The Scalable Petascale Data-Driven Approach for the Cholesky Factorization with multiple GPUs

Yuki Tsujita, Toshio Endo, Katsuki Fujisawa

Tokyo Institute of Technology, Japan

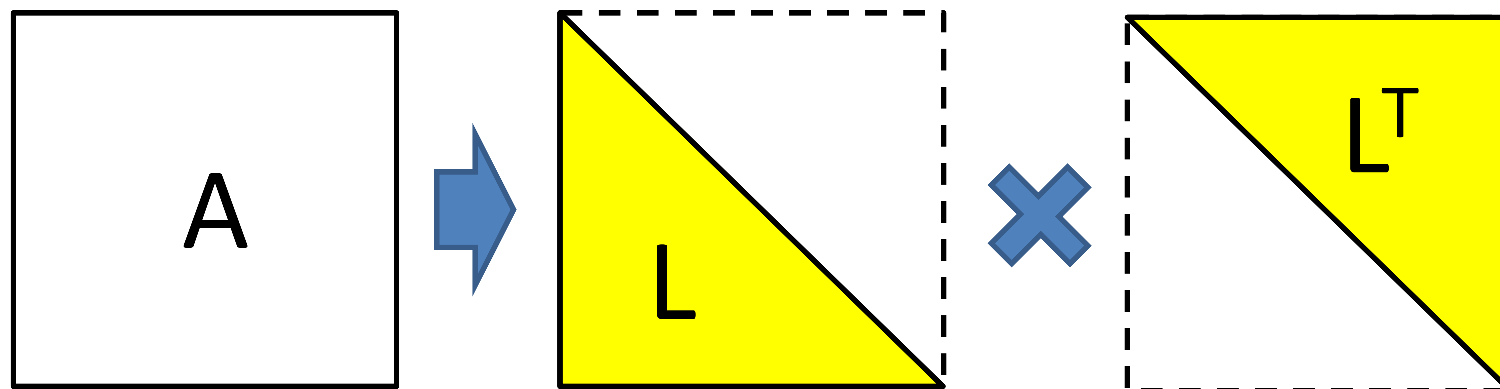
ESPM2 2015@Austin Texas, USA

What is the Cholesky factorization?

- The Cholesky factorization is a factorization of a real symmetric positive-definite matrix into the product of a lower triangular matrix and its transpose
- Statement

$$A=LL^T (A \in R^{m \times m})$$

- The time complexity of the Cholesky is $O(m^3)$



SDPARA: Our Target application

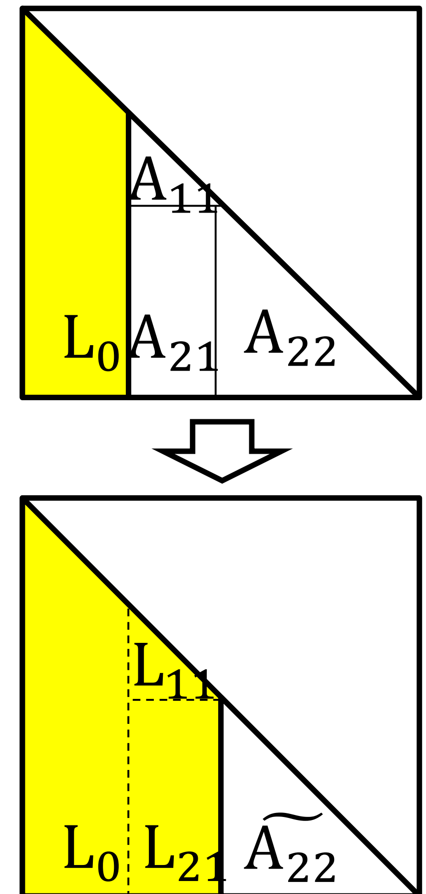
- Dense Cholesky factorization is the important kernel of SDPARA GPU ver.[Fujisawa et al. 2011]
- SDPARA GPU ver.
  - Application to solve SDP(SemiDefinite Program)
  - Offload a part of its calculations to GPU

Table: Performance record of CHOLESKY of SDPARA

Year	n	m	CHOLESKY (Flops)
2003	630	24,503	78.58 Giga
2010	10,462	76,554	2.414 Tera
2012	1,779,204	1,484,406	0.533 Peta
2014	2,752,649	2,339331	1.713 Peta

