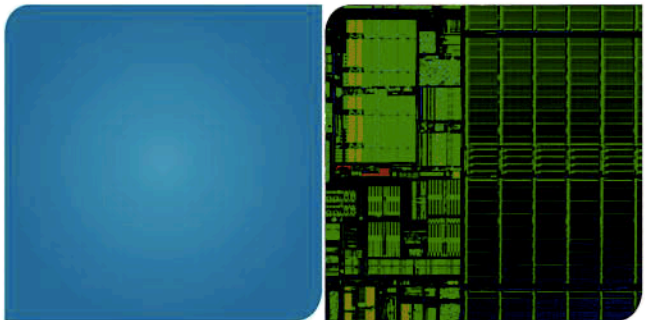
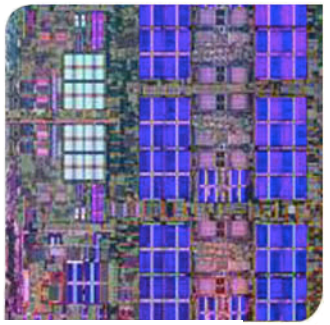




Design Space Exploration of Memory Model for Heterogeneous Computing

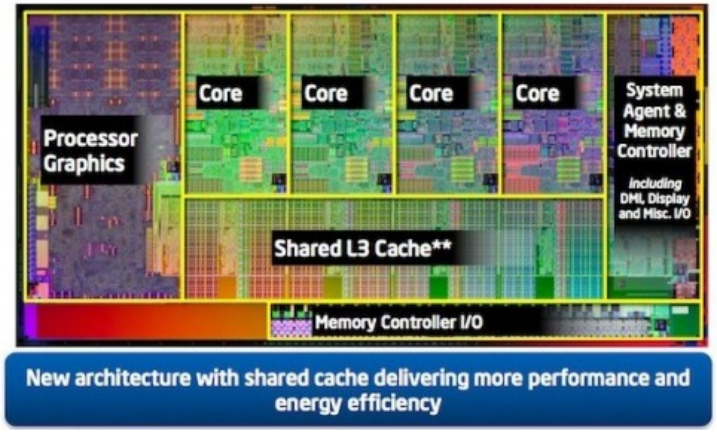
Jieun Lim*, **Hyesoon Kim+**

*Seoul National University, South Korea
+ Georgia Tech, USA

Georgia Tech comparch

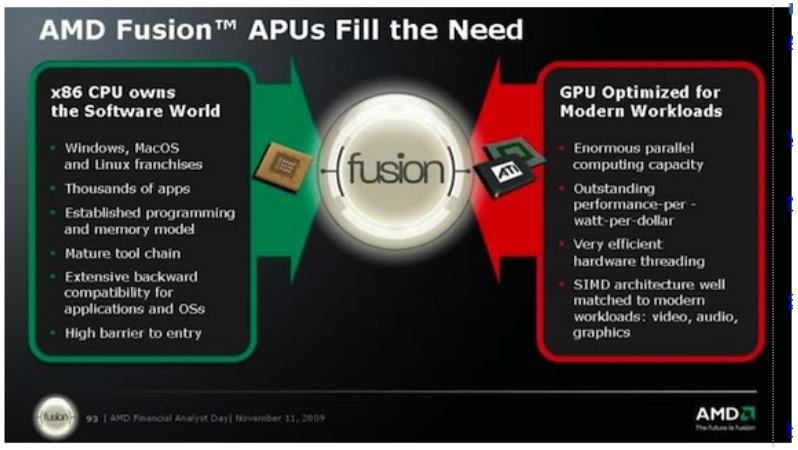
Introduction

- Heterogeneous computing has become a major architecture trend
  - CPU+GPU, Near data processing systems (NDP)
- How to design memory system
  - Strongly coupled with architecture design and programming model
  - Difficult to compare models
- Goal
  - Understand a trade-off in memory system design decisions
  - Evaluate the overhead of design options



New architecture with shared cache delivering more performance and energy efficiency

