A Message-Driven, Multi-GPU Parallel Sparse Triangular Solver

Nan Ding, Samuel Williams, Yang Liu, Xiaoye S. Li
nanding@lbl.gov
May 03, 2021

U.S. DEPARTMENT OF ENERGY

UNIVERSITY OF CALIFORNIA

BERKELEY LAB
Lawrence Berkeley National Laboratory

Sparse Direct Solvers

- Sparse direct solvers are used to solve challenging systems:
  - LU factorization (SpLU)
  - L- and U-solve (SpTRSV)

- SpLU factorization can be extremely expensive:
  - Memory hungry
  - Computationally expensive

- Solution:
  - Factor once and use as a preconditioner across multiple solves
    i.e., multiple GMRES solves call SpTRSV on every iteration all using the same factors
  - Shifts the focus to SpTRSV performance

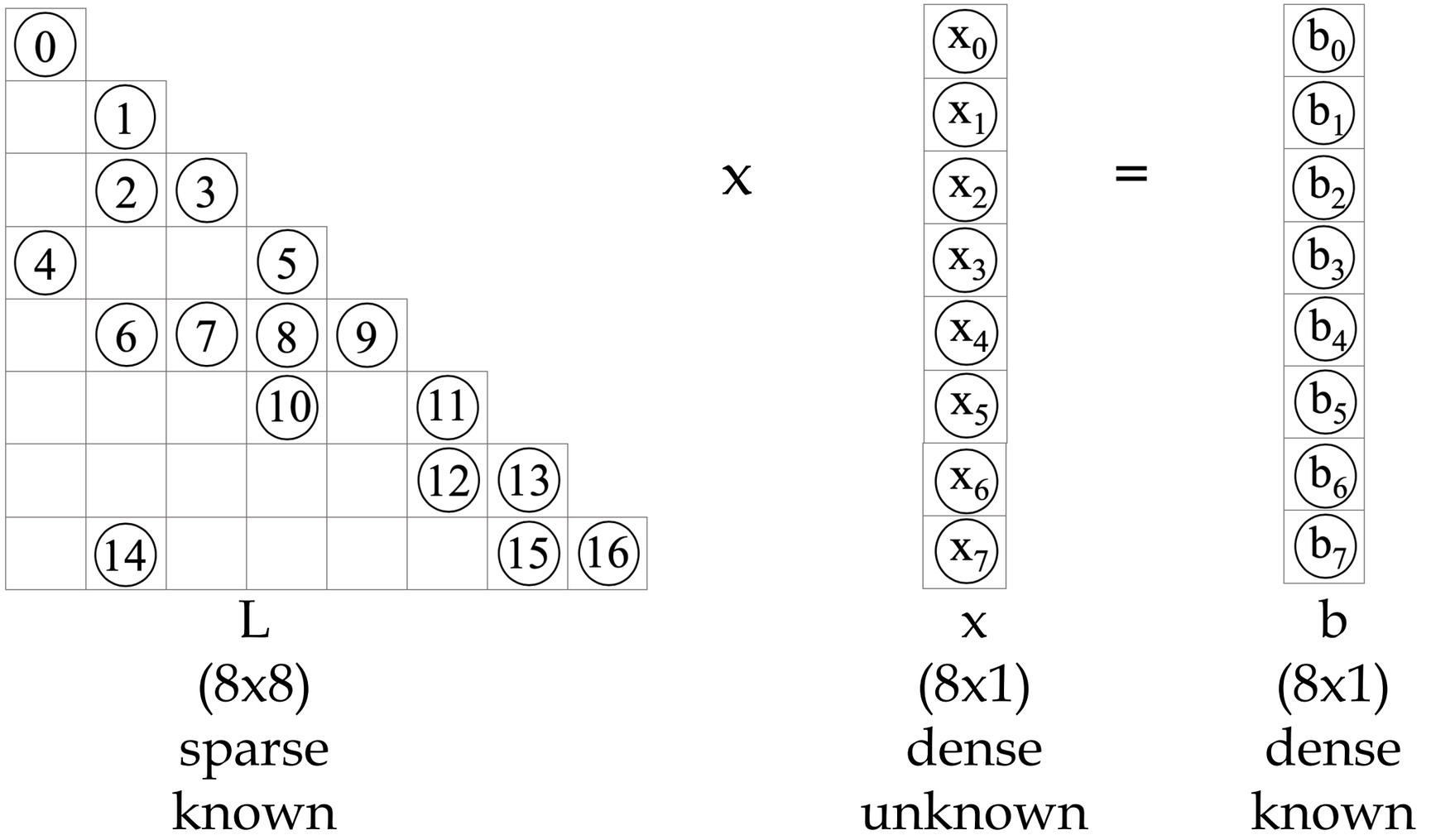
multi-GPU SpTRSV is imperative but challenging

- SpTRSV is an important kernel for a variety numerical simulations
- GPUs have become a first-class compute citizen
  - 110/147 system use NVIDIA Volta chips in 2020, Top500 list[1]
- Demand for ever finer-resolution problems calls for SpTRSV to exploit larger scales of parallelism
  - the slow pace in HBM memory capacity scaling
  - Can not always fit into a single GPU's memory

[1] https://www.top500.org/

Sparse Triangular Solver (SpTRSV)

