OpenSHMEM as a Portable Communication Layer for PGAS Models

A Case Study with Coarray Fortran

Naveen Namashivayam, Deepak Eachempati, Dounia Khaldi, Barbara Chapman

University of Houston

{nravi, dreachem, dounia, chapman}@cs.uh.edu

September 10, 2015

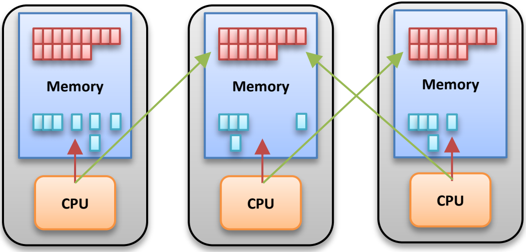
Introduction

- PGAS Programming Models
  - Emerging model in recent years
  - Focus on addressing programming challenges for scalable parallel systems
  - Language- and Library-based
- OpenSHMEM
  - Library-based light weight PGAS Model
  - Highly portable library – available in different systems
- Can OpenSHMEM be efficiently used as a runtime layer for other PGAS models – especially language-based PGAS models?

What ?	Why ?	How ?	What now?
Introduction to PGAS, CAF and OpenSHMEM	Using OpenSHMEM as transport layer for CAF	Implement CAF over OpenSHMEM	Experimental Analysis and Future Work

Partitioned Global Address Space (PGAS)

- A new class of languages and libraries
- Alternative to MPI for programming large-scale parallel systems
- Attempts to combine both the distributed and shared memory systems
- Logical shared memory – across distributed systems



- PGAS Languages – Unified Parallel C (UPC), Coarray Fortran (CAF), Chapel and X10, ….
- PGAS Libraries – OpenSHMEM, Global Arrays, MPI-3.0(RMA), ….

OpenSHMEM - Overview

- PGAS Library
- Culmination of unification effort among different SHMEM implementations

OpenSHMEM (GASNet)	Cray SHMEM	SGI SHMEM	OSU SHMEM (MVAPICH2-X)
OpenSHMEM (UCCS)	OSHMPI (over MPI-3.0)	Open MPI SHMEM	Portals SHMEM
Quadrics SHMEM	Mellanox Scalable SHMEM	Intel SHMEM (libfabrics)	other implementations

- Light-weight and portable library
- SPMD-like style of programming
- Properties available in recent OpenSHMEM-1.2 specifications
    - Symmetric Data Object management
    - Remote Read and Write using Put and Get operations
    - Barrier and Point-Point synchronization
    - Atomic memory operations and
    - Collective reduction and broadcast operations

Coarray Fortran (CAF) - Overview

- PGAS Language
- Integral part of Fortran 2008 standards
- As a language depends on compiler

- Properties of Fortran 2008 CAF standards
  - Symmetric memory object management
  - Remote Read and Write operations
  - Barrier and Point-Point synchronization
  - Critical section and locks
  - Atomic memory operations

Cray CAF (DMAPP)

Intel CAF (MPI)

OSU CAF (MVAPICH2-X)

UHCAF (GASNet / ARMCI)

Open Coarrays (GASNet / MPI-3.0)

CAF and OpenSHMEM Example

Example CAF program

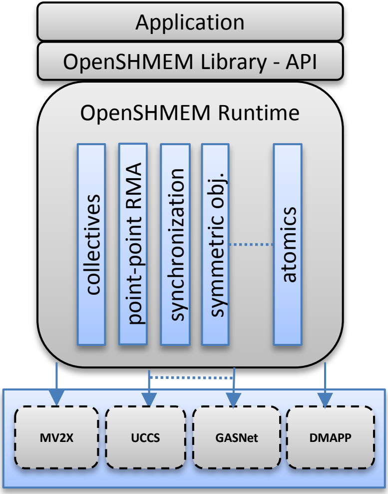
```
integer :: i

integer :: np
integer :: me

integer, allocatable :: coarray_x(:)[:]
integer, allocatable :: coarray_y(:)[:]
allocate ( coarray_x (4)[*] )
allocate ( coarray_y (4)[*] )

np   = num_images()
me   = this_image()

do i = 1, 4
      coarray_x (i)= me
      coarray_y (i) = 0
end do

coarray_y(2) = coarray_x(3)[4]

sync all
```

