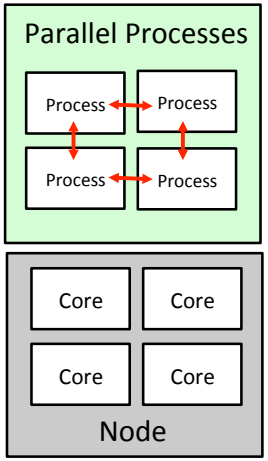
Optimizing MPI Intra-node Communication with New Task Model for Many-core Systems

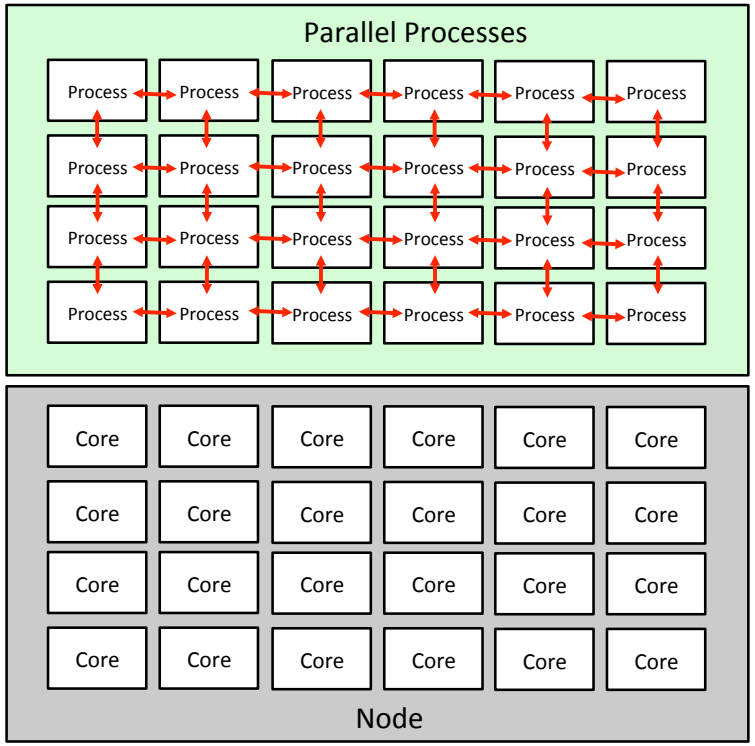
Research and Development Group, Hitachi, Ltd.
Akio SHIMADA

Background

- A large number of parallel processes can be invoked within a node on a many-core system
  - MPI and some PGAS language runtimes invokes multiple processes
- Fast Intra-node communication is required
  - Many researches proposed a variety of intra-node communication schemes(e.g. KNEM, LiMIC) since the appearance of multi-core processor and try to accelerate intra-node communication on many-core systems (e.g. hybrid MPI)



Communication on Multi-core Node



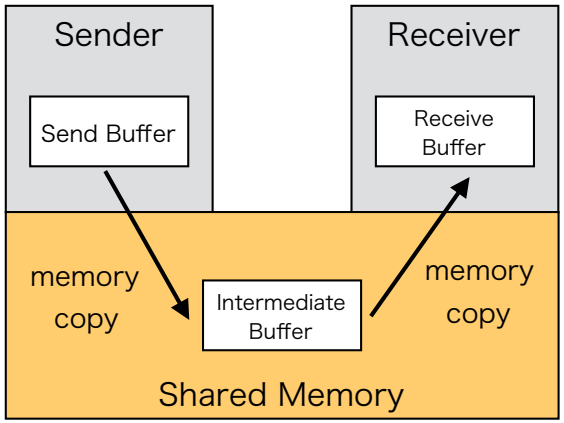
Communication on Many-core Node

Conventional Intra-node Communication Schemes

- Overheads for “crossing address space boundaries among processes” are produced
  - There are address space boundaries among processes

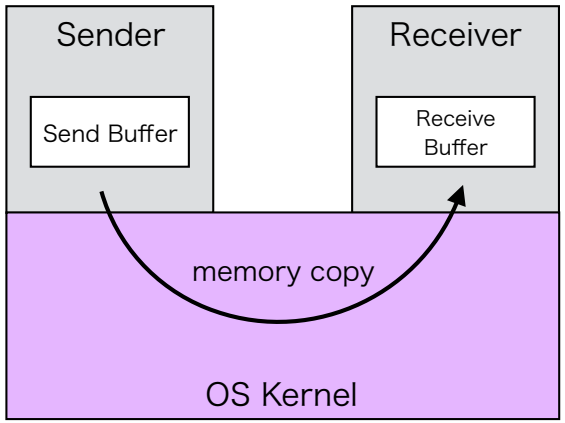
Shared Memory

- Double-copy via shared memory is required for every communication



OS kernel assistance (KNEM, LiMIC, etc.)

