An OpenSHMEM Implementation for the Adapteva Epiphany Coprocessor

James Ross
james.a.ross176.civ@mail.mil
US Army Research Laboratory

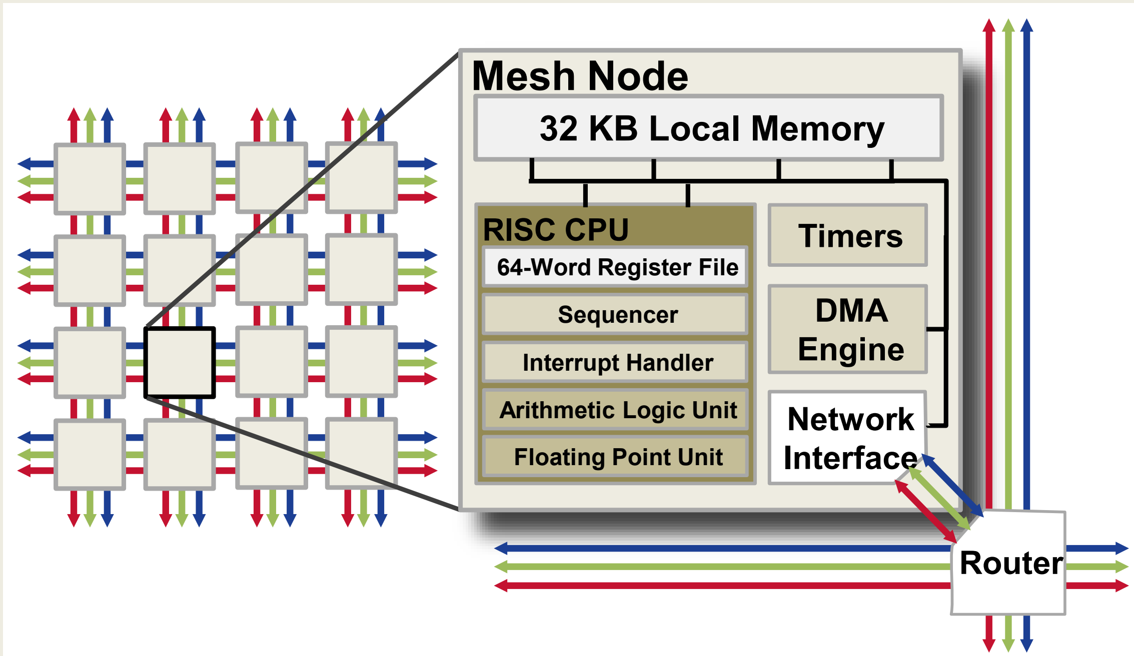
David Richie
drichie@browndeertechnology.com
Brown Deer Technology

Outline

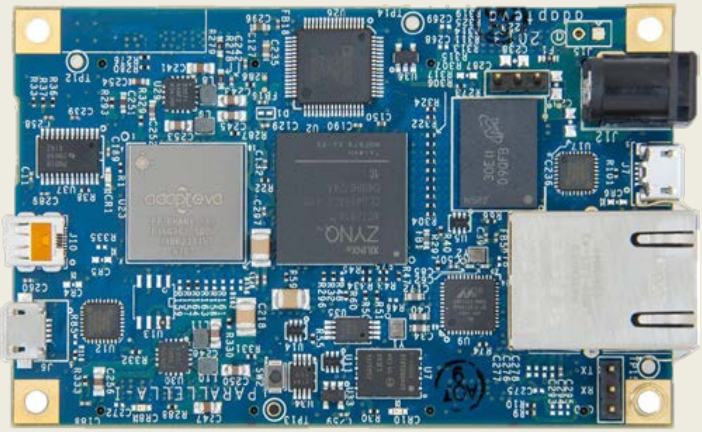
- Epiphany Architecture
- Programming Challenge
- Hardware/Software Setup
- OpenSHMEM Interface
- Benchmark Performance
- Conclusions & Future Work

Adapteva Epiphany Architecture

- Design emphasizes simplicity, scalability, power-efficiency
- 2D array of RISC cores, 2D Network on Chip (NoC)
- 512 KB shared global scratch memory (32 KB/core, Epiphany-III)
- Fully divergent cores
- Minimal un-core functionality, e.g., no data or instruction cache
- Existing design scales to thousands of cores
- High performance/power efficiency ~50 GFLOPS/W (Epiphany-IV)



Hardware/Software Setup



Hardware

- Parallella development board ($99 'Micro-Server')
- Dual-core ARM host processor
- 16-core Epiphany-III co-processor (@ 600 MHz)
  - 19.2 GFLOPS
  - 76.8 GB/s bandwidth

Software at time of publication

- OpenSHMEM 1.3 (our implementation)
- Parallella Linux image (2015.1)
  - Epiphany SDK 2015.1
  - GCC 4.8
- Brown Deer Technology
  - COPRTHR-2 Beta

Programming Challenge

<u>Programming challenge:</u>

- Can the OpenSHMEM API be efficiently used with embedded PGAS architectures like Epiphany?
- Distributed memory mapped cores with 32KB local memory per core
- Best viewed as a "distributed cluster on chip"
- Non-uniform memory access (NUMA) to mapped local memory
- Typical many-core programming models (OpenMP, OpenCL, CUDA) leverage common cache / memory hierarchy for SMP
- Here, there is no hardware data/instruction cache

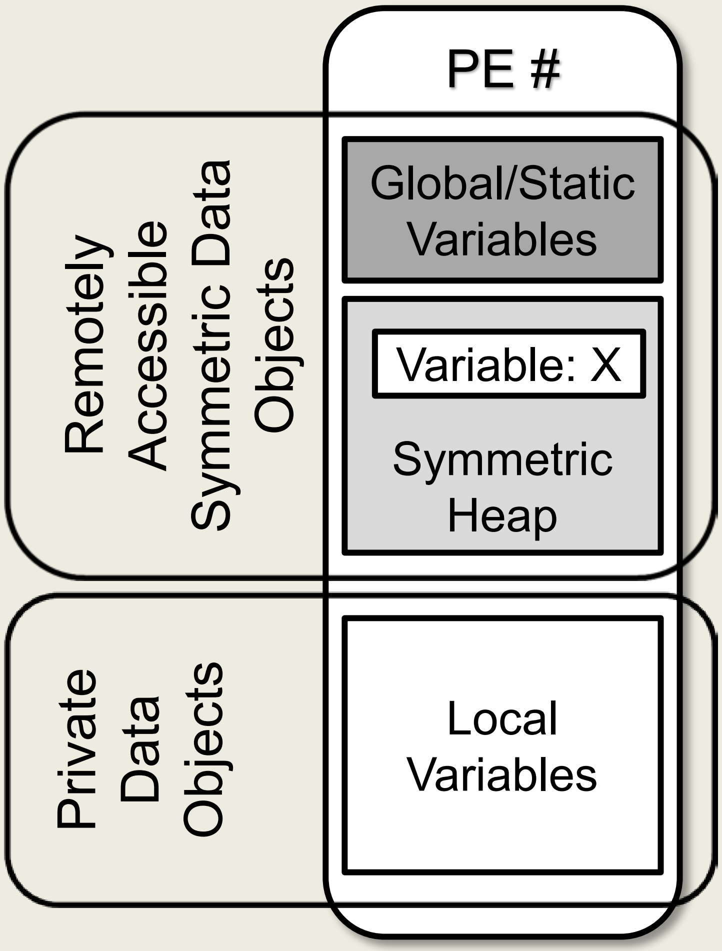
<u>Two key observations:</u>

  - Architecture resembles cluster on chip
  - Inter-core data movement is key to performance