Example OpenSHMEM program

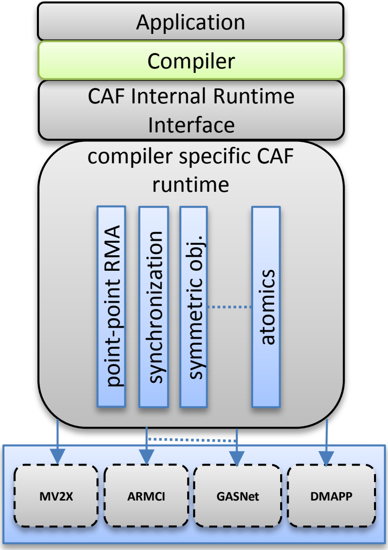
```
start_pes (0);

int i;
int np, me;

int *coarray_x, *coarray_y;
coarray_x = shmem_malloc (4 * sizeof (int));
coarray_y = shmem_malloc (4 * sizeof (int));

np   = shmem_my_pe();
me   = shmem_n_pes();

for ( i = 1; i <= 4; i++) {
      coarray_x [ i ] = me;
      coarray_y [ i ] = 0;
}

shmem_int_get(&coarray_y[2], &coarray_x[3], 1, 3);

shmem_barrier_all();
```

Communication Layer – System Interfaces

Simple OpenSHMEM architecture

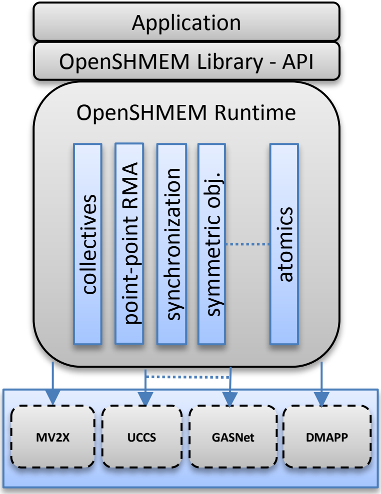


Simple CAF architecture

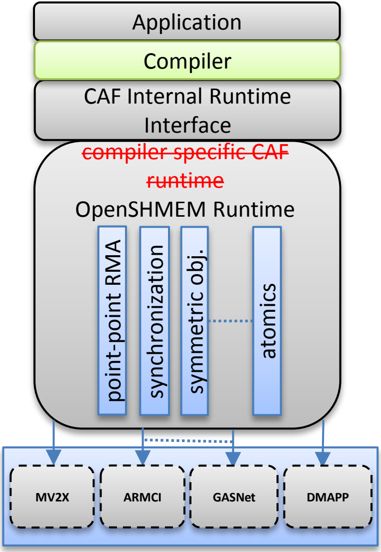


Communication Layer – System Interfaces

Simple OpenSHMEM architecture



Simple CAF architecture



Motivation – Functionality similarities

- Functionality and Features – Similar features as in CAF

Properties	CAF	OpenSHMEM
Symmetric data allocation	allocate	shmem_malloc()
Total image count	num_images()	shmem_n_pes()
Current image ID	this_image()	shmem_my_pe()
Collectives – reduction*	co_***operator***	shmem_***operator***_to_all()
Collectives – broadcast*	co_broadcast	shmem_broadcast
Barrier Synchronization	sync_all	shmem_barrier_all()
Remote memory Put/Get	[]	shmem_put/get
Single dimensional strided put	[]	shmem_iput/iget
Multi dimensional strided put	[]	X
Remote Locks	*lock*	X
Other Properties	……….	………

