Brown Deer Technology

OpenCL + OpenSHMEM Hybrid Programming Model for the Adapteva Epiphany Architecture

David A. Richie
Brown Deer Technology

James A. Ross
U.S. Army Research Lab

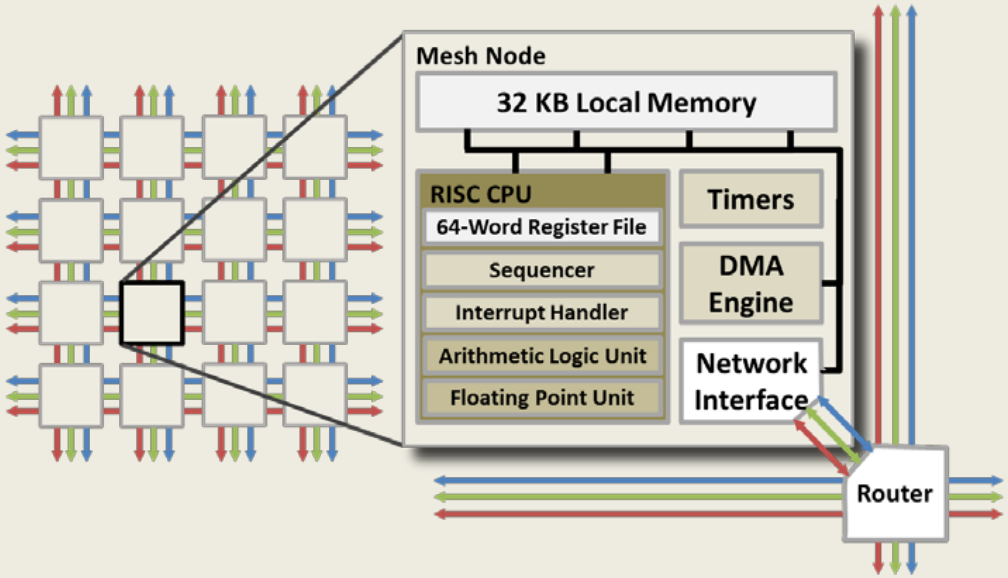
U.S. ARMY RDECOM®

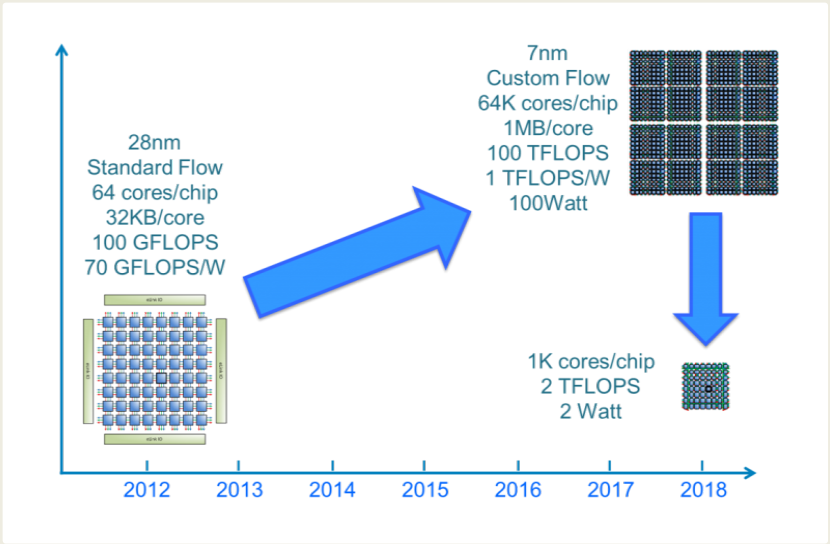
Outline

ARL

- Epiphany architecture
- OpenCL for Epiphany
- OpenSHMEM for Epiphany
- OpenCL + OpenSHMEM hybrid programming model
- Initial results
- Summary