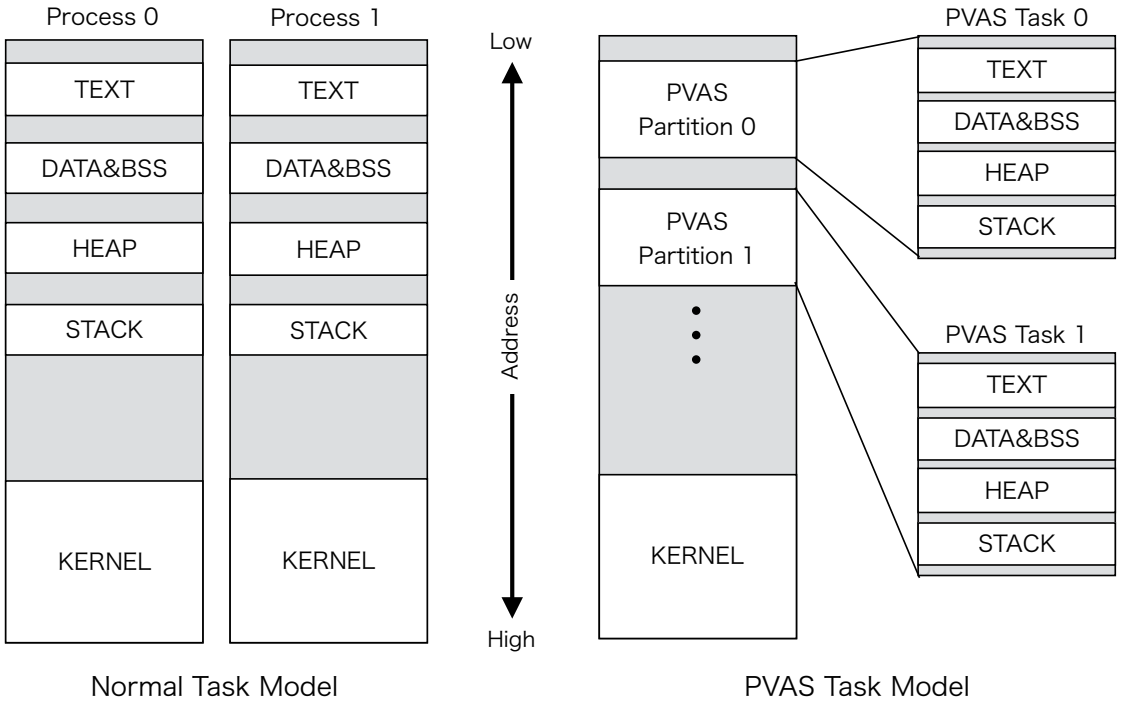
- System call overhead is produced for every communication



Proposal

- Partitioned Virtual Address Space (PVAS)
  - A new task model for efficient parallel processing on many-core systems
  - PVAS make it possible for parallel processes within the same node to run in the same address space
  - PVAS can remove overheads for crossing address space boundary from intra-node communication

Address Space Layout



- PVAS partitions a single address space into multiple segments (PVAS partition) and assigns them to parallel processes (PVAS tasks)
- Parallel processes uses the same page table for managing memory mapping informations
- PVAS task can use only its own PVAS partition as its local memory (cannot allocate memory within a PVAS partition assigned to the other PVAS task)
- PVAS task is almost same as normal process except sharing the same address space with other processes

