Adapting Sparse Triangular Solution to GPUs

Brad Suchoski, Caleb Severn, Manu Shantharam, and Padma Raghavan

The Pennsylvania State University

September 10, 2012

PENNSTATE

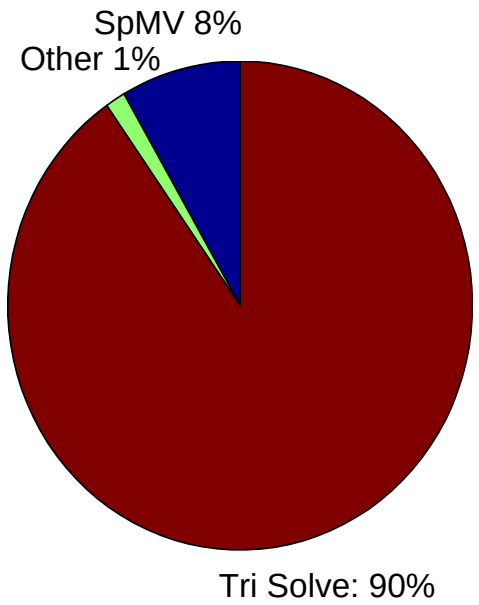
Introduction

- Trends in recent years show an increasing number of high performance computing systems are including GPU hardware to accelerate scientific calculations
- Many scientific applications, such as numerical PDE solvers, involve solving linear systems of equations where the underlying matrix is sparse
- Preconditioned iterative solvers, such as preconditioned conjugate gradients, are often used to solve these systems

PENNSTATE

Introduction

- The computation time of preconditioned iterative methods is mainly dominated by two operations
  - Sparse Matrix Vector multiply (SpMV)
  - Sparse Triangular Solve
- Recently, Naumov proposed a dependency graph based "level set" approach, but sparse triangular solve still remains the dominant operation at up to 90% of the PCG iteration time



PENNSTATE

Introduction

- Several researchers have considered parallel sparse triangular solve on earlier generations of single instruction multiple data (SIMD) systems
- Schreiber and Tang were the first to propose graph coloring to extract large amounts of fine-grained parallelism
- We expect that graph coloring will be effective at extracting the large amounts of fine-grained parallelism needed by a GPU

PENNSTATE

PENNSTATE

NVidia GPU Architecture

- Single Instruction Multiple Data (SIMD)
- Up to 16 streaming multiprocessors (SMP)
  - Share a single instruction pipeline
  - Runs groups of 32 threads called a warp in lock-step
  - Diverging branches cause execution the warp to be serialized
- Up to 48 warps (1536 threads) form a block
  - Runs on a single SMP
  - Threads can cooperate using barrier synchronization
- Groups of blocks called grid are launched in a single kernel call
  - No synchronization between threads in different blocks

PENNSTATE

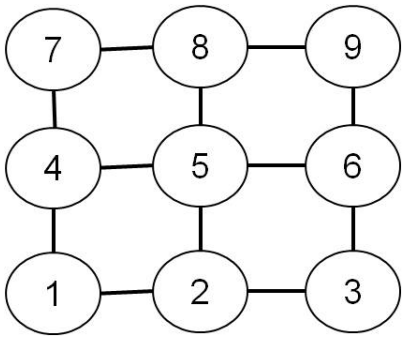
GPU Memory Characteristics

- With so many threads running concurrently bandwidth is major concern
- Interface to DRAM 384 bits (48 bytes) wide
- Bandwidth is maximized when all threads access memory in the same contiguous 48 byte block
- Threads on an SMP share 64 KB block of memory that can be configured as 16 KB shared memory and 48 KB L1 cache, or 48 KB shared memory and 16 KB L1 cache
- L2 cache is shared by all SMP

PENNSTATE

Sparse Triangular Solve Example

- 3×3 grid, 5 point finite difference matrix

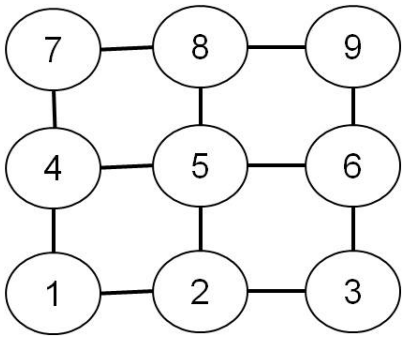


a_{11}	a_{12}		a_{14}					
a_{21}	a_{22}	a_{13}		a_{15}				
	a_{32}	a_{33}			a_{36}			
a_{41}			a_{44}	a_{45}		a_{47}		
	a_{52}		a_{54}	a_{55}	a_{56}		a_{58}	
		a_{63}		a_{65}	a_{66}			a_{69}
			a_{74}			a_{77}	a_{78}	
				a_{85}		a_{87}	a_{88}	a_{89}
					a_{96}		a_{98}	a_{99}

PENNSTATE 1855

Sparse Triangular Solve Example

- 3×3 grid, 5 point finite difference matrix

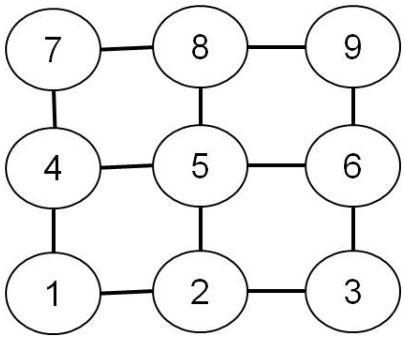


l_{11}								
l_{21}	l_{22}							
	l_{32}	l_{33}						
l_{41}			l_{44}					
	l_{52}		l_{54}	l_{55}				
		l_{63}		l_{65}	l_{66}			
			l_{74}			l_{77}		
				l_{85}		l_{87}	l_{88}	
					l_{96}		l_{98}	l_{99}

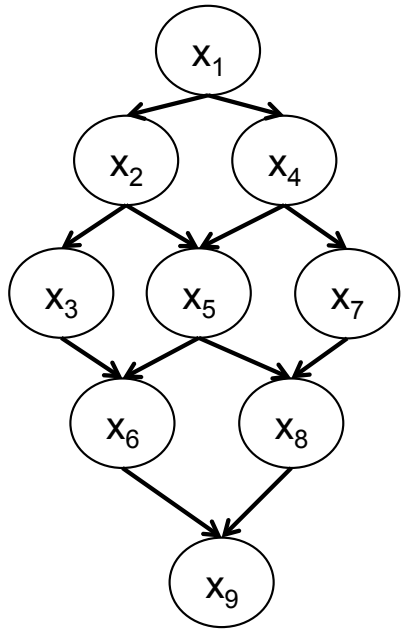
PENNSTATE

Sparse Triangular Solve Example

- 3×3 grid, 5 point finite difference matrix



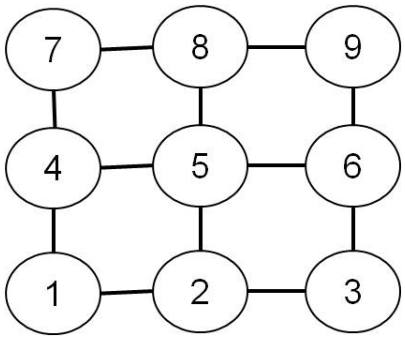
l_{11}								
l_{21}	l_{22}							
	l_{32}	l_{33}						
l_{41}			l_{44}					
	l_{52}		l_{54}	l_{55}				
		l_{63}		l_{65}	l_{66}			
			l_{74}			l_{77}		
				l_{85}		l_{87}	l_{88}	
					l_{96}		l_{98}	l_{99}



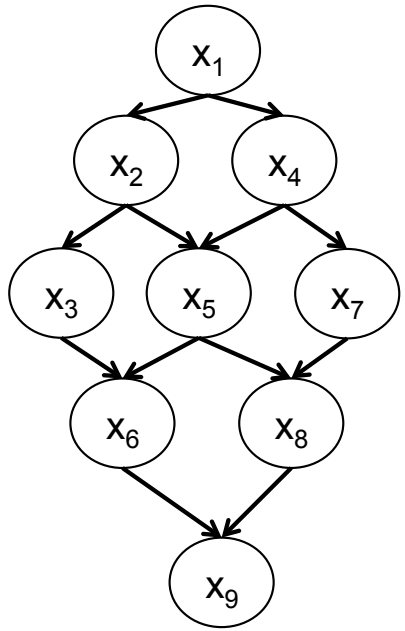
PENNSTATE

Sparse Triangular Solve Example

- 3×3 grid, 5 point finite difference matrix
- Nvidia's LS algorithm: modified BFS groups independent rows into levels. Each level can be solved in parallel
  - $L_1 = \{\mathbf{x}_1\}$, $L_2 = \{\mathbf{x}_2, \mathbf{x}_4\}$, $L_3 = \{\mathbf{x}_3, \mathbf{x}_5, \mathbf{x}_7\}$, $L_4 = \{\mathbf{x}_6, \mathbf{x}_8\}$, $L_5 = \{\mathbf{x}_9\}$