Sandy Bridge





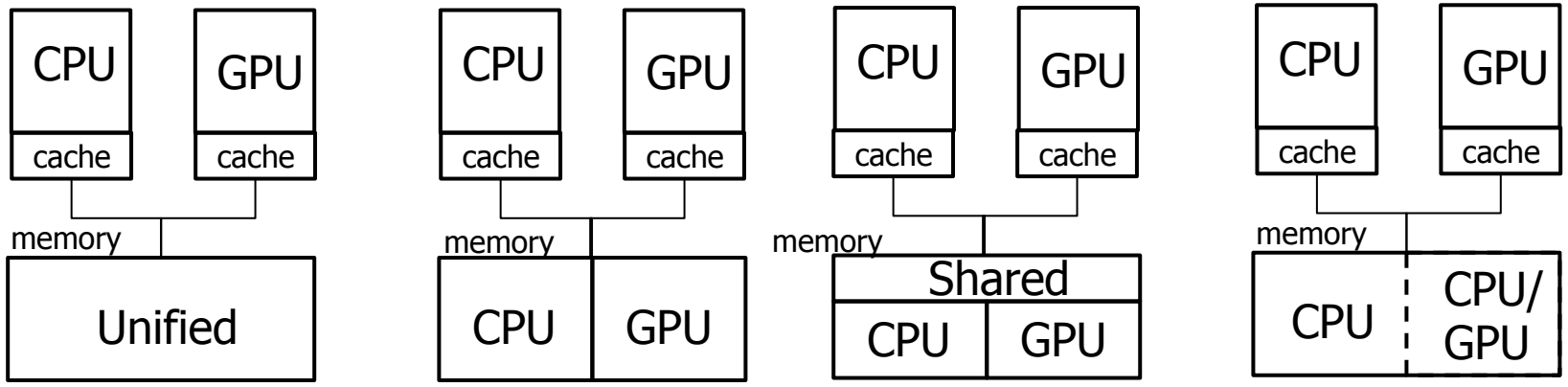
Study Objective

- Evaluate various memory system design options
- Decouple hardware architecture issues and programming models issues

- Evaluation categories
  - Memory space
  - Locality management
  - Communication overhead

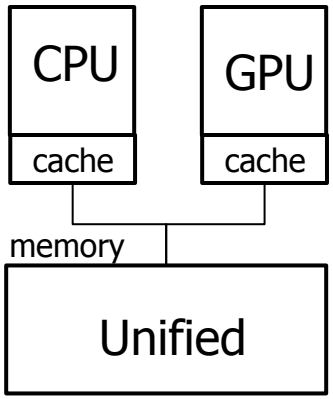
Georgia Tech comparch

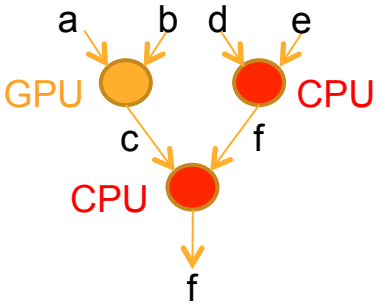
Memory Address Space Design Options



(Unified) (Disjoint) (Partially Shared) (ADSM)

Unified Memory Address Space

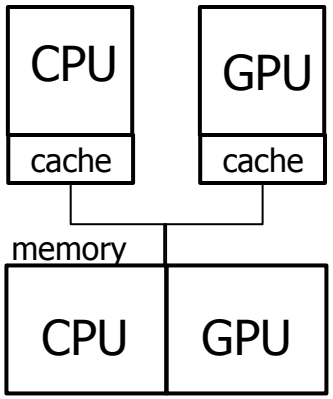


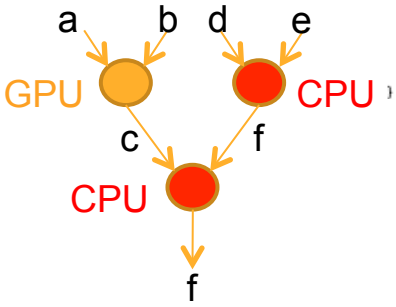



```
int addTwoVectors(int *a, int *b, int *c)
{
  for (i = 1 to 64) {
    c[i] = a[i]+b[i];
   }
}

int kernel(...)
{
   int *a = malloc(...);  int *b = malloc(...);
   int *c = malloc(...);  int *d = malloc(...);
   int *e = malloc(...);  int *f = malloc(...);

    for (i=1; i< 64; i++) // initialize
    {
    // initialize a, b, d, e
    }
    addTwoVectors (a, b, c);    // c = a+b
    addTwoVectors (d, e, f);    // f = d+e
    addTwoVectors (c, f, f);    // f = c+f
    ....
}
```