PVAS Feature

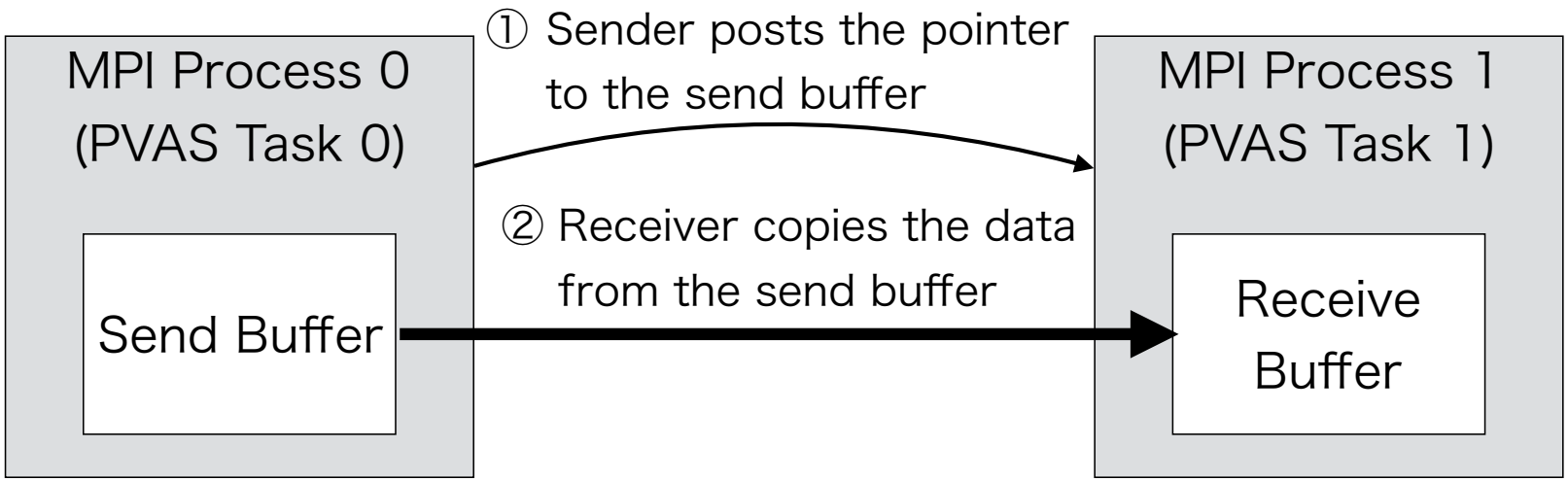
- All memory of the PVAS task is exposed to the other PVAS tasks within the same node
  - PVAS task can access the memory of the other PVAS tasks by load/store instructions (There are no address space boundaries among them)
  - A pair of PVAS tasks can exchange the data without overheads for crossing an address space boundary

Optimizing Open MPI by PVAS

- PVAS BTL component is implemented in the Byte Transfer Layer (BTL) of the Open MPI
  - SM BTL
    - Supporting double-copy communication via shared memory
    - Supporting single-copy communication with OS kernel assistance (using KNEM)
  - PVAS BTL (developed on the basis of the SM BTL)
    - Copying the data from send buffer to receive buffer without OS kernel assistance by using PVAS facility

PVAS BTL

- Invoking MPI process as PVAS task
- Copying the data from send buffer to receive buffer directly
- The overheads for crossing address space boundary is not produced when transferring the data
  - Single-copy communication (avoiding extra memory copy)
  - OS kernel assistance is not necessary (avoiding system call overhead)