* Future revision of Fortran standards – Fortran 2015

Motivation – Performance

- Relative comparison with other PGAS libraries and PGAS runtimes
- Compared OpenSHMEM, MPI-3.0 and GASNet in 2 different systems
- PGAS Microbenchmark Test Suite from University of Houston

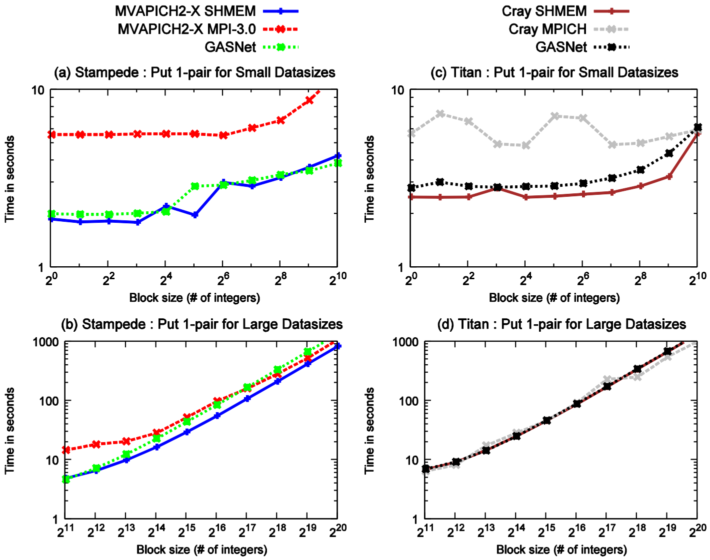
- System – 1: Stampede(TACC) – Infiniband Cluster

Nodes (cores)	OpenSHMEM	MPI-3.0 (RMA)	GASNet (conduit)
6400 (16)	MVPICH2-X SHMEM	MVPICH2-X	GASNet 1.24.0 (IB)

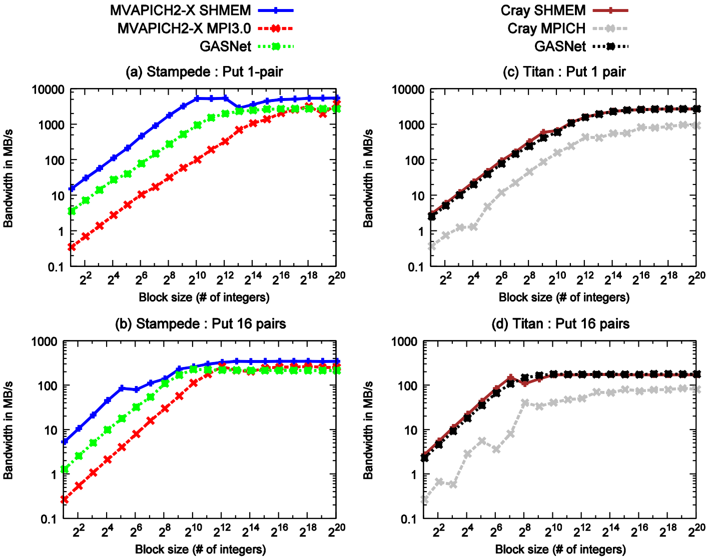
- System – 2: Titan(Cray XK7, ORNL) – Gemini system

Nodes (cores)	OpenSHMEM	MPI-3.0 (RMA)	GASNet (conduit)
18688 (16)	Cray SHMEM in Cray MPT	Cray MPICH in Cray MPT	GASNet 1.24.0 (Gemini)

