Pacific Northwest
NATIONAL LABORATORY

Fast and Scalable Sparse Triangular Solver For Multi-GPU Based HPC Architectures

Chenhao Xie[1], Jieyang Chen[2], Jesun S Firoz[1],

Jiajia Li[1,3], Shuaiwen Leon Song[4],

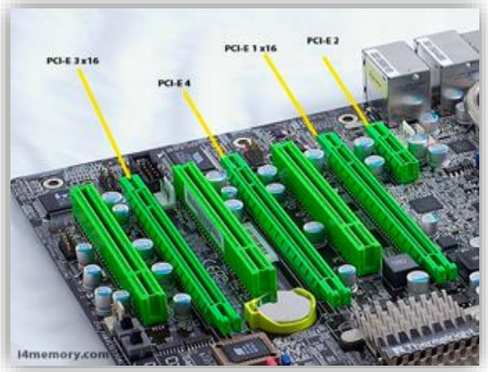
Kevin Barker[1], Mark Raugas[1], Ang Li[1]

[1]Pacific Northwest National Laboratory
[2]Oak Ridge National Laboratory
[3]William & Mary
[4]University of Sydney

U.S. DEPARTMENT OF ENERGY BATTELLE

PNNL is operated by Battelle for the U.S. Department of Energy

Multi-GPU Interconnected & Platforms



PCI-e

NVLink V1&V2

SLI Bridge

NVSwitch

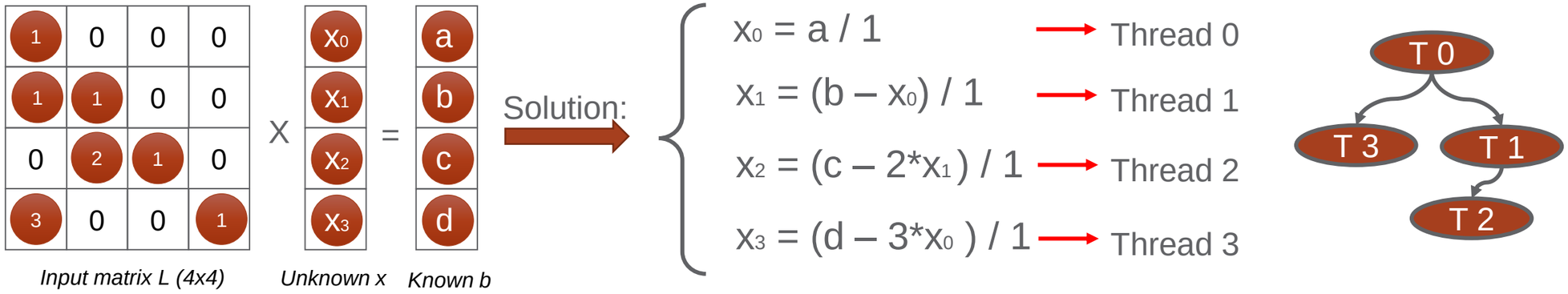
Platform	Configuration	Interconnect
P100-DGX-1	Single node, 8 GPUs	NVLink-V1
V100-DGX-1	Single node, 8 GPUs	NVLink-V2
DGX-2	Single node, 16 GPUs	NVSwitch
SLI	Single node, 2 GPUs	NVLink-SLI
SummitDev	54 nodes, 4 GPUs/node	NVLink-V1
Summit	4600 nodes, 6 GPUs/node	NVLink-V2

IB-GPUDirect-RDMA

Why Parallelizing SpTRSV Is Not Trivial

Sparse Triangle Solver: Lx = b or Ux = b

- Compute a dense solution vector *x* from the sparse linear system, where *L(U)* is a square lower(upper) triangular sparse matrix, and *b* is a dense right-hand side.

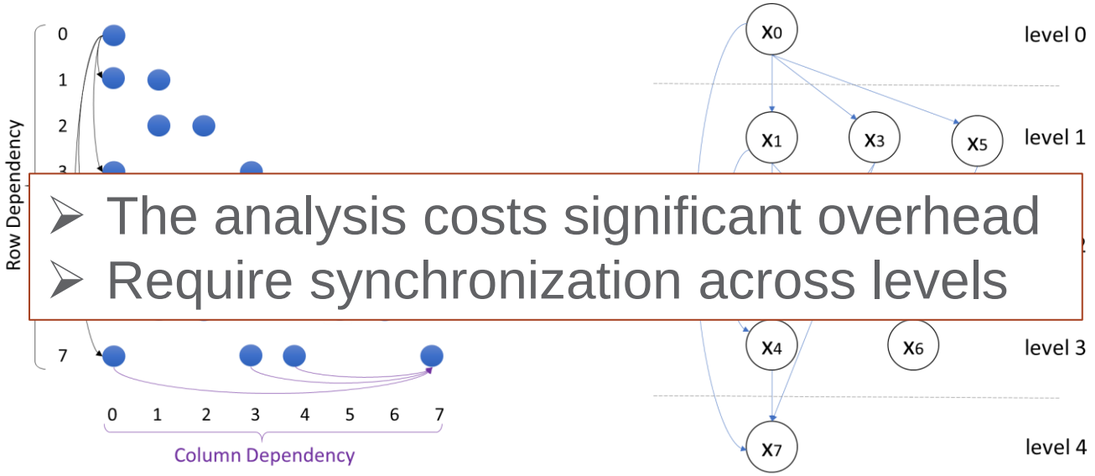


Input matrix L (4x4) *Unknown x* *Known b*

- SpTrsv is inherently sequential. In this case, Component 2 has to wait for Component 1 to finish, and Component 3 has to wait for Component 0, meaning that the all four cores cannot work in parallel.