- Unified space: identical address space both for CPU and GPU
- Fully coherent memory space or virtually unified memory space (CUDA 4.0)
  - No explicit data transfer, but complicated TLB/MMU designs

Disjoint Address Space






```
int kernel(...)
{
    // initialize a, b, c, d, e, f  in CPU
   int *a = malloc(...);  int *b = malloc(...);
   int *c = malloc(...);  int *d = malloc(...);
   int *e = malloc(...);  int *f = malloc(...);

   // duplicated pointer for a, b, c
   int *gpu_a, *gpu_b, *gpu_c;

   // allocate mem space in the GPU
   int gpu_a = GPUmemallocate (gpu_a, gpu_b, gpu_c);

   // send data from CPU to GPU
    Memcpy(gpu_a, a, MemcpyHosttoDevice);
    Memcpy(gpu_b, b, MemcpyHosttoDevice);

    addGPUTwoVectors (a, b, c);    // c = a+b;
    addTwoVectors (d, e, f);    // f = d+e

   // send data from GPU to CPU
    Memcpy(gpu_a, a, MemcpyDevicetoHost);

    addTwoVectors (c, f, f);    // f = c+f

   // free spaces in both CPU and GPU
    ....
}
```

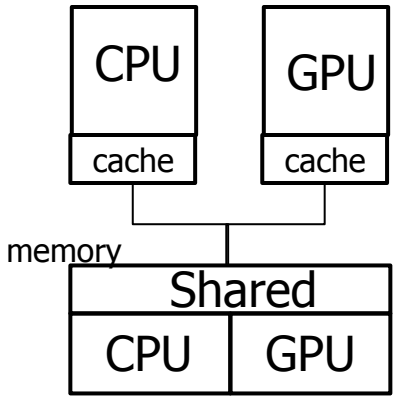
Explicit data transfer: C2G

Explicit data transfer: G2C

Disjoint memory space

- Scalable and easy to implement, but need explicit data transfer
- c.f.) physically shared cache can be used as disjoint address space (e.g. Intel's Sandia)

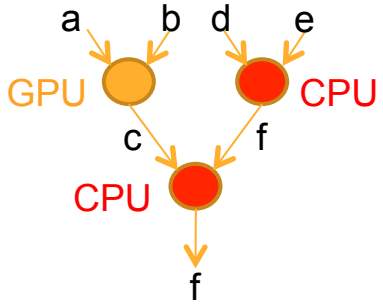
Partially-shared Address Space



Ownership control

```
attribute(GPU)
int addGPUTwoVectors(shared int *a,
    shared int *b, shread int *c)
{
  acquireOwnership(a,b,c);
  for (i=1 to 64) {
    c[i] = a[i] + b[i];
  }
  releaseOwnership(a,b,c);
}

int kernel(...)
{
  // allocate in shared region
  int *a = sharedmalloc(...);
  int *b = sharedmalloc(...);
  int *c = sharedmalloc(...);

  int *d = malloc(...);
  int *e = malloc(...);
  int *f = malloc(...);

 for (i=1; i< 64; i++) // initialize
   {
   // initialize a, b, d, e
   }
  releaseOwnership(a, b, c);
  addGPUTwoVectors(a,b,c); // c = a+b in GPU
  addTwoVectors(d,e,f);    // f = d+e in CPU
  acquireOnwership(c);
  addTwoVectors(c,f,f);    // f = c+f in CPU
```