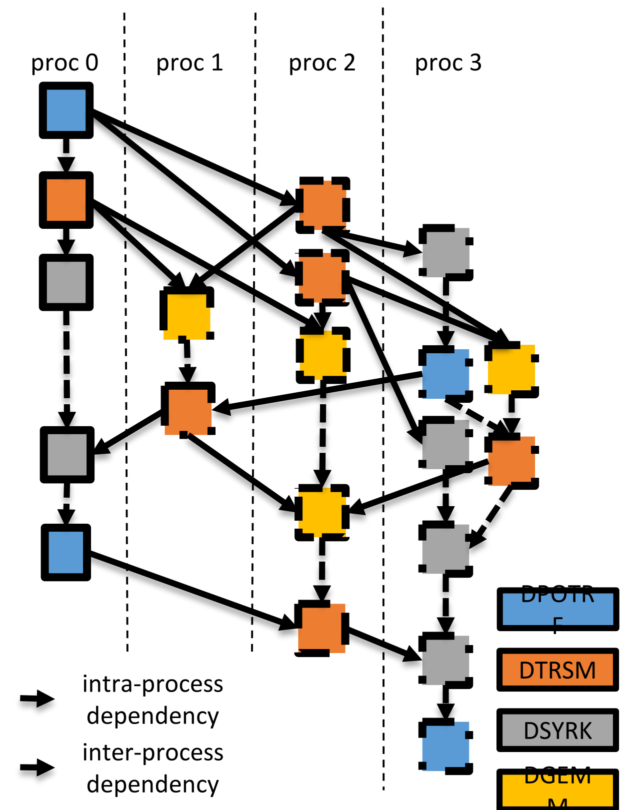
Existing approach I : Synchronous Implementation [Fujisawa et al. IPDPS 2014]

- Block Cholesky factorization
  - The input data is divided into the blocks
    - The calculations proceed in each block
  - The blocks are assigned to the processes by two dimensional block cyclic division
    - Processes do calculations of the only assigned data
- Each iteration proceeds synchronously
  - The data are transferred from CPU to GPU at the beginning of each iteration
  - If a process has no task in a certain iteration, it has to wait for the other processes finishing without doing anything



Existing approach II : Data-Driven Implementation [Tsujita&Endo JSSPP 2015]

- Kernels are divided into fine-grained tasks
  - Basically each task proceeds asynchronously
- PCIe comm. performs only when it needs
- Inter-process comm. performs in Point-to-Point way
- We found the performance may decrease in extremely large scale case



The DAG of the Cholesky factorization

Our Target

- Large problem size(m>2M)
  - Use capacity of host memory to put the matrix data[Tsujita,Endo JSSPP2015]
- High performance(>1.7PFlops)
  - Use multiple GPUs and reduce PCIe communication by GPU memory aware scheduling[Tsujita,Endo JSSPP2015]
  - Solve the communication bottleneck by introducing the scalable communication

Contribution

- Goal
  - Performance improvement of the multi-node multi-GPU Cholesky factorization
- Approach
  - Data-Driven scheduling to reduce data movement (presented@JSSPP2015)
    - Scheduling tasks in an application
    - Task selection to improve GPU memory reusability
  - The MPI communication pattern for the scalability improvement

- Achieve the performance of 1.77PFlops with 1360 nodes

Implementation overview

	Existing method Ⅰ (Synchronous)	Existing method Ⅱ (Data-Driven)	Proposed method
Data driven	×	✓	✓
PCIe Comm reducing	× (Naïve)	✓ (Swap)	✓ (Swap)
MPI Comm Scalability	✓ (Group)	× (Point-to-point)	✓ (Scalable communication)
Overlap of calculations & communications	✓	✓	✓

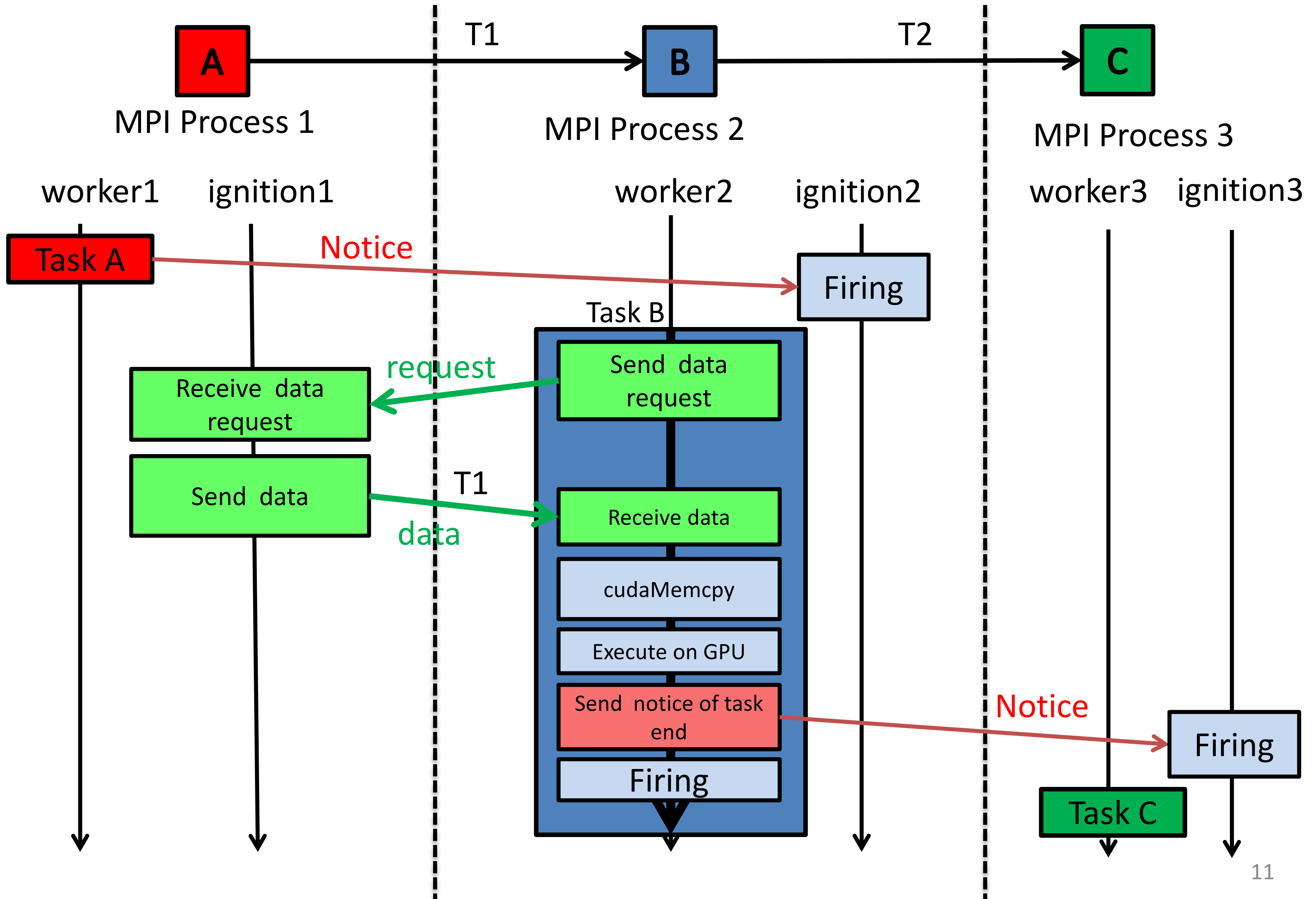
Our Basic Data-Driven Implementation (Existing Approach II)

