeForce 280 GTX 1GB RAM PCIe card
- Test conditions
  - Tests were run >10 times each to assure accurate results
  - Test time is end to end – includes all data transfer and setup overheads involved in GPU version
- Test data
  - Heart Ischemia Model
  - University of Florida’s Sparse Matrix Collection

Northeastern

Input Data

- The sparse matrices vary in size from 6.3K to 3.5M rows

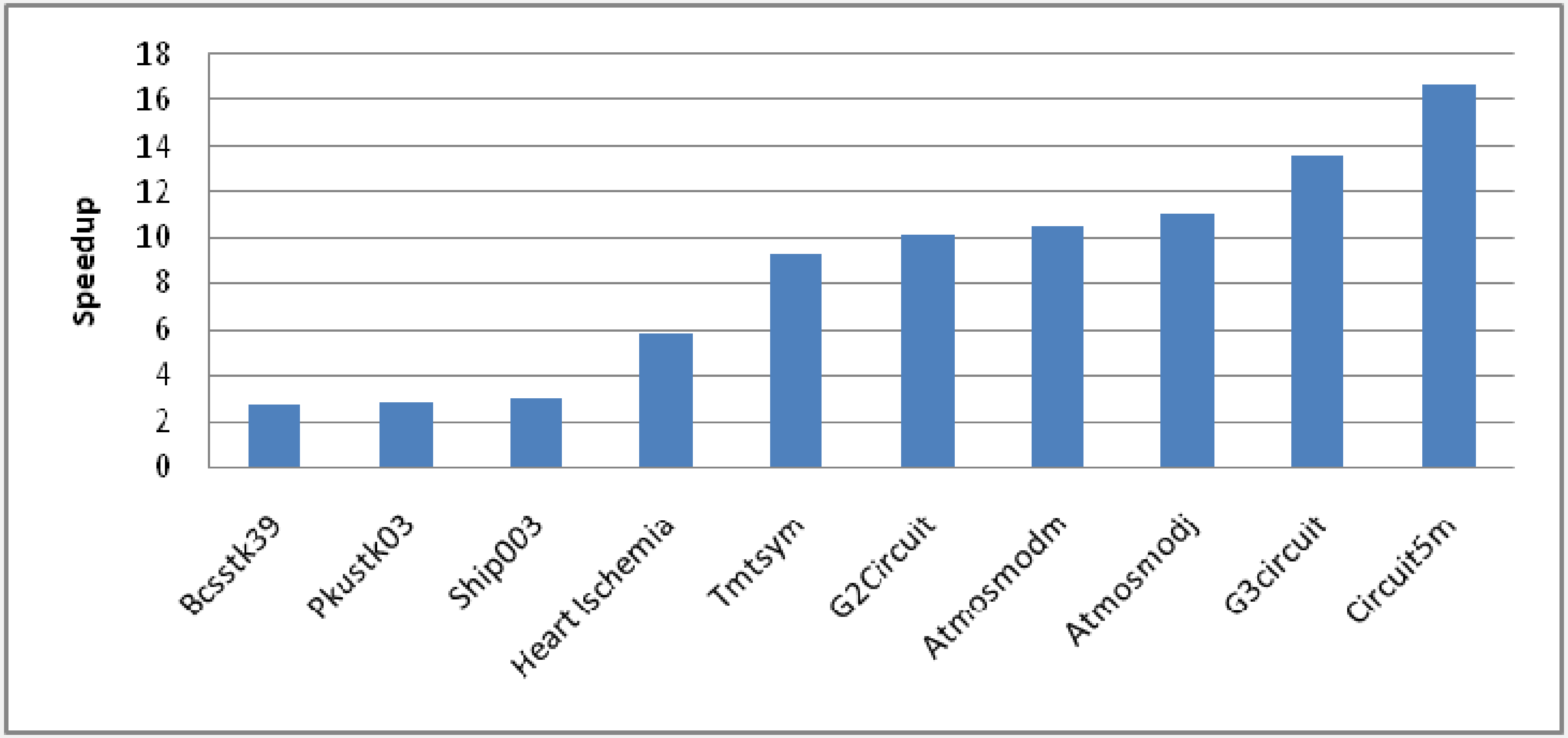




- Nonzeros vary from 42K to 14.8M

Conjugate Gradient

GPU/CPU End to End Speedup – Nearly identical performance in each of 10 runs



CPU: Intel Core 2 1.86GHz
GPU: NVIDIA GeForce GTX 280
Double Precision is used in all examples

Jacobi and Minimal Residual

CPU: Intel Core 2 1.86GHz