- Compute solution vector x from a sparse linear system, Lx=b

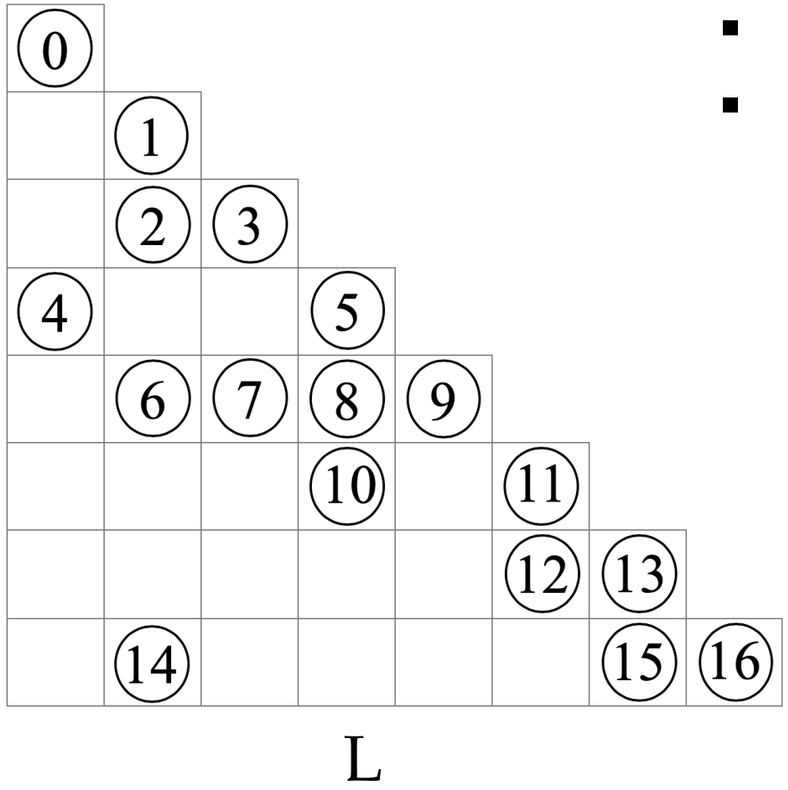


L
(8x8)
sparse
known

x
(8x1)
dense
unknown

b
(8x1)
dense
known

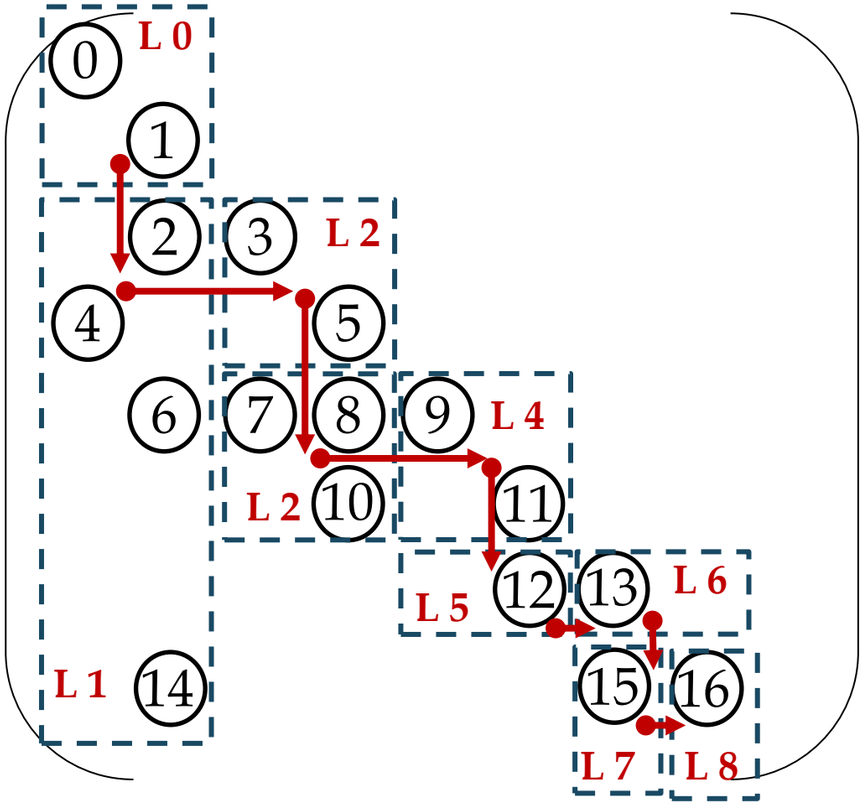
Naïve BSP SpTRSV

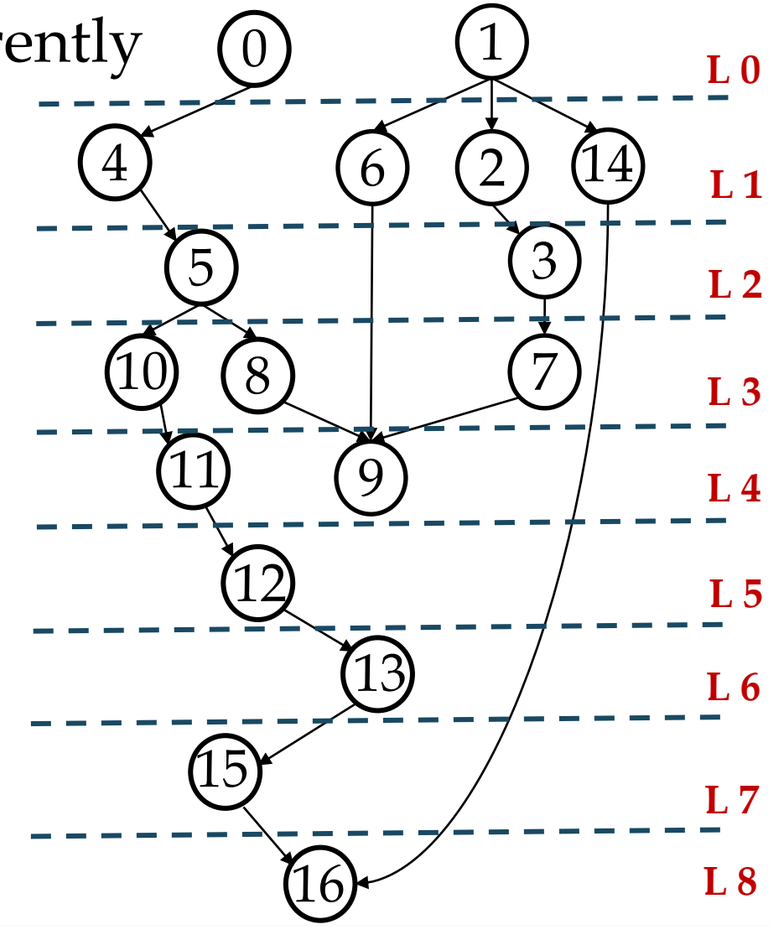


- Naive approach:
  - solve the system one equation (row) at a time