Epiphany Architecture



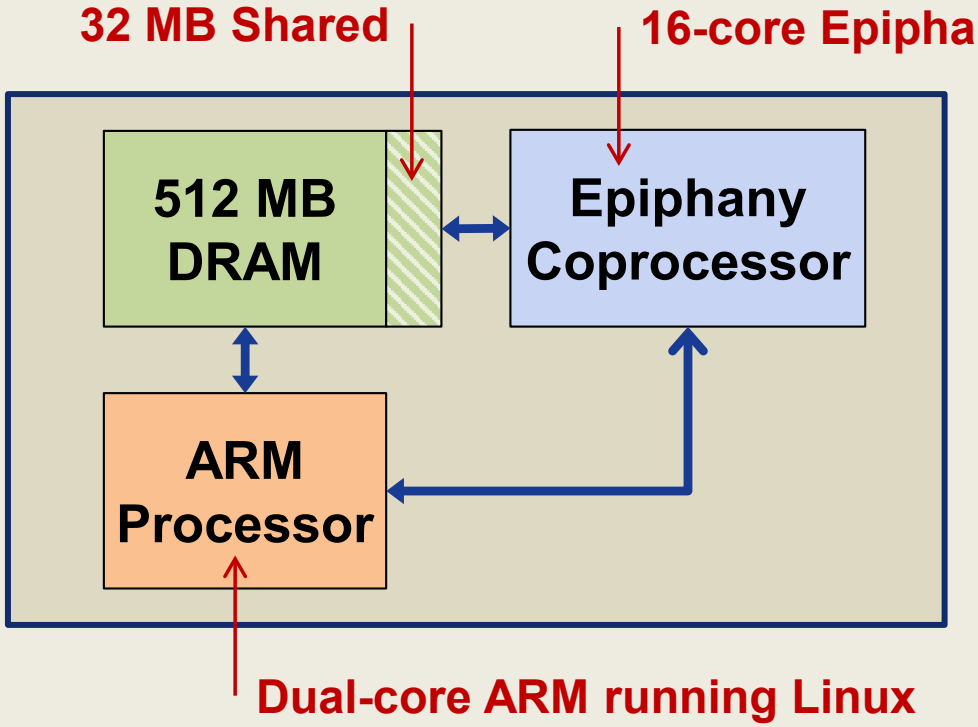


- Design emphasizes simplicity, scalability, power-efficiency
- 2D array of RISC cores, 32KB per core, 2D Network on Chip (NoC)
- Fully divergent cores with distributed local memory
- Minimal un-core functionality, e.g., no data or instruction cache
- Existing design scales to thousands of cores
- Highest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

Epiphany Memory Architecture

- Each core has 32KB of local SRAM for both instructions and data
- Local SRAM is mapped into a single flat address space
  - global_addr = (ROW << 26) | (COL << 20) | local_addr
- Can be thought of as shared distributed memory
  - Access to local SRAM of core is fast
  - Access to local SRAM of another core negotiated over 2D NoC
- Off-chip DRAM accessible to all cores negotiated over 2D NoC
  - Access is much slower than local SRAM
- Two (2) DMA engines per core
- NO cache

Parallella Platform: ARM + Epiphany



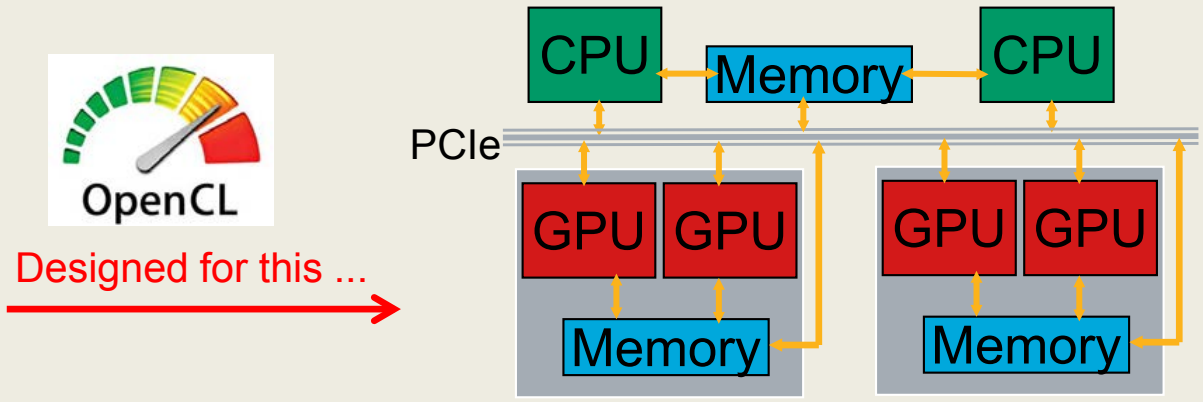
- Heterogeneous platform architecture
- Dual-core ARM processor running Linux
- 16-core Epiphany coprocessor
- 512 MB global memory, 32 MB shared memory
- Appears to fit the accelerator-offload platform model of OpenCL ...