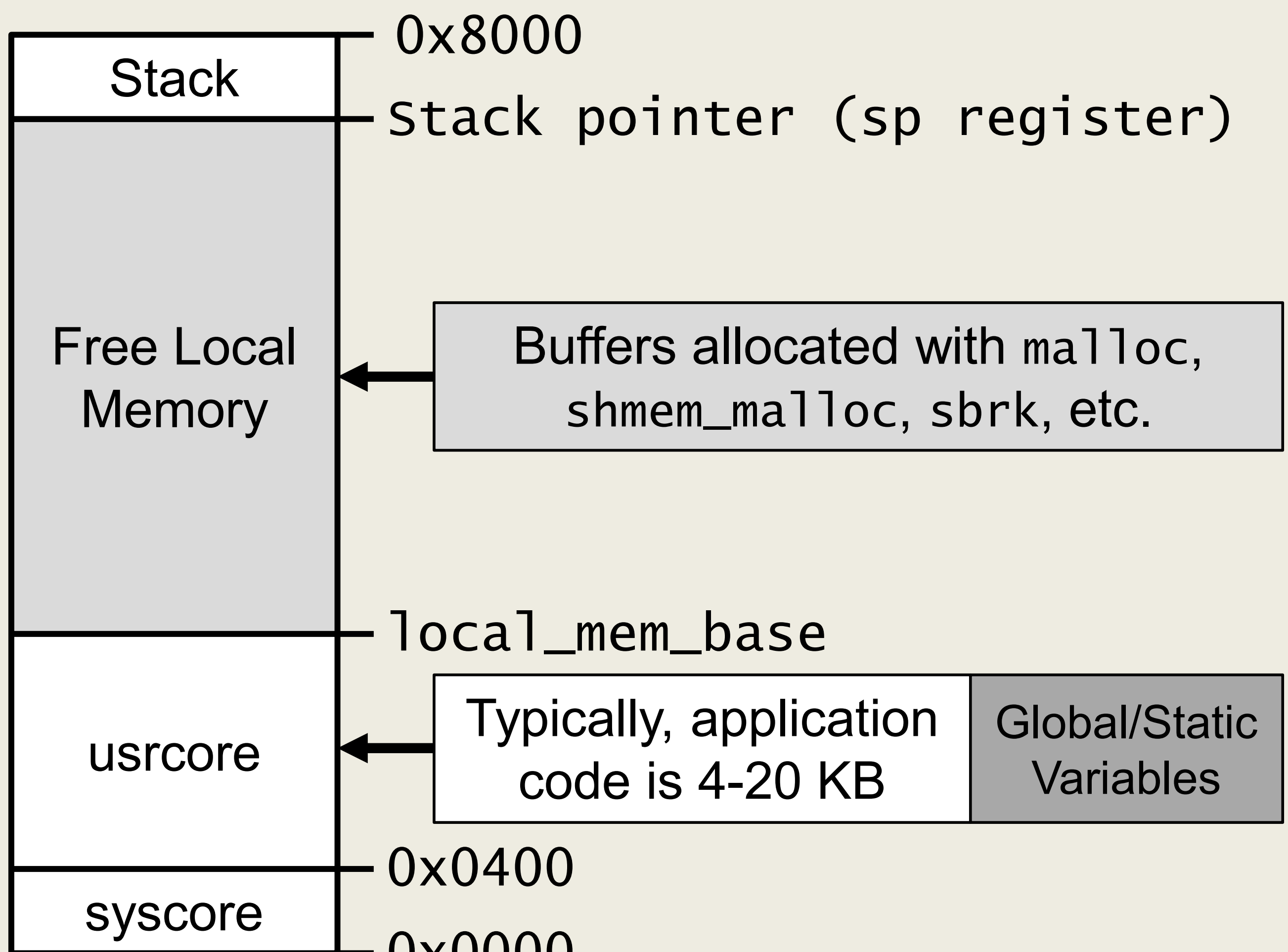
<u>Proposition:</u> Use SHMEM as device-level programming model

We are using the OpenSHMEM API to enable the device-level parallelism of a 2D RISC array processor

PGAS Memory Model



Epiphany-III Core Memory Layout (Typical)



OpenSHMEM

Why OpenSHMEM for PGAS?

- Unified Parallel C, Split-C, Fortress, Coarray Fortran, Chapel, X10 require building/modifying a compiler instead of a library

So why not MPI?

- MPI substantially larger library with higher-level abstraction and two-sided communication means more code (source and binary)
- OpenSHMEM has improved data referencing semantics and reduced interface complexity
  - No message tags, no status, no special types, no I/O
  - MPI makes no assumptions for symmetric memory allocation. Requires correct remote address calculation

Why not eSDK (e-lib)? (See Supplemental Slides)

- Non-standard, less portable code
- No abstraction between physical row/column and virtual process
- Memory management, collectives, reductions, atomics missing

U.S. ARMY RDECOM

OpenSHMEM Design

ARL

Objectives:

- Maintain/conform to OpenSHMEM API
- Complete implementation of API
- Design must lead to efficient implementation for architecture
  - Tree-based algorithms, high performance, small code size

Challenges:

- Significant local memory constraint (32KB per core)
  - Typical OpenSHMEM uses significantly larger amounts of memory
  - Entire library is 200+ routines, typical applications use just several

Boons:

- Extremely low latency on-chip inter-core communication
- No middleware (transparent network interface for memory access)
- Simple DMA engine for asynchronous communication

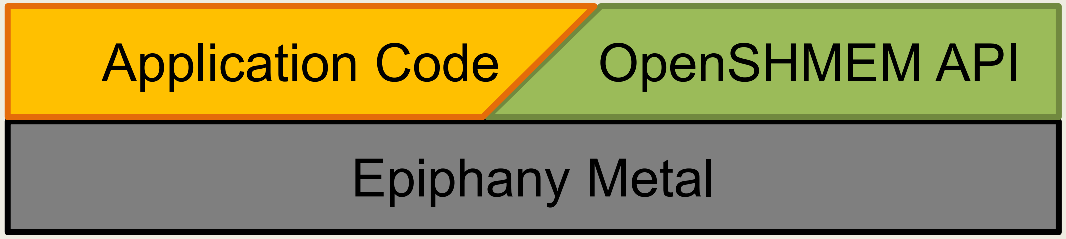
U.S. ARMY RDECOM ARL

Implementation

- Currently implemented in header-only library (~1800 LOC C/inline asm)
  - Compiler optimizes for constant arguments, reduces application code size
  - Data type differentiation handled by macros
- No software stack layers for low-level communication primitives
  - Typical implementations based on additional software stacks (GASNet, network interfaces, other software layers)
- Optimized for low latency performance, efficient hierarchical communication (dissemination barrier, recursive-doubling, etc)

Implementation

OpenSHMEM / Application / Epiphany software stack:

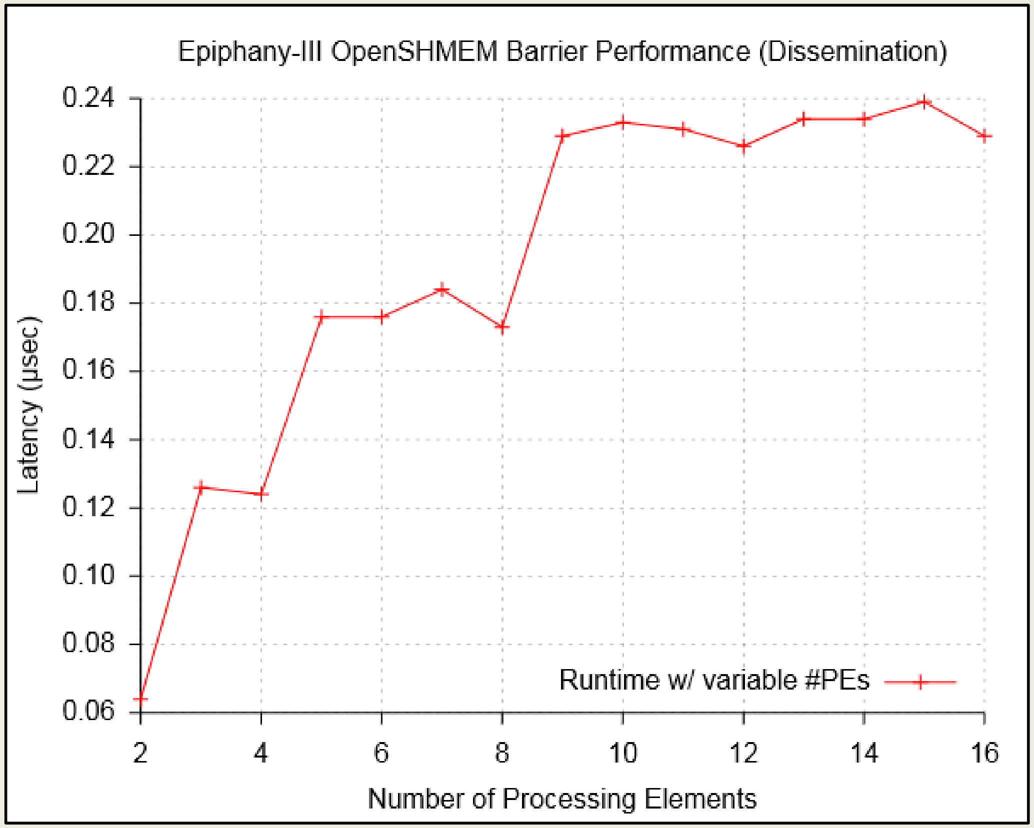


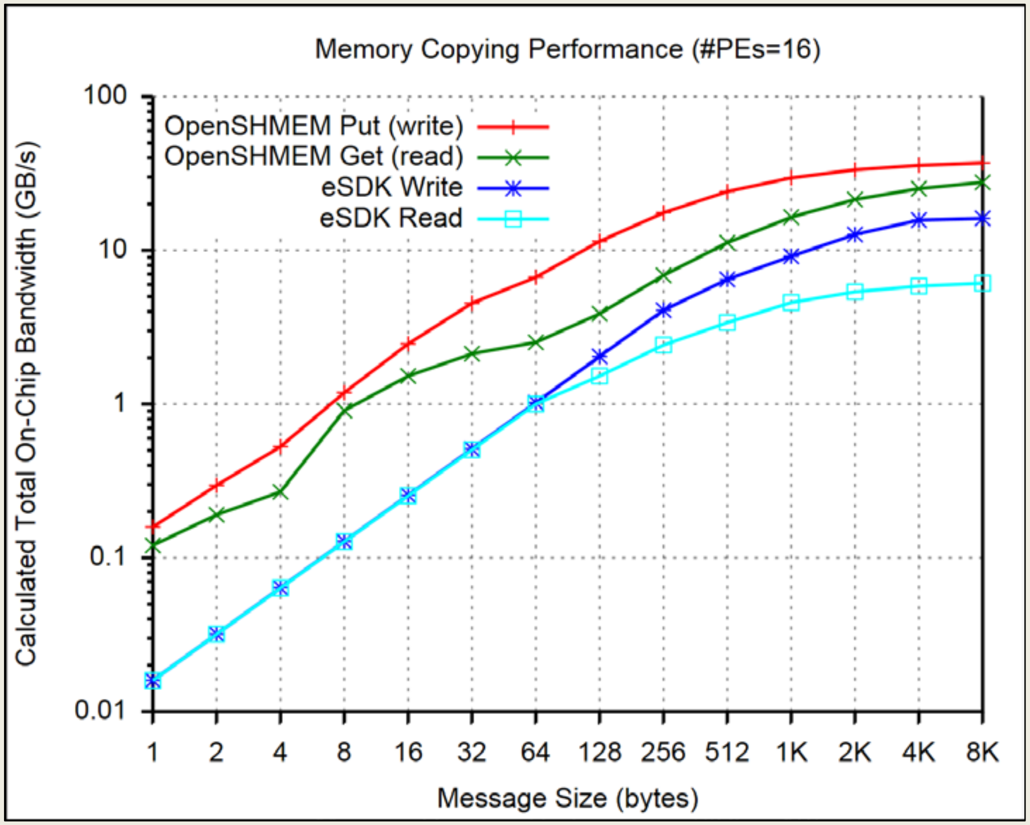
- OpenSHMEM library replaces device-side eSDK (e-lib) code.
- No consideration for coprocessor offloading (e-hal, libcoprthr)
- Message passing between cores literally corresponds to load or store instruction

...not this:

Performance Results

- Few directly comparable routines between eSDK (e-lib) and OpenSHMEM



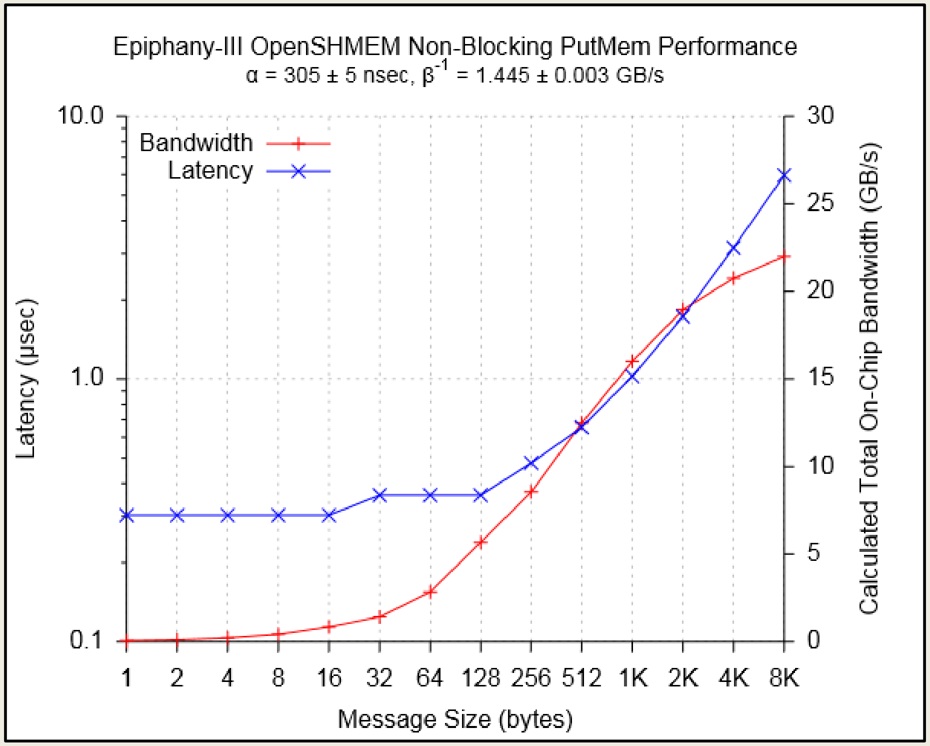


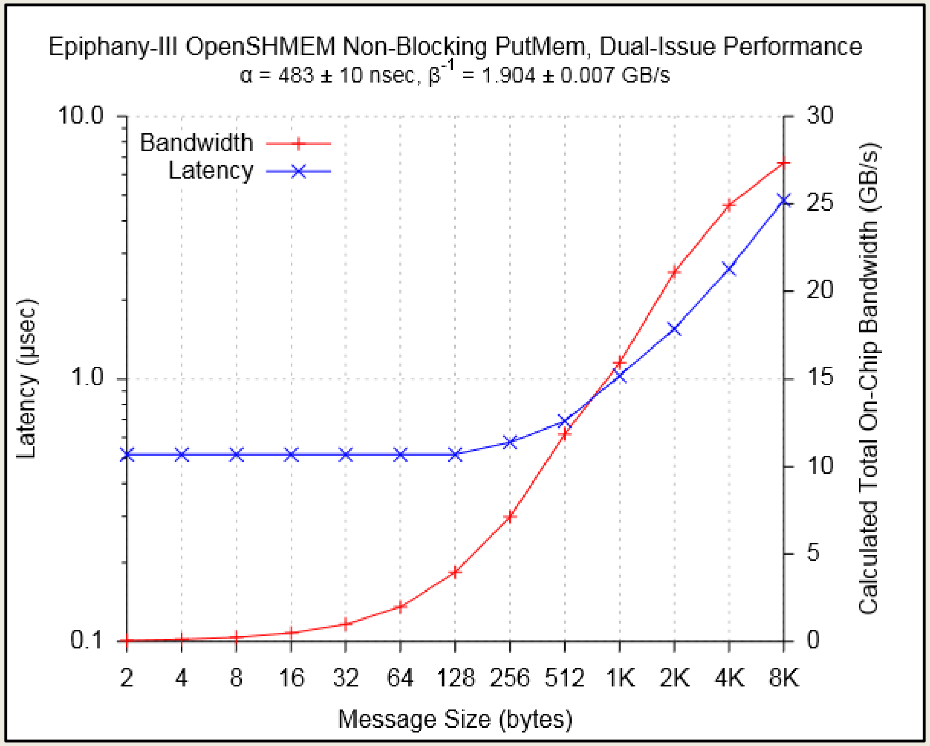
- **>9.1x speedup** for software barrier
- **20x speedup** for fixed 16 core case (hardware WAND barrier = 0.1 µsec)
- eSDK barrier = **2.0 µsec**