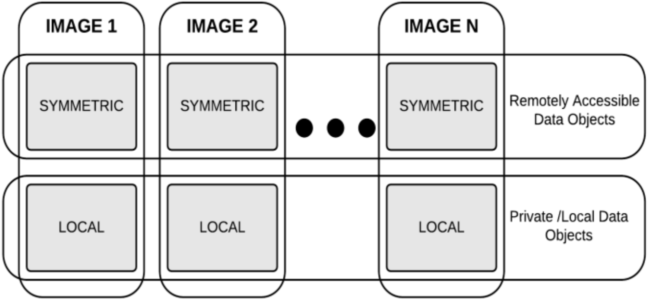
Motivation – Comparing Latency



Motivation – Comparing Bandwidth



Implementing UHCAF over OpenSHMEM
Symmetric Data Allocation



- How to make the data remotely accessible?
  - CAF – **save** or **allocatable**
  - OpenSHMEM – **static/global** variables or use **shmalloc()**
- Symmetric Heap : **allocate** and **deallocate** -> **shmalloc() and shfree()**
  - Allocate as a single chunk initially
  - Allocate with **shmalloc()** when it is necessary

Implementing UHCAF over OpenSHMEM
Remote Memory Accesses

- Remote Memory Access (RMA) in OpenSHMEM is done by **shmem_putmem()** and **shmem_getmem()**
- Similar properties in CAF, but not exactly matching
- Same location and from the same image
  - CAF ensures it is ordered
  - OpenSHMEM allows out-of-order with respect to other remote access

1. Local Completion

coarray_y(:)[2] = coarray_x(:)
coarray_x(:) = 0

2. Remote Completion

coarray_a(:)[2] = coarray_b(:)
coarray_c(:) = coarray_a(:)[2]

- For RMA Put : **shmem_putmem()** followed by **shmem_quiet()**
- For RMA Get : **shmem_getmem()** preceded by **shmem_quiet()**

Implementing UHCAF over OpenSHMEM

Strided Data Transfer – Single Dimensional Arrays

- Element-wise strided data of same basic data type (yes, with TYPE_iput/ iget)

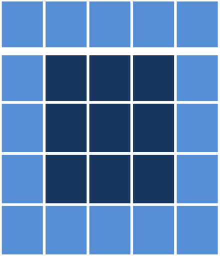
A[10] =	A [0]	A [1]	A [2]	A [3]	A [4]	A [5]	A [6]	A [7]	A [8]	A [9]	✓

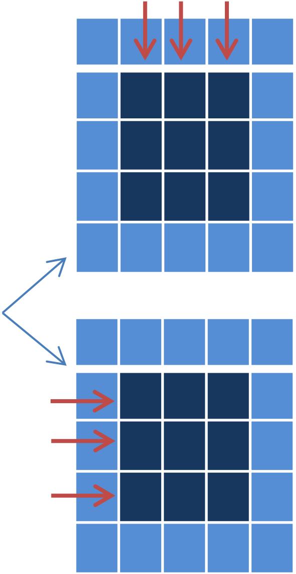
- Block-wise strided data of same basic data type

A[10] =	A [0]	A [1]	A [2]	A [3]	A [4]	A [5]	A [6]	A [7]	A [8]	A [9]	✘

Implementing UHCAF over OpenSHMEM
Strided Data Transfer – Multidimensional Arrays

Matrix Oriented Strides





<u>Algorithm 1</u>
Naïve Algorithm
shmem_putmem()
shmem_getmem()
All possible stride size

<u>Algorithm 2</u>
2-DIM Algorithm
shmem_TYPE_iput()
shmem_TYPE_iget()
Not all possibilities

- Size is even
- Size if odd (no 1 byte transfer in SHMEM)

Performance Evaluations

- University of Houston – CAF Test Suite
  - PGAS Microbenchmark Test Suite*
  - Distributed Hash Table Benchmark and^
  - Himeno Benchmark^

System	Stampede	Cray XC30
Interconnect	Infiniband	Aries
Native CAF Implementation	-	Cray Compiler (Cray CAF)
UHCAF over GASNet (conduit)	GASNet-1.24.0 (IB)	GASNet-1.24.0 (Aries)
UHCAF over OpenSHMEM	MVAPICH2-X	Cray SHMEM