Epiphany Software Support

- Vendor provides low-level Epiphany SDK for basic support
  - Simple host and device calls, no programming model support
- COPRTHR-1 targeted Epiphany with several programming APIs
  - OpenCL, STDCL, direct device API, Pthreads for coprocessors
- Threaded MPI was developed later for Epiphany
  - Built on COPRTHR-1 software stack, very good performance
- ARL OpenSHMEM recently developed
  - Uses COPRTHR-2 stack, slightly better performance (~10%)
- *... What about OpenCL?*
  - *OpenCL exhibited performance issues related to the API itself*
  - *We are revisiting OpenCL with a hybrid programming model that uses OpenSHMEM to address these issues*

OpenCL

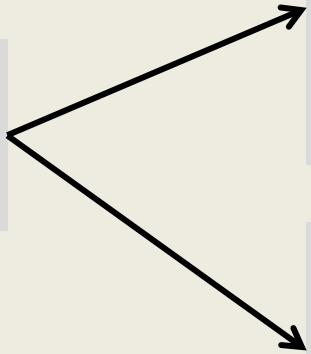
- Industry standard API for heterogeneous platforms
- Designed primarily for GPUs, can be used with other accelerators
- Programming model based on offloading parallel work
  - A kernel is executed over many threads (SIMD/SIMT)
- Two parts to the OpenCL API:
  - Run-time host API used to coordinate execution (useful)
  - Kernel programming API for offload device (useful? Why not use C?)



OpenCL in Pseudo-code

- **Example: offloading simple outer product (not efficient in practice)**
- **OpenCL uses a *co-design model* with host code and device code**

```
for( i=0; i<n; i++ ) {
   c[i] = a[i] * b[i];
}
```



```
device float *a, *b, *c; // device_malloc(n)
copy_to_device(a);
copy_to_device(b);
offload_threads( my_kern, n);
copy_from_device(c);
```

Host CPU

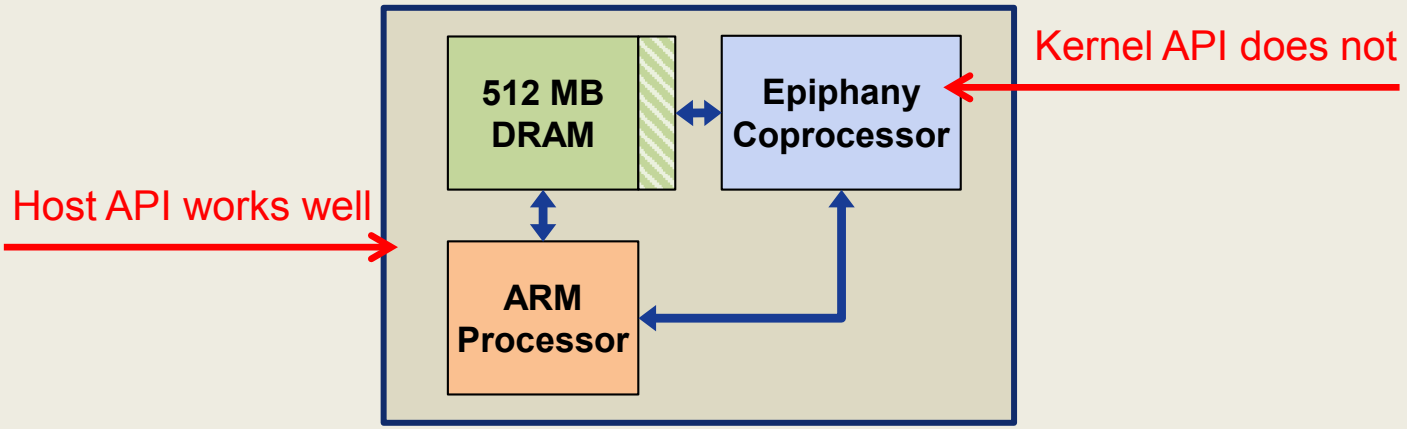
```
my_kern(float* a, float* b, float* b) {
   i = get_thead_id();
   c[i] = a[i] * b[i];
}
```