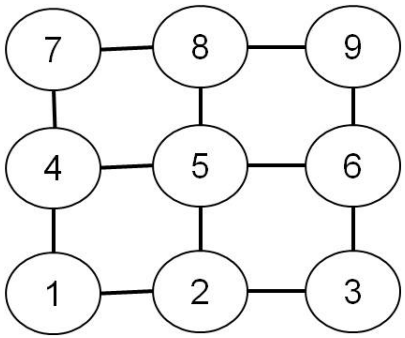
l_{11}								
l_{21}	l_{22}							
	l_{32}	l_{33}						
l_{41}			l_{44}					
	l_{52}		l_{54}	l_{55}				
		l_{63}		l_{65}	l_{66}			
			l_{74}			l_{77}		
				l_{85}		l_{87}	l_{88}	
					l_{96}		l_{98}	l_{99}



PENNSTATE

Coloring Example

- Color vertices in $G(A)$ so that any two vertices V_i and V_j have a different color if edge $(i, j) \in E \text{ of } G(A)$ using minimum number of colors

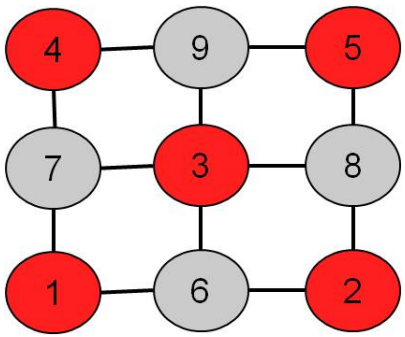


a_{11}	a_{12}		a_{14}					
a_{21}	a_{22}	a_{13}		a_{15}				
	a_{32}	a_{33}			a_{36}			
a_{41}			a_{44}	a_{45}		a_{47}		
	a_{52}		a_{54}	a_{55}	a_{56}		a_{58}	
		a_{63}		a_{65}	a_{66}			a_{69}
			a_{74}			a_{77}	a_{78}	
				a_{85}		a_{87}	a_{88}	a_{89}
					a_{96}		a_{98}	a_{99}

PENNSTATE 1855

Coloring Example

- Color vertices in $G(A)$ so that any two vertices V_i and V_j have a different color if edge $(i, j) \in E \text{ of } G(A)$ using minimum number of colors
- Apply symmetric permutation π_{color} so that vertices with same color are numbered contiguously

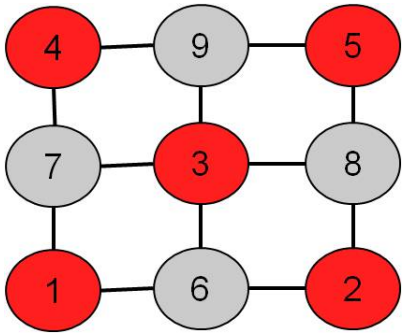


a_{11}	a_{12}		a_{14}					
a_{21}	a_{22}	a_{13}		a_{15}				
	a_{32}	a_{33}			a_{36}			
a_{41}			a_{44}	a_{45}		a_{47}		
	a_{52}		a_{54}	a_{55}	a_{56}		a_{58}	
		a_{63}		a_{65}	a_{66}			a_{69}
			a_{74}			a_{77}	a_{78}	
				a_{85}		a_{87}	a_{88}	a_{89}
					a_{96}		a_{98}	a_{99}

PENNSTATE 1855

Coloring Example

- Color vertices in $G(A)$ so that any two vertices V_i and V_j have a different color if edge $(i, j) \in E \text{ of } G(A)$ using minimum number of colors
- Apply symmetric permutation π_{color} so that vertices with same color are numbered contiguously

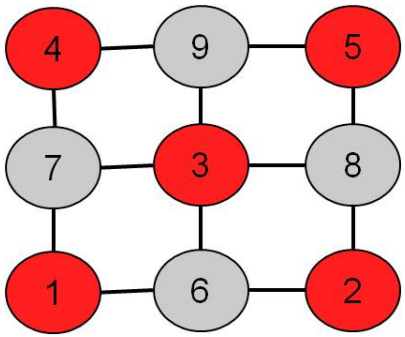


a_{11}					a_{16}	a_{17}		
	a_{22}				a_{26}		a_{28}	
		a_{33}			a_{36}	a_{37}	a_{38}	a_{39}
			a_{44}			a_{47}		a_{49}
				a_{55}			a_{58}	a_{59}
a_{61}	a_{62}	a_{63}			a_{66}			
a_{71}		a_{73}	a_{74}			a_{77}		
	a_{82}	a_{83}		a_{85}			a_{88}	
		a_{93}	a_{94}	a_{95}				a_{99}

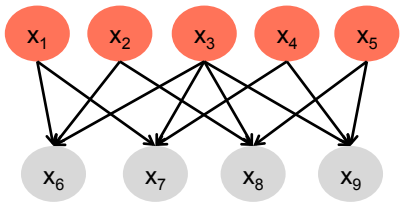
PENNSTATE

Coloring Example

- Color vertices in $G(A)$ so that any two vertices V_i and V_j have a different color if edge $(i, j) \in E \text{ of } G(A)$ using minimum number of colors
- Apply symmetric permutation π_{color} so that vertices with same color are numbered contiguously
- CS algorithm: each color can be solved in parallel
  - $C_1 = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5\}$, $C_2 = \{\mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8, \mathbf{x}_9\}$,



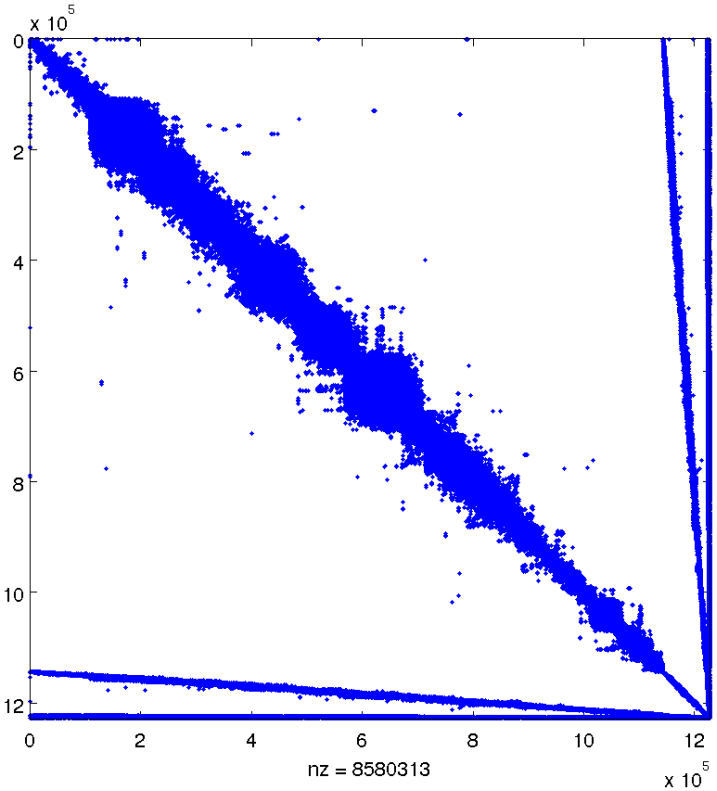
l_{11}								
	l_{22}							
		l_{33}						
			l_{44}					
				l_{55}				
l_{61}	l_{62}	l_{63}			l_{66}			
l_{71}		l_{73}	l_{74}			l_{77}		
	l_{82}	l_{83}		l_{85}			l_{88}	
		l_{93}	l_{94}	l_{95}				l_{99}



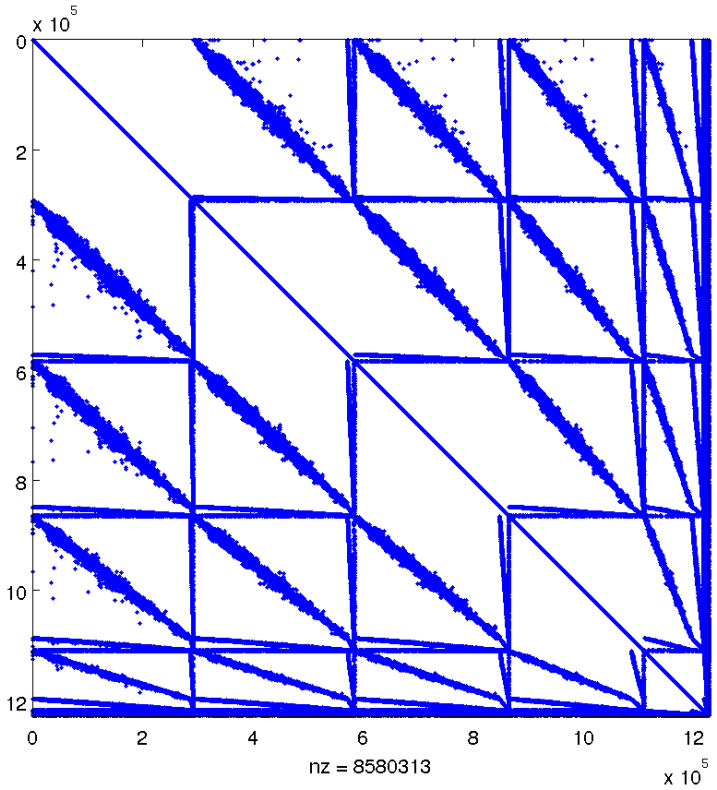
PENNSTATE

Coloring Example

- `Schmid thermal2` from UFL sparse matrix collection
- 1,228K rows
- LS on $\pi_{natural}$ has 1,240 levels. Average of 990 rows per level
- CS on π_{color} has 7 colors. Average of 175,435 rows per color.