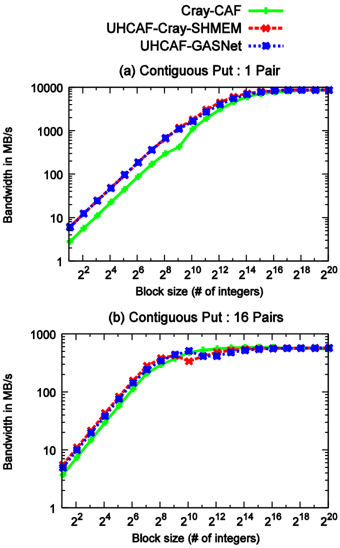
* https://github.com/uhhpctools/pgas-microbench.git
^ https://github.com/uhhpctools/caf-testsuite.git

OpenSHMEM as runtime for PGAS Languages

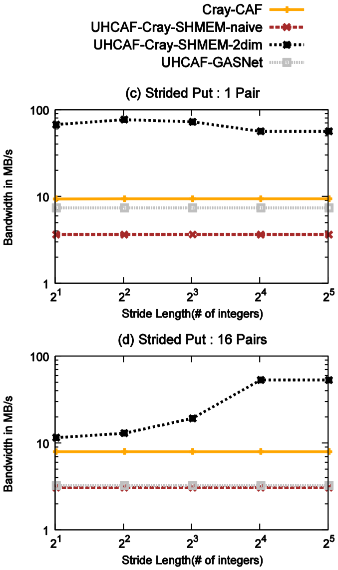
Experimental Analysis – Contiguous and Strided

- PGAS Microbenchmark - Cray XC30 Aries System

Put Bandwidth



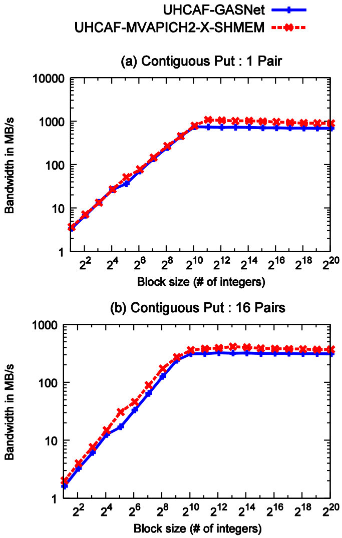
2-DIM Strided Put Bandwidth



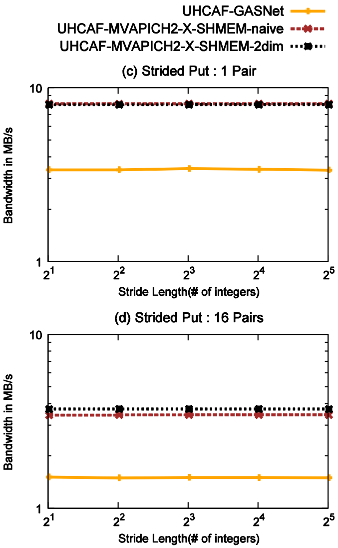
Experimental Analysis – Contiguous and Strided