GPU: NVIDIA GeForce GTX 280
Double Precision is used in all examples

107K x 107K Heart Ischemia Model			
Algorithm	Time (seconds)		Speedup
	CPU	GPU	
CG	164.57	31.05	5.3x
Jacobi	7.42	1.46	3.4x
MinRes	81.96	11.80	6.9x

Third Party Implementations

- Many third party packages are available as open source
- They may perform better but are more difficult or impossible to incorporate into the user experience of SCIRun
- CNC Number Cruncher (CG implementation)
    - Gocad Research Group – Nancy University, France

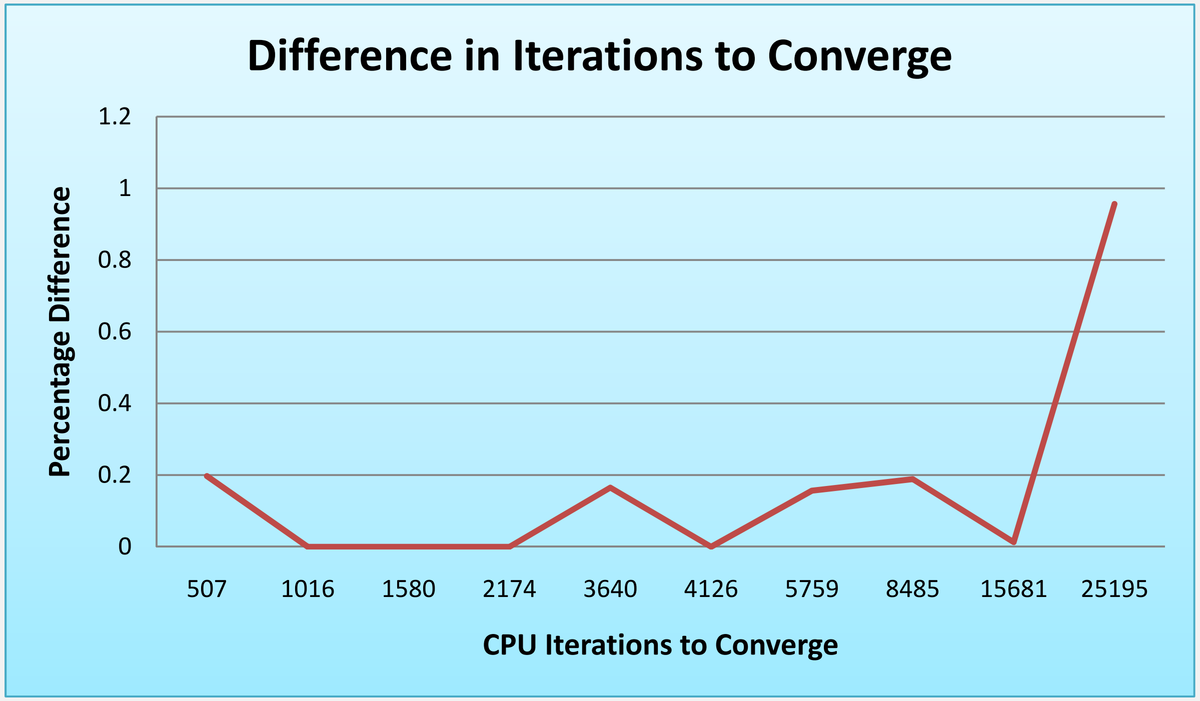
107K x 107K Heart Ischemia Model

Algorithm	Time (seconds) CPU	Time (seconds) GPU	Speedup
CG	164.57	31.05	5.3x
3rd Party CG	164.57	27.98	5.9x