Coprocessor

(Pseudo-code, not OpenCL syntax)

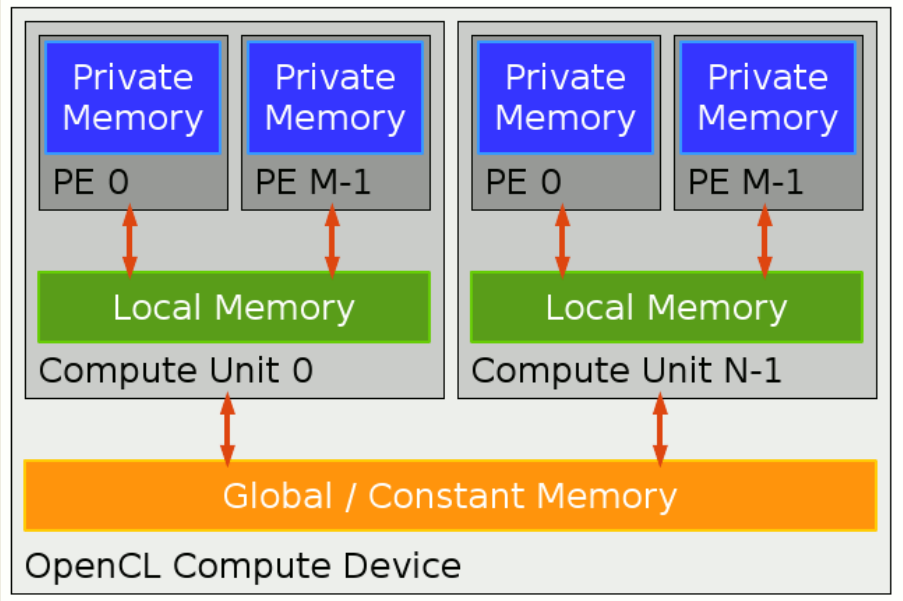
OpenCL for Epiphany

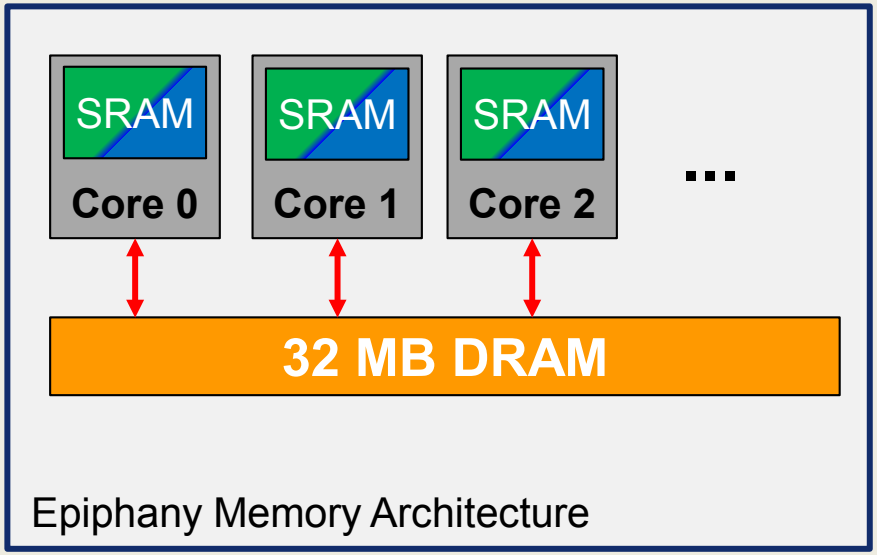
- First parallel programming API implemented for Epiphany
- Goal was to provide baseline OpenCL 1.1 functionality
  - Portability of GPU code was *not* a goal
- Most significant challenge was matching OpenCL memory model to the Epiphany memory architecture (they are inconsistent)
- Basic issue is that architecture is more capable than the OpenCL programming model *AND* ignoring this leads to poor performance