Special malloc function



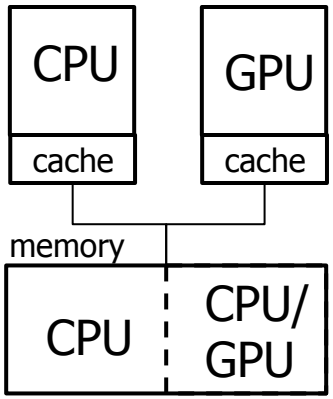
Ownership control

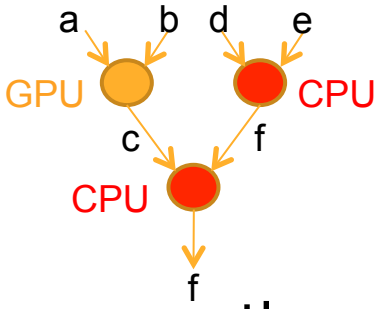
Partially-shared memory space

- Partially-shared: only part of the space is shared
- Introduced at Intel's LRB programming model
  - Ownership is maintained by programmers
  - Convenience of using shared memory, but overhead of managing between spaces

Georgia Tech comparch

Asymmetric Distributed Shared Memory (ADSM)

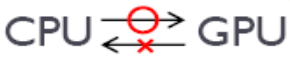





```
int kernel(...)
{

   // initialize a, b, c, d, e, f
   int *a = malloc(...);  int *b = malloc(...);
   int *c = malloc(...);  int *d = malloc(...);
   int *e = malloc(...);  int *f = malloc(...);

   // no duplicated GPU pointers
    a = adsmAlloc (64B);    // allocate mem space in the GPU
    b = adsmAlloc (64B);    // allocate mem space in the GPU
    c = adsmAlloc (64B);    // allocate mem space in the GPU

    copyfromCPUtoGPU(a,b,c);  // send data from CPU to GPU
    addGPUTwoVectors (a, b, c);    // c = a+b;

    addTwoVectors (d, e, f);    // f = d+e

    addTwoVectors (c, f, f);    // f = c+f
    accfree(a); accfree(b); accfree(c);
    ....
}
```

Explicit data transfer: C2G

ADSM



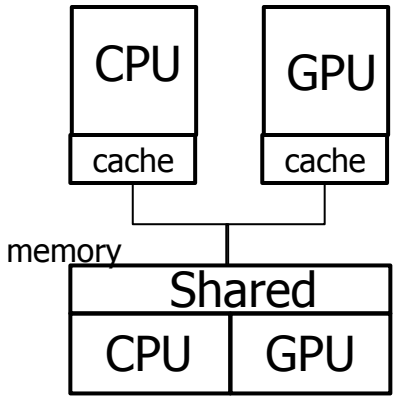
Gelado et al. ASPLOS10 (GMAC)

- ADSM: one PU can access the entire memory, but the other cannot
  - Provide a shared space with discrete memories
- ADSM uses a special memory allocation function, `adsmAlloc`, to allocate data into the shared memory space
- Unlike the disjoint memory address space, there is no need to transfer data back to the host memory space