- **2.1-9.9x speedup** for all message sizes
- Peak Put bandwidth = **2.4 GB/s** per core
- Corresponds to alternating ldrd/strd

Performance Results

- Non-blocking DMA performance (automatic dual-issue DMA scheduling)
  - Universally underperforms blocking message passing
  - Dual-issue DMA increases latency for marginal gain for large transfers
  - Hardware errata on E3 prevents full DWORD/clock (4.8 GB/s)



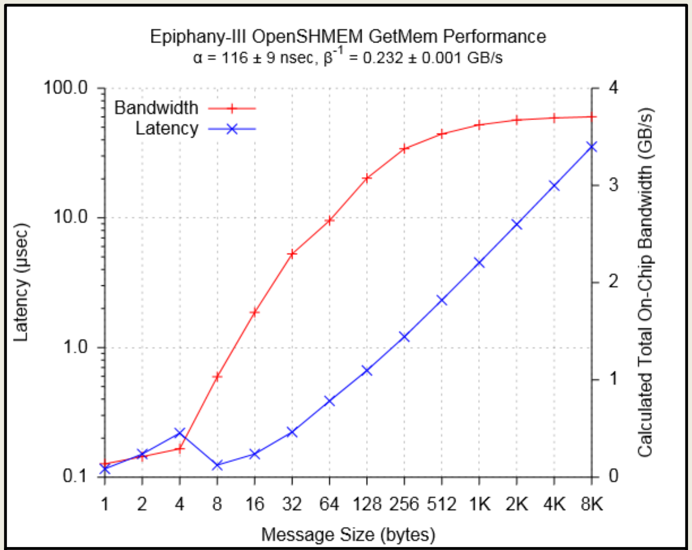


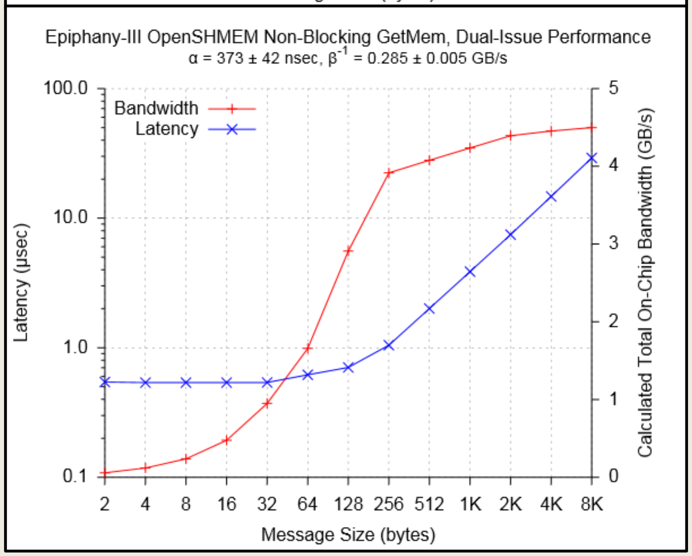
- **1.45 GB/s** peak performance per core

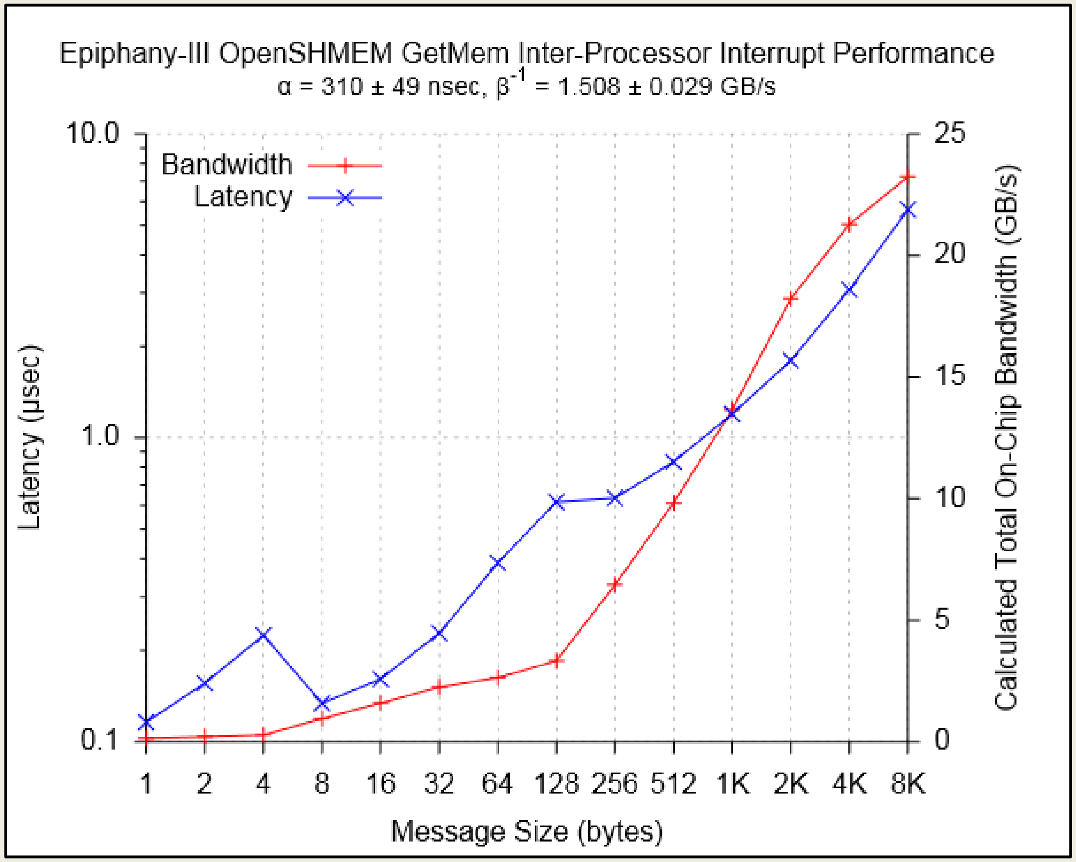
- **1.91 GB/s** per core peak performance
- Code performs *shmem_putmem_nbi* twice with half total data

Performance Results

- *Get performance worse than Put* because read network requires response/stall
- Use Inter-Processor Interrupt (experimental feature) to cause remote Put