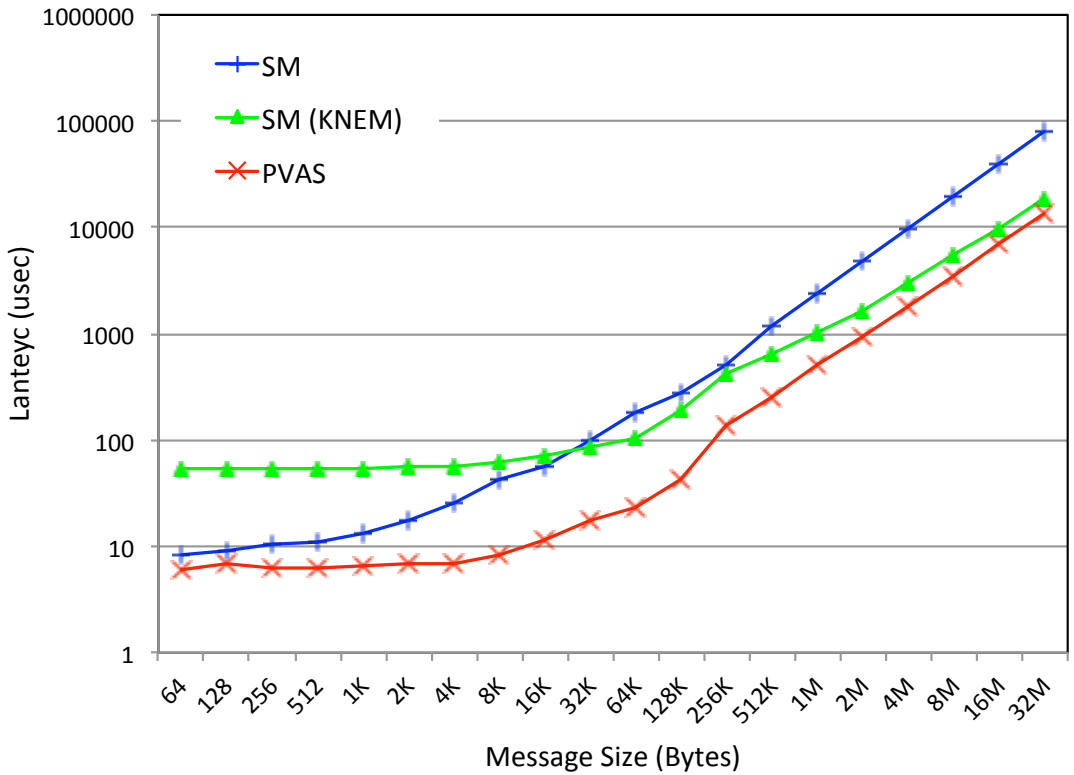


Evaluation Environment

- Intel Xeon Phi 5110P
  - 1.083 GHZ, 60 cores (4HT)
  - 32 KB L1 cache, 512 KB L2 cache
  - 8 GB of main memory
- OS
  - Intel MPSS linux 2.6.38.8 with PVAS facility
- MPI
  - Open MPI 1.8 with PVAS BTL

Latency Evaluation

- Ping-pong communication latency was measured by running Intel MPI Benchmarks

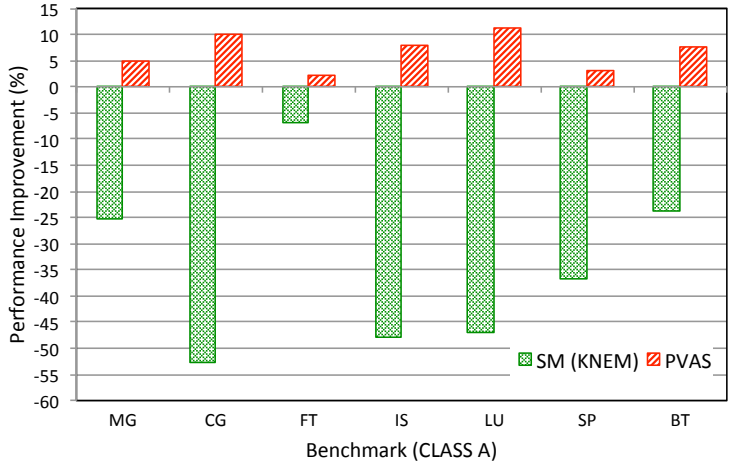


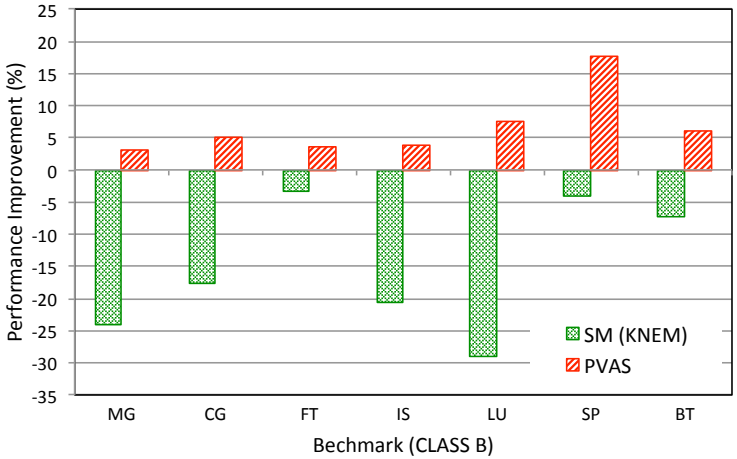
- PVAS BTL outperforms others regardless of the message size

