

Heterogeneous-Race-Free Memory Models

Derek R. Hower[†], Blake A. Hechtman^{†§}, Bradford M. Beckmann[†], Benedict R. Gaster[†],
Mark D. Hill^{‡†}, Steven K. Reinhardt[†], David A. Wood^{‡†}

[†]AMD Research

{derek.hower, brad.beckmann,
benedict.gaster,steve.reinhardt}@amd.com

[§]Duke University

Electrical and Computer Engineering

blake.hechtman@duke.edu

[‡]University of Wisconsin-Madison

Computer Sciences

{markhill, david}@cs.wisc.edu

Jyh-Jing (JJ) Hwang, Yiren (Max) Lu

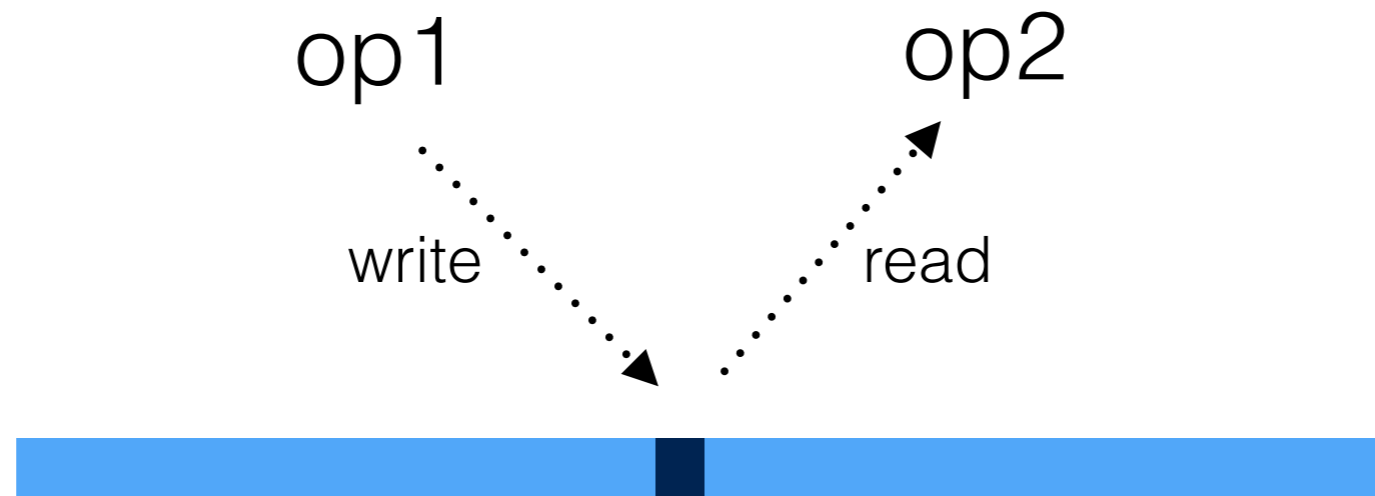
02/28/2017



Outline

- 1. Background**
- 2. HRF-direct**
- 3. HRF-indirect**
- 4. Experiments**

Data Race Condition



Sequential Consistency for Data Race Free (1990/C++)

```
atomic<int> A = {0};  
atomic<int> B = {0};  
int X;
```

Thread t1

```
X = 1;  
A.store(1, memory_order_seq_cst);    // release
```

Thread t2

```
while(!A.load(memory_order_seq_cst)); // acquire  
int R2 = X; // R2 will receive X=1 from t1  
B.store(1, memory_order_seq_cst);    // release
```

Thread t3

```
while(!B.load(memory_order_seq_cst)); // acquire  
int R3 = X; // R3 will receive X=1 from t1
```

