ECE408 Spring 2020

Applied Parallel Programming

Lecture 18: Parallel Sparse Methods

Objective

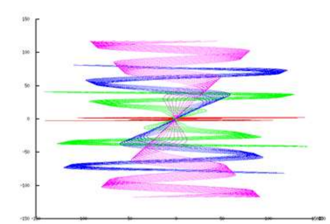
- To learn the key techniques for compacting input data in parallel sparse methods for reduced consumption of memory bandwidth
  - better utilization of on-chip memory
  - fewer bytes transferred to on-chip memory
  - Better utilization of global memory
  - Challenge: retaining regularity

Sparse Matrix

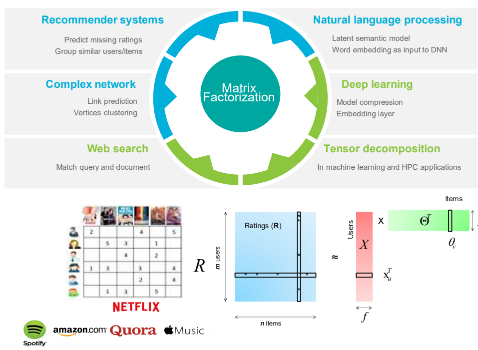
- Many real-world systems are sparse in nature
  - Linear systems described as sparse matrices
- Solving sparse linear systems
  - Traditional inversion algorithms such as Gaussian elimination can create too many “fill-in” elements and explode the size of the matrix
  - Iterative Conjugate Gradient solvers based on sparse matrix-vector multiplication is preferred
- Solution of PDE systems can be formulated into linear operations expressed as sparse matrix-vector multiplication

Sparse Data
Motivation for Compaction

- Many real-world inputs are sparse/non-uniform
- Signal samples, mesh models, transportation networks, communication networks, etc.

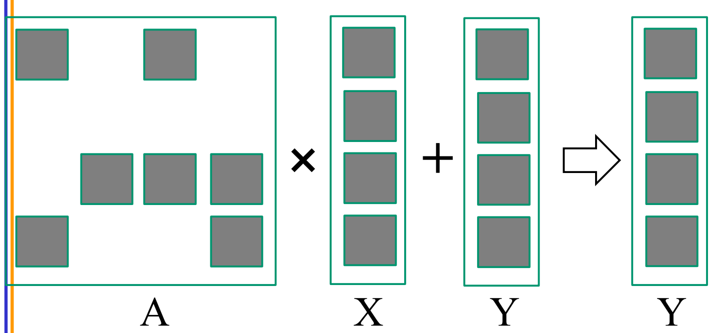


Sparse Matrix in Analytics and AI



Science Area	Number of Teams	Codes	Struct Grids	Unstruct Grids	Dense Matrix	Sparse Matrix	N-Body	Monte Carlo	FFT	PIC	Sig I/O
Climate and Weather	3	CESM, GCRM, CM1/WRF, HOMME	X	X		X		X			X
Plasmas/ Magnetosphere	2	H3D(M),VPIC, OSIRIS, Magtail/UPIC	X				X		X		X
Stellar Atmospheres and Supernovae	5	PPM, MAESTRO, CASTRO, SEDONA, ChaNGa, MS-FLUKSS	X			X	X	X		X	X
Cosmology	2	Enzo, pGADGET	X			X	X				
Combustion/ Turbulence	2	PSDNS, DISTUF	X						X		
General Relativity	2	Cactus, Harm3D, LazEV	X			X					
Molecular Dynamics	4	AMBER, Gromacs, NAMD, LAMMPS				X	X		X		
Quantum Chemistry	2	SIAL, GAMESS, NWChem			X	X	X	X			X
Material Science	3	NEMOS, OMEN, GW, QMCPACK			X	X	X	X			
Earthquakes/ Seismology	2	AWP-ODC, HERCULES, PLSQR, SPECFEM3D	X	X			X				X
Quantum Chromo Dynamics	1	Chroma, MILC, USQCD	X		X	X					
Social Networks	1	EPISIMDEMICS									
Evolution	1	Eve									
Engineering/System of Systems	1	GRIPS,Revisit						X			
Computer Science	1	[illegible]		X	X	X			X		X

Sparse Matrix-Vector Multiplication (SpMV)