- PGAS Microbenchmark - Stampede Infiniband Cluster

Put Bandwidth



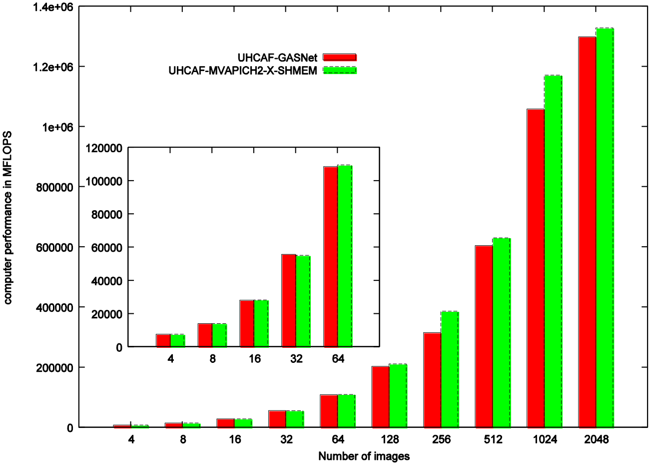
2-DIM Strided Put Bandwidth



OpenSHMEM as runtime for PGAS Languages

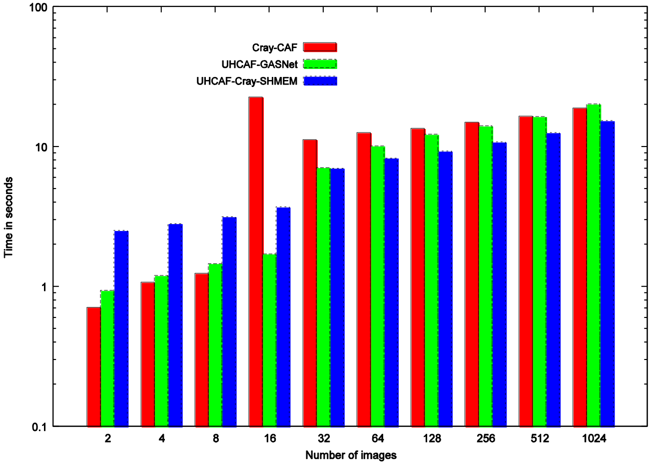
Experimental Analysis – Himeno Benchmark

- 2-Dimensional data decomposition and strided data communication
- Advantage in using Naïve strided algorithm (**shmem_putmem**)
- Average 6% improvements with UHCAF(MV2-X SHMEM) and maximum 22% improvements



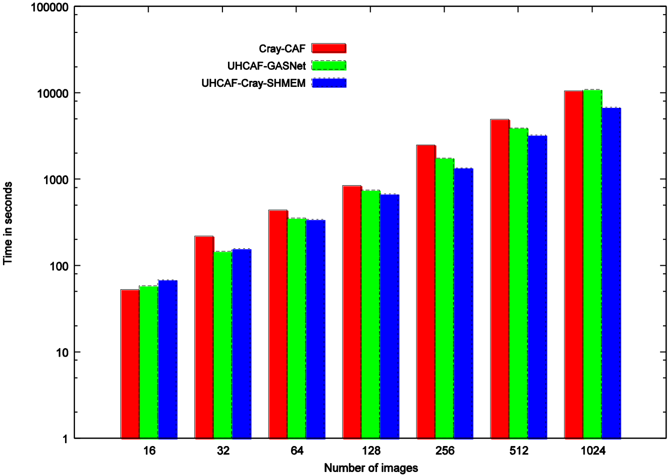
Experimental Analysis – Distributed Hash Table

- Titan Supercomputer, Distributed Hash Table Benchmark
- Latency measurement
- UHCAF(Cray SHMEM) 28% faster than Cray CAF and 18% faster than UHCAF(GASNet)



Experimental Analysis – Locks

- PGAS Microbenchmark Test Suite - Titan (OLCF, ORNL)
- Latency measurement for Locking and Unlocking Operation
- UHCAF(Cray SHMEM) is 22% faster than Cray CAF and 10% faster than UHCAF(GASNet)



OpenSHMEM as runtime for PGAS Languages

Conclusion

- Implemented UHCAF over OpenSHMEM and evaluated performance
  - UHCAF(GASNet) Vs UHCAF(MV2-X SHMEM)
  - UHCAF(GASNet) Vs Cray CAF Vs UHCAF(Cray SHMEM)
- Put Bandwidth – 18% improvement
- 2-Dimensional Strided Put Bandwidth
  - 3X improvements – UHCAF(Cray SHMEM) Vs Cray CAF
  - 9X improvements – UHCAF(Cray SHMEM) Vs UHCAF(GASNet)
- Why OpenSHMEM ?
  - Availability in many systems
  - System specific optimizations
  - OpenSHMEM specifications committee – new extensions
- Future Work
  - Perform tests for strong scaling
  - Optimize Symmetric memory heap maintenance
  - Optimize using native OpenSHMEM features like **shmem_ptr**