- GPU memory-aware scheduling
  - Task selection considering the reusability of GPU memory
- Point-to-Point asynchronous MPI communication
- GPU memory management by swapping
  - select an unnecessary data as a victim

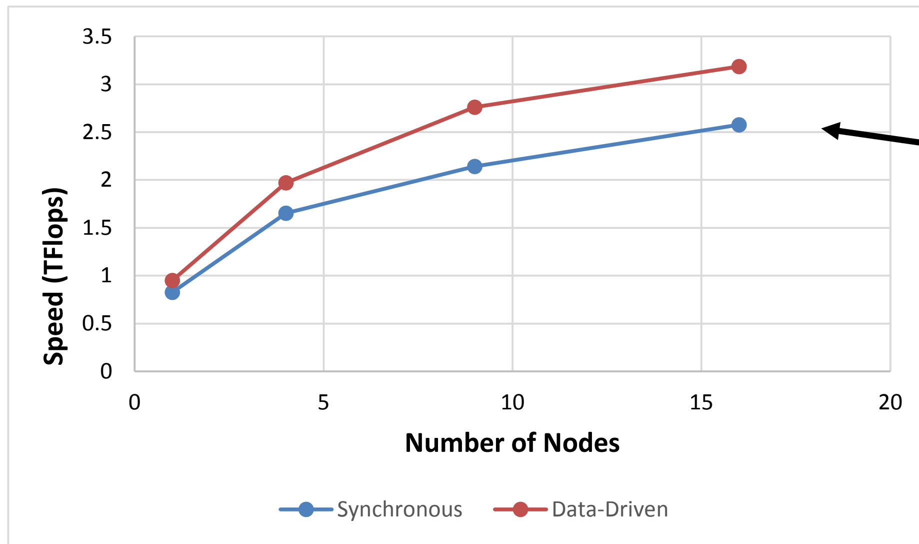
Worker thread & Ignition thread

- MPI process has several worker thread and one ignition thread
- Worker
  - Executes tasks
  - Process has two or three worker per one GPU in order to achieve overlapping of calculation, PCIe and MPI simply
- Ignition
  - checks arrival of notice messages from other processes
  - handles data request
- All threads in a process shares a single task queue