  - can be optimized to (selectively) parallelize over column updates or row reductions

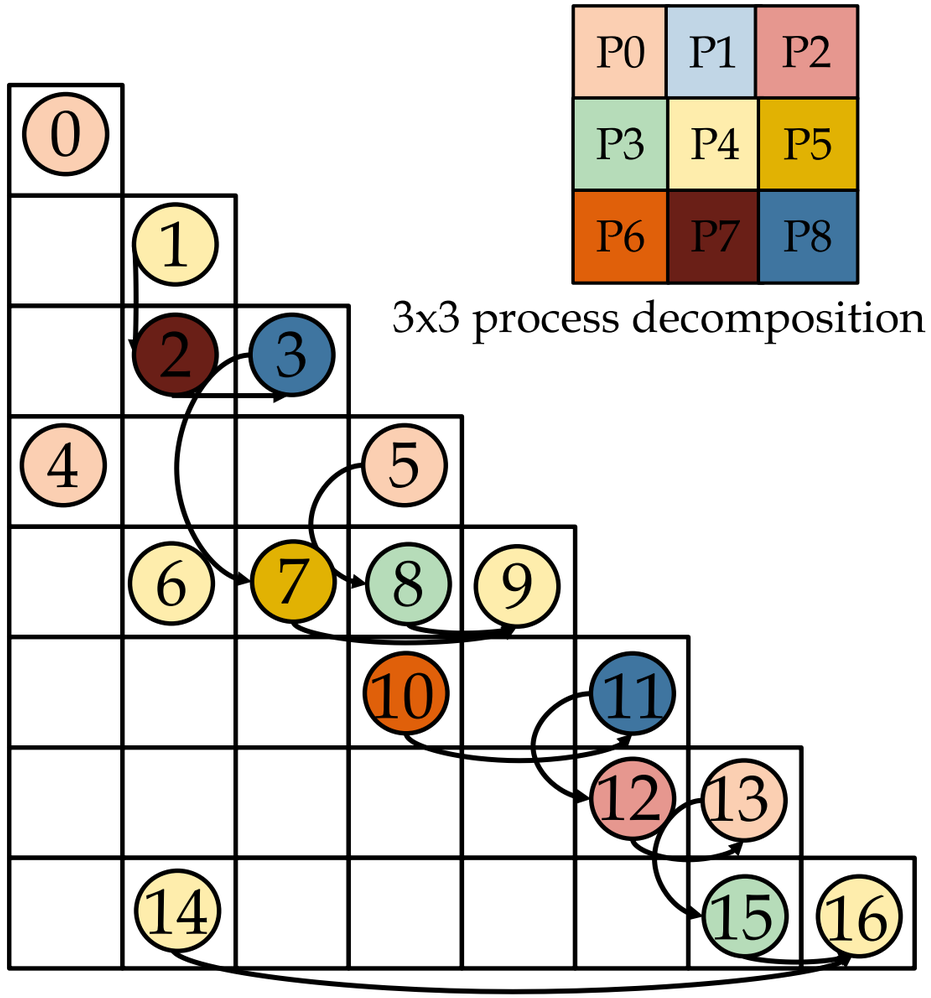
Recast SpTRSV as a DAG

- Computation = Directed Acyclic Graph
- Each node in the DAG is a small dense matrix-vector
- Use level sets
- Blocks in the same level can be solved concurrently