- Acknowledgements

  OLCF – ORNL (Titan) TOTAL(Cray XC30) TACC (Stampede)

OpenSHMEM as a Portable Communication Layer for PGAS Models

A Case Study with Coarray Fortran

Naveen Namashivayam, Deepak Eachempati, Dounia Khaldi, Barbara Chapman

University of Houston

{nravi, dreachem, dounia, chapman}@cs.uh.edu

September 10, 2015

Backup Slides

Conclusion

- Implemented UHCAF over OpenSHMEM and evaluated performance for:
    - UHCAF(GASNet) vs UHCAF(MV2-X SHMEM)
    - UHCAF(GASNet) vs Cray CAF vs UHCAF(Cray SHMEM)
- Around 18% improvement for Put Bandwidth with UHCAF(Cray SHMEM) against Cray CAF and UHCAF(GASNet)
- For 2-D Strided Put Bandwidth with UHCAF(Cray SHMEM) against Cray CAF we get 3X improvements and 9X improvements against UHCAF(GASNet)
- Similar improvements in DHT and Himeno benchmarks
- OpenSHMEM is light weight and has minimum overhead and can be used to implement other PGAS Language based models

Implementing Strided Data Transfer

Algorithm 1

```
for ( i = 1, i ≤ nelems, i++) do
   shmem putmem(dest_ptr, src_ptr, blksize, pe_id)
   dest_ptr += dest_stride
   src_ptr  += src_stride
end for
```

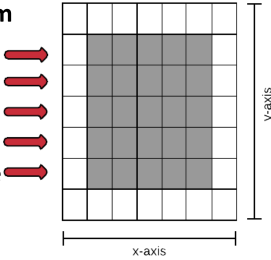
Algorithm 2

```
if (check whether shmem_iput32 can be used)
  if (check whether shmem_iput128 can be used)
    call shmem_iput128 for each 16 byte chunk
    update dest_ptr and src_ptr
  if (check whether shmem_iput64 can be used)
    call shmem_iput64 for each remaining 8 byte chunk
    update dest_ptr and src_ptr
  if (check whether shmem_iput32 can be used)
    call shmem_iput32 for each remaining 4 byte chunk
else
  use ALGORITHM 1
```

Implementing Strided Data Transfer

```
for ( i = 1, i ≤ nelems, i++) do
   shmem putmem(dest_ptr, src_ptr, blksize, pe_id)
   dest_ptr += dest_stride
   src_ptr  += src_stride
end for
```

- Implementing using multiple **shmem_putmem**
- Call **shmem_putmem** along the y-axis
- Number of **shmem_putmem** will be equal to the number of elements (**nelems**)
- More number of SHMEM calls
- Normal implementation – support all possible **src_stride**, **dest_stride** and **blksize**



Implementing Strided Data Transfer

Algorithm 2

```
if (check whether shmem_iput32 can be used)
  if (check whether shmem_iput128 can be used)
    call shmem_iput128 for each 16 byte chunk
    update dest_ptr and src_ptr
  if (check whether shmem_iput64 can be used)
    call shmem_iput64 for each remaining 8 byte chunk
    update dest_ptr and src_ptr
  if (check whether shmem_iput32 can be used)
    call shmem_iput32 for each remaining 4 byte chunk
else
  use ALGORITHM 1
```

- Implementing using multiple **shmem_iput128, shmem_iput64,** and/or **shmem_iput32** calls
- Call ***iput*** routines along the x-axis
- Shift between different types
- Suitable for small data sizes
- Does not support all the values for src_stride, dest_stride and blksize