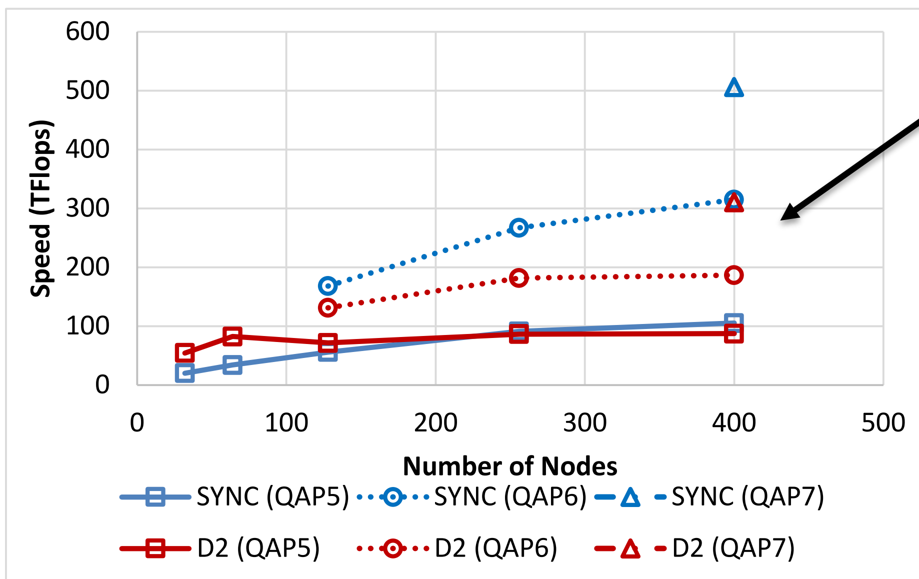
Task Execution



The pitfall of Data-Driven



By Data-Driven implementation, we get better performance



As problem size or number of the nodes increases, the performance decreases in data-driven execution

The suspected bottleneck is a concentration of MPI communication

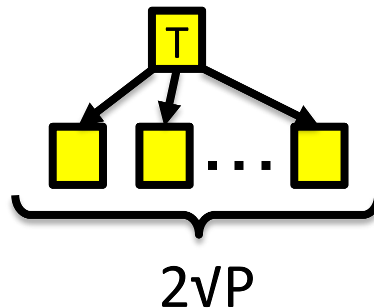
Not only does our approach suffer from this problem !

The pitfall of Data-Driven

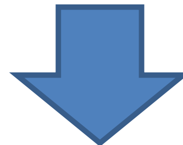
Synchronous implementation uses MPI_Bcast for data transfer

But in Data-Driven implementation

- Each task runs asynchronously -> MPI_Bcast, MPI_Ibcast: ×
- When many processes request the same tile,
  Point-to-Point communication is executed 2√P times



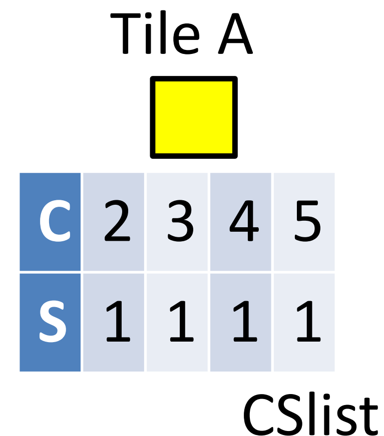
The existing data-driven shows less performance in high parallel situation



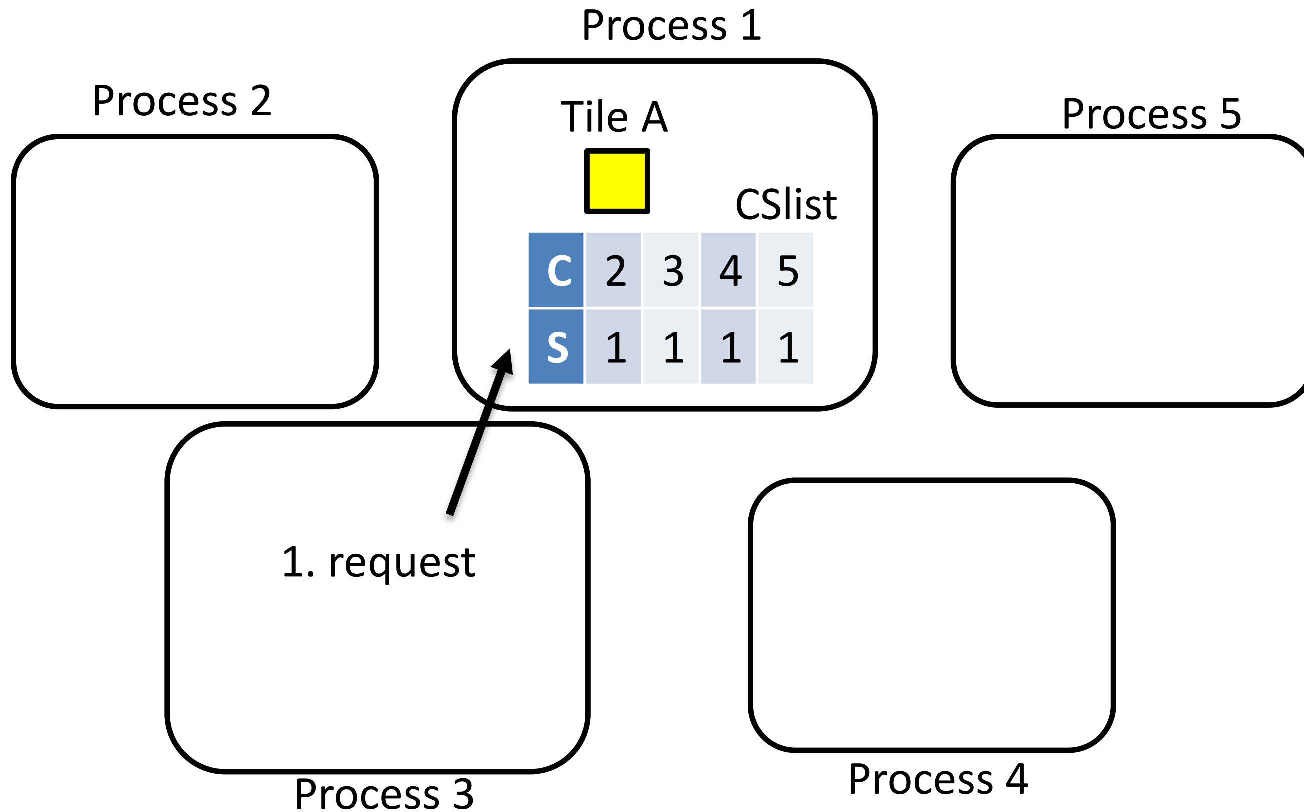
For scalable data transfer, we create
a broadcast tree structure dynamically