SpTRSV in SuperLU: Message Driven



- SuperLU imposes a 2D block cyclic process layout
- Two types of computation:
  - Solves (on-diagonal blocks)
  - MatVec (off-diagonal blocks)
- Two types of communication:
  - Block column broadcast
  - Block row reduction

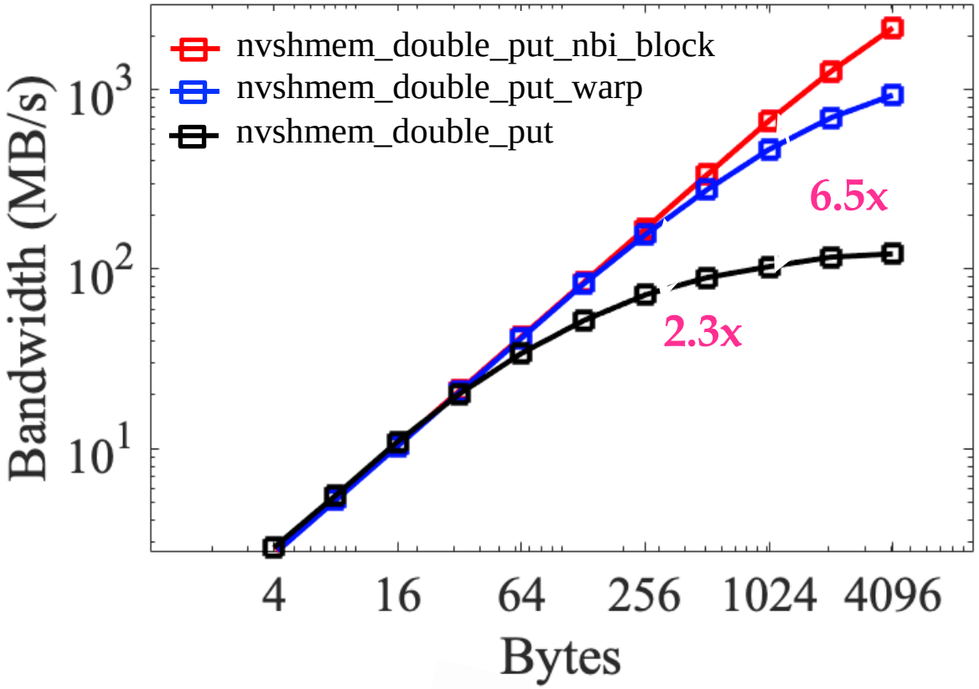
NVSHMEM has potential
(but bad implementations can destroy it)

- NVSHMEM is a parallel programming interface based on OPENSHMEM
  - one-sided communication for NVIDIA GPU clusters

✓ **uses GPU-initiated data transfers**

✓ **provides signaling operations and point-to-point synchronization operations to notify receivers**

✗ **point-to-point synchronization operations limits the number of thread blocks that can be concurrently scheduled on one V100 GPU to 80 (the number of SMs) to avoid potential deadlocks**