$\pi_{natural}$



π_{color}

PENNSTATE 1855

Performance Model

- Attempt to model $T_{nnz}(A, x) = \frac{T(A,x)}{nnz(A)}$ where $T(A, x)$ is the total time to perform 1 triangular solve using method x (LS, CS, or CPLS).

PENNSTATE

Performance Model

- Attempt to model $T_{nnz}(A, x) = \frac{T(A,x)}{nnz(A)}$ where $T(A, x)$ is the total time to perform 1 triangular solve using method x (LS, CS, or CPLS).

N	Number of rows in A
$N_r(A)$	Average number of nonzeros per row in A, $\frac{nnz(A)}{N}$
$\delta(A, x)$	The number of dependency sets in A using method x, i.e. the number of levels in LS or the number of colors in CS
$\phi(A, x)$	Average parallel workload, average number of rows per dependency set $\frac{N}{\delta(A,x)}$
P	Hardware parallel capacity, i.e. the number of CUDA cores available (512 for Tesla M2090)

PENNSTATE

Performance Model

- Attempt to model $T_{nnz}(A, x) = \frac{T(A,x)}{nnz(A)}$ where $T(A, x)$ is the total time to perform 1 triangular solve using method x (LS, CS, or CPLS).

N	Number of rows in A
$N_r(A)$	Average number of nonzeros per row in A, $\frac{nnz(A)}{N}$
$\delta(A, x)$	The number of dependency sets in A using method x, i.e. the number of levels in LS or the number of colors in CS
$\phi(A, x)$	Average parallel workload, average number of rows per dependency set $\frac{N}{\delta(A,x)}$
P	Hardware parallel capacity, i.e. the number of CUDA cores available (512 for Tesla M2090)

$$T_{nnz}(A, x) = \begin{cases} \frac{1}{\alpha\phi(A,x)}, & \phi(A, x) \leq P \\ \frac{1}{\alpha P}, & \text{otherwise.} \end{cases}$$

PENNSTATE

Performance Model

- Attempt to model $T_{nnz}(A, x) = \frac{T(A,x)}{nnz(A)}$ where $T(A, x)$ is the total time to perform 1 triangular solve using method x (LS, CS, or CPLS).

N	Number of rows in A
$N_r(A)$	Average number of nonzeros per row in A, $\frac{nnz(A)}{N}$
$\delta(A, x)$	The number of dependency sets in A using method x, i.e. the number of levels in LS or the number of colors in CS
$\phi(A, x)$	Average parallel workload, average number of rows per dependency set $\frac{N}{\delta(A,x)}$
P	Hardware parallel capacity, i.e. the number of CUDA cores available (512 for Tesla M2090)

$$N_r(A)\,T_{nnz}(A, x) = \begin{cases} \frac{1}{\alpha\phi(A,x)}, & \phi(A, x) \leq P \\ \frac{1}{\alpha P}, & \text{otherwise.} \end{cases}$$

PENNSTATE 1855

Performance Model

- Attempt to model $T_{nnz}(A, x) = \frac{T(A,x)}{nnz(A)}$ where $T(A, x)$ is the total time to perform 1 triangular solve using method x (LS, CS, or CPLS).

N	Number of rows in A
$N_r(A)$	Average number of nonzeros per row in A, $\frac{nnz(A)}{N}$
$\delta(A, x)$	The number of dependency sets in A using method x, i.e. the number of levels in LS or the number of colors in CS
$\phi(A, x)$	Average parallel workload, average number of rows per dependency set $\frac{N}{\delta(A,x)}$
P	Hardware parallel capacity, i.e. the number of CUDA cores available (512 for Tesla M2090)

$$T_{nnz}(A, x) = \begin{cases} \frac{1}{\alpha N_r(A)\phi(A,x)}, & \phi(A, x) \leq P \\ \frac{1}{\alpha N_r(A)P}, & \text{otherwise.} \end{cases}$$