Scalable Communication

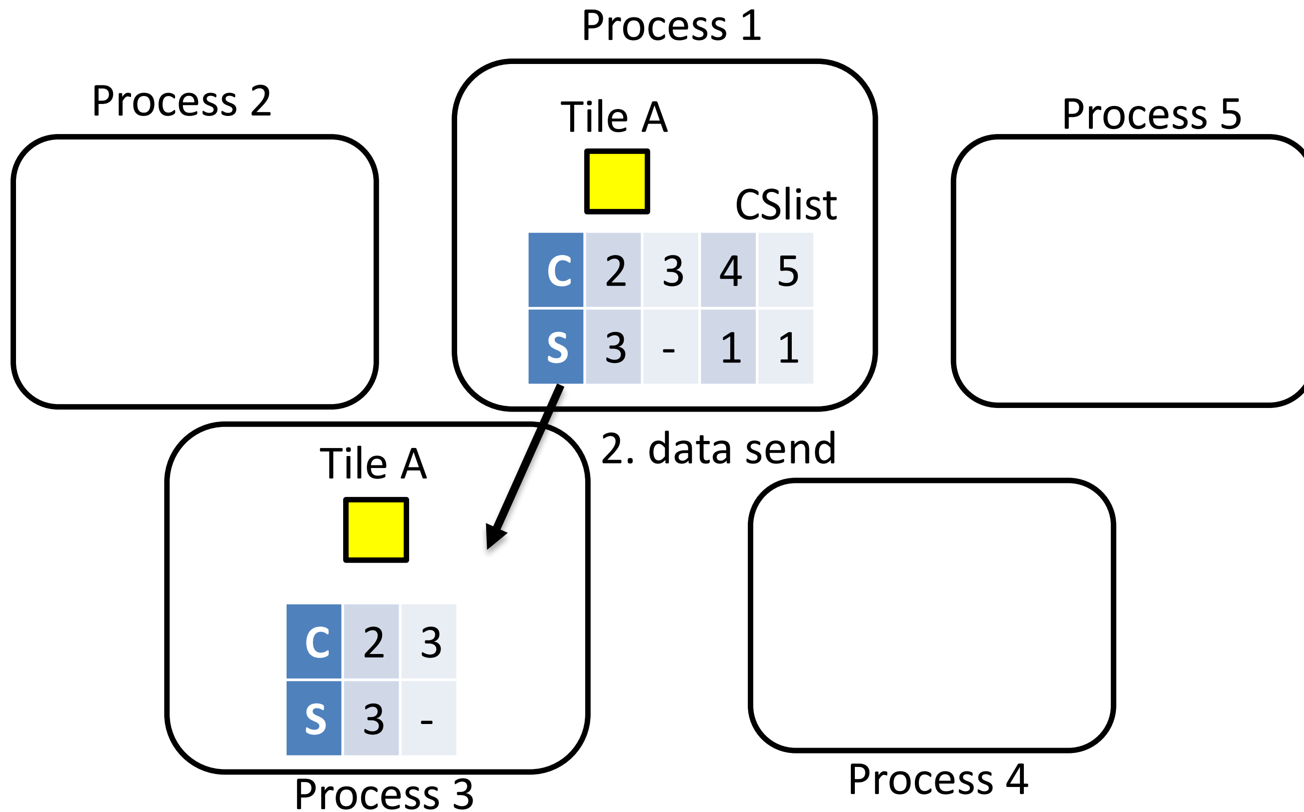
- Presupposition
  - Data send is occurred only when a process receive requests from other processes
  - The order of data requests is unsettle
- For scalable data transfer,
  We make CSlist(Client-Server list)
  - one CSlist for one tile
  - CSlist has clients and corresponding servers
  - When a process receives requests, checks CSlist
    ・Server: send data ・Others: forward to its server
  - When a process sends data, forces a part of its clients on requestor