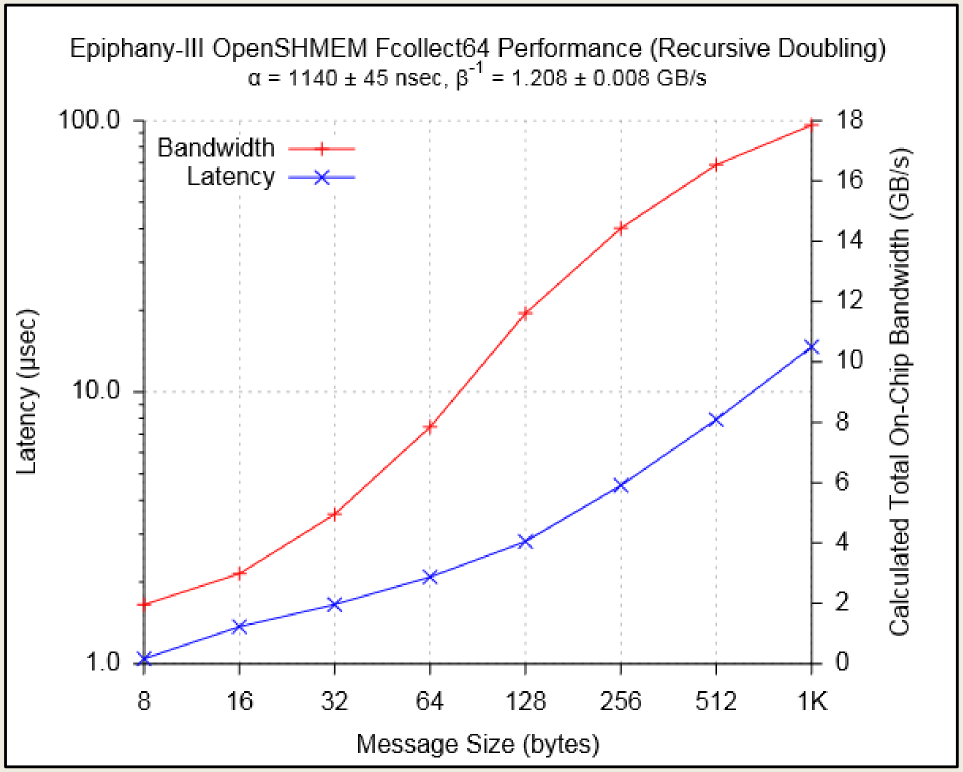


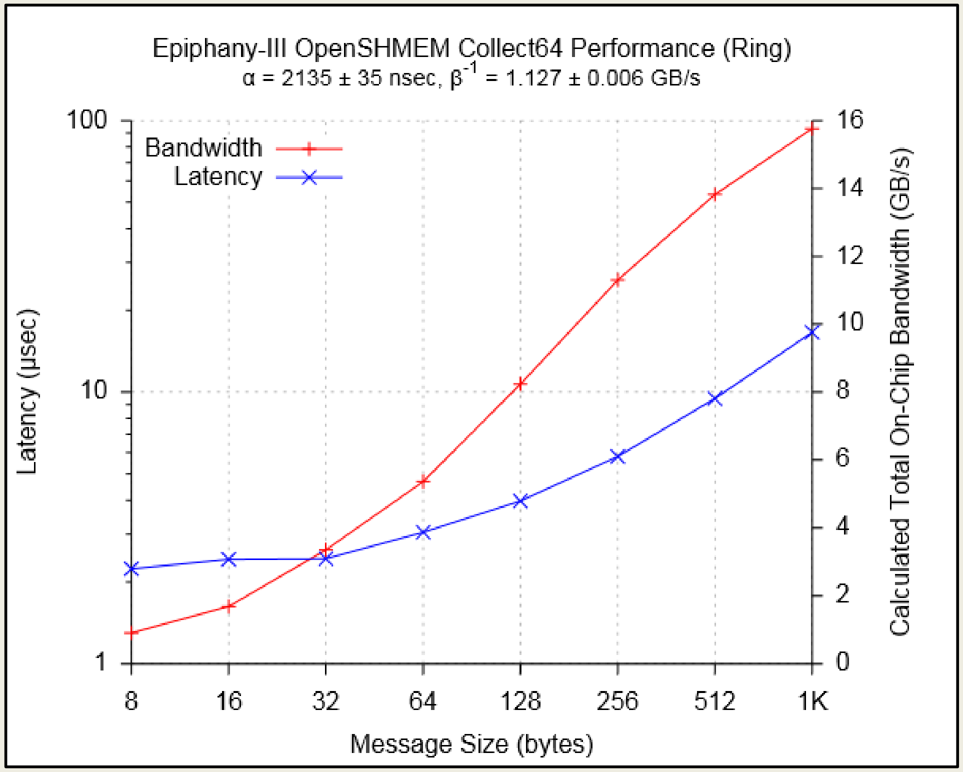


- **200-300 MB/s** per core peak Get performance
- **1.5 GB/s** peak performance with **IPI**
  - Crossover at N >= 64 bytes

Performance Results

- Collect peak bandwidth good, but with relatively high latency for small sizes
  - Concatenates blocks of data from multiple PEs to all PEs

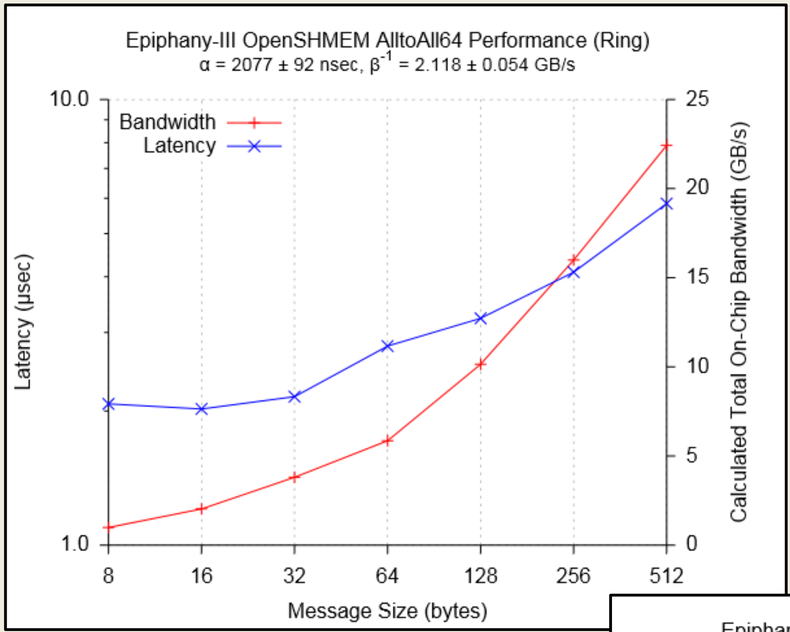




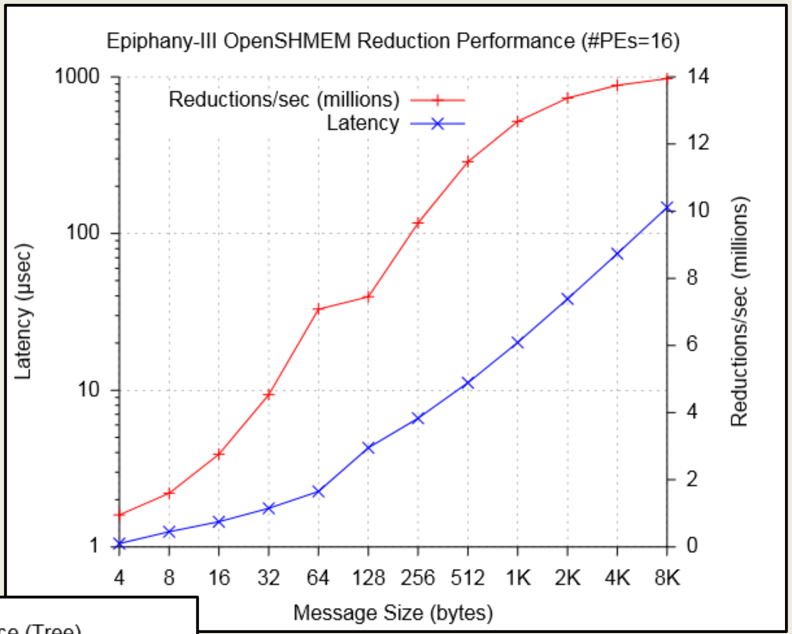
- **1.21 GB/s** peak performance per core
- **1.13 GB/s** peak performance per core