Challenges

- Compared to dense matrix multiplication, SpMV
  - Is irregular/unstructured
  - Has little input data reuse
  - Benefits little from compiler transformation tools
- Key to maximal performance
  - Maximize regularity (by reducing divergence and load imbalance)
  - Maximize DRAM burst utilization (layout arrangement)

A Simple Parallel SpMV

Row 0	3	0	1	0	Thread 0
Row 1	0	0	0	0	Thread 1
Row 2	0	2	4	1	Thread 2
Row 3	1	0	0	1	Thread 3

- Each thread processes one row

Compressed Sparse Row (CSR) Format

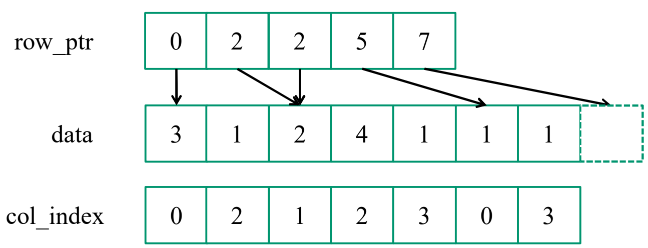
CSR Representation

		Row 0	Row 2	Row 3
Nonzero values	data[7]	{ 3, 1,	2, 4, 1,	1, 1 }
Column indices	col_index[7]	{ 0, 2,	1, 2, 3,	0, 3 }
Row Pointers	ptr[5]	{ 0, 2, 2, 5, 7 }		

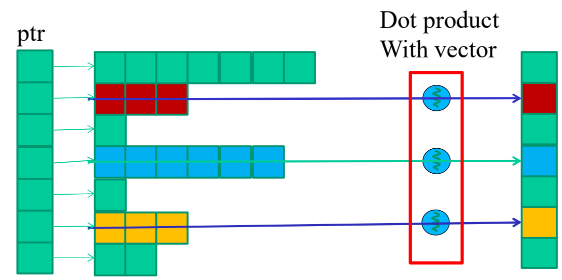
Dense representation

Row 0	3	0	1	0	Thread 0
Row 1	0	0	0	0	Thread 1
Row 2	0	2	4	1	Thread 2
Row 3	1	0	0	1	Thread 3

CSR Data Layout



CSR Kernel Design



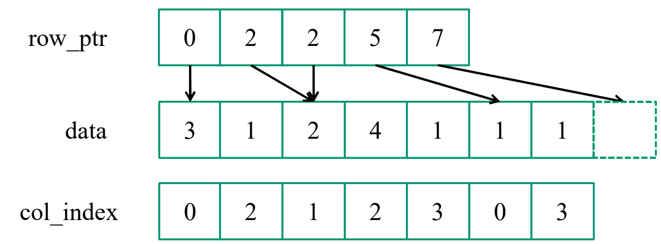
A Parallel SpMV/CSR Kernel (CUDA)

```
__global__ void SpMV_CSR(int num_rows, float *data,
    int *col_index, int *row_ptr, float *x, float *y) {
     int row = blockIdx.x * blockDim.x + threadIdx.x;
     if (row < num_rows) {
       float dot = 0;
       int row_start = row_ptr[row];
       int row_end =   row_ptr[row+1];
       for (int elem = row_start; elem < row_end; elem++) {
         dot += data[elem] * x[col_index[elem]];
       }
       y[row] = dot;
     }
   }
```

		Row 0	Row 2	Row 3
Nonzero values	data[7]	{ 3, 1,	2, 4, 1,	1, 1 }
Column indices	col_index[7]	{ 0, 2,	1, 2, 3,	0, 3 }
Row Pointers	row_ptr[5]	{ 0, 2, 2, 5, 7 }		