PENNSTATE

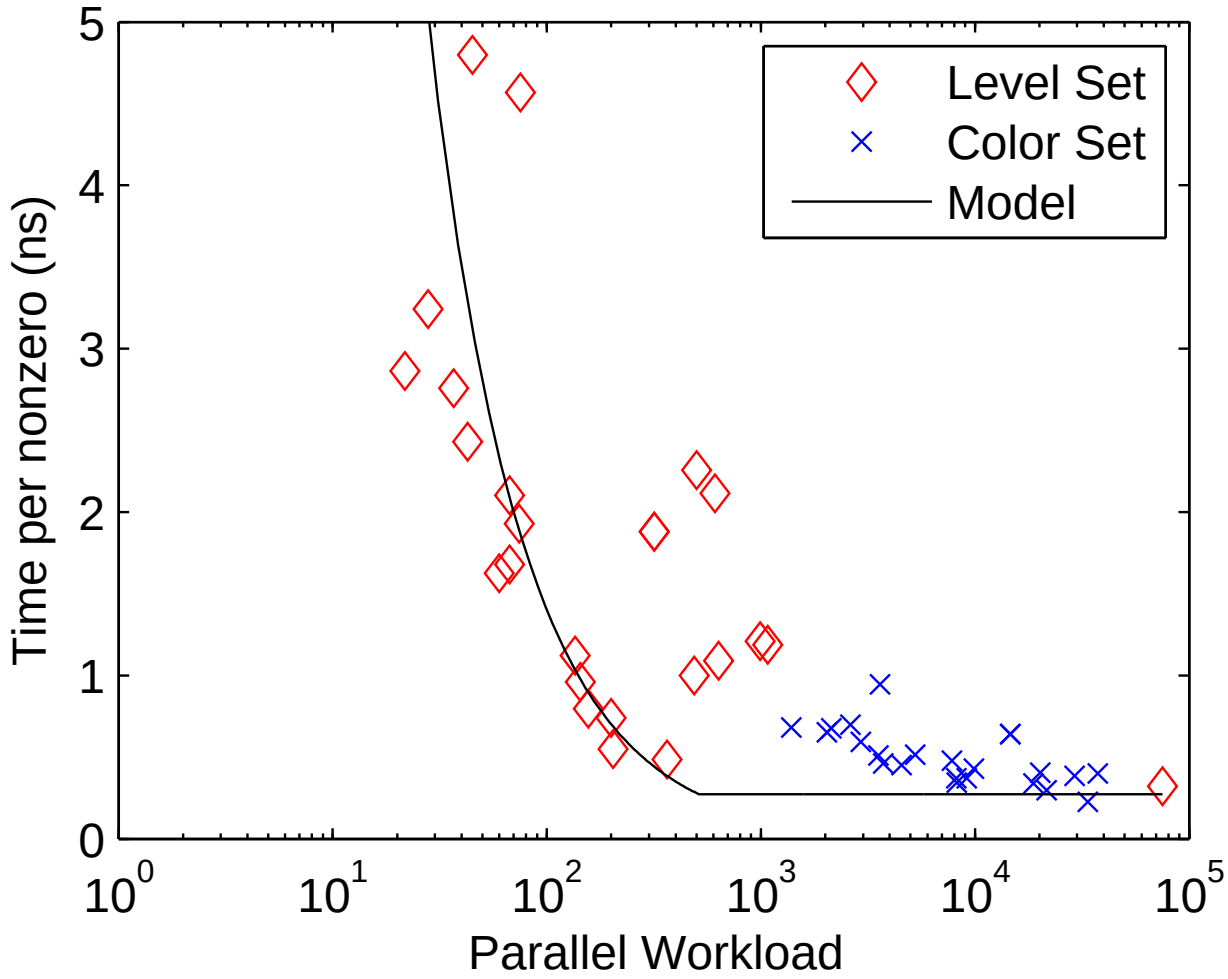
Experimental Setup

ID	Matrix	N	NNZ	# Colors	$\phi(A, CS)$	# Levels	$\phi(A, LS)$	NNZ/N
1	2cubes_sphere	101,492	1,647,264	13	7,807.1	14,330	7.1	16.2
2	thermomech_TK	102,158	711,558	7	14,594.0	323	316.3	7.0
3	thermomech_TC	102,158	711,558	7	14,594.0	323	316.3	7.0
4	x104	108,384	10,167,624	78	1,389.5	4,993	21.7	93.8
5	shipsec8	114,919	6,653,399	54	2,128.1	3,128	36.7	57.9
6	ship_003	121,728	8,086,034	60	2,028.8	4,368	27.9	66.4
7	cfd2	123,440	3,087,898	15	8,229.3	4,358	28.3	25.0
8	boneS01	127,224	6,715,152	36	3,534.0	814	156.3	52.8
9	shipsec1	140,874	7,813,404	48	2,934.9	2,101	67.1	55.5
10	bmw7st_1	141,347	7,339,667	54	2,617.5	708	199.6	51.9
11	Dubcova3	146,689	3,636,649	16	9,168.1	3,267	44.9	24.8
12	bmwcra_1	148,770	10,644,002	40	3,719.3	730	203.8	71.5
13	G2_circuit	150,102	726,674	4	37,525.5	734	204.5	4.8
14	shipsec5	179,860	10,113,096	50	3,597.2	2,689	66.9	56.2
15	thermomech_dM	204,316	1,423,116	7	29,188.0	323	632.6	7.0
16	pwtk	217,918	11,634,424	48	4,540.0	142,169	1.5	53.4
17	hood	220,542	10,768,436	42	5,251.0	605	364.5	48.8
18	BenElechi1	245,874	13,150,496	30	8,195.8	5,755	42.7	53.5
19	offshore	259,789	4,242,673	12	21,649.1	3,453	75.2	16.3
20	msdoor	415,863	20,240,935	42	9,901.5	6,938	59.9	48.7
21	af_x_k101	503,625	17,550,675	15	33,575.0	6,766	74.4	34.8
22	af_shellx	504,855	17,588,875	25	20,194.2	3,726	135.5	34.8
23	parabolic_fem	525,825	3,674,625	5	105,165.0	8	65,728.1	7.0
24	Fault_639	638,802	28,614,564	34	18,788.3	4,463	143.1	44.8
25	apache2	715,176	4,817,870	3	238,392.0	665	1,075.5	6.7
26	ecology2	999,999	4,995,991	2	499,999.5	2,000	500.0	5.0
27	thermal2	1,228,045	8,580,313	7	175,435.0	1,240	990.4	7.0
28	StocF-1465	1,465,137	21,005,389	11	133,194.3	3,005	487.6	14.3
29	G3_circuit	1,585,478	7,660,826	4	396,369.5	2,595	611.0	4.8
30	500 model grid	250,000	1,248,000	2	125,000	999	250.3	5.0

PENNSTATE

Model Verification

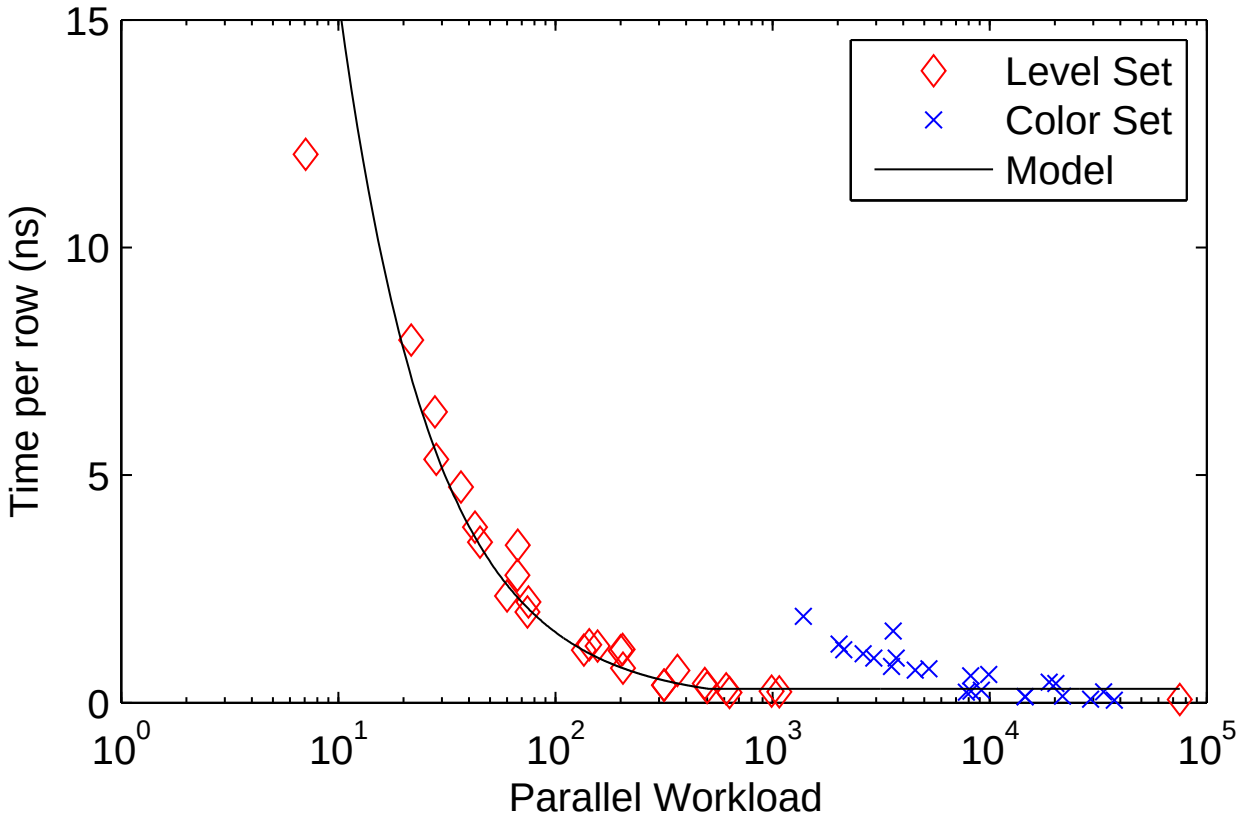
- $\alpha = 7.134e{-}3$ using linear least squares to fit model to points with $\phi(A, x) < 512 = 10^{2.71}$



PENNSTATE 1855

Model Verification

- To cancel out $N_r(A)$ compare $N_r(A) \times T_{nnz}(A, x)$ vs $\phi(A, x)$
- $\alpha = 6.448e{-}3$ using linear least squares to fit model to points with $\phi(A, x) < 512 = 10^{2.71}$



PENNSTATE

Scalability

- Now consider a series of $K \times K$ model grids
  - Representative of broader class of finite difference and finite element grids

PENNSTATE

Scalability

- Now consider a series of $K \times K$ model grids
  - Representative of broader class of finite difference and finite element grids
- LS algorithm on $\pi_{natural}$ will have $2K - 1$ parallel steps and average parallel workload $\phi(A, LS) = \frac{K^2}{2K-1} \approx \frac{1}{2}K$

PENNSTATE

Scalability

- Now consider a series of $K \times K$ model grids
  - Representative of broader class of finite difference and finite element grids
- LS algorithm on $\pi_{natural}$ will have $2K - 1$ parallel steps and average parallel workload $\phi(A, LS) = \frac{K^2}{2K-1} \approx \frac{1}{2}K$
- CS algorithm will have 2 parallel steps independent of K and average parallel workload $\phi(A, CS) = \frac{1}{2}K^2$