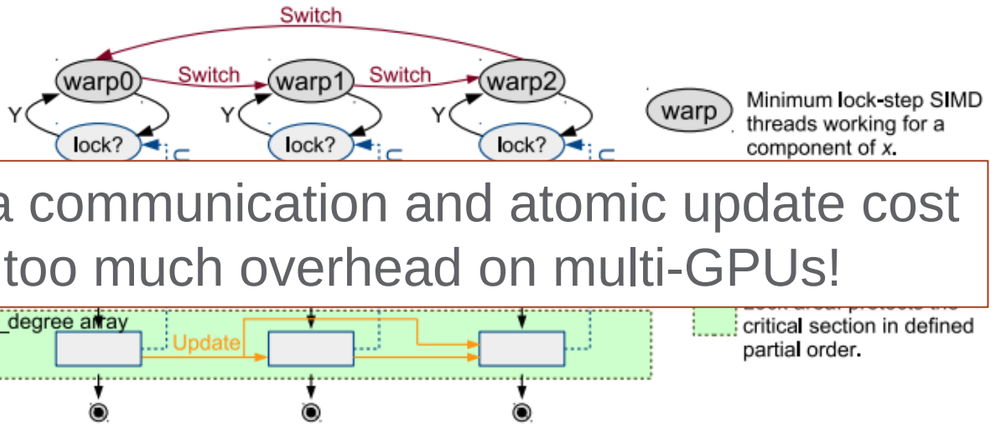
The SOTA SpTrsv on GPUs

- cuSparse lib: ***csrsv2()***
  - Basic idea: some components are independent and can be processed simultaneously.



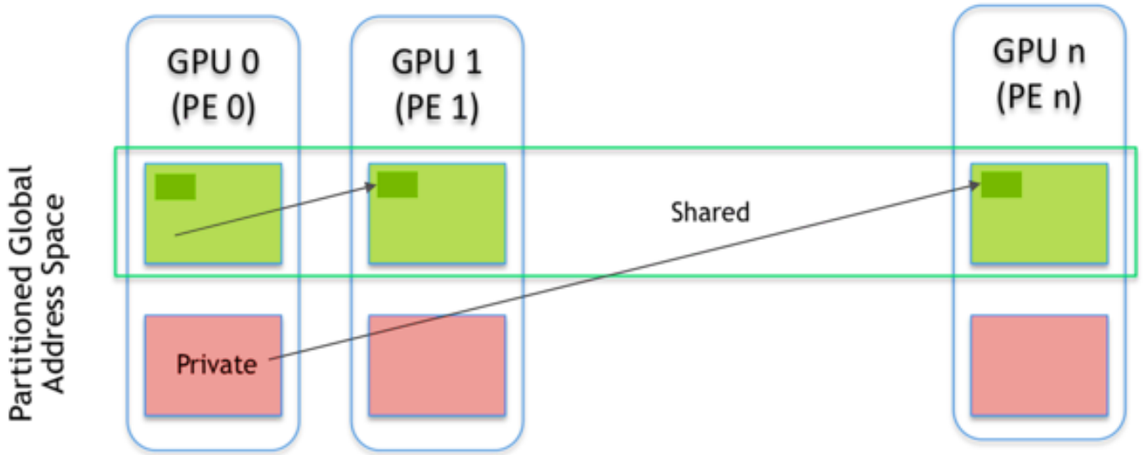
- The analysis costs significant overhead
- Require synchronization across levels

- A Synchronization-Free SpTrsv
  - Basic idea: Migrating the level-sets analysis at runtime
  - Components are scheduled by the hardware warp-switch of the GPU
  - Update the intermediate value (in_degree and left_sum) using GPU atomic operations



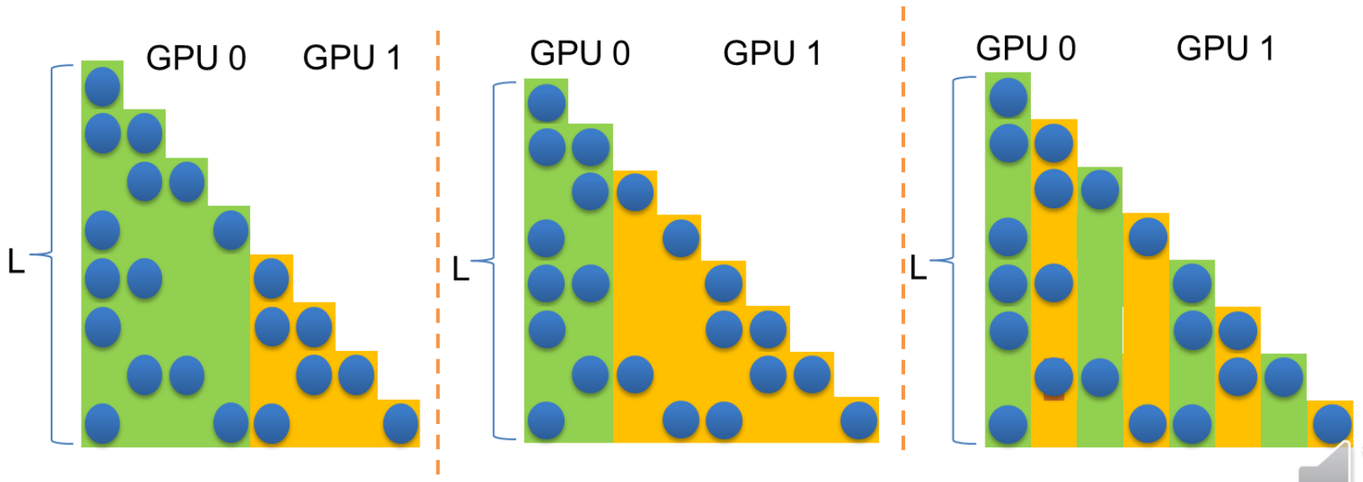
Data communication and atomic update cost too much overhead on multi-GPUs!