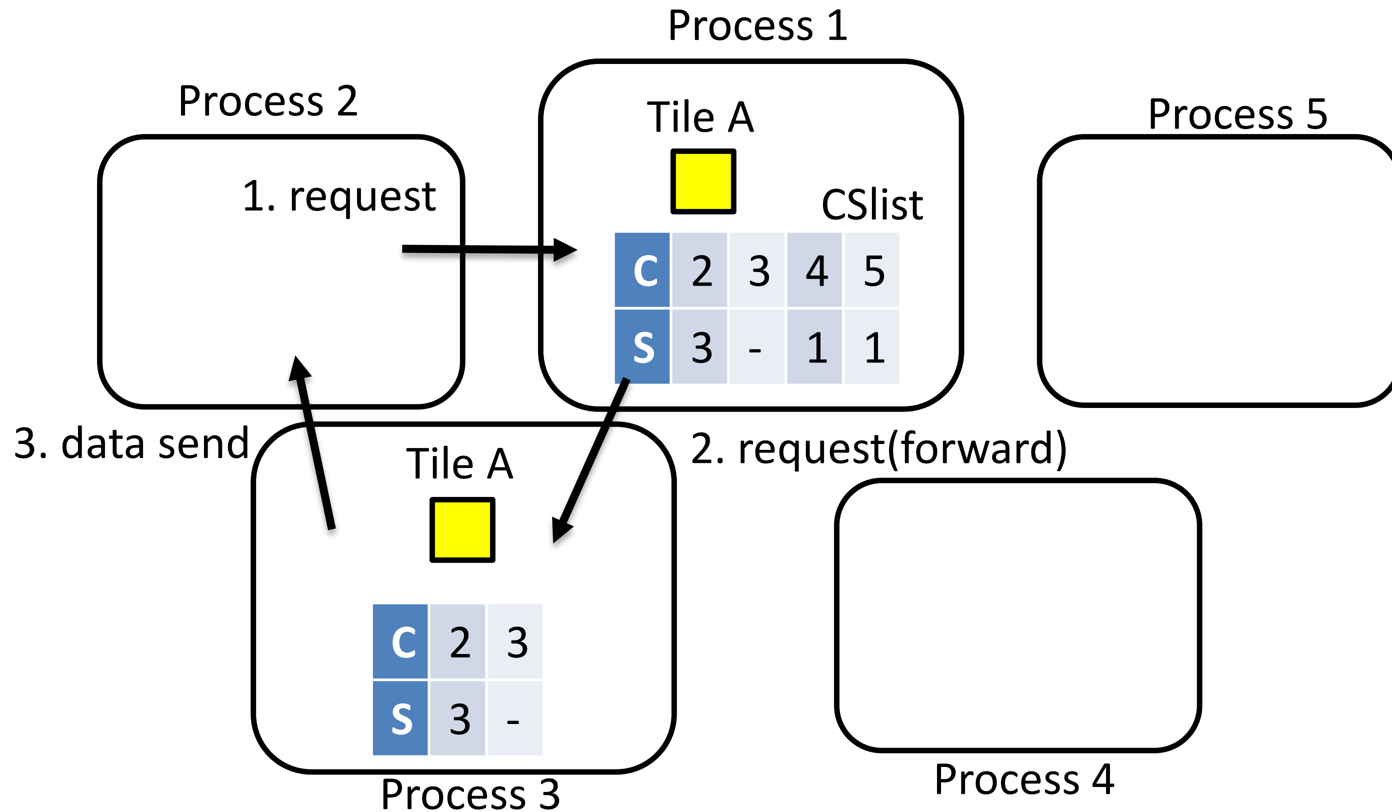
Scalable Communication



Scalable Communication



Scalable Communication

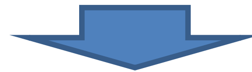


Scalable termination detection

A process cannot exit even if all of its tasks has been finished

→ Process may still receive requests for its owned data from other running processes

The detection of process's termination becomes difficult !

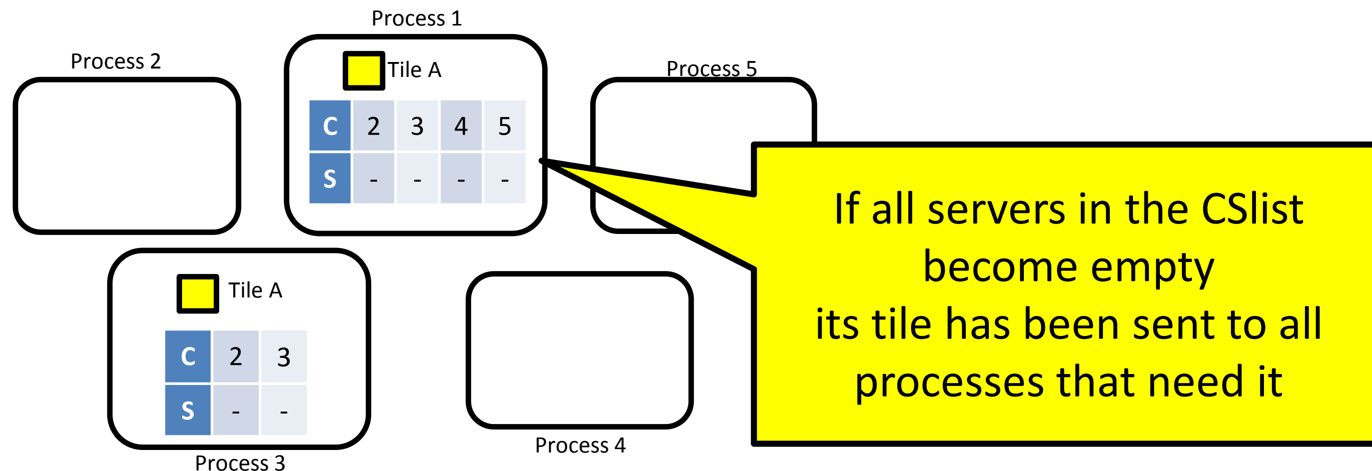


we solve this by using CSlist

CSlist shows "which client has been requested this tile, or not yet"