L.Buatois, G.Caumon & B.Lévy, Concurrent Number Cruncher: An Efficient Sparse Linear Solver on the GPU, High Performance Computing and Communications, Third International Conference, HPCC, 2007.

Northeastern

Validation of Results



- Different orders of operations still affect the iterations necessary to achieve desired error
  - Double precision is necessary to limit this
  - The CPU and GPU differ in the number of iterations needed to converge by less than 1%

Discussion

- Speedup was achieved using the original CPU algorithm
  - The only added operations are transferring the data to the GPU
  - The algorithms were accelerated by a simple technique that can be applied to algorithms throughout SCIRun

Iterative portion of the Jacobi Method solver

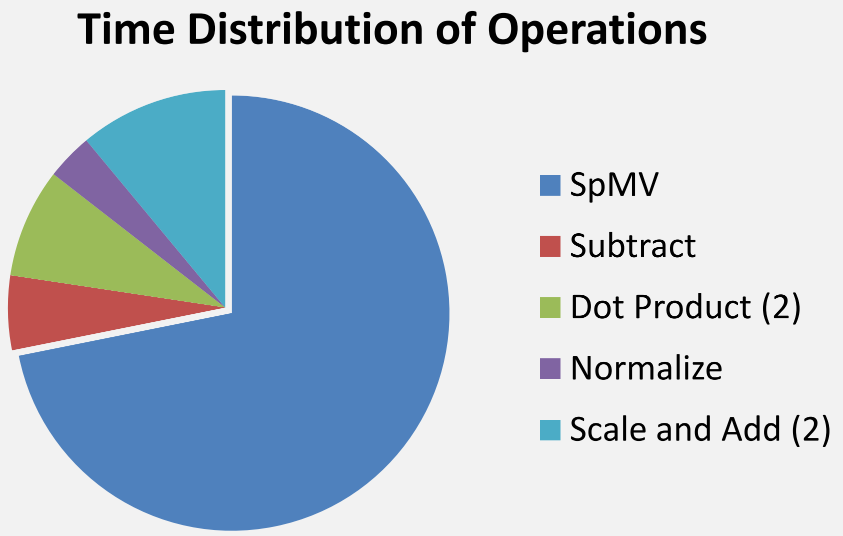
```
PLA.mult(DIAG,Z,Z);
PLA.sub(Z,X,X);
PLA.mult(A,X,Z);
PLA.sub(Z,B,Z);

error = PLA.norm(Z) / bnorm;

if (error < xmin)
{
    PLA.copy(X,XMIN);
    xmin = error;
}

if (PLA.first())
    convergence_[niter] = xmin;

niter++;
```

```
GPUPLA.mult(DIAG,Z,Z);
GPUPLA.sub(Z,X,X);
GPUPLA.mult(A,X,Z);
GPUPLA.sub(Z,B,Z);

error = GPUPLA.norm(Z) / bnorm;

if (error < xmin)
{
    GPUPLA.copy(X,XMIN);
    xmin = error;
}

if (GPUPLA.first())
    convergence_[niter] = xmin;

niter++;
```