PENNSTATE

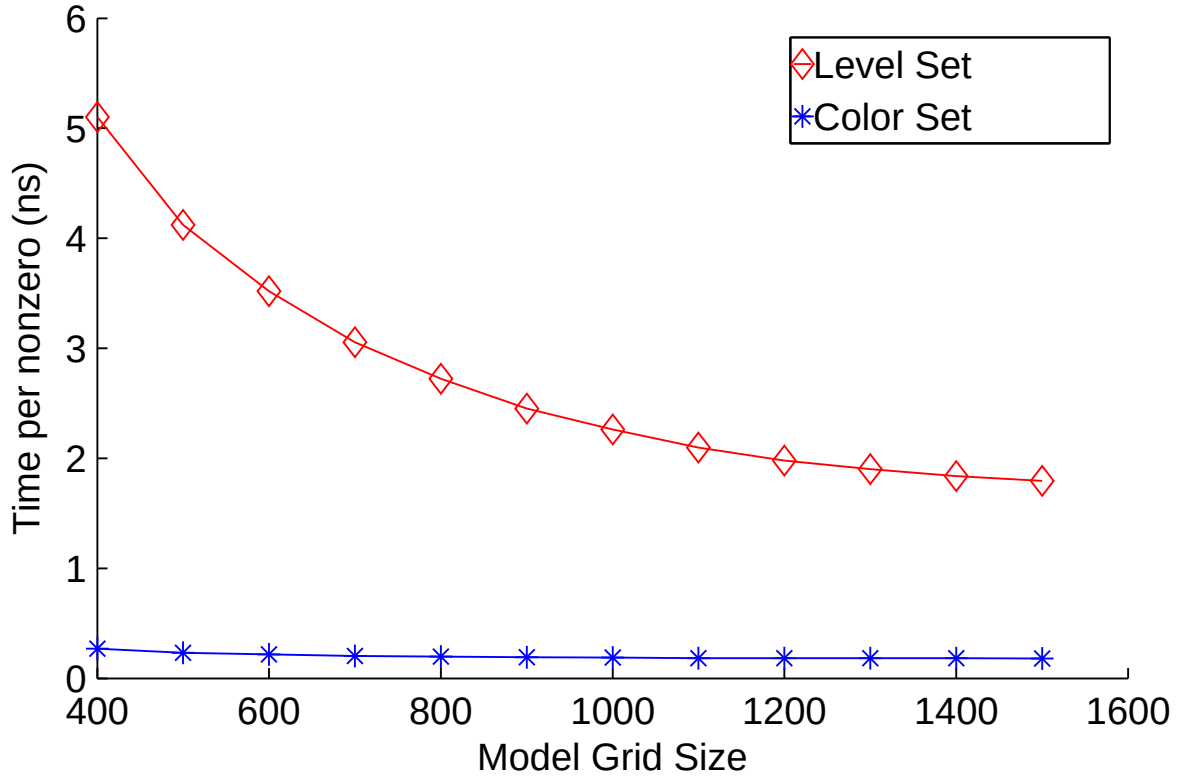
Scalability

- Now consider a series of $K \times K$ model grids
  - Representative of broader class of finite difference and finite element grids
- LS algorithm on $\pi_{natural}$ will have $2K - 1$ parallel steps and average parallel workload $\phi(A, LS) = \frac{K^2}{2K-1} \approx \frac{1}{2}K$
- CS algorithm will have 2 parallel steps independent of K and average parallel workload $\phi(A, CS) = \frac{1}{2}K^2$
- CS is much more scalable than LS algorithm.
  - Even for 400×400 model grid $\phi(A, LS) \approx 200 < 512$
  - For same model grid $\phi(A, CS) = 80,000$
  - CS algorithm bound by hardware parallel capacity when $K < \sqrt{2 \times 512} = 32$

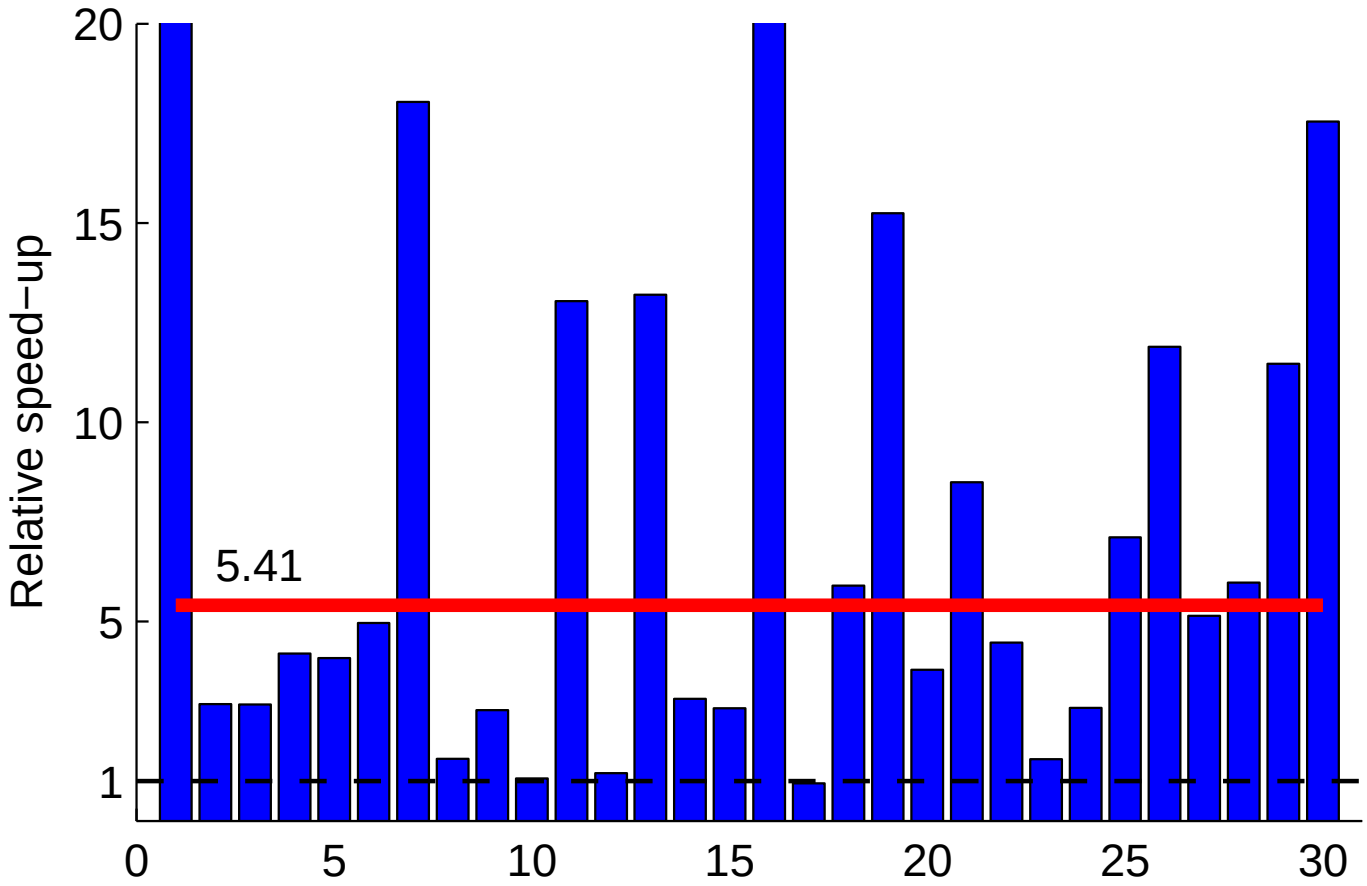
PENNSTATE

Scalability

- LS incremental performance improvements occur up through $K = 1500$
  - At $K = 1500$, $\phi(A, LS) \approx 750$ but still 1022 levels with fewer than 512 rows
- CS Relatively constant
  - Both colors contain > 512 rows

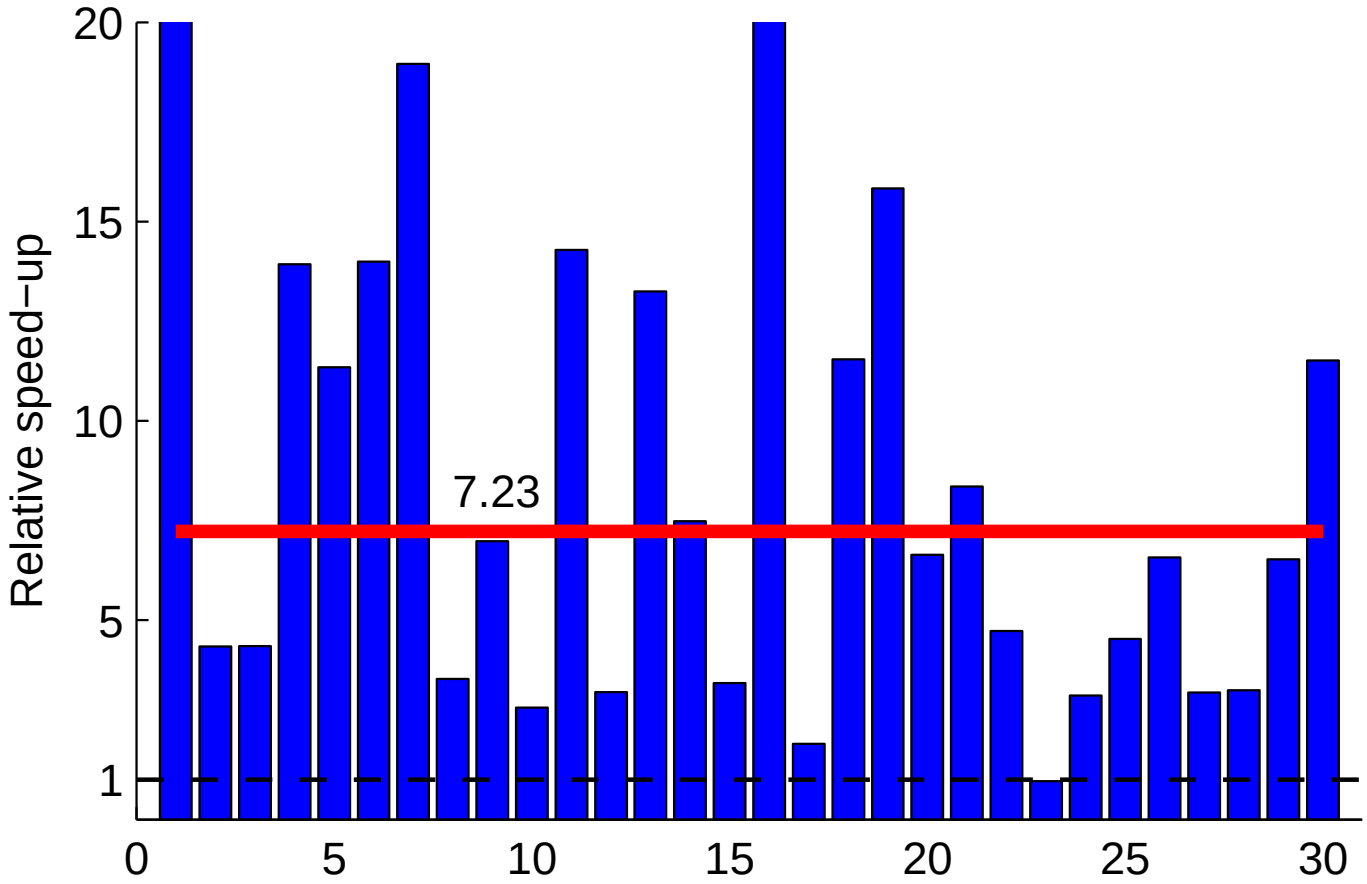


- Speedup of CS relative to LS $\frac{T(A,CS)}{T(A,LS)}$
- Geometric mean of 5.41 across all matrices