The "Local" Memory Problem

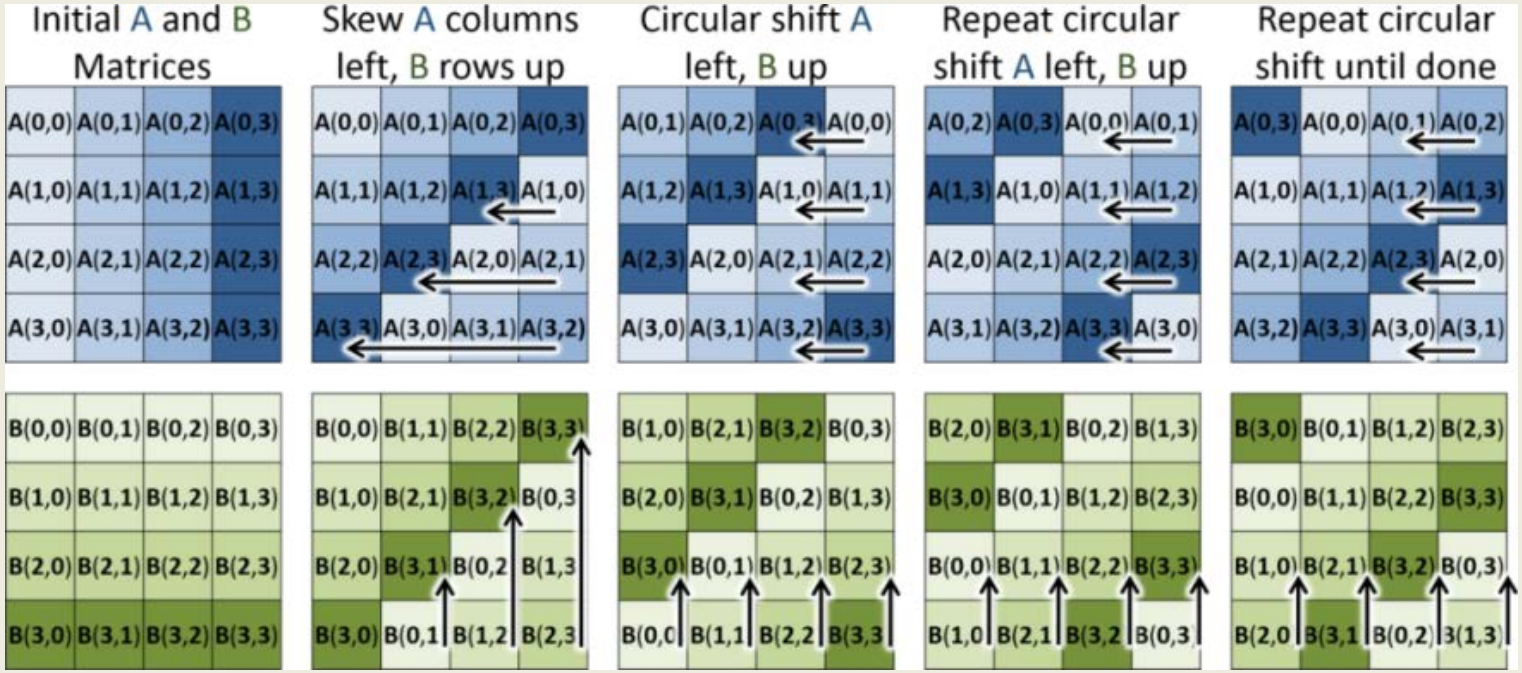
- Epiphany coprocessor treated as a single OpenCL compute unit
  - Based on fact that cores can be synchronized
- Question is whether core-local SRAM is OpenCL *local* or *private*
  - Treating as OpenCL local is inefficient – NUMA issue
  - Treating as OpenCL private is incorrect – all cores have access
- Best view would be *symmetric shared memory*
  - No support for this in OpenCL





