KALRAY

Kalray MPPA®

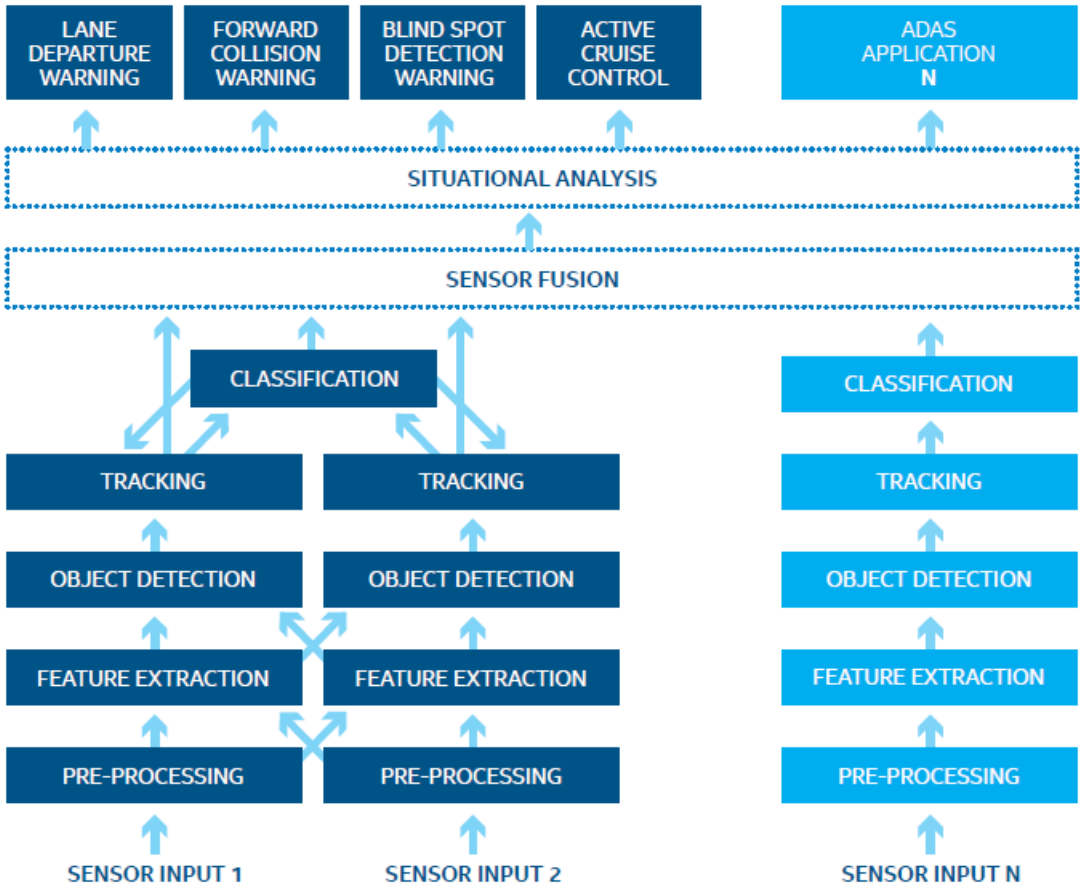
Massively **P**arallel **P**rocessor **A**rray

MPPA Asynchronous Operations API

Benoît Dupont de Dinechin / Julien Hascoët / Pierre Guironnet de Massas

June 2016

Target Application: Sensor Fusion (source Intel)