Zero-Copy SpTrsv for Multi-GPUs



- Efficiency and asynchronously data communication using **NVSHMEM**

- **Task-based** workload balance approaches to ensures that the parallel components are scheduled on different GPUs



Pacific Northwest
NATIONAL LABORATORY

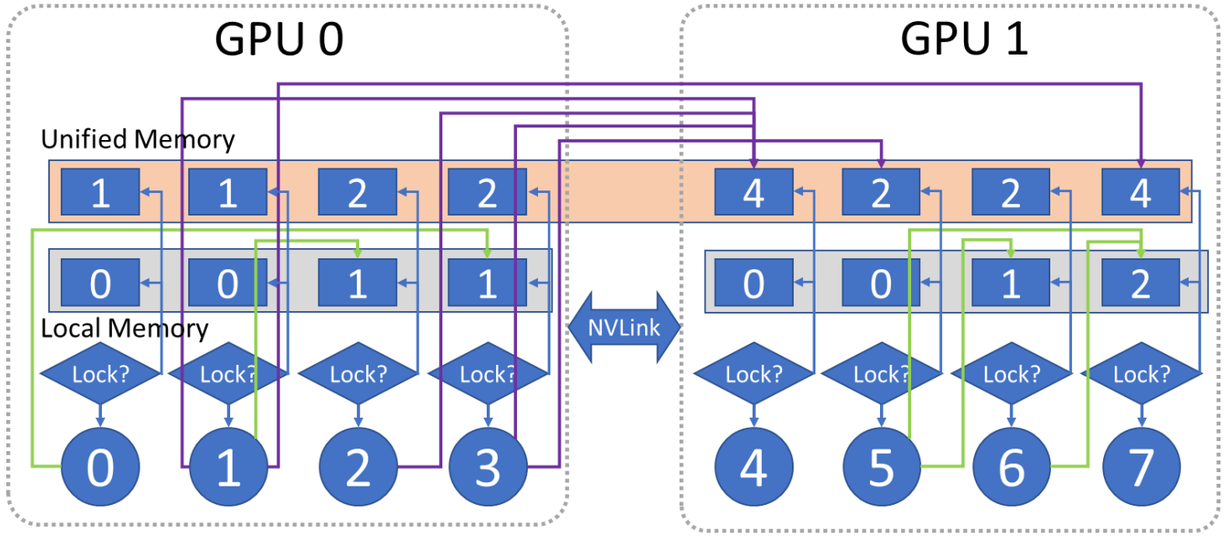
Agenda

- Motivation and Introduction
- Fast and Scalable Sparse Triangular Solver For Multi-GPU Based HPC Architectures
  - Limitation of Unified Memory Solution of SpTrsv
  - Read-Only Inter-GPU Communication with NVSHMEM
  - Fine Turning Workload Dependency
- Evaluation and Results
- Conclusion

Pacific Northwest NATIONAL LABORATORY

SpTrsv with Unified Memory

- Convenient
  - Hide complexity from users
- Page fault mechanism
  - Coarse-grained data copy
- Data contentions
  - System-wide atomic update will access the data simultaneously
- Workload unbalance
  - Dependency are unidirectional

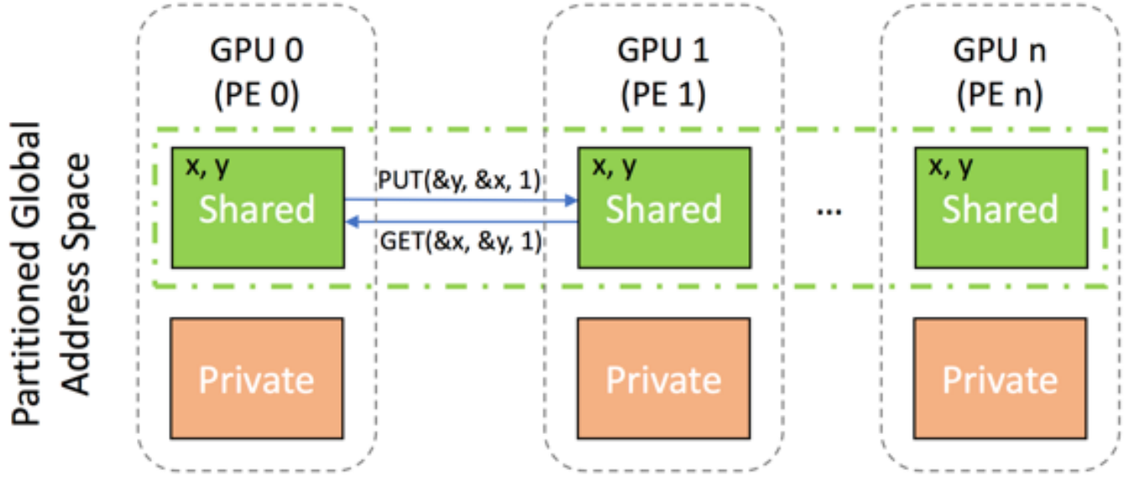


Require Low Overhead Fine-Grained Communications!!

Pacific Northwest NATIONAL LABORATORY

GPU CUDA NVSHMEM

- OpenSHMEM-based PGAS programming interface for multi-GPUs
- GPU-side interface allows GPU threads to
  1. Access distributed memory via data movement API
  2. Direct load/store (LD/ST) where GPUs are P2P-accessible
  3. **Highly-concurrent fine-grained messaging**
  4. **Asynchronously one-side data communication**