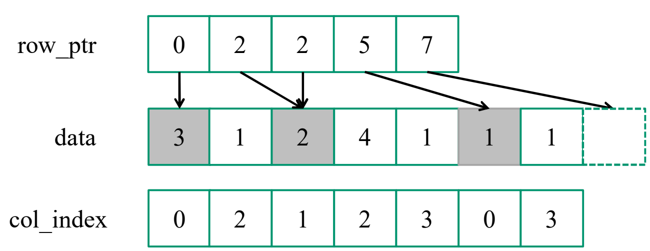
CSR Kernel Control Divergence

- Threads execute different number of iterations in the kernel for-loop

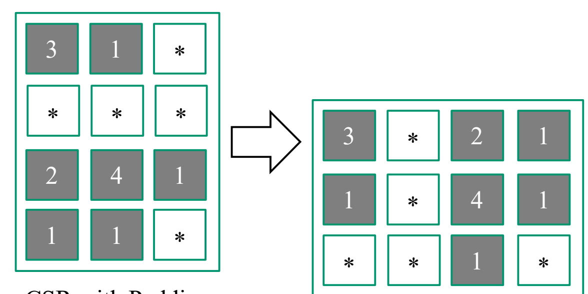


CSR Kernel Memory Divergence (Uncoalesced Accesses)

- Adjacent threads access non-adjacent memory locations
  - Grey elements are accessed by all threads in iteration 0



Regularizing SpMV with ELL(PACK) Format

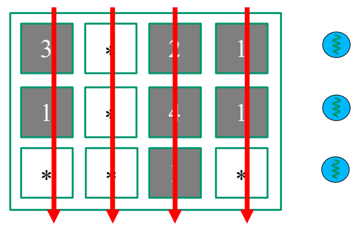


CSR with Padding

Transposed

- Pad all rows to the same length
  - Inefficient if a few rows are much longer than others
- Transpose (Column Major) for DRAM efficiency
- Both data and col_index padded/transposed

ELL Kernel Design

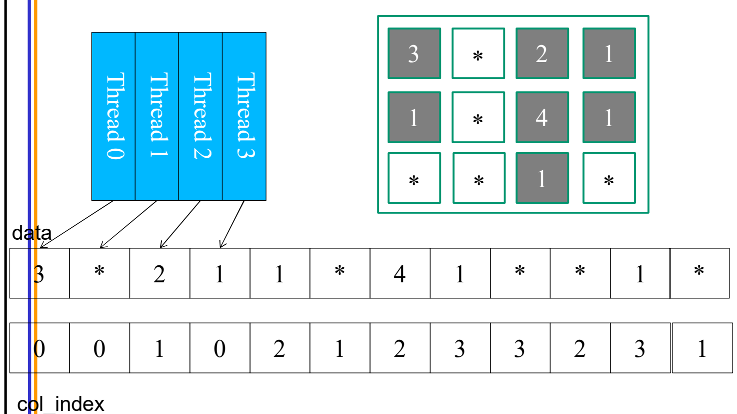


A parallel SpMV/ELL kernel

```
__global__ void SpMV_ELL(int num_rows, float *data,
    int *col_index, int num_elem, float *x, float *y) {

  int row = blockIdx.x * blockDim.x + threadIdx.x;
  if (row < num_rows) {
    float dot = 0;
    for (int i = 0; i < num_elem; i++) {
      dot += data[row+i*num_rows]*x[col_index[row+i*num_rows]];
    }
    y[row] = dot;
  }
}
```


Memory Coalescing with ELL



Coordinate (COO) format

- Explicitly list the column and row indices for every non-zero element

		Row 0	Row 2	Row 3
Nonzero values	data[7]	{ 3, 1,	2, 4, 1,	1, 1 }
Column indices	col_index[7]	{ 0, 2,	1, 2, 3,	0, 3 }
Row indices	row_index[7]	{ 0, 0,	2, 2, 2,	3, 3 }

COO Allows Reordering of Elements

		Row 0	Row 2	Row 3
Nonzero values	data[7]	{ 3, 1,	2, 4, 1,	1, 1 }
Column indices	col_index[7]	{ 0, 2,	1, 2, 3,	0, 3 }
Row indices	row_index[7]	{ 0, 0,	2, 2, 2,	3, 3 }

Nonzero values	data[7]	{ 1 1, 2, 4, 3, 1 1 }
Column indices	col_index[7]	{ 0 2, 1, 2, 0, 3, 3 }
Row indices	row_index[7]	{ 3 0, 2, 2, 0, 2, 3 }


```
for (int i = 0; i < num_elem; row++)
  y[row_index[i]] += data[i] * x[col_index[i]];
```

a sequential loop that implements SpMV/COO

READ CHAPTER 10