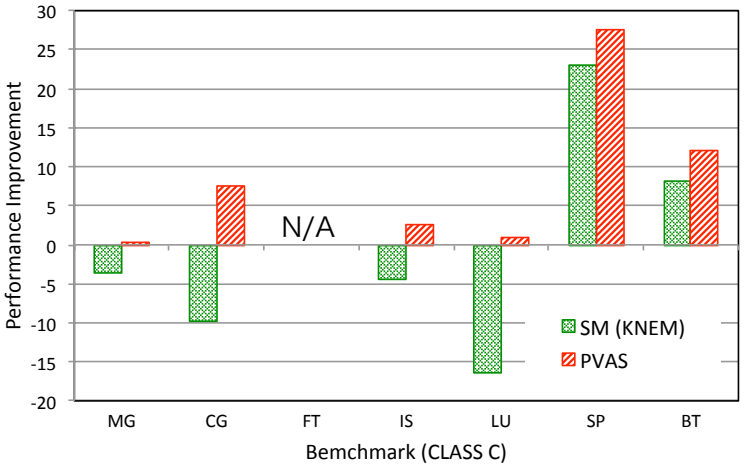
- Latency of the SM BTL (KNEM) is higher than that of SM BTL when message size is small because of the system call overhead

NAS Parallel Benchmarks (NPB)

- Running NPB on a single node
  - Number of Processes
    - 128 (MG, CG, FT, IS, LU)
    - 225 (SP, BT)
  - Problem size
    - CLASS A, B, C (A < B < C)
- PVAS BTL improves benchmark performance by up to 28%
  - SP (CLASS C)