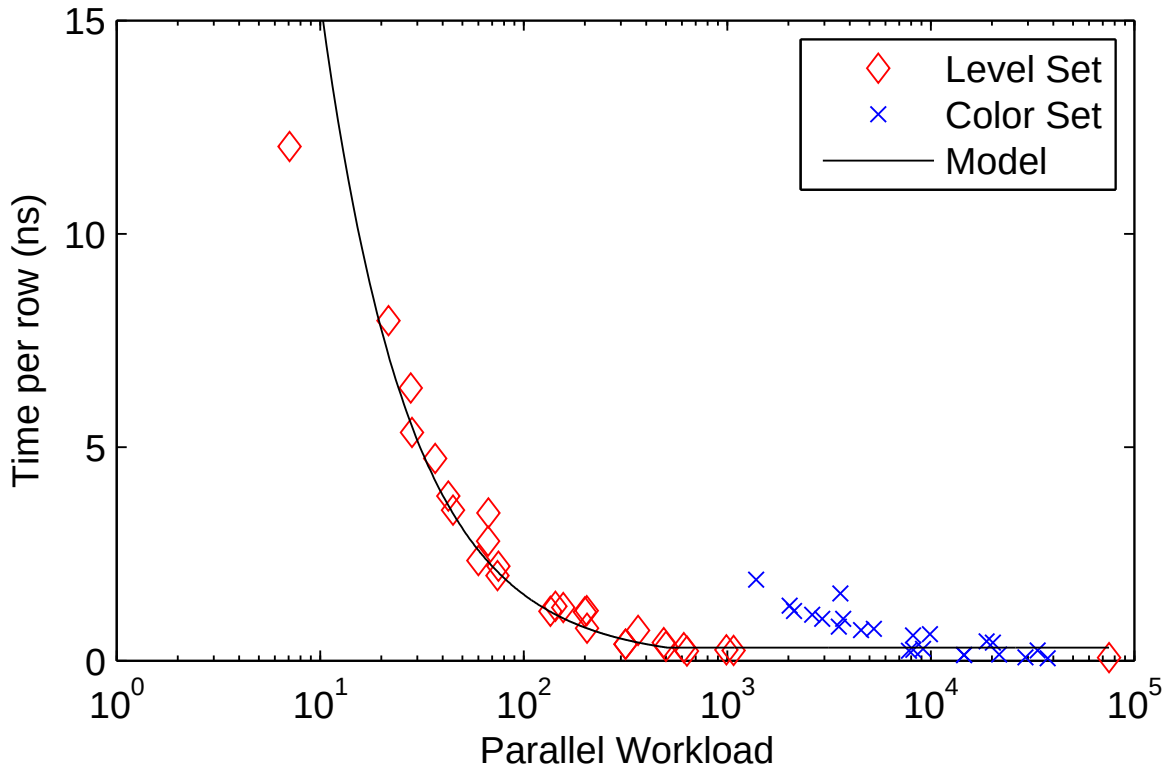
PENNSTATE

- Further speedup achieve by first applying color permutation π_{color} then using LS (CPLS)
- Mean speedup of 7.23 relative to $T(A, LS)$
- Incurs preprocessing cost of both CS and LS

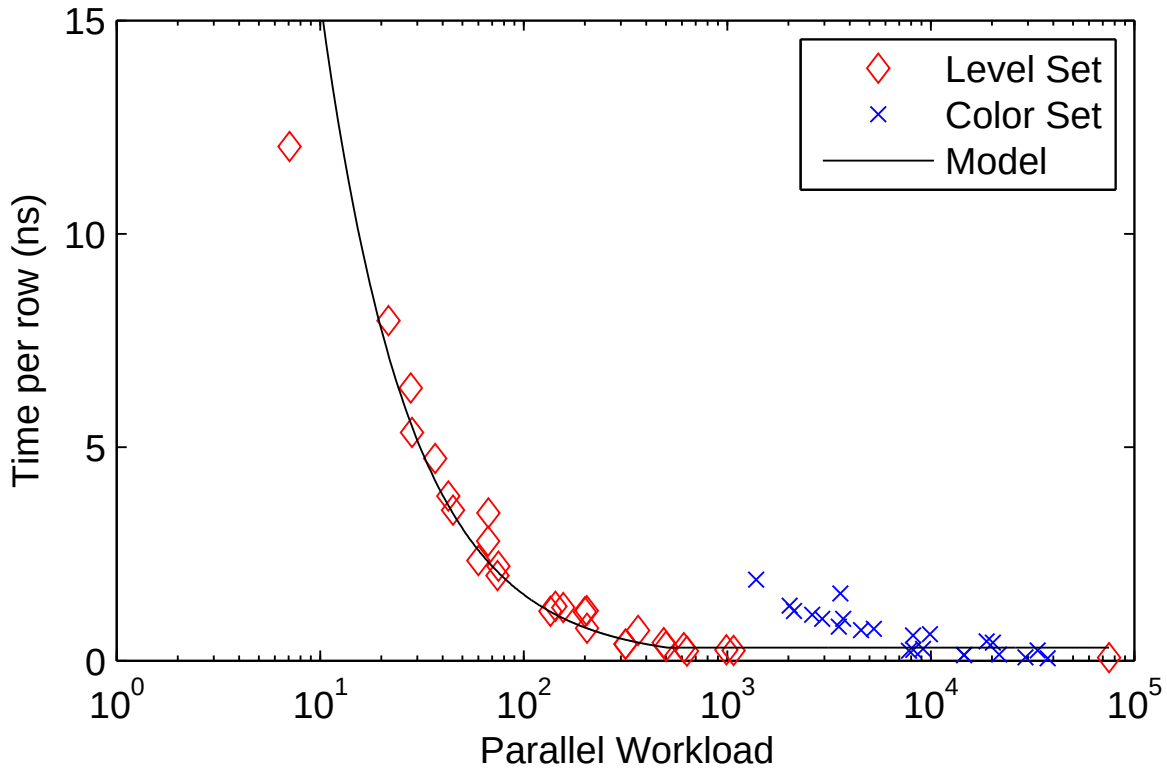


PENNSTATE

- Conjecture: additional speedups due to secondary effects of optimizations not in CS algorithm, not increased parallel workload