So there is no further request message, when CSlists for all local tiles become empty

By using CSlist we can detect process's termination without especial communications



Experiment Conditions

- We use 1360 nodes of TSUBAME2.5

node architecture of TSUBAME 2.5	
CPU	Intel Xeon 2.93 GHz (6 cores) x 2
CPU memory	54GiB
GPU	NVIDIA Tesla K20X × 3
GPU memory	6GiB

- Three MPI processes per a node
- One GPU per a MPI process(3 GPU/node)
- Tile Size:2,048 x 2,048
- GPU memory:5,000MiB per a GPU
- NVIDIA CUDA 7.0 and CUBLAS 7.0

Performance Evaluation

- Evaluation
  - Scalability evaluation
  - Extremely Large Scale
- Problem size

  QAP5: m=379,350 QAP6: m=709,275

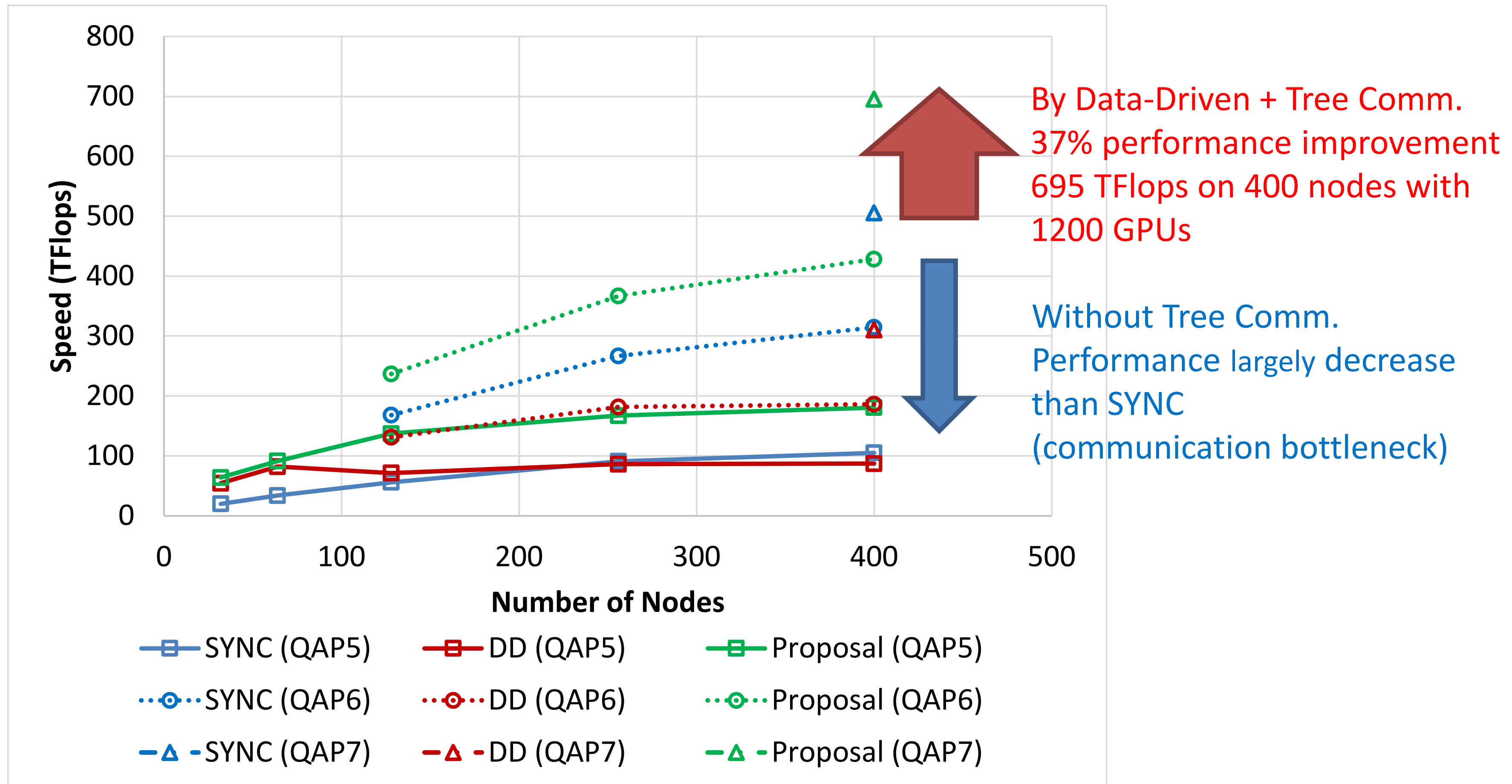
  QAP7: m=1,218,400 QAP9: m=1,962,225

- Compared Implementations

	Existing approach Ⅰ (Synchronous: SYNC)	Existing approach Ⅱ (Data-Driven:DD)	Proposed method (Proposal)
PCI Comm Reducing	×	✓	✓
MPI Comm Scalability	✓ (Group)	× (Point-to-point)	✓ (scalable communication)