Where the Performance is Realized

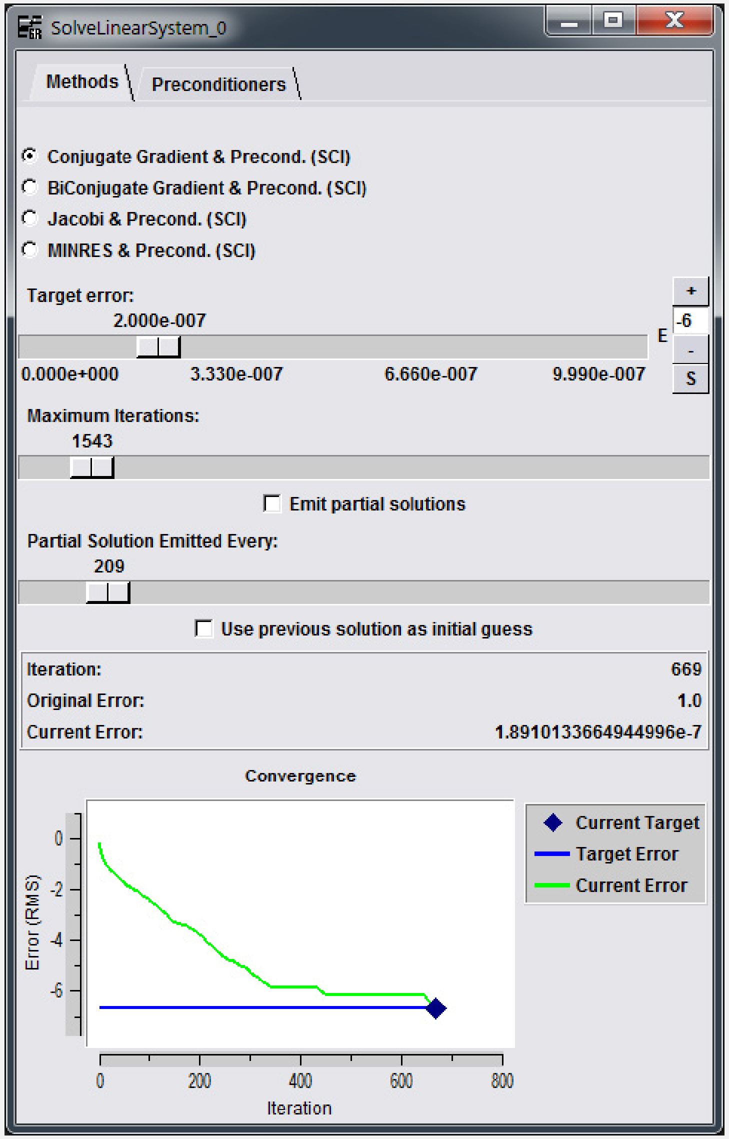
- In SpMV, each row is computed by one thread
    - Small number of rows = low utilization of GPU
- The other vector operations (mostly accelerated via the CUBLAS library) are relatively fast but occupy a low % of total time

Calculation	CPU (ms)	GPU (ms)	Speedup
Data Copy and Setup	0.08	190.22	**-2377.75x**
Preconditioner	145.11	8.26	**17.57x**
SpMV	130.68	9.37	**13.95x**
Subtract	21.09	0.72	**29.29x**
Dot Product (2 per iter)	12.97	0.53	**24.47x**
Norm	6.77	0.45	**15.04x**
Scale and Add (2 per iter)	19.62	0.72	**27.25x**
Total time per iteration	223.72	13.04	**17.15x**



Northeastern

SolveLinearSystem Module Modifications



SolveLinearSystem_0

Methods | Preconditioners

- [✓ Conjugate Gradient & Precond. (SCI)]
- BiConjugate Gradient & Precond. (SCI)
- Jacobi & Precond. (SCI)
- MINRES & Precond. (SCI)