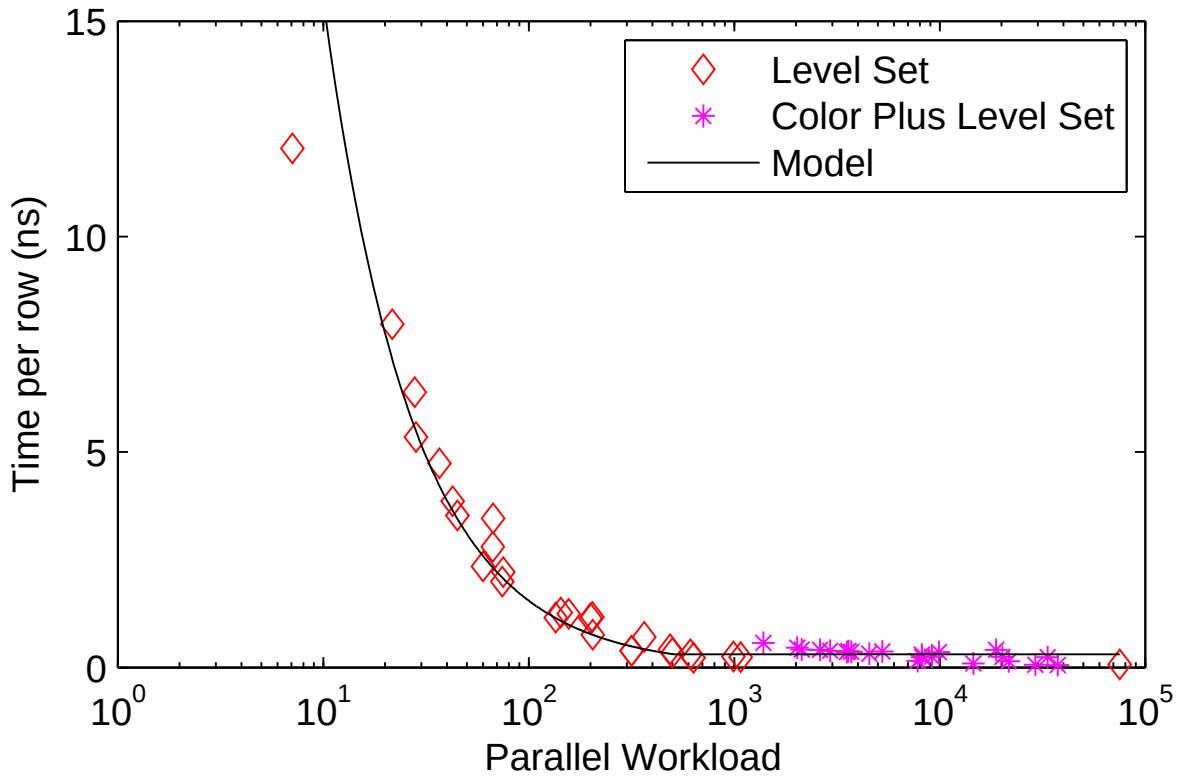


- Conjecture: additional speedups due to secondary effects of optimizations not in CS algorithm, not increased parallel workload
- Same optimizations should be in LS and CPLS. So does CPLS match model?



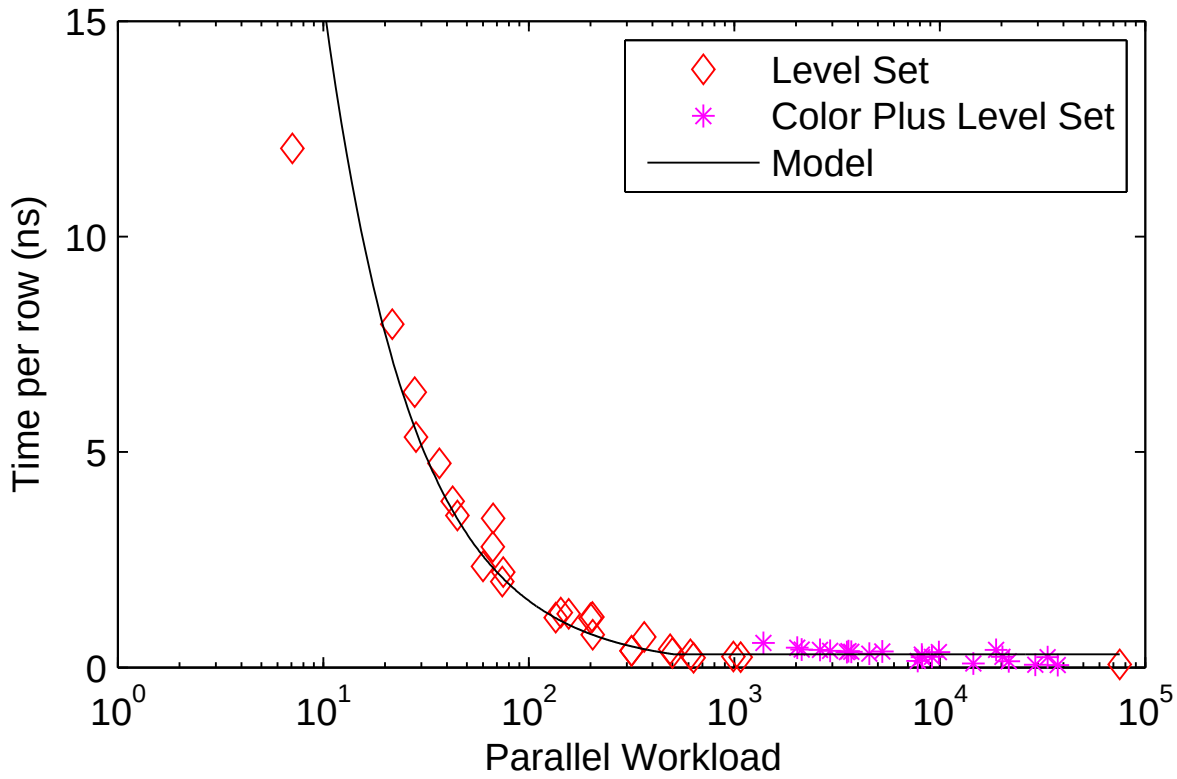
PENNSTATE

- Conjecture: additional speedups due to secondary effects of optimizations not in CS algorithm, not increased parallel workload
- Same optimizations should be in LS and CPLS. So does CPLS match model?



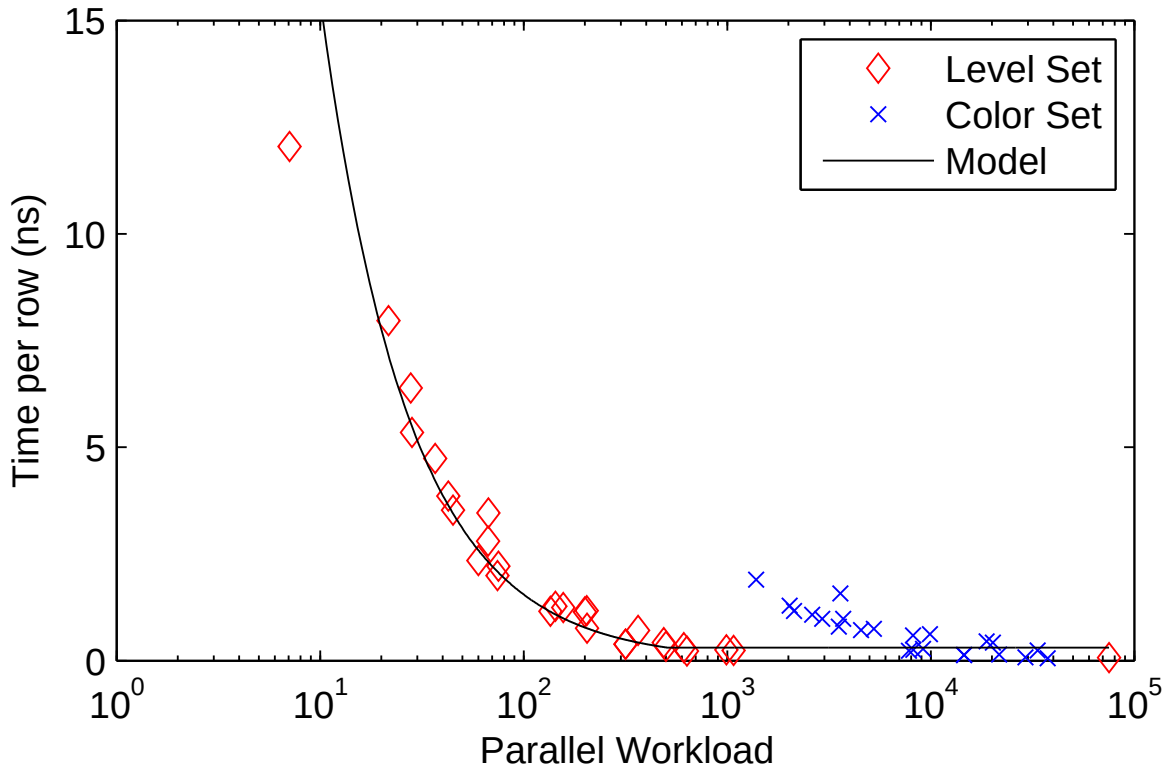
PENNSTATE

- Conjecture: additional speedups due to secondary effects of optimizations not in CS algorithm, not increased parallel workload
- Same optimizations should be in LS and CPLS. So does CPLS match model?
- CS used hard coded 4 threads per row



PENNSTATE

- Conjecture: additional speedups due to secondary effects of optimizations not in CS algorithm, not increased parallel workload
- Same optimizations should be in LS and CPLS. So does CPLS match model?
- CS used hard coded 4 threads per row
- Matrices with high N_r are dense and will typically have many colors, thus low $\phi(A, CS)$. More fragmented access pattern.