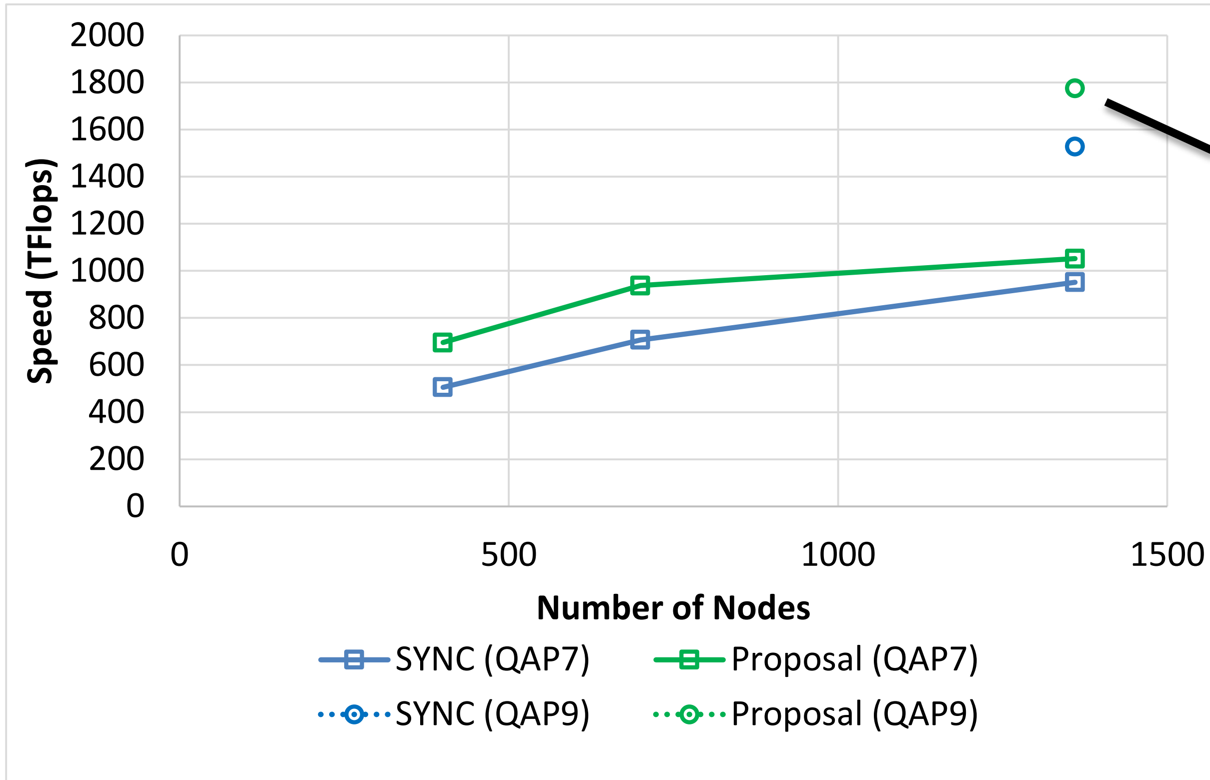
Scalability Evaluation

Conduct scalability evaluation on TSUBAME2.5 using until 400 nodes(3 GPUs per a node)



Extremely Large Scale

- Conduct scalability evaluation on from 400 nodes to 1360 nodes (3GPUs per a node)



1.775PFlops on 1360 nodes with 4080 GPU by our approach

Related work

- StarPU: a unified platform for task scheduling on heterogeneous multicore Architectures[Cédric Augonnet et al.]
  - A DAG scheduling framework for heterogeneous environments
  - Allows for each task to run either on CPUs or GPUs according to the resource utilization in order to improve the performance
  - But StarPU does not have scalability improvement techniques as our approach
- DAGuE: A generic distributed DAG engine for high performance computing[George Bosilca et al.]
  - DAG(Direct Acyclic Graph) scheduler for distributed environments with GPUs
  - The Cholesky factorization is one of their target application
  - But it is not clear how DAGuE treats memory objects when GPU memory is full

Conclusion

- The solution of the scalability issues in typical data driven implementation by introducing scalable data transfer method and termination detection method
- Compared with the synchronous implementation, 37% performance improvement on 400 nodes and 1,200 GPUs of TSUBAME2.5 supercomputer
- Achieved 1.775PFlops on 1360 nodes and 4080 GPUs of TSUBAME2.5 supercomputer

Future Work

- Use both CPU & GPU for kernel calculations
- Comparative experiments with related works
- Construct the ideal task selection model and conduct comparative experiments with it
- Apply our approach to other applications