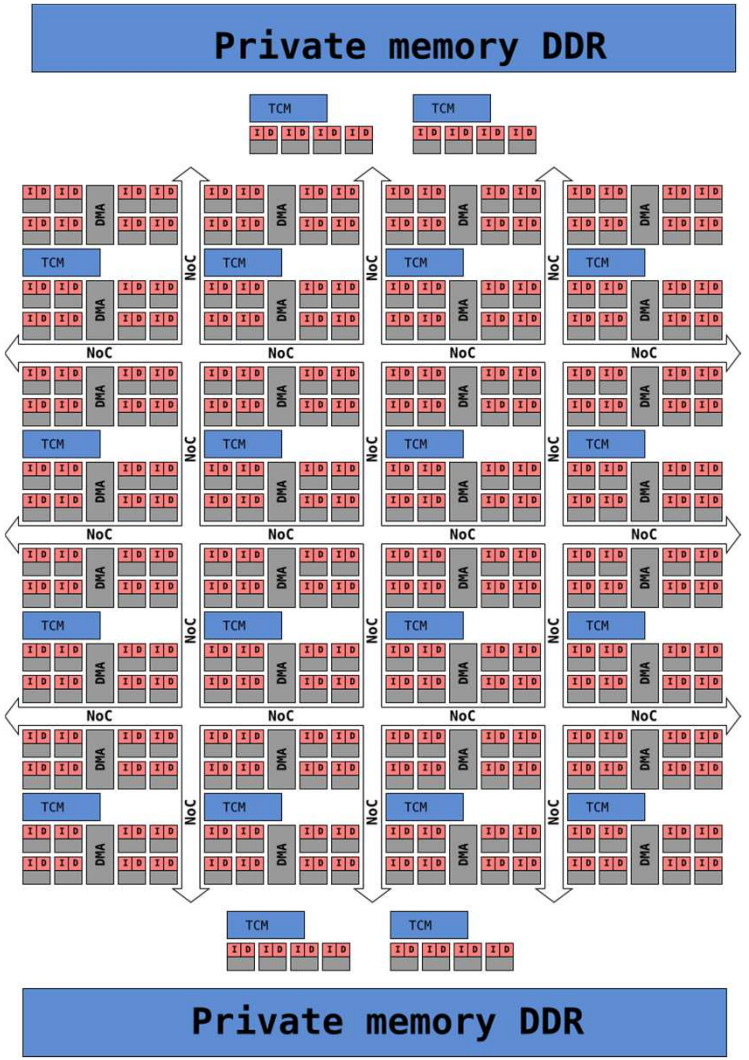


Automotive Sensor Processing

- Low-level processing
  - Operates locally on the pixels
  - Typical algorithms include image transforms, image filtering, computing gradients, and morphological operations.
- Mid-Level Processing
  - Operates globally on the image. The main tasks are detection of interest points (disparity, detection of corners or edges), feature extraction by various techniques (HOG, SURF, SIFT, ORB), integral image (Viola Jones)
  - Other tasks correlate images, such as image disparity in stereo vision and optical flow for image sequences
- High-Level Processing
  - Implements object detection, tracking and classification by leveraging machine learning techniques: Adaptive Boosting (AdaBoost), Support Vector Machines (SVM), K-Nearest Neighbors (KNN), Artificial Neural Network (ANN).

MPPA Distributed Memory Architecture Challenge

- Classic approaches to communication
  - Load/Store to unified memory
  - Remote DMA to unified memory
  - Remote DMA to private memories
  - Send/receive to private memories
- Classic approaches to synchronization
  - Locks, semaphores, conditional variables
  - Barriers, collectives (reduce/scan/all2all)
  - Remote atomic operations
  - Remote atomic queues
- Classic HPC clusters vs MPPA
  - The working memory of HPC clusters is the collection of identical node memories
  - The working memory of the MPPA is the collection of SMEMs backed by the DDRs



MPPA Distributed Memory Architecture + Runtime

- MPPA local memory benefits
  - Local memory accesses are more energy-efficient than a Last-Level Cache (LLC)
  - Local memory access interferences can be managed for time-critical applications
- MPPA DSM for implicit DDR access
  - Escape code size and data size restrictions
  - Mostly used for application porting
  - Some implicit global memory accesses cannot be converted to remote DMA
- MPPA Asynchronous Operations
  - API designed to simplify use of local memories and sharing of the DDR memories through remote DMA over NoC
  - Co-exist with the DSM or run stand-alone