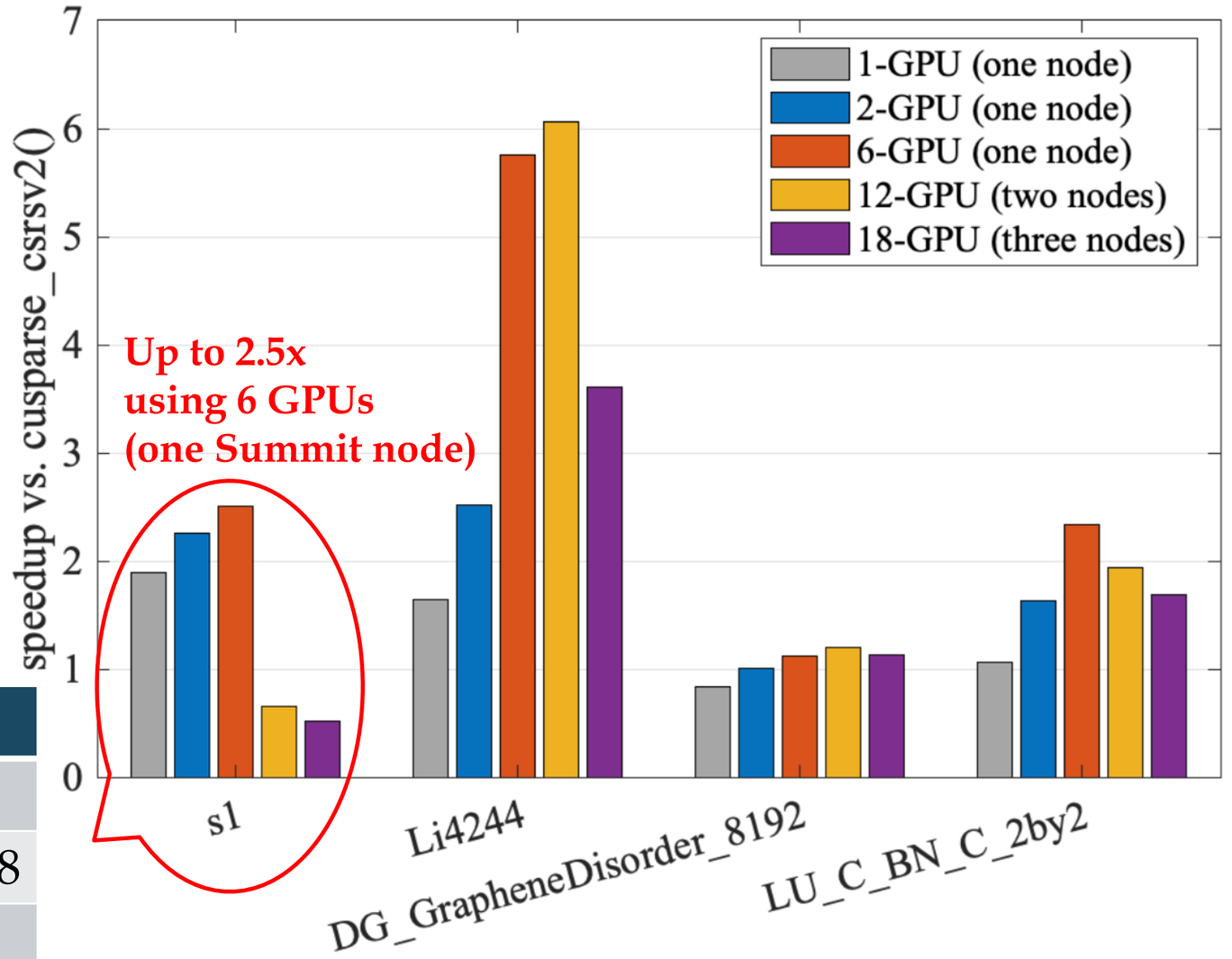


Multi-GPU SpTRSV vs. cusparse_csrsv2()

Experimented on Summit:

- Cuda 10
- Nvshmem 1.1.3
- Grdcopy 2.0

- bind one process to one GPU
- Px1 process layout (column broadcast)
- use nvshmem_double_put_nbi_block()



M3DC1 matrix	
s1_mat_0_507744	
nnz in L	8.80E+08
Nsupers	9827

Multi-GPU SpTRSV using two CUDA streams

WAIT (for notifications), stream[0]

thread block 0 (column broadcast)	thread block 1 (row reduction)

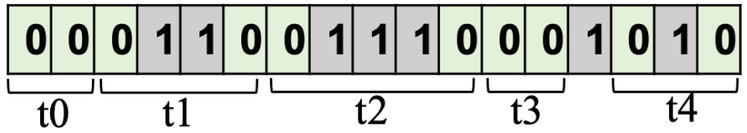
for each thread *t*:

While expecting more tasks: *(recorded in the pre-processing pharse)*

idx=nvshmem_int_wait_until_any(**flag_x**, mask of the buffer)

M_c[***idx***]=1

0: active in wait, 1: masked off



additional work except waiting:

local GEMV

fmod(I)--

if **fmod(I)**==0 NVSHMEM send (by thread) *lsum*

NVSHMEM send:

nvshmemx_double_put_nbi_block()

nvshmem_fence()

nvshmemx_int_signal()

for each thread block *K*: (SOLVE)

if I am the diagonal process in charge of K:

spin wait until the diagonal block is unlocked **fmod(I)**==0

local TRSV

else: (I am off-diagonal process)

While **flag_x**[***idx***] !=1;

NVHSMEM send *x*

local GEMV

fmod(I)--

if **fmod(I)**==0 NVSHMEM send (by thread) lsum

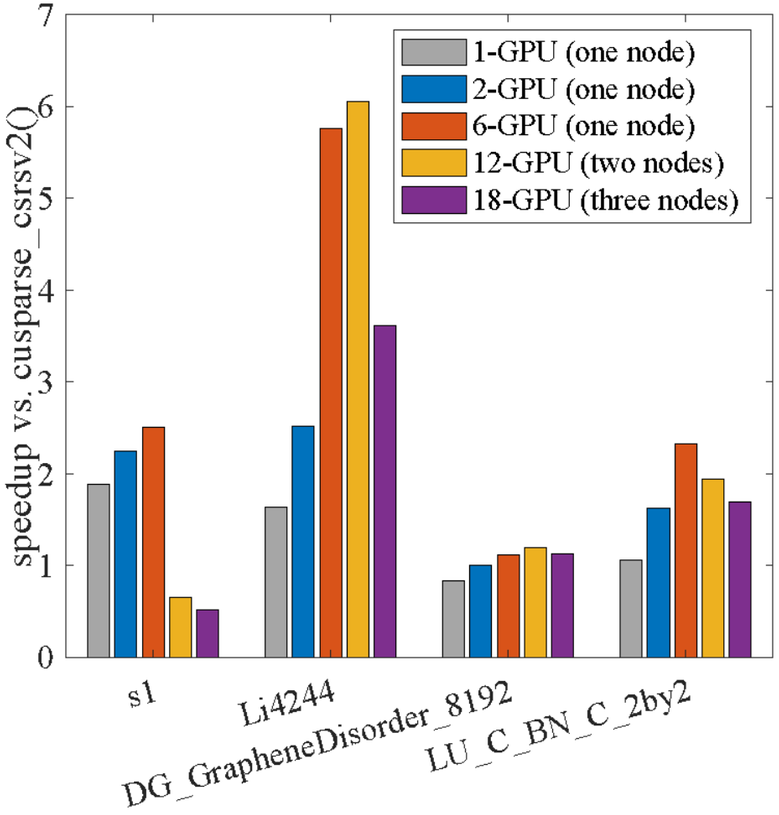
wait for WAIT successfully launched

SOLVE (GEMV/TRSV, send data and notification), stream[1]