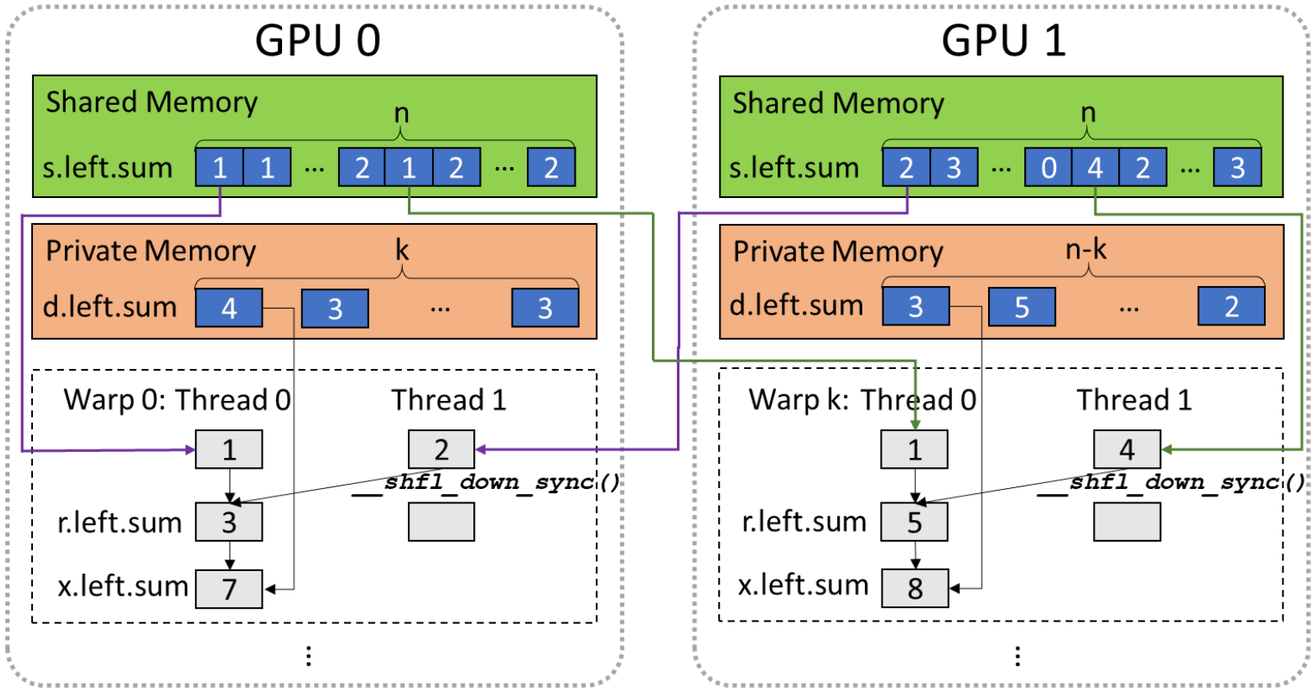


Peer Direct LD/ST and Global Remote Access

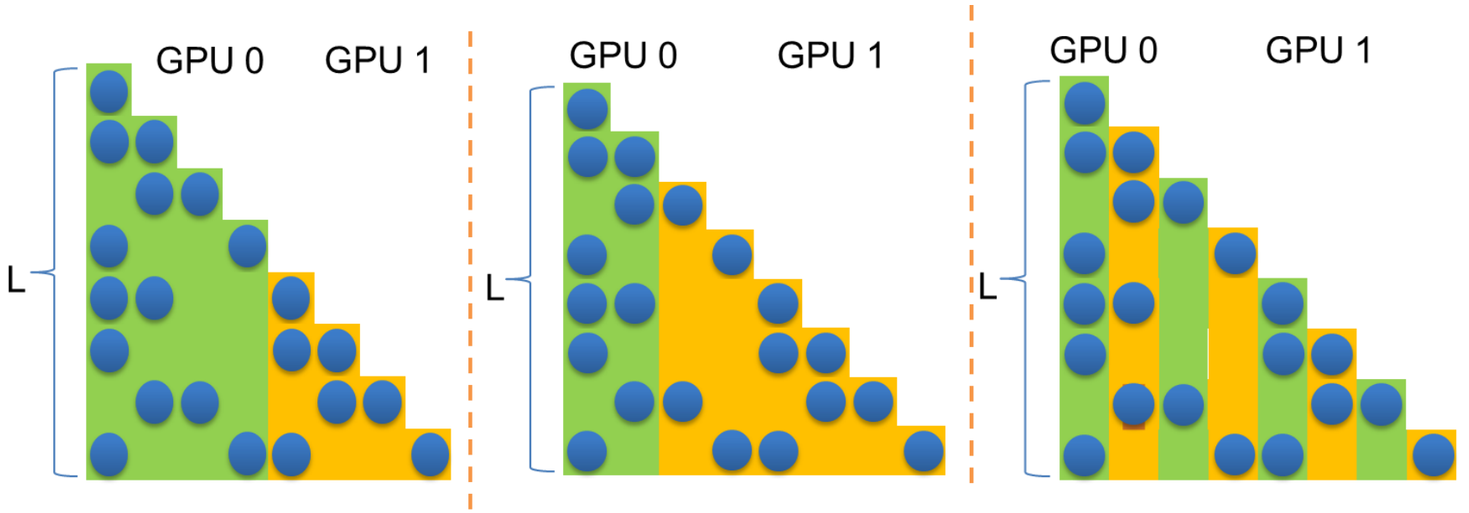
Using NVSHMEM for Resolving Dependency

- For SpTrsv, we convert the system-wide atomic update from unified memory to NVSHMEM shared memory space

```
Solve device x:
for all i ∈ dev.x do                                  ▷ parallel in warps
    x.in.degree[i] ← 0
    while d.in.degree[i]+1 ≠ x.in.degree[i] do
        for all d ∈ ngpu do                           ▷ parallel in threads
            if r.in.degree[i][d] ≠ 0 then
                r.in.degree[i][d] ← get(s.in.degree[i],d)
        x.in.degree[i] ← reduction(*r.in.degree[i])
    for all d ∈ ngpu do                               ▷ parallel in threads
        r.left.sum[i][d] ← get(s.left.sum[i],d)
    x.left.sum[i] ← reduction(*r.left.sum[i])
    x[i] ← b[i] - d.left.sum[i] - x.left.sum[i]
    x[i] ← x[i]/val[col.ptr[i]]
    for all j ∈ nnz_i do                              ▷ parallel in threads
        rid ← row.idx[j]
        if rid ∈ dev.x then
            d.atomic.add(&d.left.sum[rid], val[j]x[i])
            d.atomic.incr(&d.in.degree[rid])
        else
            d.atomic.add(&s.left.sum[rid], val[j]x[i]))
            d.atomic.decr(&s.in.degree[rid])
```