MPPA Asynchronous Operations Principles

- Inspired by HPC clusters one-sided communication & synchronization
  - Cray SHMEM, ORNL ARMCI, Berkeley GasNet, MPI-3 one-sided subset
  - Cannot directly reuse these libraries because of the MPPA architecture
- Asynchronous data transfers
  - All data transfer operations return immediadely to caller
  - An event structure can be used to wait/test for **local completion**
- Remote DMA between local and remote memory segments
  - Put (remote write) and Get (remote read) operations with data reshaping
  - The default memory segment is the local address space of each cluster
- Point-to-point synchronization operations
  - Remote fence (**global completion**), peek, poke, post-add, fetch-clear, fetch-add
  - Local wait on the comparison between a local variable and a value
- Remote queues (in development)
  - Push on a remote queue-like memory segment, with atomicity if possible

MPPA Asynchronous Operations API Overview

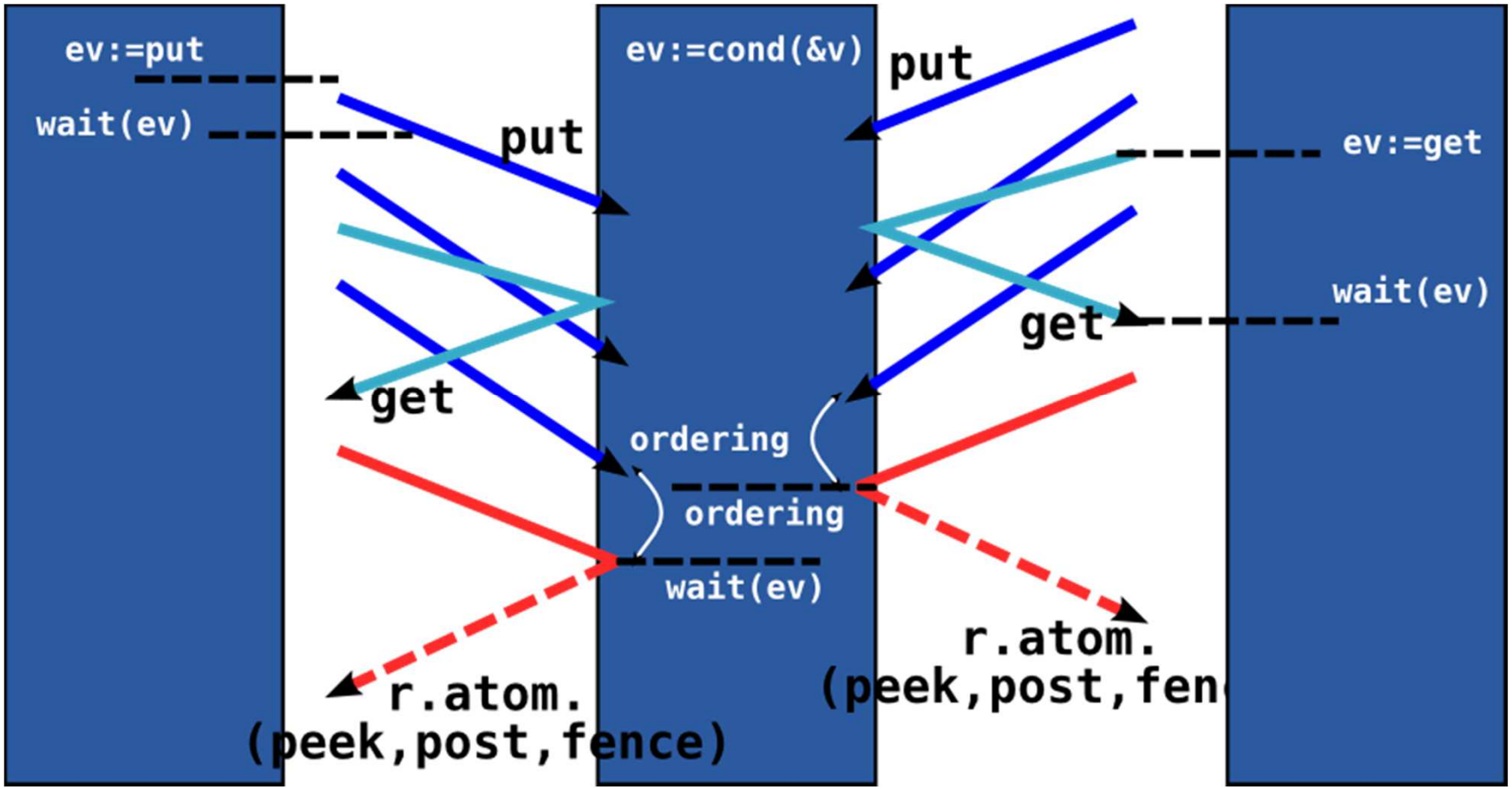
- Dense Transfers
  - mppa_async_get
  - mppa_async_put
  - mppa_async_get_spaced
  - mppa_async_put_spaced
  - mppa_async_get_vectored
  - mppa_async_put_vectored
- Sparse Transfers
  - mppa_async_sget_spaced
  - mppa_async_sput_spaced
  - mppa_async_sget_blocked2d
  - mppa_async_sput_blocked2d
  - mppa_async_sget_blocked3d
  - mppa_async_sput_blocked3d
- Asynchronous Events
  - mppa_async_event_wait
  - mppa_async_event_test
- Global Synchronization
  - mppa_async_fence
  - mppa_async_peek
  - mppa_async_poke
  - mppa_async_postadd
  - mppa_async_fetchclear
  - mppa_async_fetchadd
  - mppa_async_evalcond
- Remote queues
  - mppa_async_enqueue
  - mppa_async_dequeue