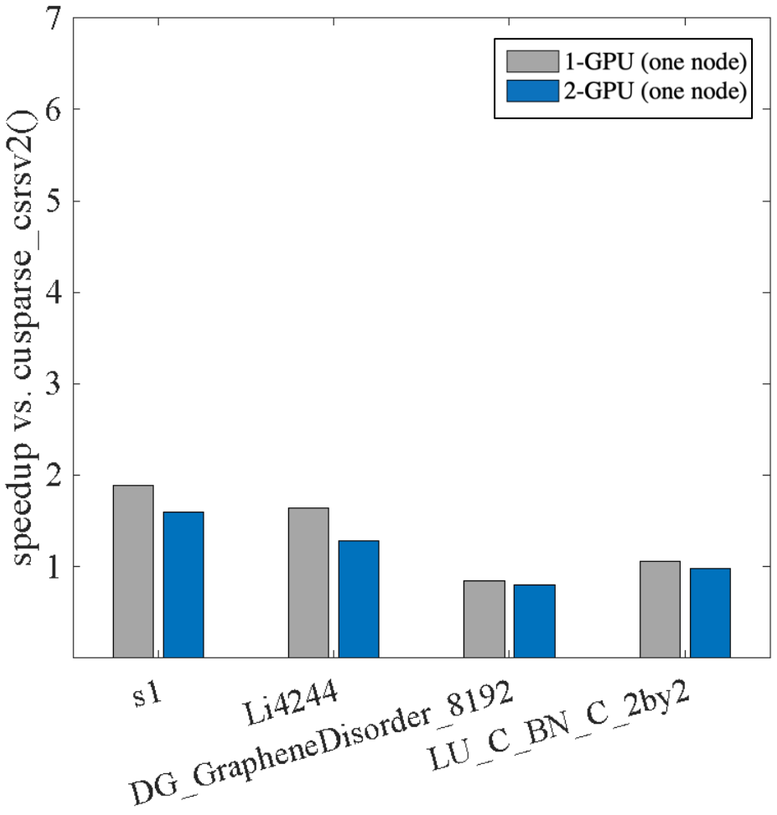
Number of local super nodes in that GPU

Multi-GPU SpTRSV performance on Summit GPUs
using different process layouts

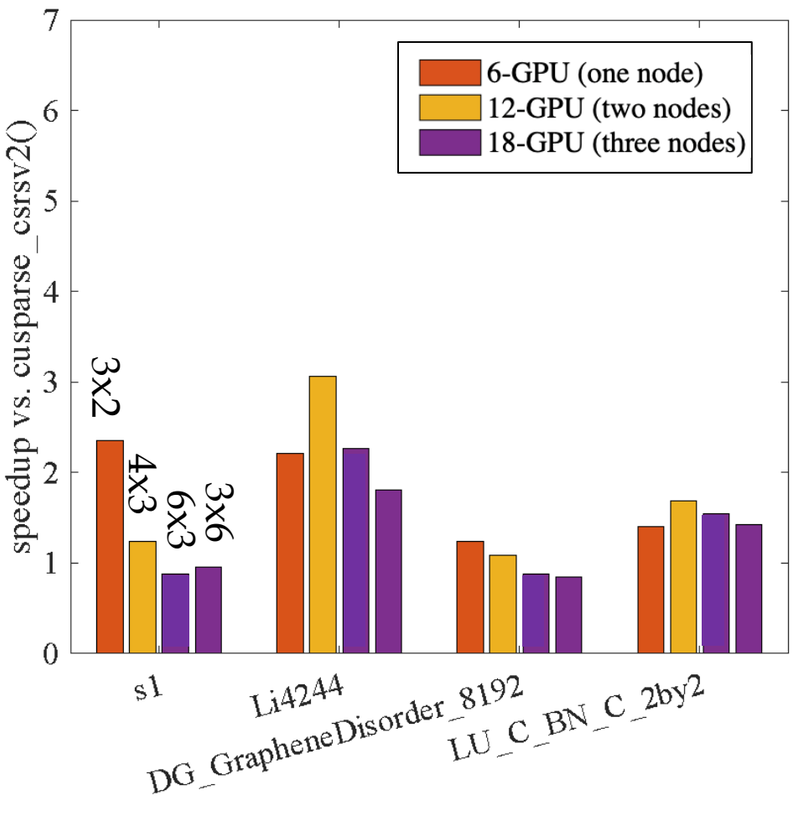
Px1 process layout (column broadcast)
- NVSHMEM send using thread blocks