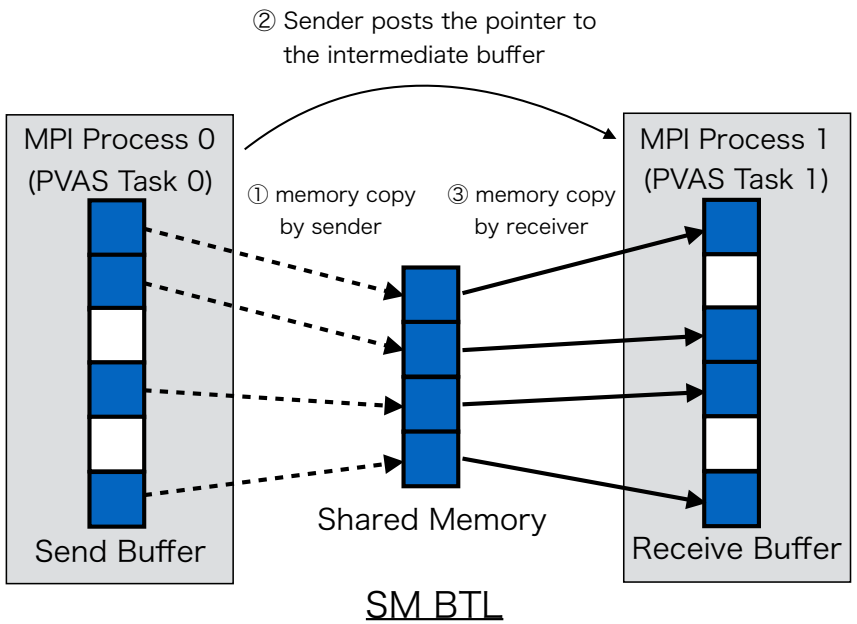


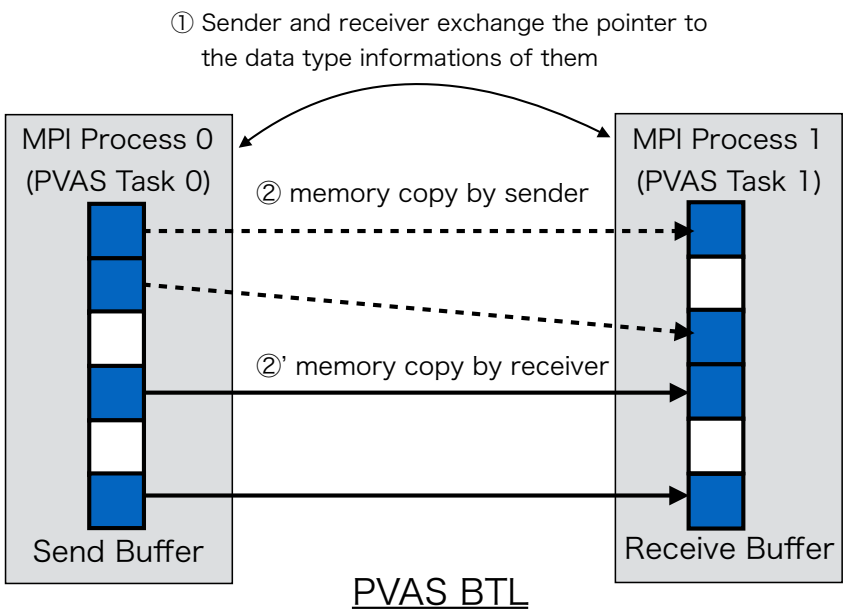




Optimizing Non-contiguous Data Transfer Using Derived Data Types

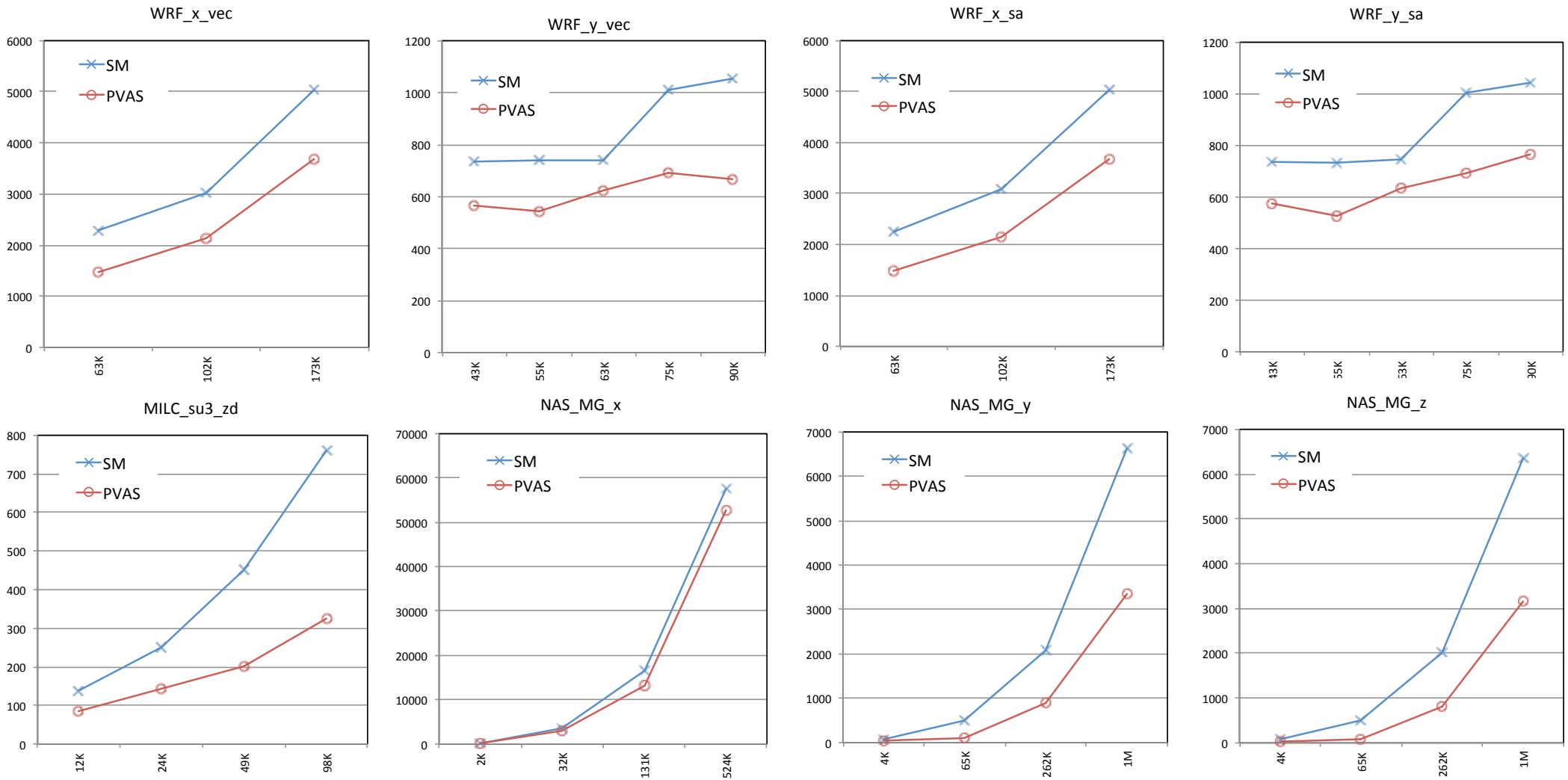
- Sender and receiver exchange the pointer to the data type informations of them
  - MPI process can access the MPI internal objects of the other MPI process when using PVAS facility
- Sender and receiver copies the data from the send buffer to the receive buffer consulting the data type informations of them
  - Sender and receiver copy the data in parallel