Target error:
2.000e-007
0.000e+000 3.330e-007 6.660e-007 9.990e-007
E -6

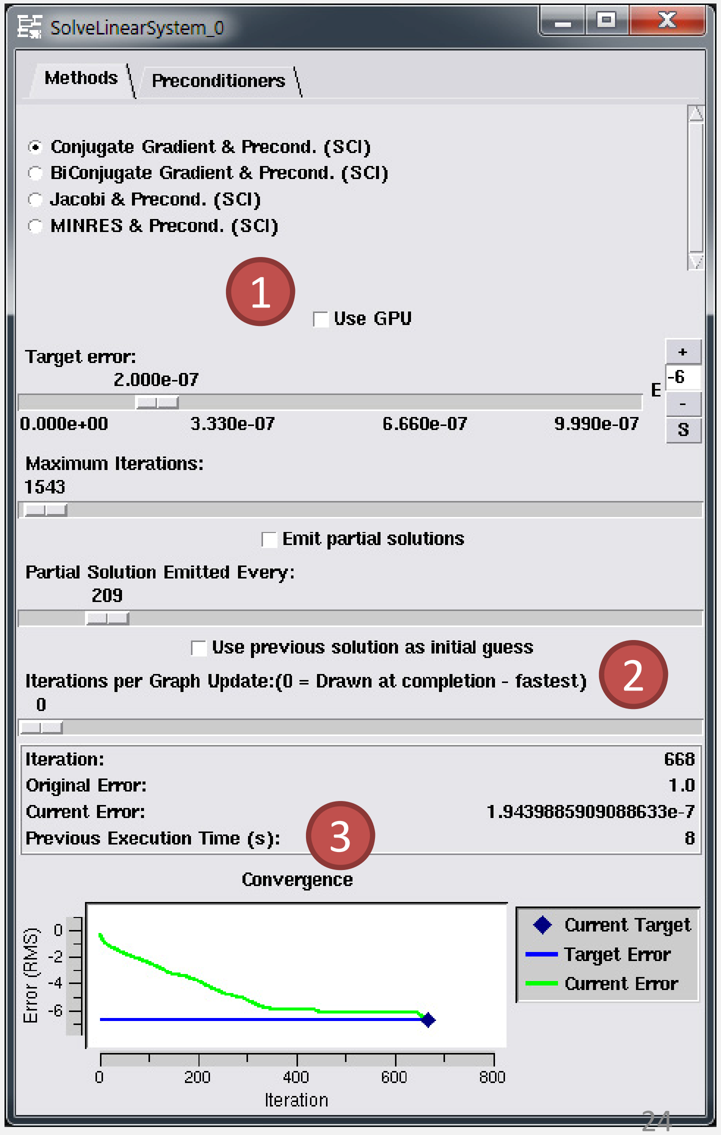
Maximum Iterations:
1543

☐ Emit partial solutions

Partial Solution Emitted Every:
209

☐ Use previous solution as initial guess

Iteration:	669
Original Error:	1.0
Current Error:	1.8910133664944996e-7



SolveLinearSystem_0

Methods | Preconditioners

- [✓ Conjugate Gradient & Precond. (SCI)]
- BiConjugate Gradient & Precond. (SCI)
- Jacobi & Precond. (SCI)
- MINRES & Precond. (SCI)

1

☐ Use GPU

Target error:
2.000e-07
0.000e+00 3.330e-07 6.660e-07 9.990e-07
E -6

Maximum Iterations:
1543

☐ Emit partial solutions

Partial Solution Emitted Every:
209

☐ Use previous solution as initial guess

Iterations per Graph Update:(0 = Drawn at completion - fastest)
0

2

Iteration:	668
Original Error:	1.0
Current Error:	1.9439885909088633e-7
Previous Execution Time (s):	8

3

Limitations

- Computation boundaries
  - Double precision availability & performance is limited
    - Even in the new Fermi generation of GPUs, double precision is still limited to 1/8 of single precision speed (1 DP unit per MP)
    - This will get better soon!
  - Sparse data
    - Memory coalescing is essential to good GPU performance
- Varying data characteristics
  - The worst possible data scenario could cause poor GPU performance