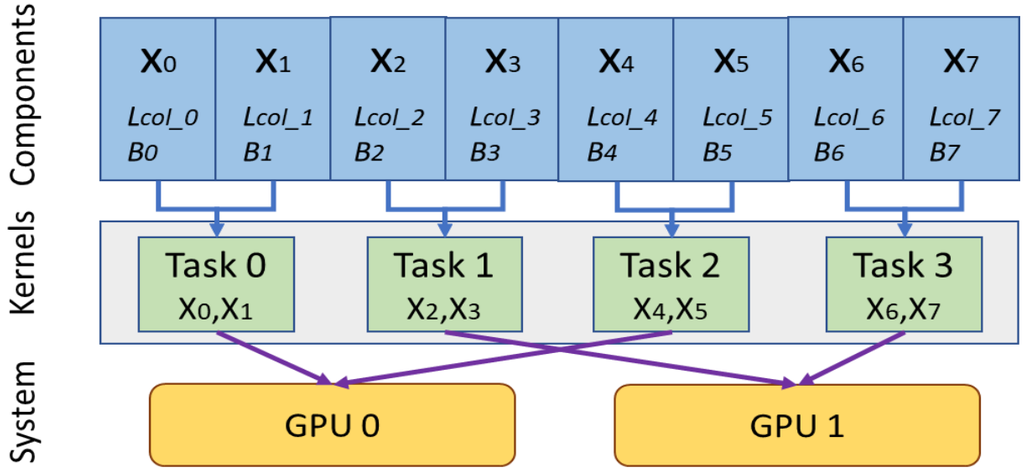
Workload Balance among GPUs



Three Approaches

- Balance components
- Balance *nnz*
- Component RR

- Task Based Workload Distribution
  - More tasks per GPU: workload becomes more balanced among GPUs
  - Less task per GPU: can exploit in-task data locality for better performance



Pacific Northwest NATIONAL LABORATORY

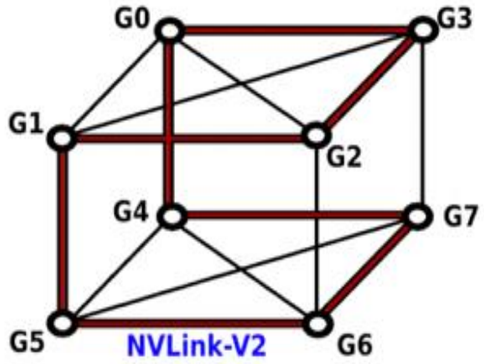
Agenda

- Motivation and Introduction
- Fast and Scalable Sparse Triangular Solver For Multi-GPU Based HPC Architectures
  - Limitation of Unified Memory Solution of SpTrsv
  - Read-Only Inter-GPU Communication with NVSHMEM
  - Fine Turning Workload Dependency
- Evaluation and Results
- Conclusion

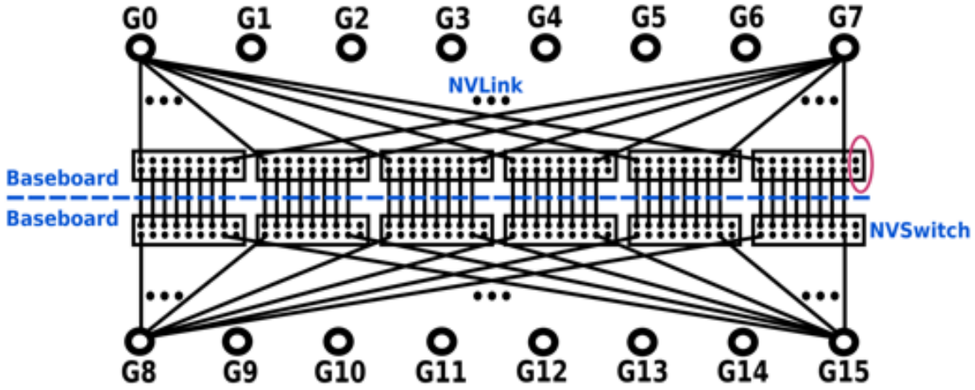
Pacific Northwest NATIONAL LABORATORY

Evaluation

- **V100-DGX-1**
  - Hybrid cube-mesh interconnection network topology
  - Up to 4 GPUs of DGX-1 due to P2P requirement of NVSHMEM
  - 25 GB/s bandwidth per NVLink in each direction
- **V100-DGX-2**
  - All-to-all connected through NVSwitch
  - Up to 16 GPUs of DGX-2
  - 100 GB/s bandwidth per GPUs
- **Benchmark Matrices**
  - 14 sparse matrices from SuiteSparse
  - 2 out of memory matrices
  - the intermediate arrays consume 10% of total memory requirement