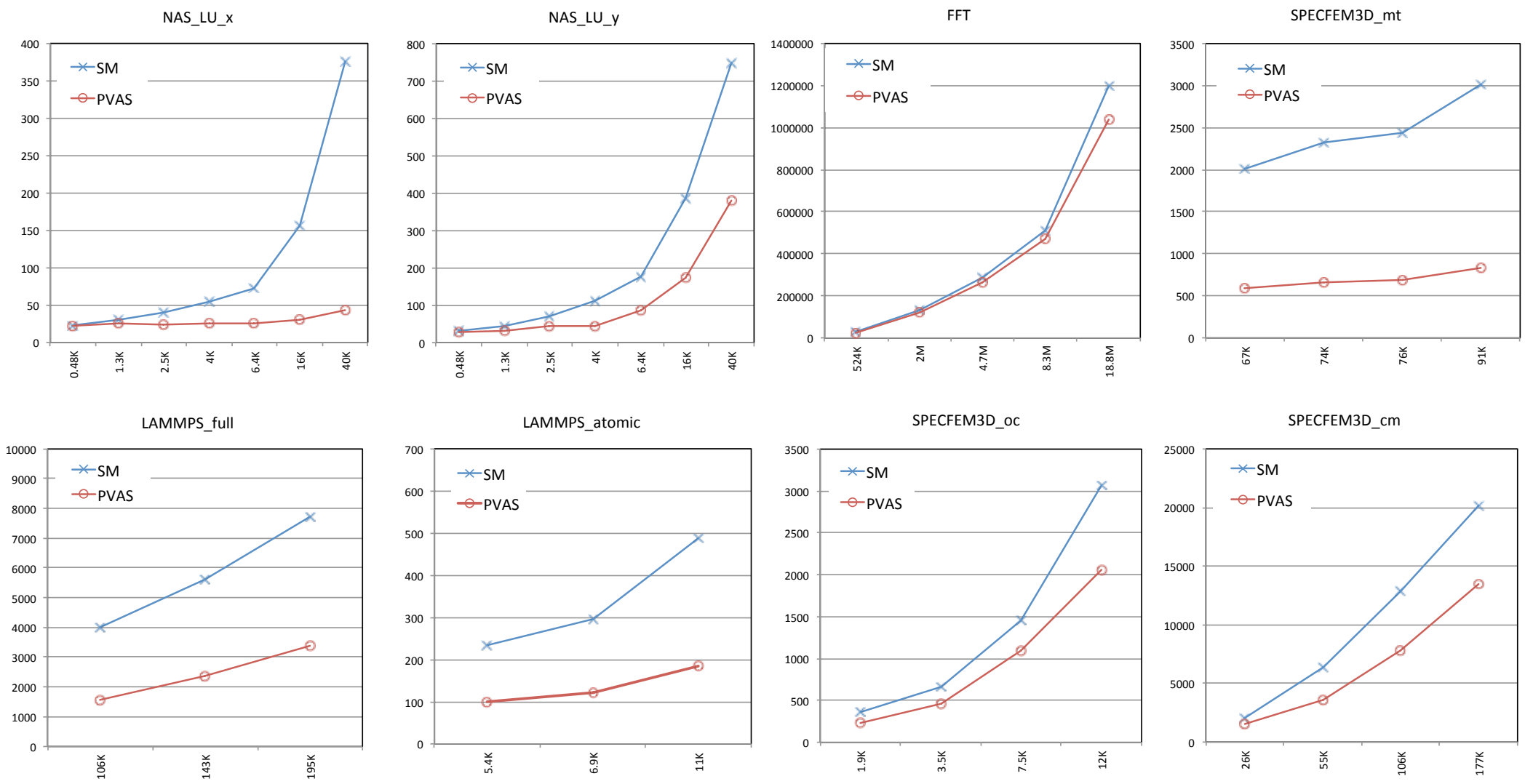
Latency Evaluation Using DDTBench(1/2)

- DDTBench [Timo et al., EurMPI'12] mimics the commutation pattern of MPI applications by using derived data types
  - MPI processes send and receive the non-contiguous data in WRF, MILC, NPB, LAMMPS, SPECFEM3D



X-axis: Data Size, Y-axis: Latency (usec)

Latency Evaluation Using DDTBench(2/2)