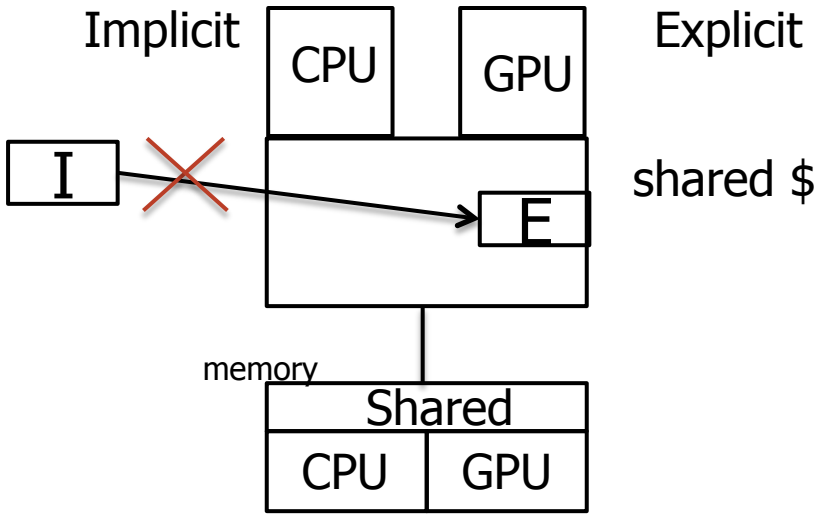
Georgia Tech comparch

Locality Management

- Implicit vs. Explicit management
  - Implicit: hardware manages locality (hardware cache)
  - Explicit: programmer manages locality (software managed cache)
- Shared memory space: all implicit, all explicit
- Various options in partially shared space
  - Implicit-private-explicit-shared
  - Explicit-private-Implicit-shared
  - Hybrid mechanisms
    - Implicit-private-explicit-private-explicit-shared (CPU and GPU have different management)
    - Implicit-private-explicit-private-implicit-shared (CPU and GPU have different management)
- Partially shared space provides the most number of options

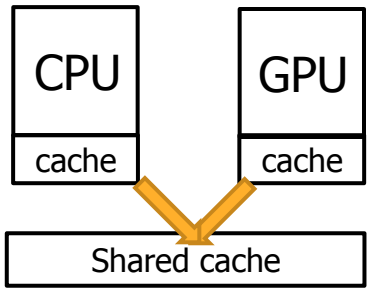


Hardware Implementation of Hybrid Mechanisms



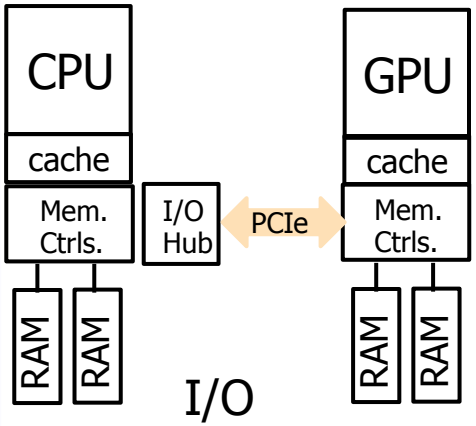
- If CPU and GPU share a cache
- Tag bit to indicate Explicit management
- Explicit cache size should be smaller than shared cache
- cache replacement policy: Implicit cache block cannot evict an explicit cache block

Communication Options (1)

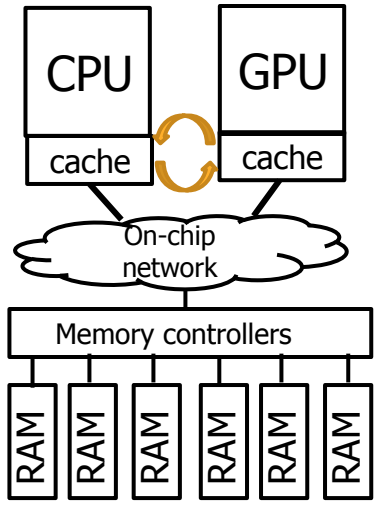


Physically shared cache

Shared object can be directly updated inside the shared cache



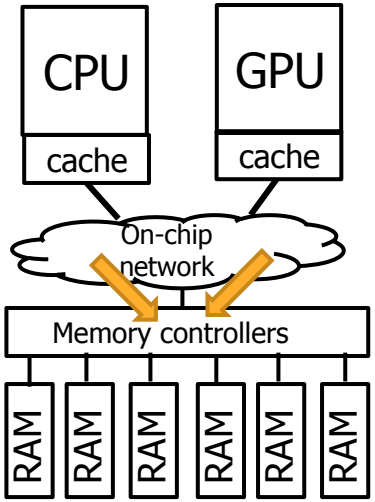
System BUS such as PCI-E, or a processor BUS

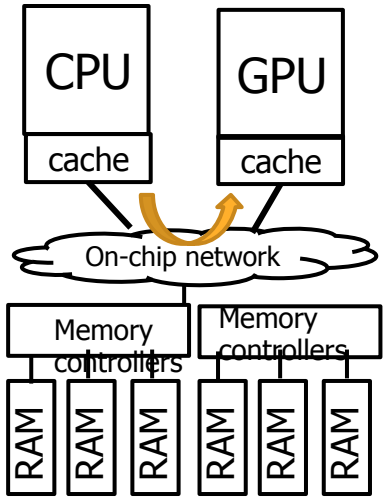


Cache coherence

Georgia Tech comparch

Communication Options (2)