The Problem Is Real

- Lack of support for symmetric shared memory is significant
- Demonstrate by example: Cannon's algorithm
- Known to be ideal matrix-matrix multiply algorithm for Epiphany
- Simple premise:
  - Perform sub-matrix multiplication
  - Shift sub-matrix of A (B) to the left (up) – requires inter-core memcopy
- *Critical that data can be copied directly from one core to another*



Non-Standard Extensions

- **In order to achieve performance, non-standard extensions were added to the OpenCL implementation for Epiphany (2012)**
- **These included family of mutex calls for synchronization and support for one-sided inter-core memcopy**

```
void* memaddr( void* ptr, threadspec_t thrs, int flags);

        Return global address of ptr within address space of thread thrs

void* memcopy( void* dst, void* src, size_t n, int flags );

        Inter-core memcopy using addresses mapped to global address space
```

Note: These extensions were delivered before OpenSHMEM 1.0 specification was released

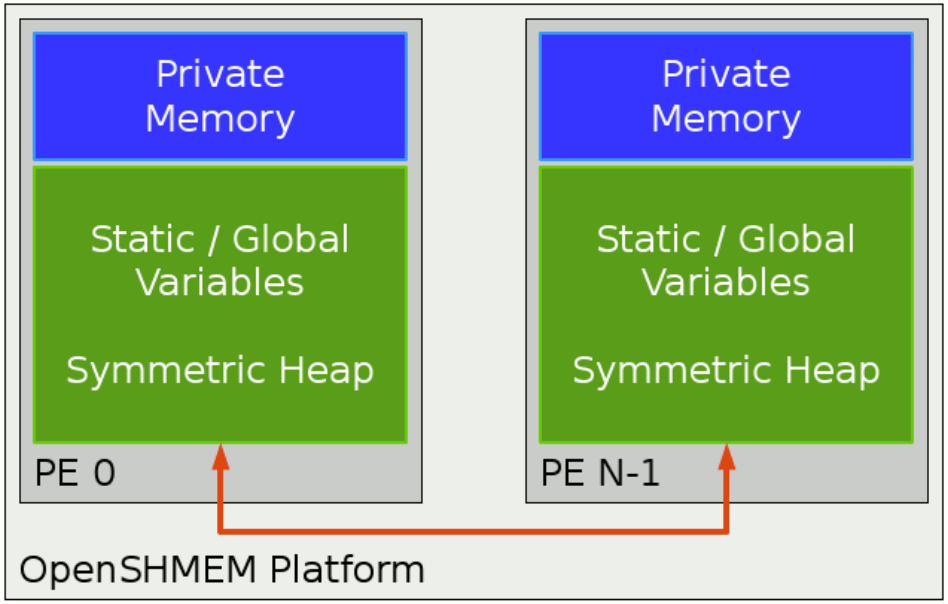
- **The memcopy extension is similar in functionality to OpenSHMEM put call (at the time OpenSHMEM was not considered)**