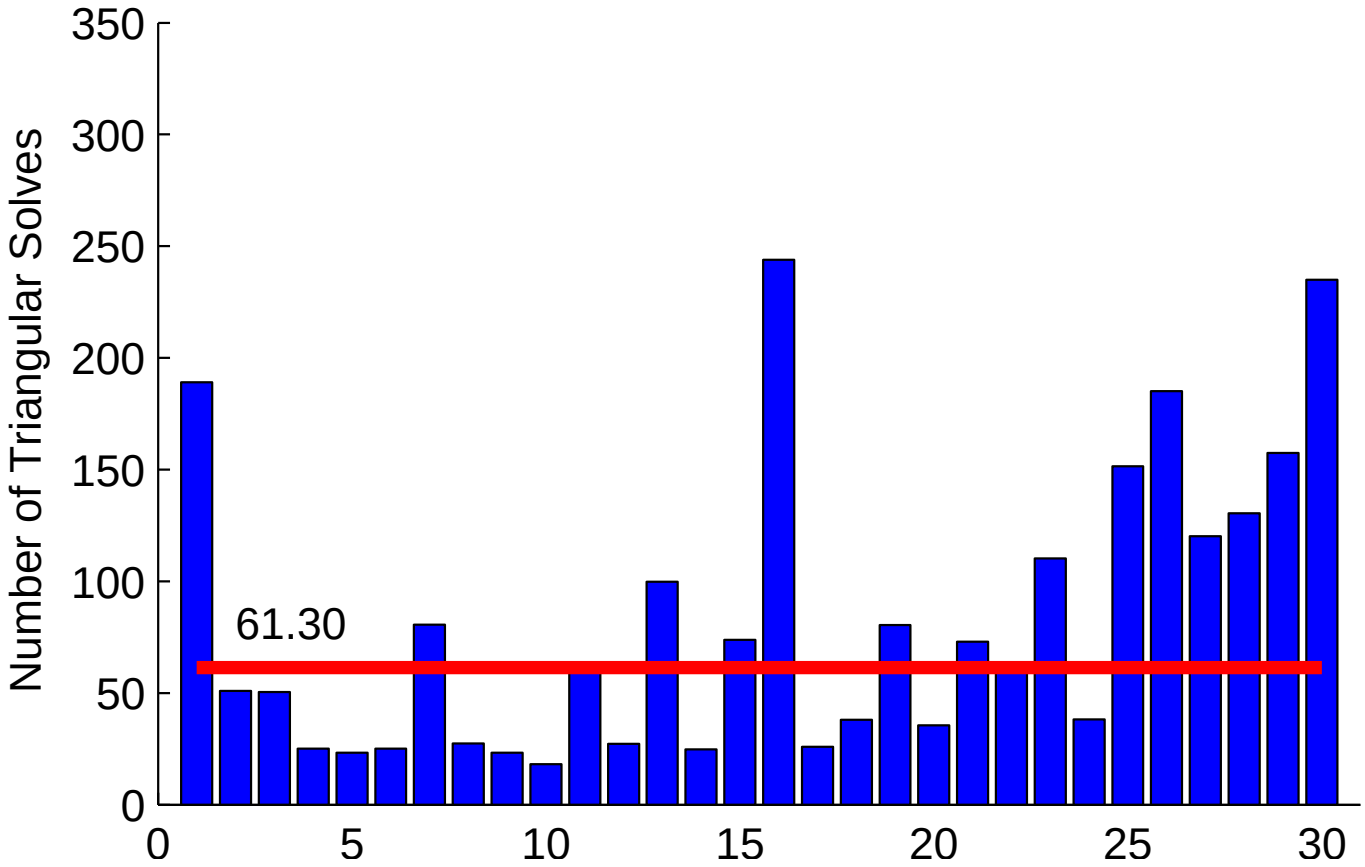
Cost of Preprocessing

- Both methods incur some overhead preprocessing the matrix

PENNSTATE

Cost of Preprocessing

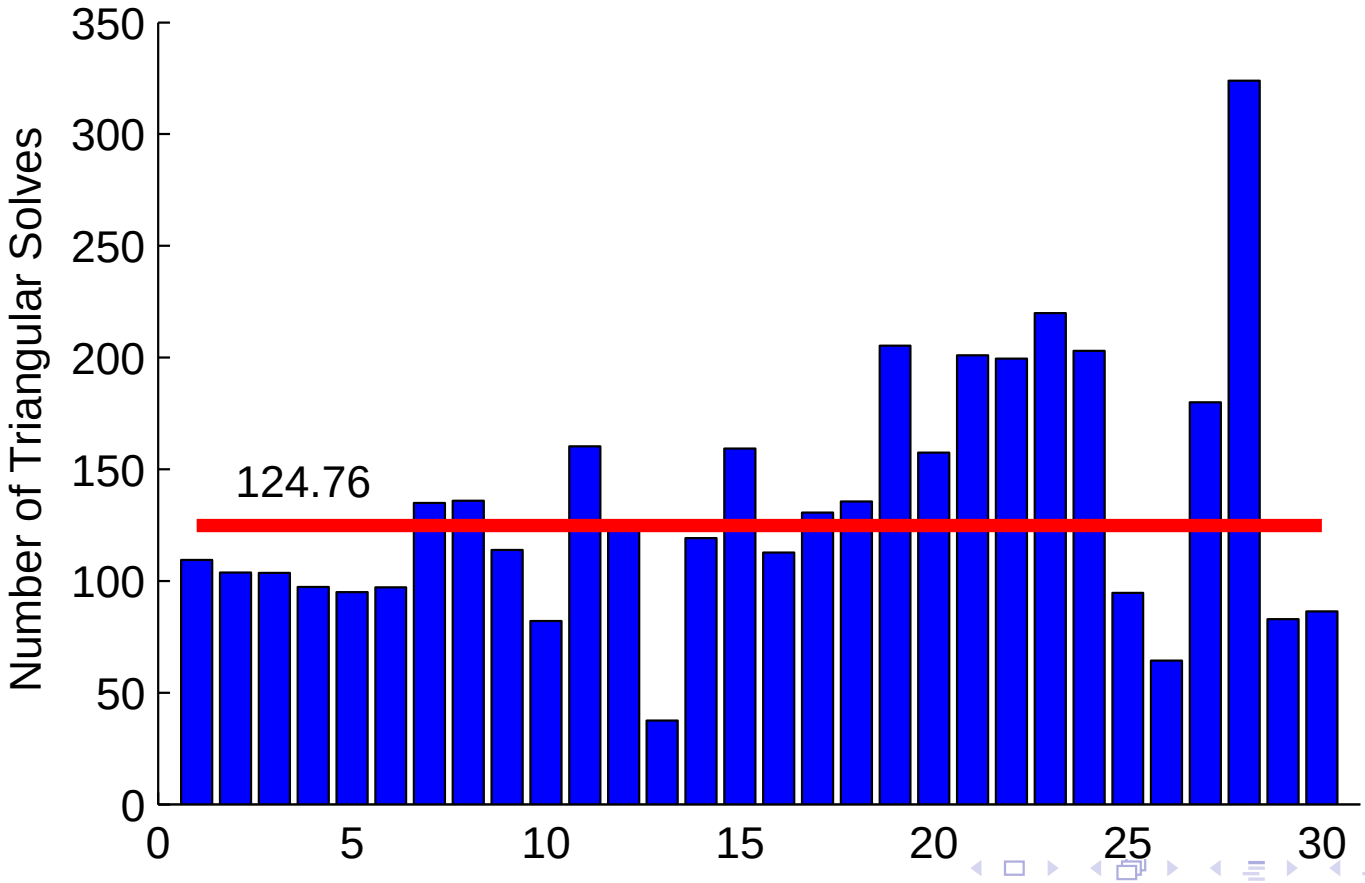
- Both methods incur some overhead preprocessing the matrix
- Cost of LS analysis relative to $T(A, CS)$
  - Geometric mean $61 \times T(A, CS)$



PENNSTATE

Cost of Preprocessing

- Cost of CS analysis relative again to $T(A, CS)$
  - Geometric mean $124 \times T(A, CS)$
  - About twice that of LS
- Overhead amortized over 2 triangular solves per iteration of PCG, many iterations per solve



Summary

- We demonstrated how graph coloring can be used to expose high levels of fine-grained concurrency required for optimal performance on GPU

PENNSTATE

Summary

- We demonstrated how graph coloring can be used to expose high levels of fine-grained concurrency required for optimal performance on GPU
- Analysis of simple 5 point finite difference matrix, representative of larger category of finite difference and finite element matrices, illustrates the scalability of CS over LS

PENNSTATE

Summary

- We demonstrated how graph coloring can be used to expose high levels of fine-grained concurrency required for optimal performance on GPU
- Analysis of simple 5 point finite difference matrix, representative of larger category of finite difference and finite element matrices, illustrates the scalability of CS over LS
- A simple performance model for triangular solve times based on the level of exposed concurrency was proposed

PENNSTATE

Summary

- We demonstrated how graph coloring can be used to expose high levels of fine-grained concurrency required for optimal performance on GPU
- Analysis of simple 5 point finite difference matrix, representative of larger category of finite difference and finite element matrices, illustrates the scalability of CS over LS
- A simple performance model for triangular solve times based on the level of exposed concurrency was proposed
- Experiments verified the performance model as well as the scalability of the CS algorithm over LS

PENNSTATE

Summary

- We demonstrated how graph coloring can be used to expose high levels of fine-grained concurrency required for optimal performance on GPU
- Analysis of simple 5 point finite difference matrix, representative of larger category of finite difference and finite element matrices, illustrates the scalability of CS over LS
- A simple performance model for triangular solve times based on the level of exposed concurrency was proposed
- Experiments verified the performance model as well as the scalability of the CS algorithm over LS
- On a single Tesla M2090, 5.41 mean speedup of CS over LS and 7.23 mean speedup of CPLS over LS

PENNSTATE