1xP process layout (row reduction)
- NVSHMEM send using threads



2D process layout
- mixed



It's important to understand what constraint the performance

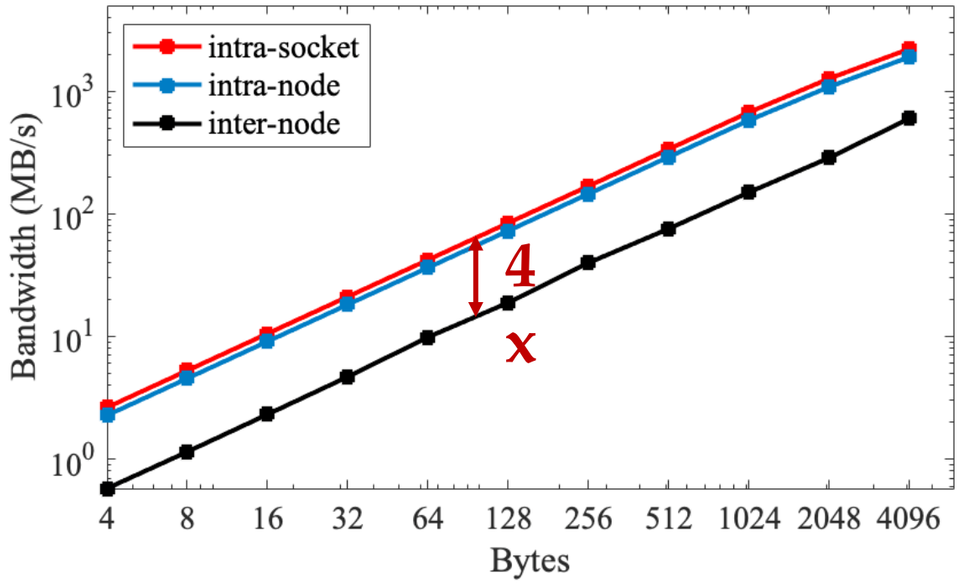
- Some numerical methods lend themselves to simple performance analysis
- DAG-based SpTRSV demands more sophistication
- Solution:
  - construct a critical path performance model
  - assess our observed performance relative to machine capabilities.

Critical path Performance model

- **Architecture Characterization**
  - Memory bandwidth

$$\frac{828\ GB/s}{80\ SM\ *64\ warps/8\ warps} = 1.3GB/s \leq bw = \frac{828\ GB/s}{local\ supernodes} \leq 5.2\ GB/s = \frac{828\ GB/s}{80\ SM\ *2\ warps}$$

  - NVSHMEM bandwidth (optimized)
    - NVSHMEM Performance differs with GPU affinities
    - Intra-socket is 4x faster than inter-node



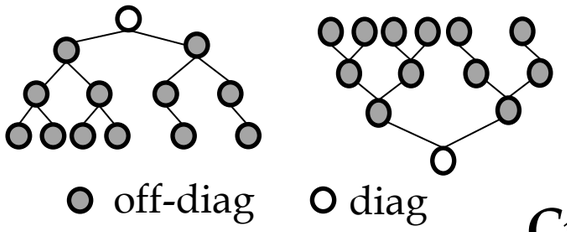
Critical path Performance model

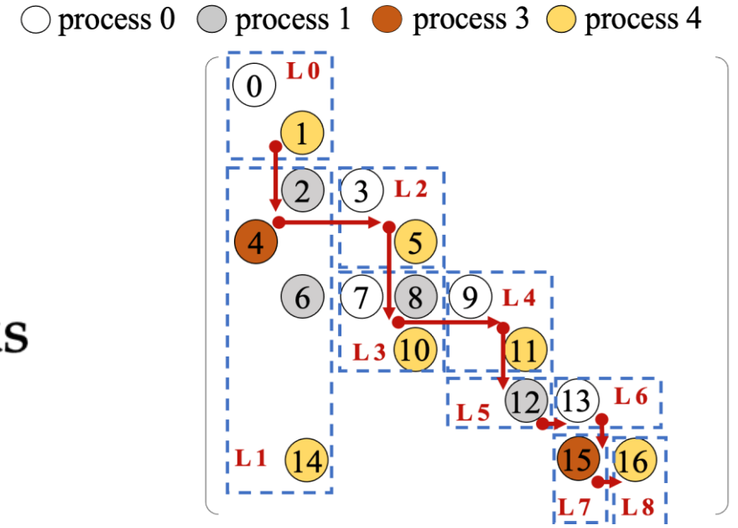
- **SpTRSV Characterization**
  - Initial Critical path: based on level-set using BFS
  - Refined Critical path: process decomposition
    - Memory bandwidth scales with the number of blocks (GEMV/TRSV) in the same level until the aggregate bandwidth reach the peak:

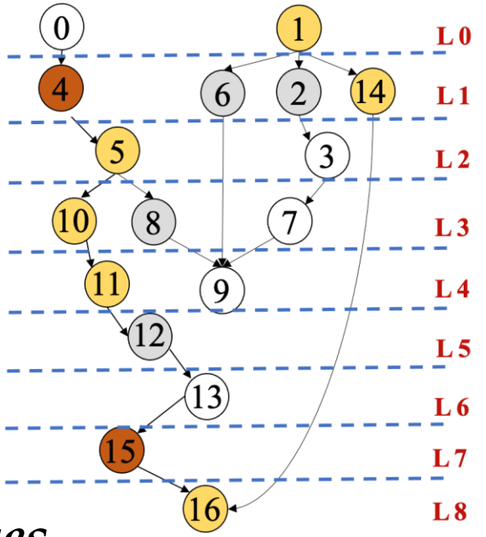
$$T_{mat-vec\ per\ gpu} = \frac{accumulated\ Bytes}{aggrated\ bw}$$