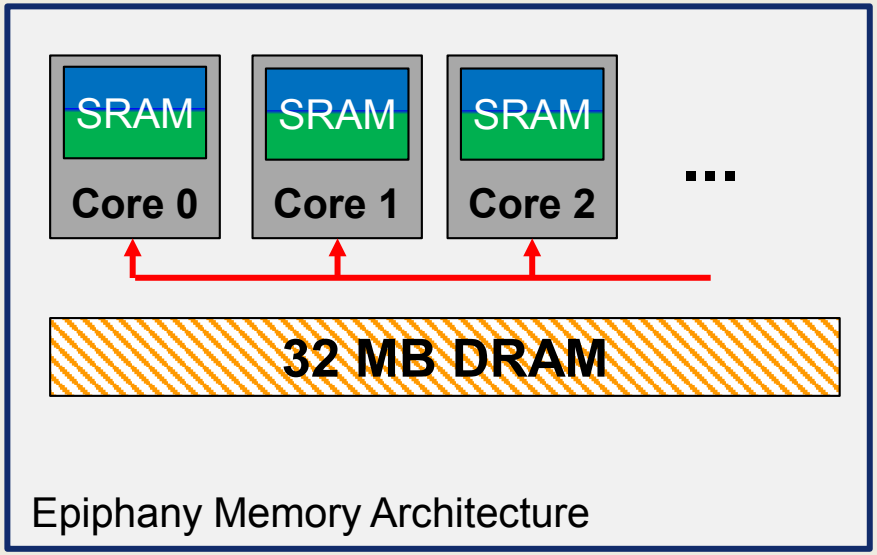
```
void shmem_TYPE_put( TYPE* target, TYPE* source, size_t nelems, int pe );

        Copy local data to remote PE
```

OpenSHMEM for Epiphany

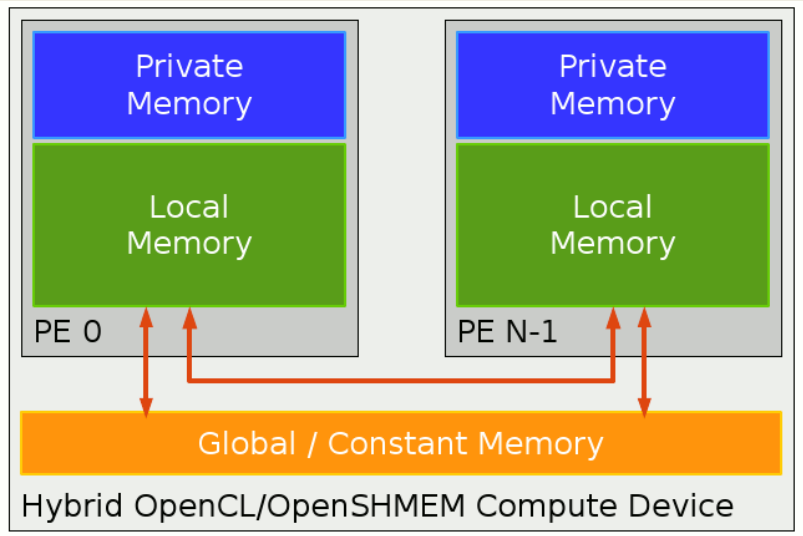
- Recently the full OpenSHMEM 1.3 specification has been implemented for Epiphany (details were presented earlier)
- Device-level programming only
  - No coprocessor offload mechanism (uses COPRTHR-2 magic trick)
  - No access to DRAM shared between host and coprocessor
- Correctly addresses the Epiphany core-local SRAM and supports inter-core memcopy that is critical for high-performance