Memory controllers

XBOX360 allows direct communication using the L2 cache for some graphics data, but they do not fully shared data using the cache

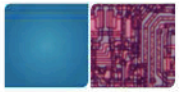
Does not require any software or hardware coherence support

Interconnection network + DMA

Uses an interconnection network system to directly communicate without necessarily going through memory controllers

e.g.)IBM cell

Georgia Tech comparch


Heterogeneous Architecture Summary

scheme	address space	Connection	coherence	how to use shared data	consistency	synchronization	Locality
CPU+CUDA* [28]	disjoint	PCI-E	-	NA	weak consistency	-	impl-pri-expl-pri
EXOCHI [32]	unified	Memory controller	can be coherent	CHI runtime API	weak consistency	unknown	impl-pri
CPU+LRB [30]	partially shared	PCI-E	coherent only in LRB/CPU	type qualifier, ownership	weak consistency	APIs	impl-pri
COMIC [21]	unified	interconnection	directory	COMIC API functions	centralized release consistency	barrier function	expl-pri-impl-pri-impl-shared
Rigel [18]	unified	interconnection	HW/SW	global memory operation	weak consistency	implicit barrier/Rigel LPI	expl
GMAC [9]	ADSM	PCI-E	GMAC protocol	global memory operation	weak consistency	sync API	expl-private-impl-shared
Sandy Bridge [14]	disjoint	Memory controller	-	-	weak consistency	-	impl-priv-exp-priv
Fusion [2]	disjoint	Memory controller	-	-	-	-	-
IBM Cell [13]	disjoint	interconnection	-	-	weak consistency	-	expl-pri-impl-priv-impl-shared
Xbox 360 [3]	disjoint	cache/FSB	-	Lock-set cache, copy	-	-	impl-priv-exp-shared
CUBA [8]	disjoint	BUS	-	direct access to local storage	weak consistency	-	exp-priv
CUDA 4.0	unified	-	-	explicit copy	weak consistency	-	exp-priv
OpenCL	unified	-	-	explicit copy	weak consistency	-	exp-priv

- None of the heterogeneous computing system has employed a unified, fully-coherent, strong-consistent memory system yet
- Most proposed/existing systems have disjoint memory systems

Georgia Tech comparch



Evaluation

- MacSim (GT) cycle-level simulator
- Intel's Sandy Bridge like configuration + NVIDIA's Fermi like GPU configuration
- Benchmarks