Performance Results

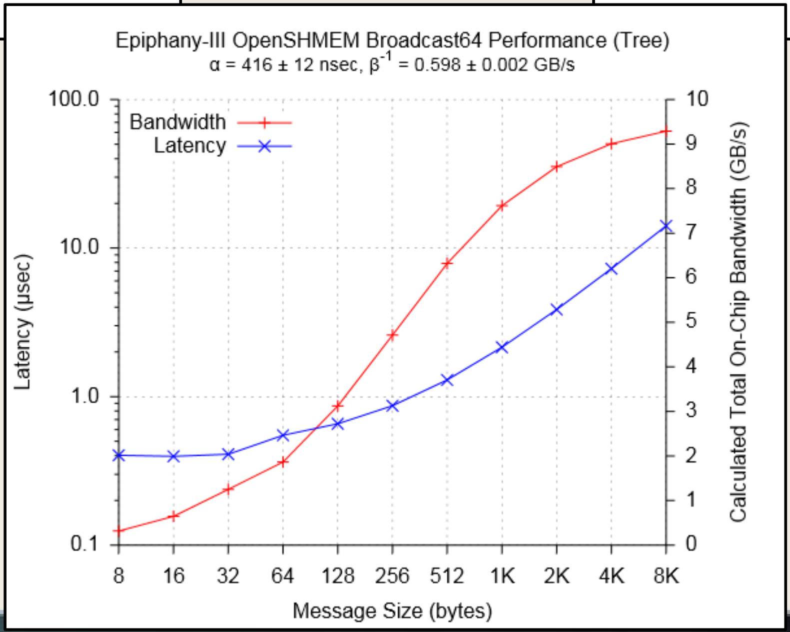
- Collective operations achieve high performance



AlltoAll Peak = **2.12 GB/s**



14 MegaReductions/s



Effective Broadcast = Peak Put / log2(NPES) = **600 MB/s**

U.S. ARMY RDECOM ARL

Example

```c
#include <shmem.h>
#define N 3040

long pSync[SHMEM_REDUCE_SYNC_SIZE] = { SHMEM_SYNC_VALUE };
int pWrk[SHMEM_REDUCE_MIN_WRKDATA_SIZE];
extern float dot_product(float* a, float* b, int n); // optimized dot product routine**
float c = 0.0f; // collective result

int main(void)
{
   int nd8m1 = (N >> 3) - 1; // N/8 - 1
   shmem_init();

   int me = shmem_my_pe();
   int npes = shmem_n_pes();

   // allocation
   float* a = (float*)shmem_malloc(N*sizeof(float));
   float* b = (float*)shmem_malloc(N*sizeof(float));

   // initialization...
   for (int i = 0; i < N; i++) a[i] = b[i] = 1.0f;

   shmem_barrier_all();

   c = dot_product(a, b, nd8m1); // assembly optimized routine
   shmem_float_sum_to_all(&c, &c, 1, 0, 0, npes, pWrk, pSync);

   shmem_free(b);
   shmem_free(a);

   shmem_finalize();
}
```

code	size (bytes)
syscore	1024
dot_product	208
main*	1916

Combined, this code achieves 87% peak performance for 16 cores:

16.8 GFLOPS

67.3 GB/s

Nelements = 48,640 (16*3040)

* Includes inlined *shmem_*(*init*, *my_pe*, *n_pes*, *malloc*, *barrier_all*, *float_sum_to_all*, *free*, *finalize*) routines

** Assembly optimized *dot_product* available in supplemental slides

U.S. ARMY RDECOM

Conclusions & Future Work

ARL

- This work reported on full OpenSHMEM 1.3 implementation for embedded PGAS architecture, Adapteva's Epiphany
- OpenSHMEM provides an effective programming model for this class of architecture
- Header-only implementation enables compiler optimizations for program size and performance
- Performance exceeds eSDK (e-lib) in speed, code size
- Performance/latencies in another performance class than traditional implementations due to on-chip network – a favorable comparison

- Distribution for community input (on ARL github account)
- Improve memory management with COPRTHR-2
- Recommendations to OpenSHMEM standard committee for embedded PGAS architectures

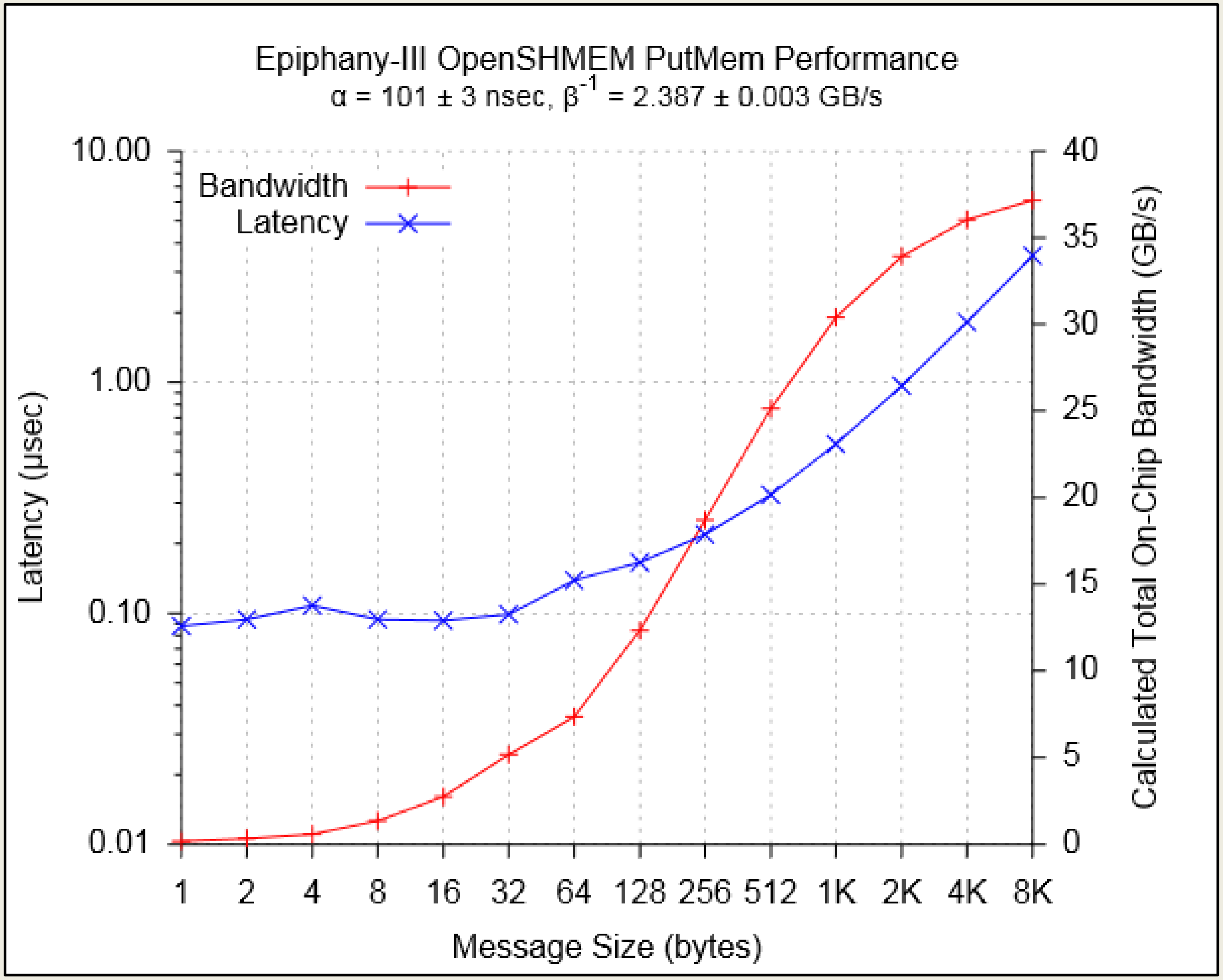
Resources

- US Army Research Laboratory GitHub
  - https://github.com/USArmyResearchLab
  - Intend to release ARL OpenSHMEM for Epiphany with example codes and benchmarks as free and open source software
- COPRTHR 2.0 Resources/Download
  - http://www.browndeertechnology.com/resources_epiphany_developer_coprthr2.htm
- Parallella (Epiphany-III host platform)
  - http://parallella.org/