Illustration of Code Transformations for Put/Get

- Example extracted from a tiled matrix multiply algorithm
  - Inner loop is first converted to a dense Put (mppa_async_put)
  - Outer loop is then converted to a sparse Put (mppa_async_sput_spaced)
  - Use of blocking calls (last Put parameter is NULL instead of event pointer)

```
void
tileto(int m, int n, dtype C[m][n], int i, int j, int p, dtype c[p][p])
{
  int mii = MIN(p, m-i);
  int mjj = MIN(p, n-j);
#ifndef useputget
  for (int ii = 0; ii < mii; ii++) {
    for (int jj = 0; jj < mjj; jj++) {
      C[i+ii][j+jj] = c[ii][jj];
    }
  }
#elif (useputget == 1)
  for (int ii = 0; ii < mii; ii++) {
    mppa_async_put(&c[ii][0], &C[i+ii][j+0], ddr0_segment, mjj*sizeof(dtype), NULL);
  }
#elif (useputget == 2)
  mppa_async_sput_spaced(&c[0][0], &C[i+0][j+0], ddr0_segment, mjj*sizeof(dtype), mii,
                         (char*)&c[1][0]-(char*)&c[0][0], (char*)&C[i+1][j+0]-(char*)&C[i+0][j+0], NULL);
#endif//useputget
}
```