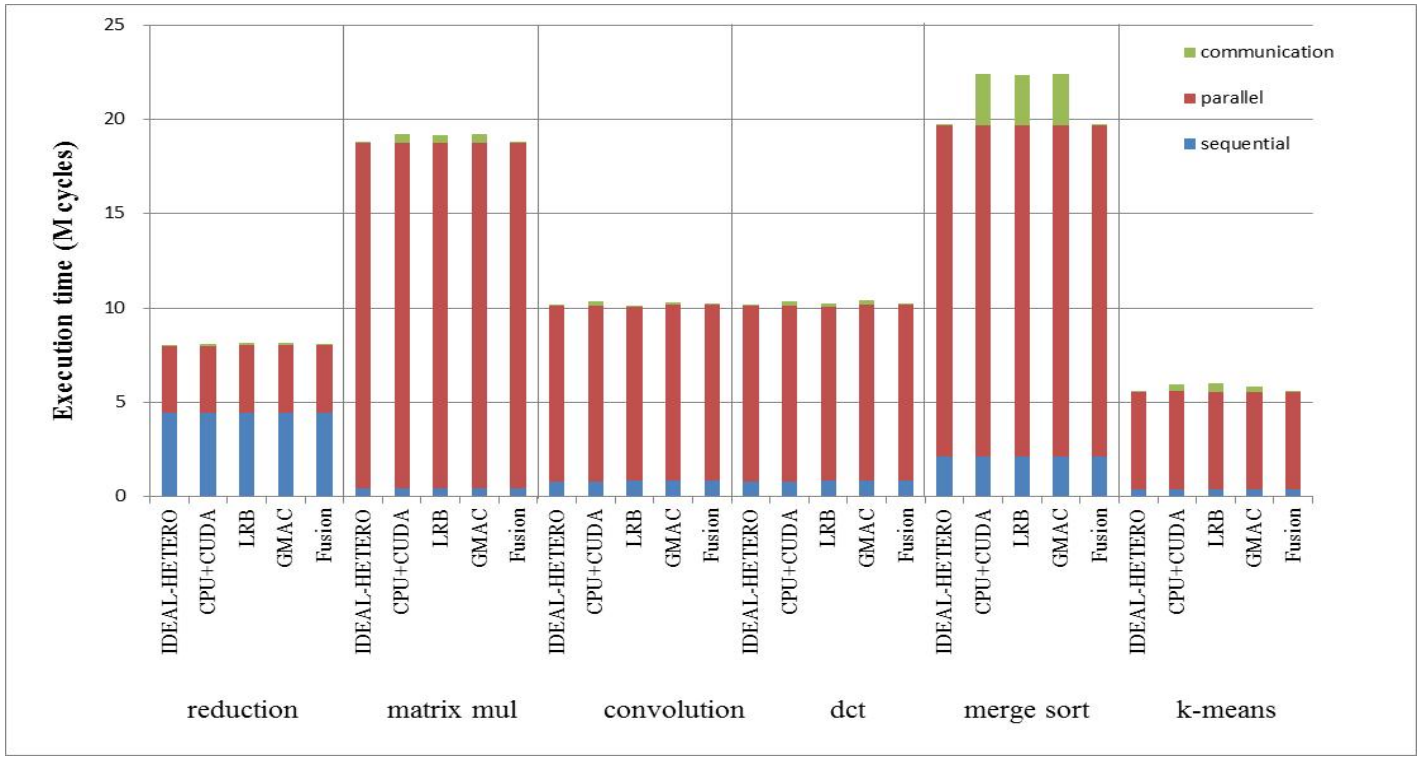
Name	compute pattern	# of instructions (CPU)	(GPU)	# (serial)	# of communications	initial transfer data size (B)
matrix mul [28]	fully parallel, no comm during computation	8585229	8585228	16384	2	524288
merge sort [28]	parallel -> merge -> sequential	161233	157233	97668	2	39936
dct [28]	fully parallel, no comm. during computation	2359298	2359298	262144	2	262244
reduction	parallel -> merge -> sequential	70006	70001	99996	2	320512
convolution [28]	parallel -> merge -> parallel	448260	448259	65536	3	65536
k-mean	parallel -> merge -> sequential (repeated)	1847765	1844981	36784	6	136192

- Parameters of modeling communication overhead

Name	Description	System	Latency (cycles)
api-pci	mem copy using PCI-E	CPU+CUDA, GMAC	33250+data_size/tr*f
api-acq	acquire action	LRB	1000
api-tr	data transfer	LRB	7000
lib-pf	page fault	LRB	42000