    - Communication:

$$T_{comm\ per\ gpu} = \sum_{\text{levels}} \left(\mathrm{L_{net}} + \frac{\mathrm{log2(\#out)} * \mathrm{sz}}{\mathrm{BW_{net}}} \right) + \sum_{levels} \left(log2(\#in) * (L_{net} + \frac{sz}{BW_{net}}) \right)$$



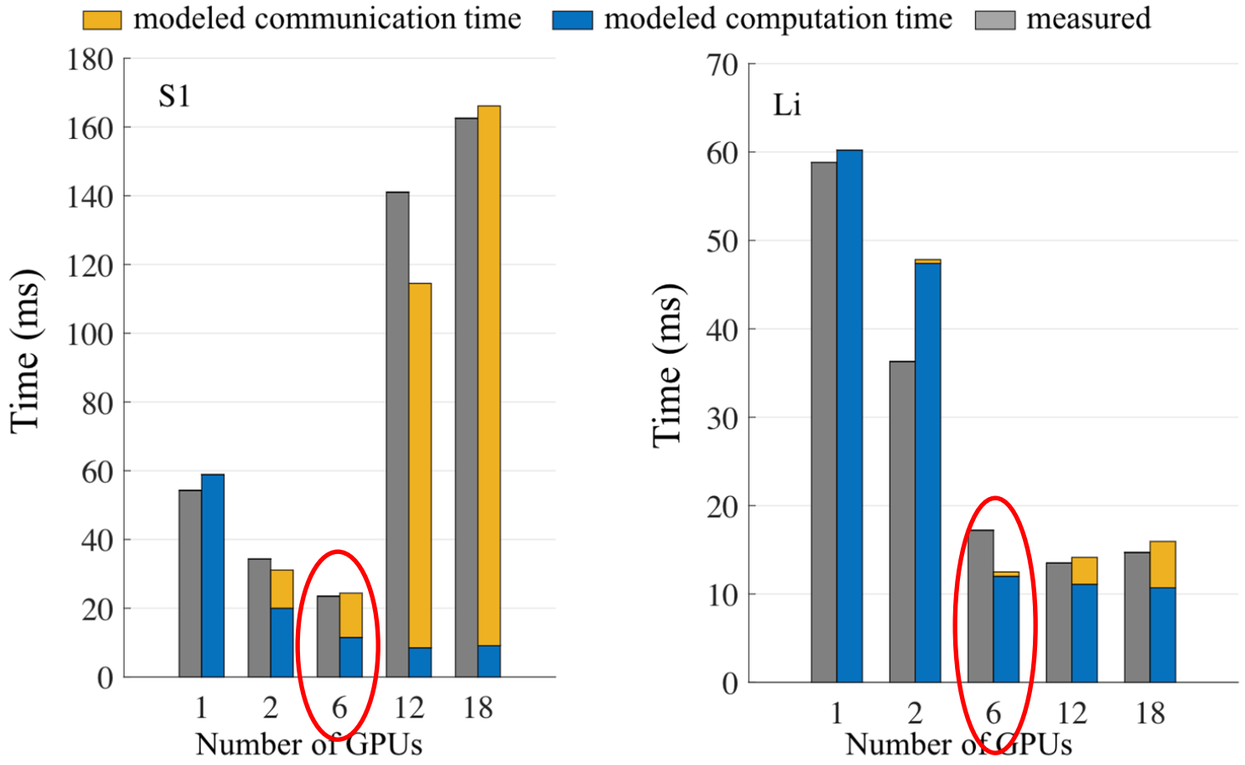




Critical path could become longer due to limited resources

SpTRSV performance differs with critical paths

Matrix	#supernodes	DAG levels	nnz L	Compare to our single GPU solution			
				One Summit node		2 Summit nodes	3 Summit nodes
				2 GPU	6GPU	12 GPU	18 GPU
s1_mat_0_507744	9,827	388	8.80E+08	1.2x	1.3x	0.7x	0.8x
Li4244	362	188	5.18E+08	1.5x	3.5x	3.7x	2.7x



6 GPU

S1:

- 7,922 messages on the critical path
- 1.3 GB/s memory bandwidth

Li:

- 270 messages on the critical path
- 5.2 GB/s memory bandwidth

Summary

- **Multi-GPU SpTRSV using CUDA streams**
  - Bring experience for DAG-based computations on emerging accelerated architectures, e.g., GPU-GPU communication using nvshmem
  - Core specialization on GPUs for DAG-based computations: more complex producer-consumer relationship than stencils ---- the producer (sender) and the consumer (receiver) can swap roles in turn to dispatch new work (message).

- **DAG performance analysis can be extremely challenging**
  - Extend our SpTRSV performance model for GPUs
  - Accounts for matrix sparsity, process layout, network/memory bandwidth/latency
  - Enables performance analysis for DAG-based computations
  - Helps understand performance bottlenecks

Acknowledgements

This research is supported in part by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) programs under Contract No. DE-AC02-05CH11231 at Lawrence Berkeley National Laboratory.

U.S. DEPARTMENT OF ENERGY

UNIVERSITY OF CALIFORNIA

Questions


U.S. DEPARTMENT OF
ENERGY
UNIVERSITY OF
CALIFORNIA
BERKELEY LAB
Lawrence Berkeley National Laboratory