MPPA Asynchronous Operations Ordering

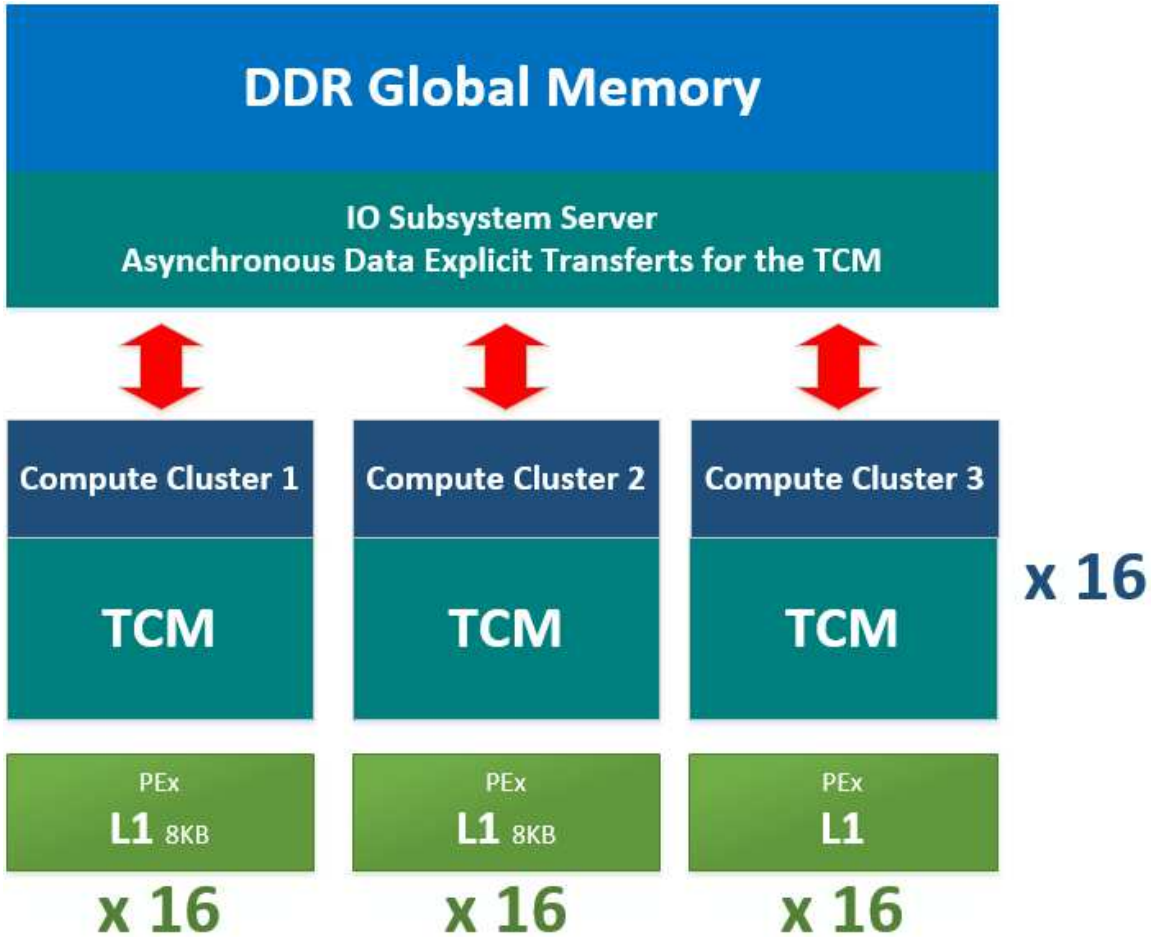


MPPA Asynchronous Operations Performances

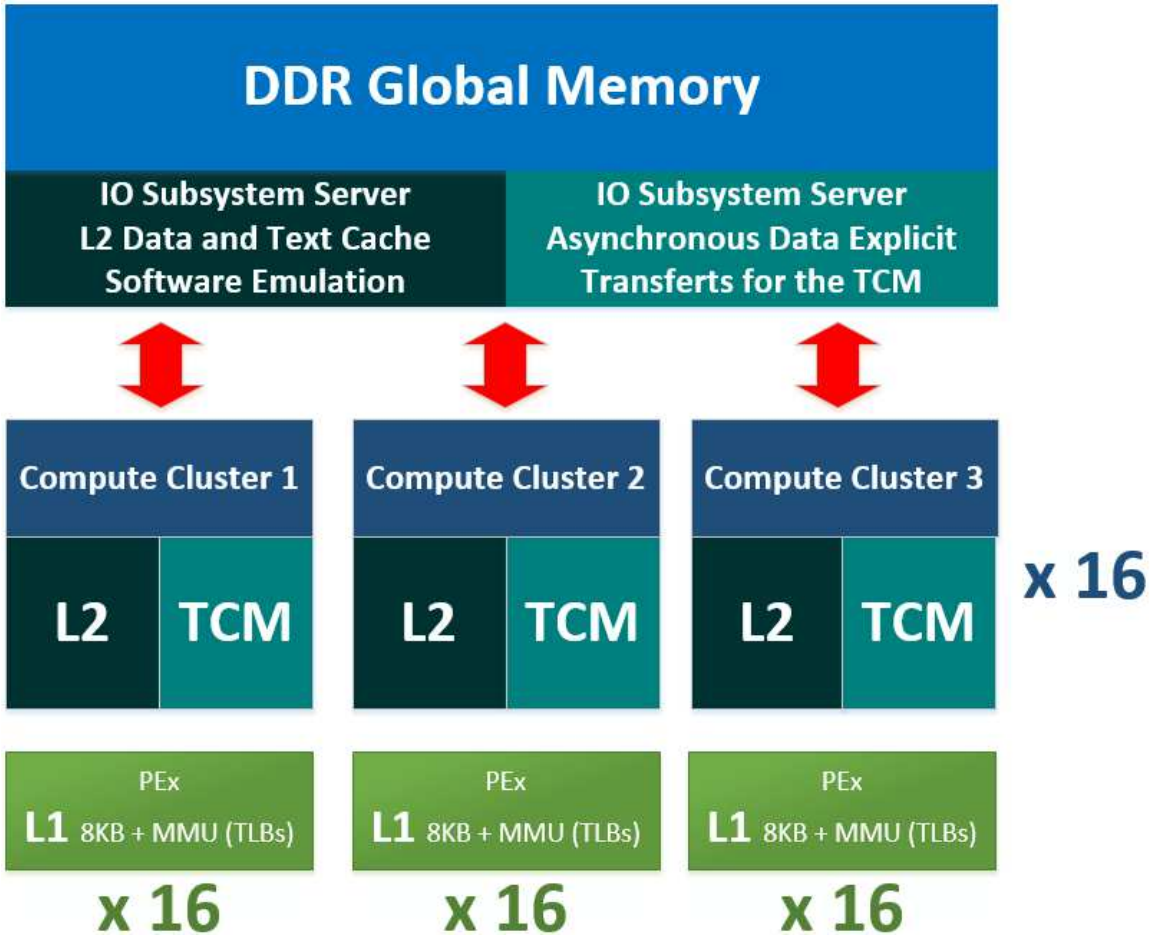
Measured	Cycles
Get DDR latency	403
Put DDR latency	600
Get DDR latency, local completion	1125
Put DDR latency, remote completion	891
Put DDR latency, local completion	700
Poke or PostAdd latency	400
FetchAdd latency, local completion	1100

Mono core in one cluster, but thread safe API
Measured with warm cache (else +200 cycles)

MPPA Runtine Environnent with Async Only



MPPA Runtine Environment with Async and DSM



Other MPPA Asynchronous Operations Benefits

- Expose the mOS exo-kernel features to user applications
  - Virtualized hardware for VLIW cores and NoC interfaces
  - Software DMA engine (emulates a « capable » DMA hardware)
    - Inter-cluster remote read (get)
    - Inter-cluster remote write (put)
    - Data reshaping (gather / scatter)
  - Active message server (emulates a « smart » control NoC)
    - Intercluster peek, poke, postadd and other atomic operations
- Provide executable specifications for the MPPA Coolidge
  - Time-consuming code paths in the runtime are candidate for hardware acceleration
  - Functional safety domain boundaries are clearly identified

MPPA Asynchronous Operations vs MPPA IPC

- RDMA operations
  - Asynchronous Put / Get
  - Do not handle PCIe
  - No serialization
  - Based on events
- Synchronizations
  - Remote atomic + local condition evaluation
  - Remote queues
- Remote memory consistency
  - Ensured from local to remote
  - No use of C-NoC except DDR credits
- Naming of architectural resources
  - Implicit in ranks and segments