Georgia Tech comparch

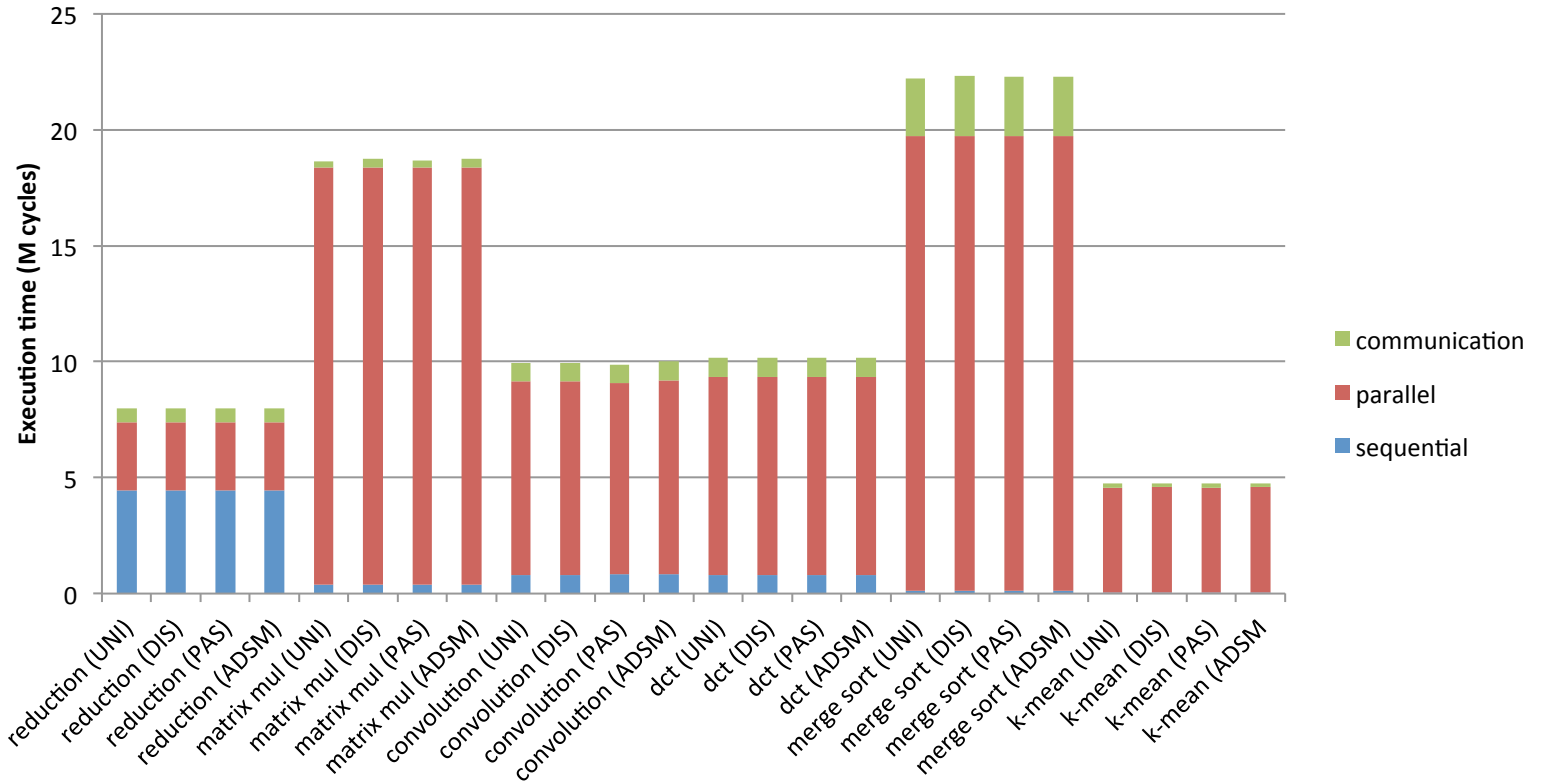
Evaluation of Five Heterogeneous Architecture Configuration



- Compare five systems
  - IDEAL-HETERO : unified and fully coherent
  - CPU+CUDA : disjoint space + PCI-E
  - LRB : partially-shared space + PCI aperture
  - GMAC : ADSM+PCI-E
  - Fusion : disjoint space + memory controller

Georgia Tech comparch

Memory Space Effects



- Not much performance difference

Programmability vs. Memory Address Space

	Comp	UNI	PAS	DIS	ADSM
matrix mul	39	0	2	9	6
merge sort	112	0	2	6	4
dct	410	0	2	6	4
reduction	142	0	2	9	6
convolution	75	0	4	9	6
k-mean	332	0	6	6	4

The number of source lines to handle data communication

- Different programming options affect how easy/difficult it is to write programs
- Use the number of source lines to indicate programmability
  - The number of additional source lines that are required to handle explicit data communication and data handling operations
- Unified < partially-shared <= ADSM < disjoint
  - Unified space does not require any special APIs
  - Disjoint memory space requires the most additional source code lines



Conclusion

- We exploited the design space of heterogeneous computing memory systems
- memory space does not affect performance significantly
- Partially shared memory space is the most promising option
  - provides many hardware design options (locality managements) and moderately good programmability

Georgia Tech comparch



Thank you!!

Georgia Tech comparch