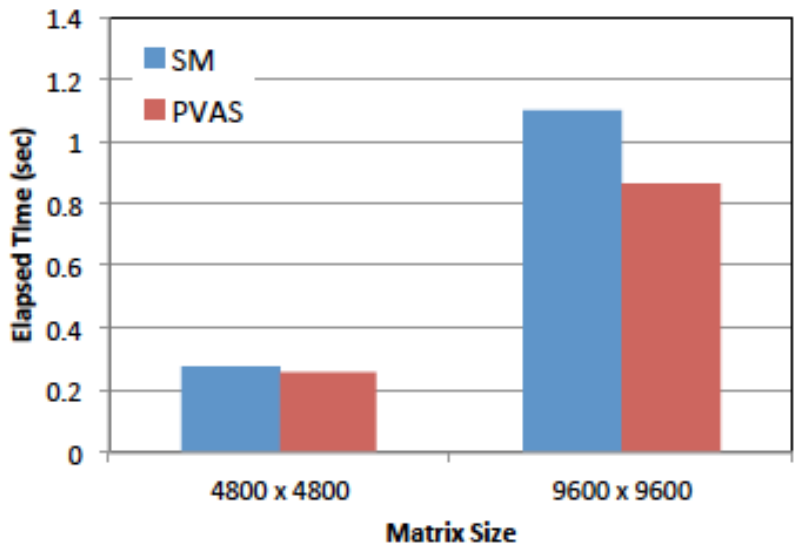
X-axis: Data Size, Y-axis: Latency (usec)

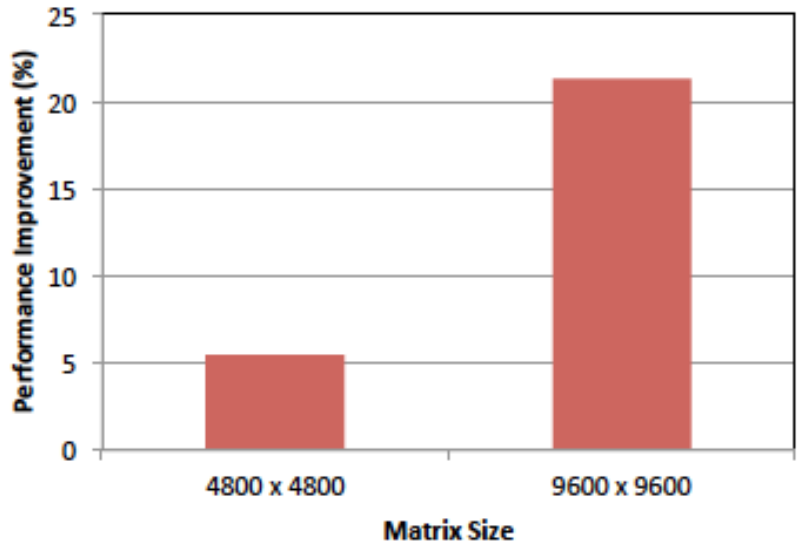
Latency Analysis

- Performance improvement can be larger when data size is large
  - PVAS implementation can accelerate data copy between processes
  - Time for data copy does not impact when message size is small
- Performance improvement can be smaller when transferring data from complex data type buffer to complex data type buffer
  - Access to the sparsely located data incurs a lot of cache misses during data copy

FFT2D_datatype

- 2D Fast Fourie Transform code
  - Using Derived Data Types for matrix transpose
  - Different vector types on send/recv side
- PVAS BTL improves benchmark performance by up to 21%





fft2d_datatype results (NP=240)

Related Work

- SMARTMAP [Ron et al., SC'08]
  - SMARTMAP enables process to map the whole memory of the other process into its address space
  - It is similar to the PVAS, but the implementation is different
  - SMARTMAP accelerates MPI Intra-node communication for transferring contiguous data
- User-mod Memory Registration
  - UMR is a function of Mellanox IB, which makes it possible to transfer non-contiguous data through one RDMA operation
  - UMR accelerates MPI inter-node communication using derived data types [Mingzhe et al., IEEE Cluster'15]

Summary

- We introduced PVAS task model
  - A new task model for efficient parallel processing on many-core systems
  - PVAS removes overheads for crossing address space boundary from intra-node communication by running the parallel processes within the same address space
- We optimized MPI intra-node communication by using PVAS facility
  - We optimized contiguous and non-contiguous data transfers in Open MPI
  - PVAS implementation outperforms SM implementation