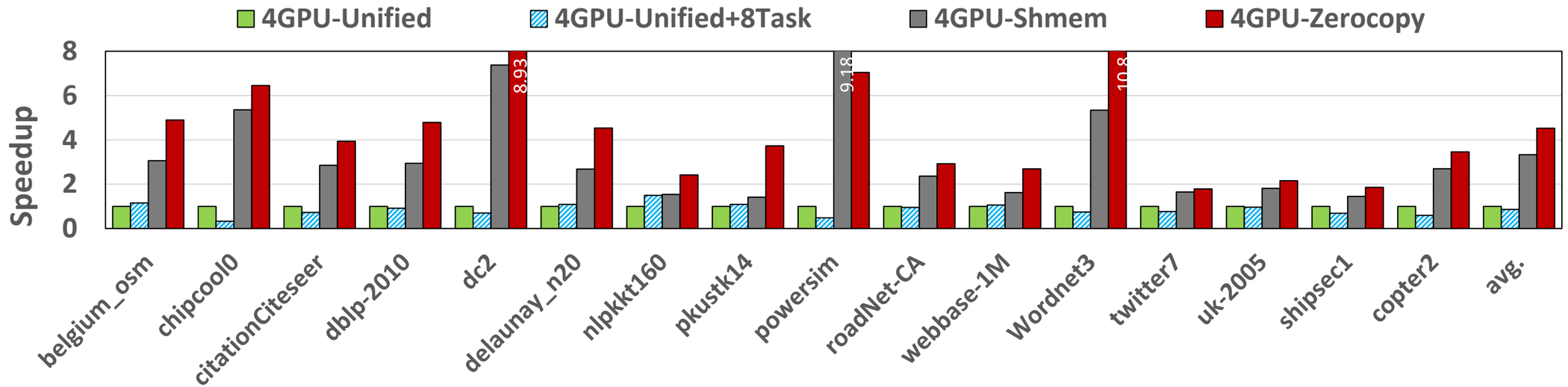
V100-DGX-1

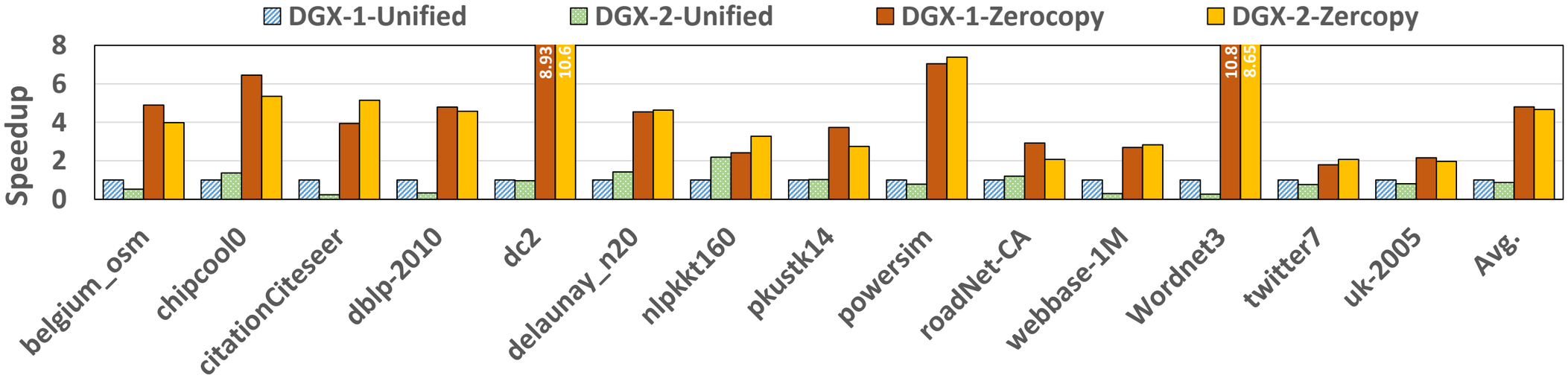


V100-DGX-2

Pacific Northwest NATIONAL LABORATORY

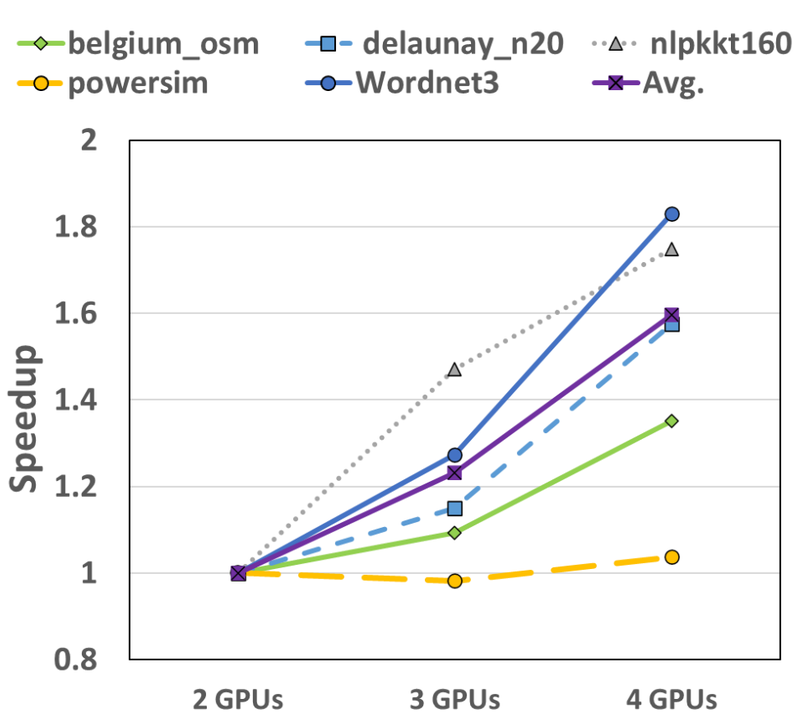
SpTrsv Results (Unified Mem VS NVSHMEM)



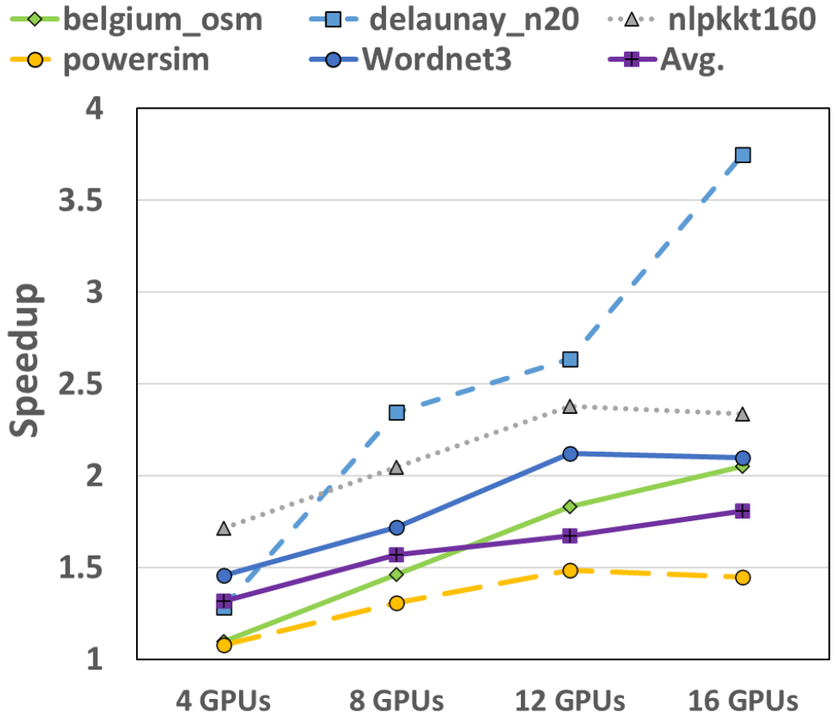


Pacific Northwest NATIONAL LABORATORY

SpTrsv Results (Scalability)