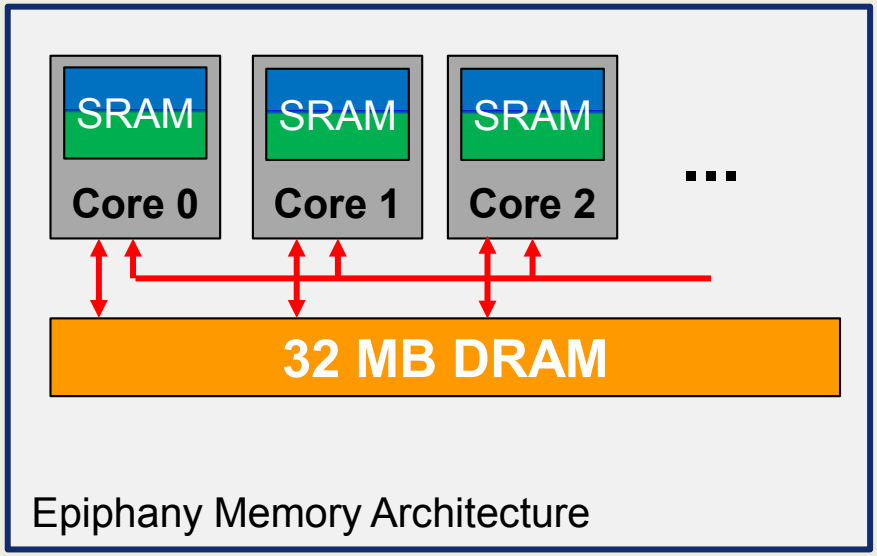




OpenCL + OpenSHMEM

- We propose a hybrid OpenCL + OpenSHMEM programming model
- Implemented for Epiphany, applicable to similar architectures
- Hybrid programming model allows efficient access to the Epiphany memory architecture and support for platform offload





- OpenSHMEM context nested within an OpenCL kernel
  - OpenCL host code is unchanged
  - Concept of an OpenSHMEM job is launched for each OpenCL kernel that is enqueued

U.S. ARMY RDECOM®

OpenCL + OpenSHMEM

ARL

- Pseudo-code showing the relation of the hybrid API
- Host code controls allocation of OpenCL global memory
- Parallel work is offloaded to coprocessor using OpenCL
- Within the OpenCL kernel OpenSHMEM symmetric shared memory allocation and data movement is supported

Host Application Code

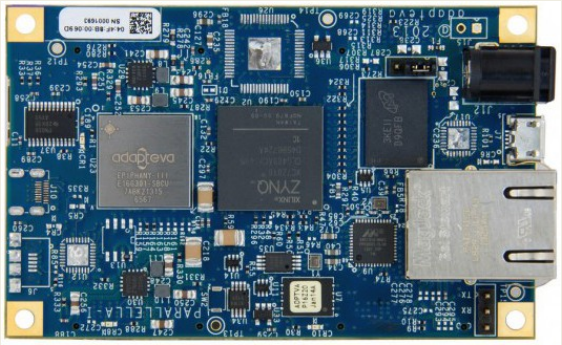
```
{
   allocate OpenCL global memory
   copy_to_global_memory
   offload_threads( my_kern, n)
   copy_from_global_memory
}
```

Coprocessor Device Code

```
my_kern( args )
{
   SHMEM init
   SHMEM allocate
   copy global to local memory
   SHMEM put
   copy local to global memory
}
```

Initial Results

- **Benchmarked two implementations of Cannon's algorithm:**
  - **Pure OpenCL using global memory for performing the shift**
  - **Hybrid OpenCL + OpenSHMEM using one-sided communication for performing the shift**
- **Hardware used was a Parallella Epiphany development board:**



- Parallella board dev kit
- Dual-core ARM host processor
- 16-core Epiphany-III co-processor
- 19.2 GFLOPS @ 600 MHz
- **~10 mm^2, 65 nm, 0.594 Watt (max) chip**

- **Results showed 2.3x overall speedup using the hybrid model**

Matrix Size	Performance (MFLOPS)	
	OpenCL	OpenCL+OpenSHMEM
32 x 32	218	504
64 x 64	424	100
128 x 128	794	1812

Summary

- Demonstrated hybrid OpenCL + OpenSHMEM programming model for device-level parallel programming of an Epiphany RISC array
- OpenSHMEM API is nested within the OpenCL kernel code
- Hybrid model directly resolves most critical deficiency encountered in the use of OpenCL alone for this architecture
- Introduction of OpenSHMEM allows proper management of on-chip distributed symmetric shared memory, critical for high performance
- OpenCL provides support lacking with OpenSHMEM alone
  - Host platform offload and access to global shared memory (between host and device)
- Benchmarks for matrix-matrix multiplication demonstrate the expected performance improvement