Successes

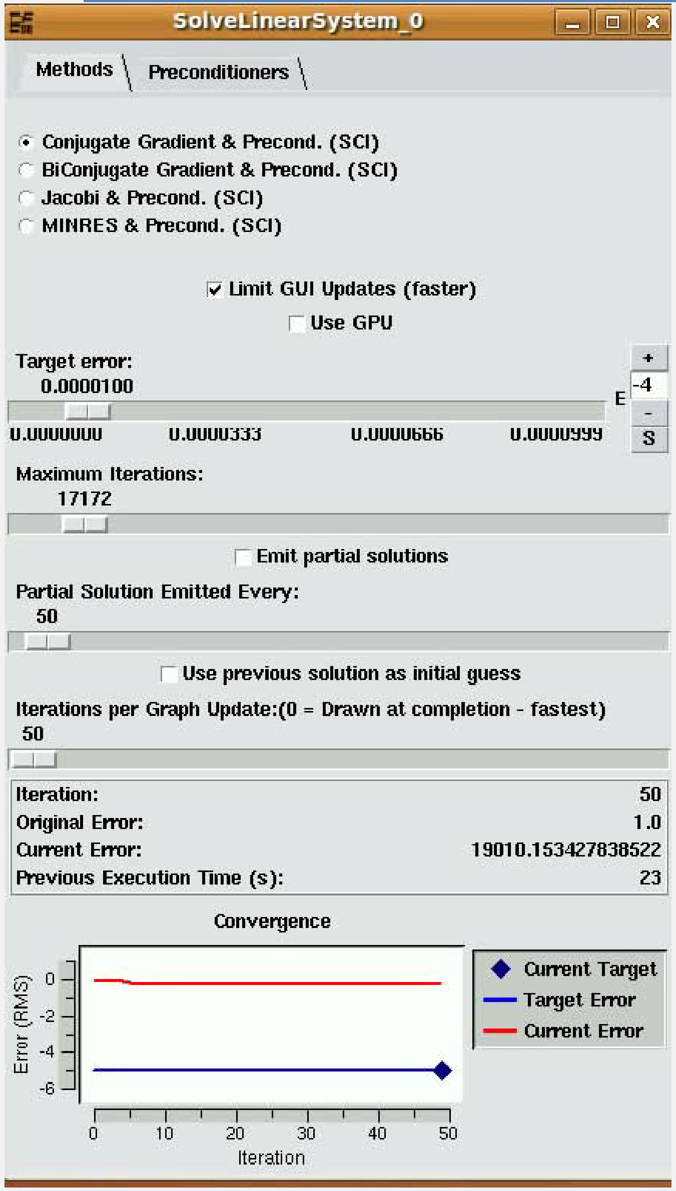
- User experience
  - The scientist using SCIRun gets results quicker
  - Transparency – Same user interaction during GPU accelerated functions
- SCIRun development – SCIRun is an open source PSE
  - GPU can be used following pre-existing programming paradigm
- Extensibility to other PSEs
  - Algorithms can be accelerated and still provide adequate interface communication by performing small individual calculations rather than complex kernels

Future Work

- Choose between CPU and GPU algorithms automatically at run-time
- Experiment with new SpMV techniques and newly released libraries for these functions
- Investigate better asynchronous techniques for inter-algorithm visualizations
- Demonstrate acceleration of algorithms outside of the linear solver module

Northeastern

A video recording of the CPU and GPU versions of the Conjugate Gradient Algorithm


SolveLinearSystem_0

Methods | Preconditioners

- [✓ Conjugate Gradient & Precond. (SCI)]
- BiConjugate Gradient & Precond. (SCI)
- Jacobi & Precond. (SCI)
- MINRES & Precond. (SCI)

☑ Limit GUI Updates (faster)