V100-DGX-1



V100-DGX-2

- Results are normalized to ***csrsv2()*** from cuSparse lib *on single GPU*
- Scalability depends on both matrices' parallelism and hardware structure

Pacific Northwest
NATIONAL LABORATORY

Conclusion

- Through performance characterization, we identify that applying the state-of-the-art Unified Memory for data sharing among GPUs may cause severe performance penalty.
- We leverage the new **NVSHMEM** technology to design an efficient and scalable algorithm for the SpTrsv kernel executing on a multi-GPU system setup. It has profound design implication for a spectrum of applications that have inherent irregular memory accesses and strong innertask dependencies.
- For better workload balancing, a **task-based** workload distribution scheme is further introduced.
- We demonstrate the performance benefit and scalability of our proposed SpTrsv design on a range of inputs that require out-of-core execution.

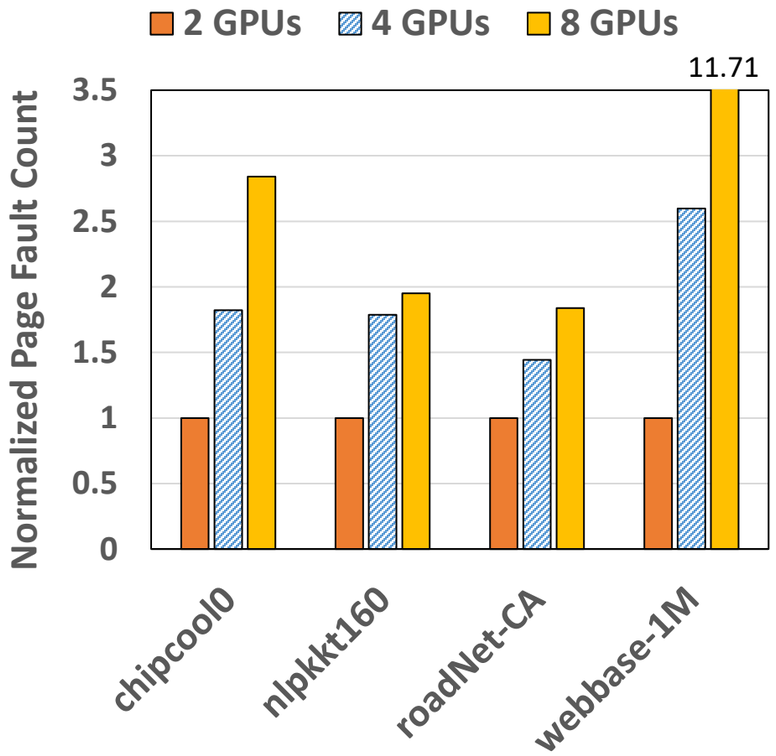
Pacific Northwest
NATIONAL LABORATORY

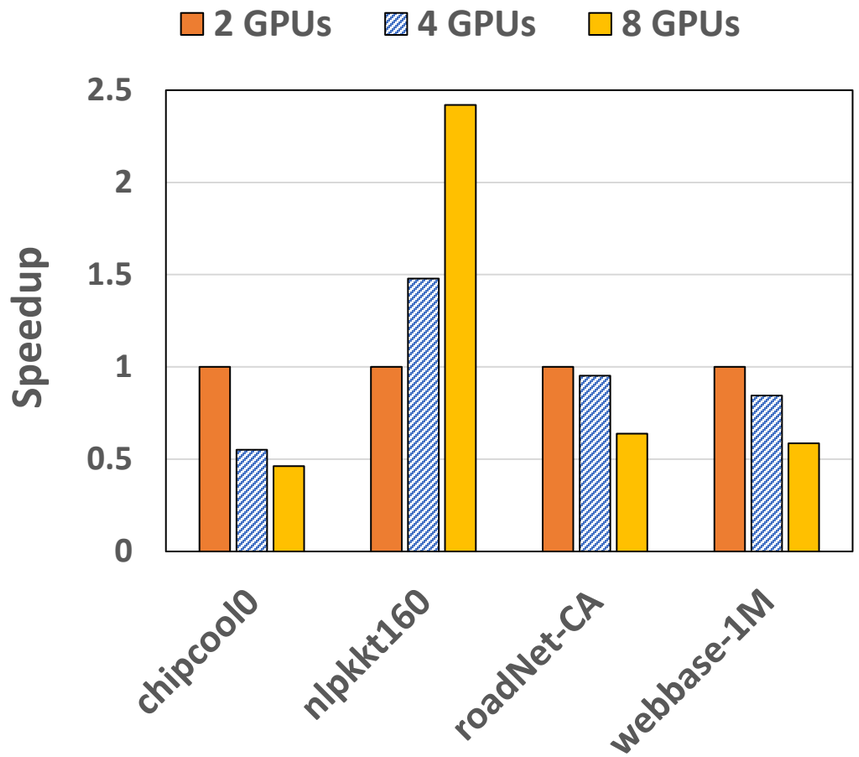