- RDMA operations
  - Read / write / pread / pwrite / aio_read / aio_write
  - Hidden serialization
  - Based on interrupts
- Synchronizations
  - Producer-consumer
  - Barrier-like
  - Remote queues
- Remote memory consistency
  - No D-NoC fence workaround
  - No ordering D-NoC / C-NoC
- Naming of architectural resources
  - Explicit in pathnames

Application Complexity for a Simple Vector Add

- MPPA IPC
  - Write IO cluster code
  - Write compute cluster code
- Code Size
  - **9 lines** on compute cluster + **10 lines** on IO cluster for NoC communications
  - Must synchronize most of the transfers (read before write for instance)

- MPPA Async
  - Write compute cluster code
- Code Size
  - **2 Lines** on compute cluster for NoC communications
  - Synchronization implied by data transfers (put, get)

Comparison of Implementations

Metric	MPPA IPC	MPPA NOC	MPPA Async
Code Size (cluster side)	89,87 KB	26,7KB	16,27 KB
Sloc Count C language	4518 (IO+CC) eppm/mppaipc/nocipc	3441 (IO+CC)	2286 (IO+CC)

- MPPA Async bypasses the libnoc (like the ODP implementation)
- MPPA Async uses mOS features (DMA Engine, Active Messages)

Comparing the DMA Job Latency with Completion (the most important metric)

Metric Local Measure in cycles	MPPA IPC (Andey) Aio write + wait	MPPA Async (Bostan) put/get + wait on cluster
Cluster 2 DDR 1B	2361	891 (put)
DDR to Cluster 1B	40 810	1165 (get)
Cluster 2 DDR 1 KB	2 687	1080 (put)
DDR 2 Cluster 1 KB	43 364	1500 (get)
Cluster 2 DDR 1 MB	288 379	287 340 (put)
DDR 2 Cluster 1 MB	1 169 115	288 178 (get)

MPPA Asynchronous Operations and OpenCL

- From TI KeyStone 'Optimization Techniques for Device (DSP) Code'
  - Prefer Kernels with 1 work-item per work-group (DSP seen as one Compute Unit)
  - Use async_work_group_copy and async_work_group_strided_copy
    - "it is almost always better to write the values to a local buffer and then copy that local buffer back to a global buffer using the OpenCL async_work_group_copy function"
- On the MPPA, expose as the standard OpenCL asynchronous copies
  - OpenCL asynchronous copies are restricted to dense local memory accesses
  - Need to provide enough local memory => 1 Work Group per Cluster preferred
  - Extensions for 2D/3D accesses in global memory (done on ST P2012 OpenCL)
  - No simple OpenCL extension for direct communication between Work Groups
- Full exploitation of MPPA Async in OpenCL requires non-standard techniques
  - Dispatch n kernels on a OpenCL Sub-Device with n Compute Units
  - Inside a kernel, revert to C/C++ or enable all MPPA Async calls from OpenCL-C

MPPA Asynchronous On-Going and Future Work

- Check applicability to Data Center applications
  - The ‘remote read’ use case corresponds to ‘mppa_async_get_vectored’
  - Could also apply as a MPPA Linux offloading API to the compute clusters
- Implicit asynchronous operations by software multi-threading
  - Run 4 software threads per PE core, yield thread on event_wait and evalcond
  - Obviate explicit software pipelining to overlap computation & communication
- Abstract physical MPPA resources (clusters, memories)
  - Use process ‘rank’s and segment IDs instead of cluster / SMEMs&DDRs
- Extensions / restrictions for time-critical applications
  - Activation of NoC QoS and disciplined use of the DDR by remote read server
- Asynchronous [non-blocking] collective operations like in MPI-3
  - Take a group of processes and compute min/max/sum etc. of distributed data
- Applications to optimized domain-specific libraries
  - BLAS, FFT, computer vision, CNN frameworks, video decoding