Questions: james.a.ross176.civ@mail.mil

Supplemental Slides

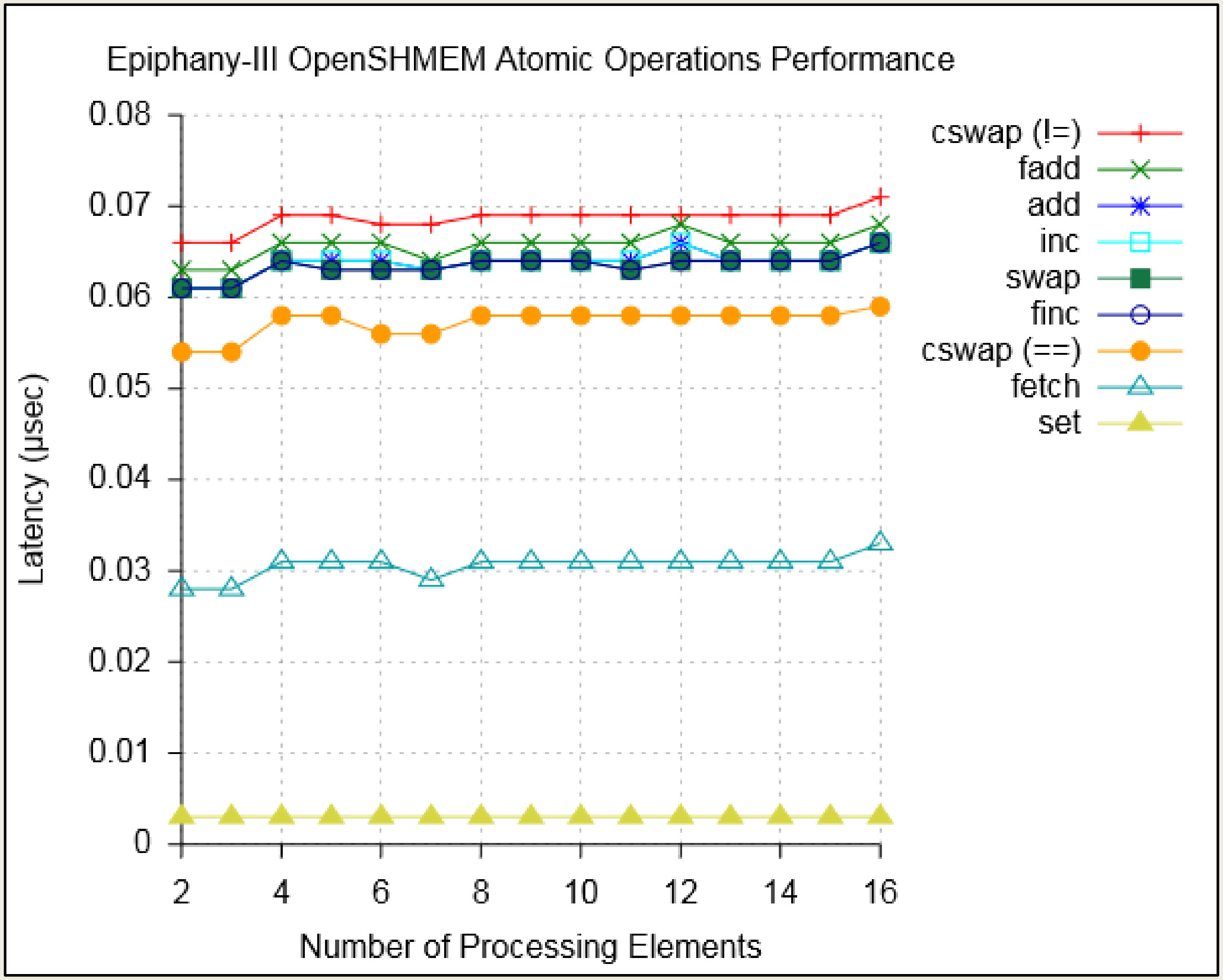
More Figures



U.S. ARMY RDECOM

More Figures

ARL

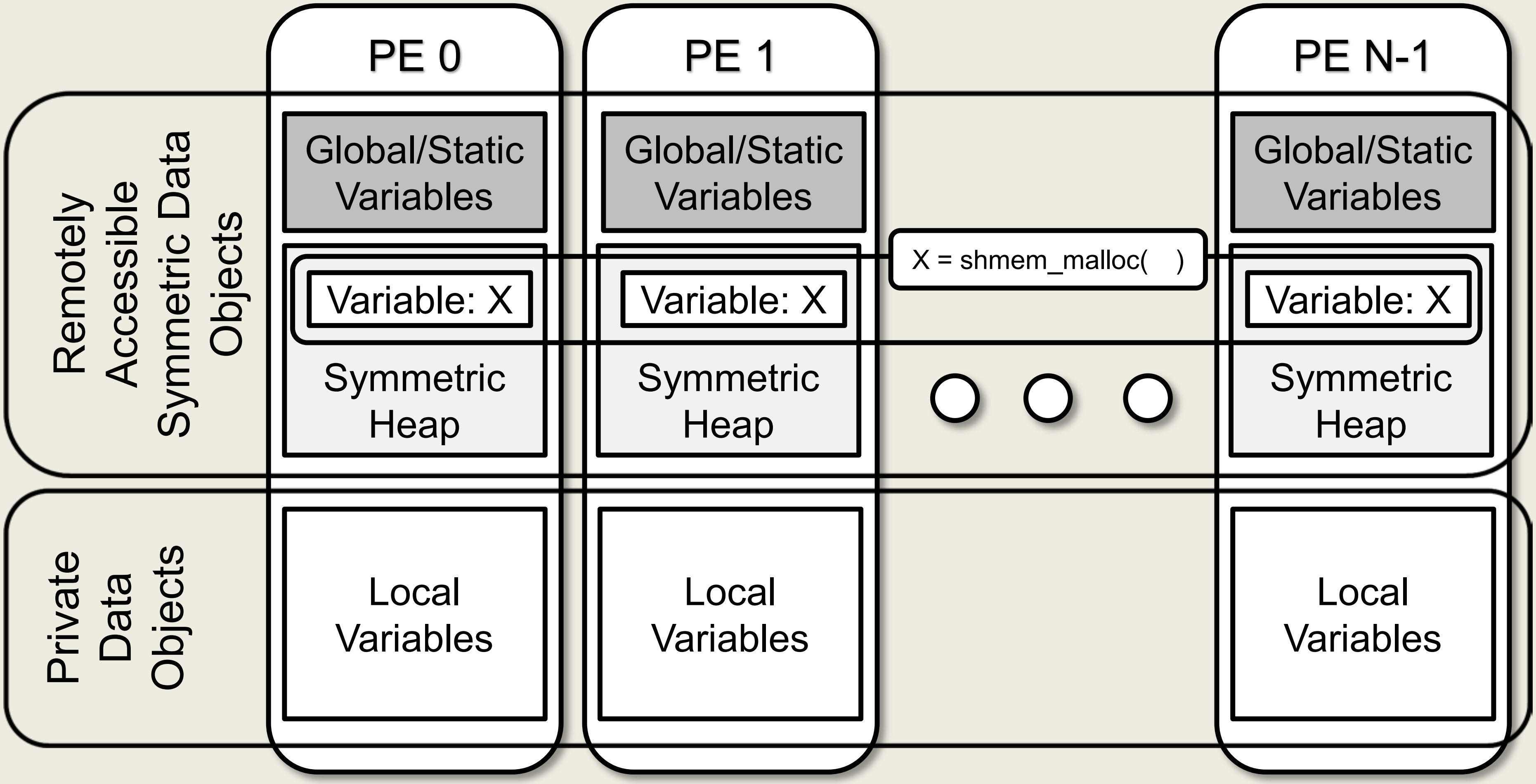


U.S. ARMY RDECOM

OpenSHMEM

ARL

- Open standard API, portable across many platforms
- Library using Partitioned Global Address Space (PGAS) programming model
- Processing Element (Epiphany core) is an OpenSHMEM process
- Symmetric Objects have same address on all PEs, global shared memory