Thank you

Q&A

Pacific Northwest NATIONAL LABORATORY

Page Thrashing of Unified Memory

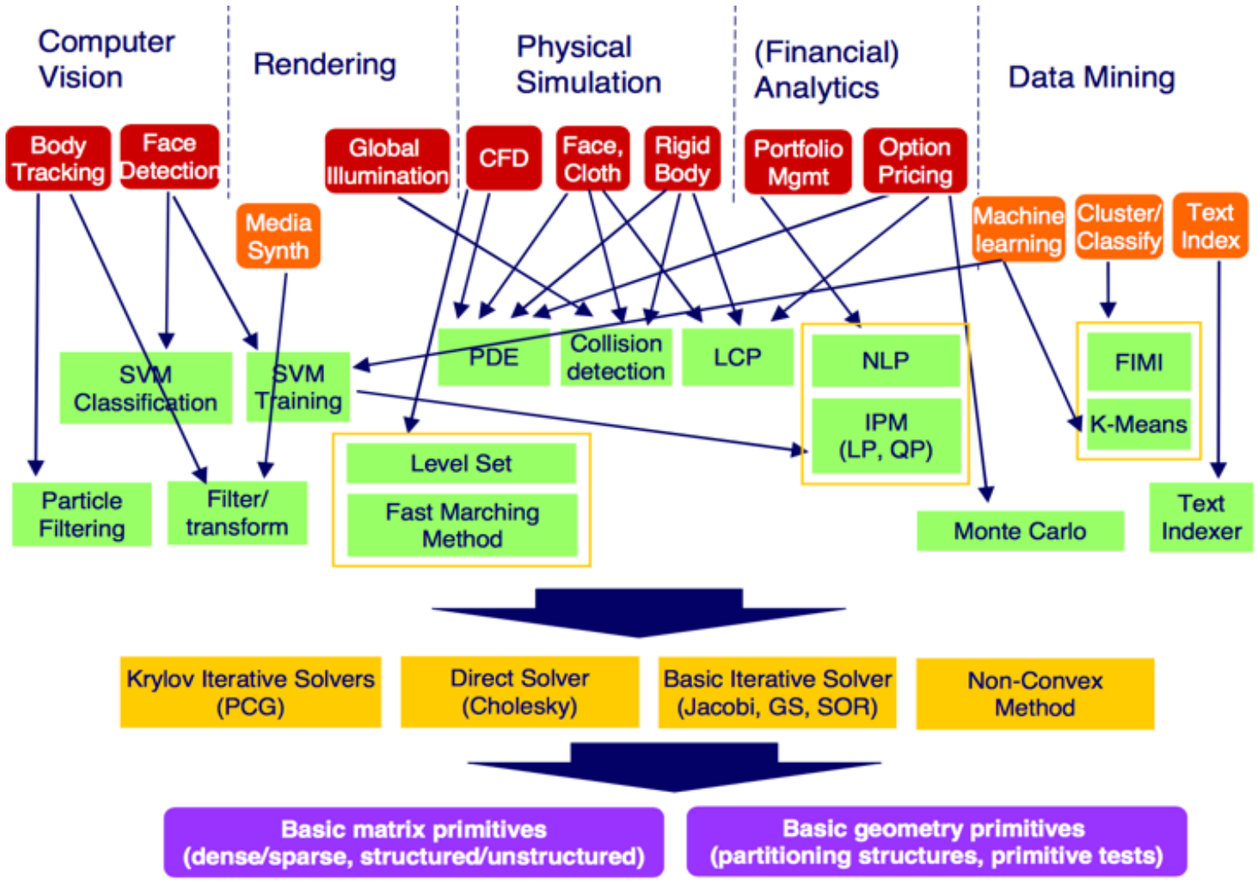




System-wide atomic updates incurs **data contentions** and **page faults,** that significantly slow down SpTrsv even with more computing resource

Major Sparse Matrix Primitives

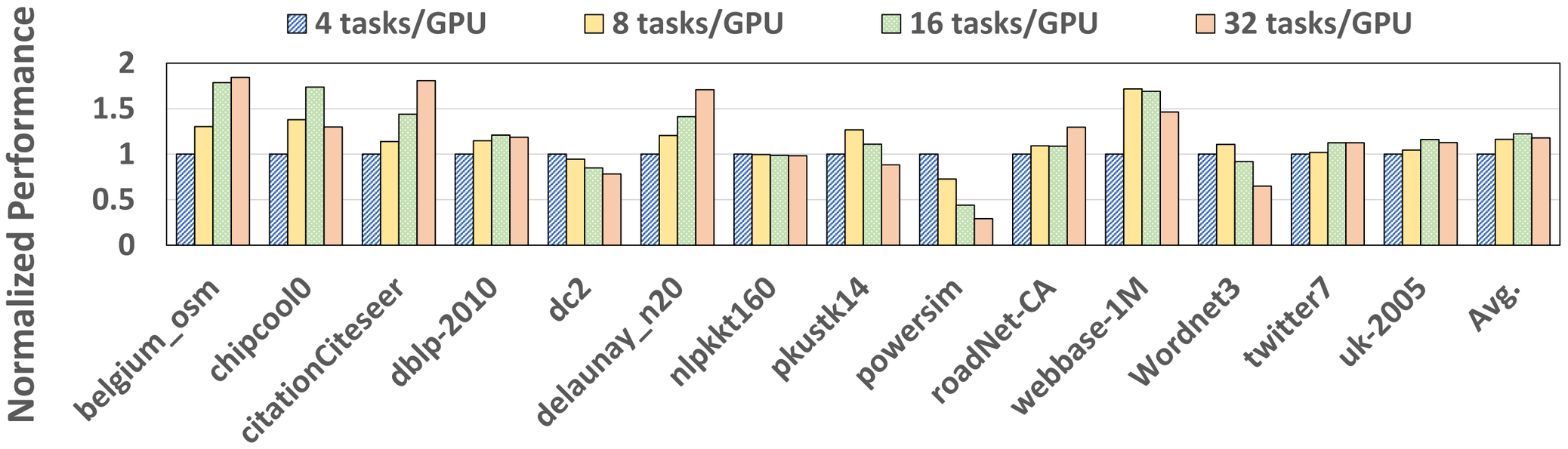
- Many scientific simulation, machine learning, and graph analytics apps can be attributed as sparse BLAS problems in the end
- Among all sparse linear algebra kernels, **Sparse Triangular Solver** (**SpTrsv),** which solves sparse triangular linear system, play fundamental roles
- However, due to the **inherently sequential** feature, parallelizing SpTrsv is not trivial, specially for multi-GPU based HPC systems



(Source: Berkeley Dwarfs Report)

Pacific Northwest
NATIONAL LABORATORY

SpTrsv Results (Sensitive Over Task_size)



More tasks per GPU leads to finer-grained communication and better workload balance, but at the same time, suffer from higher scheduling overhead to issue tasks