☐ Use GPU

Target error:
0.0000100 E -4

Maximum Iterations:
17172

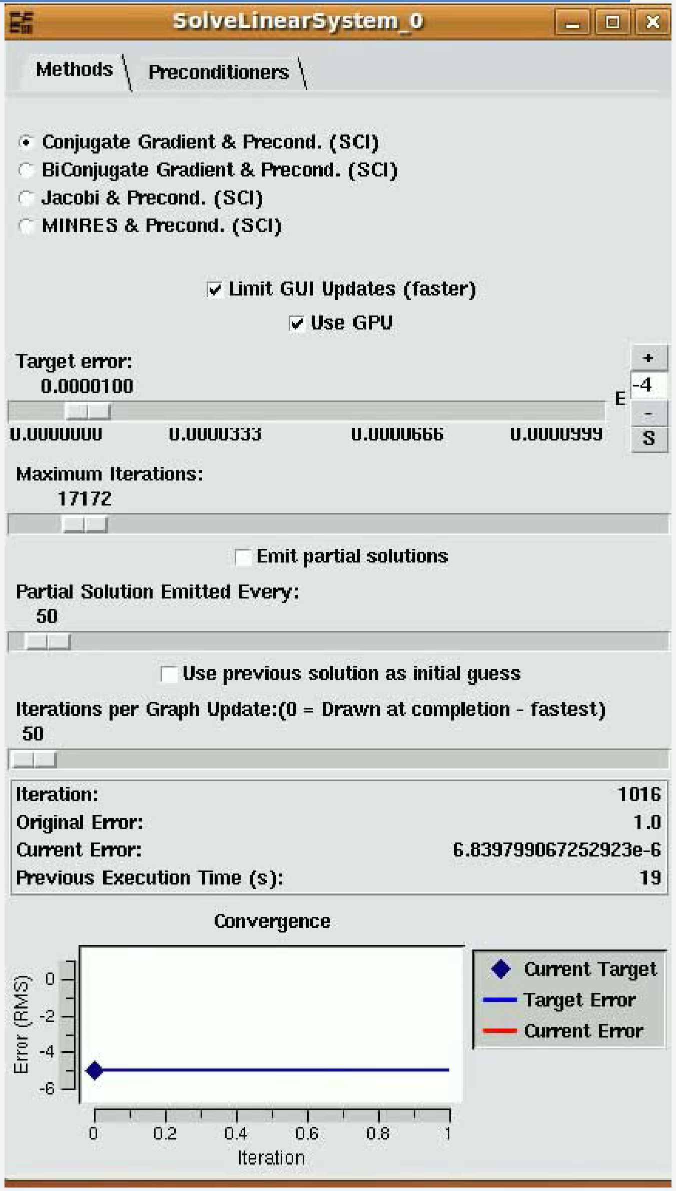
☐ Emit partial solutions

Partial Solution Emitted Every:
50

☐ Use previous solution as initial guess

Iterations per Graph Update:(0 = Drawn at completion - fastest)
50

```
Iteration:                                      50
Original Error:                                1.0
Current Error:                  19010.153427838522
Previous Execution Time (s):                    23
```


SolveLinearSystem_0

Methods | Preconditioners

- [✓ Conjugate Gradient & Precond. (SCI)]
- BiConjugate Gradient & Precond. (SCI)
- Jacobi & Precond. (SCI)
- MINRES & Precond. (SCI)

☑ Limit GUI Updates (faster)

☑ Use GPU

Target error:
0.0000100 E -4

Maximum Iterations:
17172

☐ Emit partial solutions

Partial Solution Emitted Every:
50

☐ Use previous solution as initial guess

Iterations per Graph Update:(0 = Drawn at completion - fastest)
50

```
Iteration:                                    1016
Original Error:                                1.0
Current Error:                6.839799067252923e-6
Previous Execution Time (s):                    19
```

Thank You

This work is supported by:

- NSF CenSSIS - The Center for Subsurface Sensing and Imaging
- BIAT – A Biomedical Imaging Acceleration Testbed: NSF Award Number 0946463
- CIBC, the Center for Integrative Biomedical Computing, which develops and maintains SCIRun

Devon Yablonski
Yablonski.d@husky.neu.edu