U.S. ARMY RDECOM

Dot Product

ARL

```
.global _dot_product
_dot_product:
        movts lc, r2
        mov r2, %low(.Lstart)
        movts ls, r2
        mov r2, %low(.Lend-4)
        movts le, r2
        .balignw 8,0x01a2
        mov r24, #0
        mov r25, #0
        mov r26, #0
        mov r27, #0
        ldrd r48, [r0], #1
        fsub r44, r24, r24
        ldrd r50, [r1], #1
        fsub r45, r24, r24
        ldrd r52, [r0], #1
        fsub r46, r24, r24
        ldrd r54, [r1], #1
        fsub r47, r24, r24
        ldrd r56, [r0], #1
        fmadd r24, r48, r50
        ldrd r58, [r1], #1
        fmadd r25, r49, r51
        ldrd r60, [r0], #1
        fmadd r26, r52, r54
        ldrd r62, [r1], #1
        fmadd r27, r53, r55
        .Lstart:
        ldrd r48, [r0], #1
        fmadd r44, r56, r58
        ldrd r50, [r1], #1
        fmadd r45, r57, r59
        ldrd r52, [r0], #1
        fmadd r46, r60, r62
        ldrd r54, [r1], #1
        fmadd r47, r61, r63
        ldrd r56, [r0], #1
        fmadd r24, r48, r50
        ldrd r58, [r1], #1
        fmadd r25, r49, r51
        ldrd r60, [r0], #1
        fmadd r26, r52, r54
        ldrd r62, [r1], #1
        fmadd r27, r53, r55
        .Lend:
        fmadd r44, r56, r58
        fmadd r45, r57, r59
        fmadd r46, r60, r62
        fmadd r47, r61, r63
        fadd r24, r24, r25
        fadd r26, r26, r27
        fadd r44, r44, r45
        fadd r46, r46, r47
        fadd r24, r24, r26
        fadd r44, r44, r46
        fadd r0, r24, r44
        rts
```

```c
float dot_product(const float* a, const float* b, int nd8m1) {
   float c = 0.0f;
   int n = (nd8m1 + 1)*8;
   for (int i = 0; i < n; i++) {
      c += a[i] * b[i];
   }
   return c;
}
```

The assembly-optimized dot product (left) uses a hardware loop, 8-way loop unrolling, dual-issue pipelined loads/stores, fused multiply-adds. It is functionally identical to the C code (above). The inner loop achieves peak performance of the processor in both core bandwidth and computation. For N = 3040, that's an overall 96% peak performance with the call overhead, header, and footer code.

U.S. ARMY RDECOM

Supplemental SDK Comparison

ARL

Core/Device-Level Feature	OpenSHMEM 1.3	eSDK (e-lib)
Query Routines	Yes	Yes
Memory Management Routines	Yes	None
Remote Memory Access Routines	Yes	Yes
Non-Blocking Remote Memory Access Routines	Yes	Yes (needs better abstraction)
Atomic Memory Operations	Yes	None
Collective Routines	Yes	Barrier only
Point-To-Point Synchronization Routines	Yes	None
Memory Ordering Routines	Yes	Partial
Distributed Locking Routines	Yes	Partial
Signal/Interrupt Configuration Routines	Application code	Yes
Read/Write Special Registers	Application code	Yes
Timing Routines	Library Interface	Yes

U.S. ARMY RDECOM

Supplemental SDK Comparison

ARL

eSDK (e-lib interface)	OpenSHMEM
e_coreid_t e_get_coreid(void)	int shmem_my_pe(void) int shmem_n_pes(void)
void *e_get_global_address(unsigned row, unsigned col, const void *ptr)	void *shmem_ptr(const void *target, int pe)
e_coreid_t e_coreid_from_coords(unsigned row, unsigned col) void e_coords_from_coreid(e_coreid_t coreid, unsigned *row, unsigned *col) e_bool_t e_is_on_core(const void *ptr) void e_neighbor_id(e_coreid_wrap_t dir, e_coreid_wrap_t wrap, unsigned *row, unsigned *col)	int shmem_pe_accessible(int pe) int shmem_addr_accessible(const void *addr, int pe)
unsigned e_ctimer_get(e_ctimer_id_t timer) unsigned e_ctimer_set(e_ctimer_id_t timer, unsigned int val) unsigned e_ctimer_start(e_ctimer_id_t timer, e_ctimer_config_t config)` unsigned e_ctimer_stop(e_ctimer_id_t timer) void e_wait(e_ctimer_id_t timer, unsigned int clicks)	No standard equivalent, but included in extended/experimental interface
void *e_read(const void *remote, void *dst, unsigned row, unsigned col, const void *src, size_t n) void *e_write(const void *remote, const void *src, unsigned row, unsigned col, void *dst, size_t n) int e_dma_start(e_dma_desc_t *descriptor, e_dma_id_t chan) int e_dma_busy(e_dma_id_t chan) void e_dma_wait(e_dma_id_t chan) int e_dma_copy(void *dst, void *src, size_t n) void e_dma_set_desc(e_dma_id_t chan, unsigned config, e_dma_desc_t *next_desc, unsigned strd_i_src, unsigned strd_i_dst, unsigned count_i, unsigned count_o, unsigned strd_o_src, unsigned strd_o_dst, void *addr_src, void *addr_dst, e_dma_desc_t *desc)	shmem_put shmem_get shmem_iput shmem_iget shmem_p shmem_g shmem_[TYPE]_put shmem_[TYPE]_get shmem_[TYPE]_iput shmem_[TYPE]_iget shmem_[TYPE]_p shmem_[TYPE]_g shmem_put[SIZE] shmem_get[SIZE] shmem_iput[SIZE] shmem_iget[SIZE] shmem_putmem shmem_getmem shmem_put_nbi shmem_get_nbi shmem_[TYPE]_put_nbi shmem_[TYPE]_get_nbi shmem_put[SIZE]_nbi shmem_get[SIZE]_nbi shmem_putmem_nbi shmem_getmem_nbi
void e_irq_attach(e_irq_type_t irq, sighandler_t handler) void e_irq_set(unsigned row, unsigned col, e_irq_type_t irq) void e_irq_clear(unsigned row, unsigned col, e_irq_type_t irq) void e_irq_global_mask(e_bool_t state) void e_irq_mask(e_irq_type_t irq, e_bool_t state)	No equivalent in SHMEM (but some of this occurs transparently within library, optionally)

void e_mutex_init(unsigned row, unsigned col, e_mutex_t *mutex, e_mutexattr_t *attr)
void e_mutex_lock(unsigned row, unsigned col, e_mutex_t *mutex)
unsigned e_mutex_trylock(unsigned row, unsigned col, e_mutex_t *mutex)
void e_mutex_unlock(unsigned row, unsigned col, e_mutex_t *mutex)
void e_barrier_init(volatile e_barrier_t bar_array[], volatile e_barrier_t *tgt_bar_array[])
void e_barrier(volatile e_barrier_t *bar_array, volatile e_barrier_t *tgt_bar_array[])
	void shmem_barrier_all(void);
	void shmem_barrier(int PE_start, int logPE_stride, int PE_size, long *pSync);
	void shmem_fence(void);
	void shmem_quiet(void);
	void shmem_set_lock(long *lock);
	void shmem_clear_lock(long *lock);
	int shmem_test_lock(long *lock);

No equivalent on-chip memory management interface	void *shmem_malloc(size_t size) void *shmem_align(size_t alignment, size_t size) void shmem_free(void *ptr) void *shmem_realloc(void *ptr, size_t size)

U.S. ARMY RDECOM

Supplemental

ARL

eSDK (e-lib interface)	OpenSHMEM
No equivalent atomic operations	shmem_[TYPE]_finc shmem_[TYPE]_inc shmem_[TYPE]_fadd shmem_[TYPE]_add shmem_[TYPE]_cswap shmem_[TYPE]_swap shmem_[TYPE]_fetch shmem_[TYPE]_set
No equivalent collective routines	shmem_broadcast[SIZE] shmem_fcollect[SIZE] shmem_collect[SIZE] shmem_alltoall[SIZE] shmem_alltoalls[SIZE] shmem_[TYPE]_sum_to_all shmem_[TYPE]_prod_to_all shmem_[TYPE]_min_to_all shmem_[TYPE]_max_to_all shmem_[TYPE]_and_to_all shmem_[TYPE]_or_to_all shmem_[TYPE]_xor_to_all
No equivalent point-to-point synchronization routines	shmem_[TYPE]_wait shmem_[TYPE]_wait_until