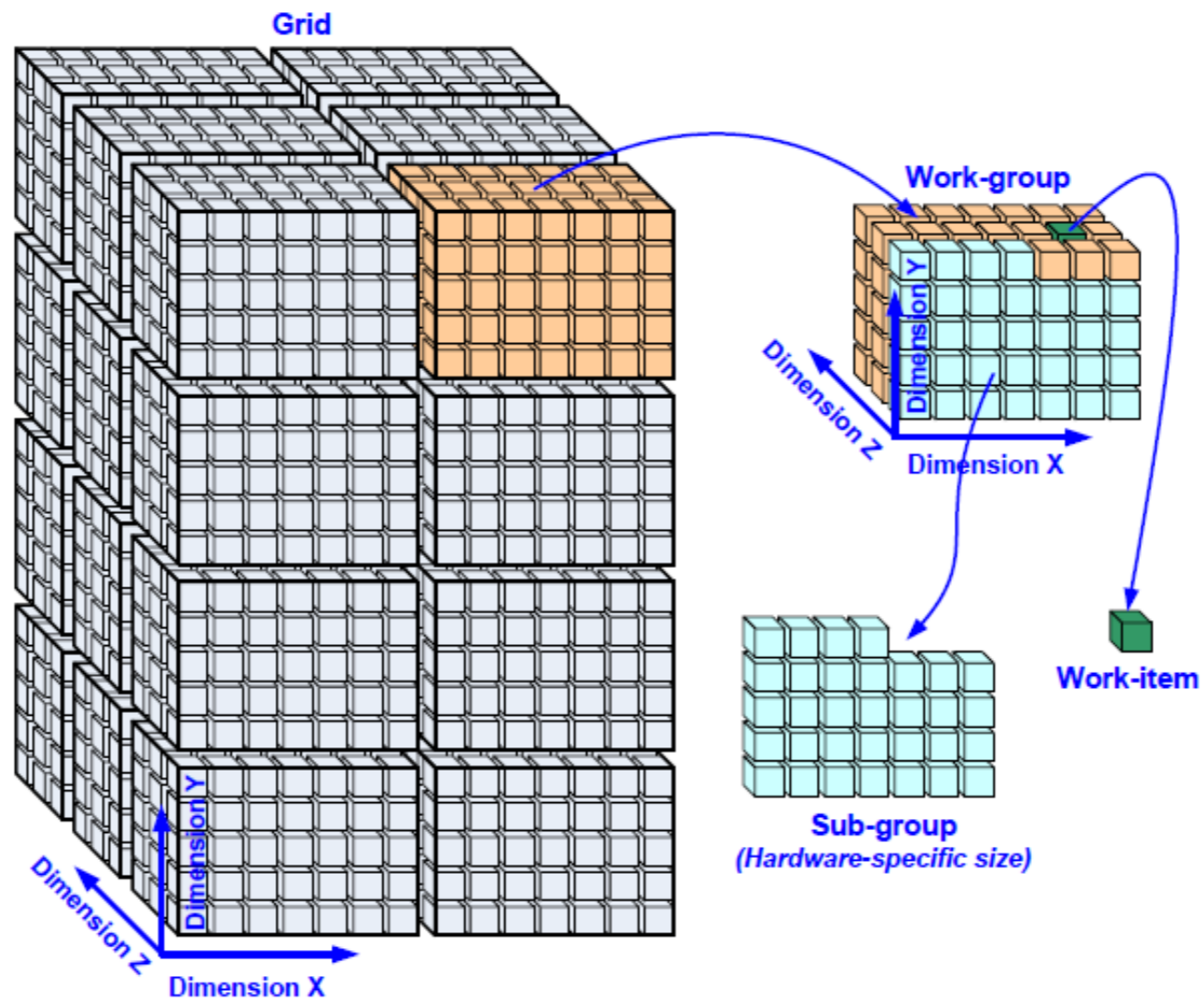
Why SC for DRF cannot be adopted for heterogeneous systems?

- Synchronization is only among a subset of threads (**scope**)
- The **latency** of global communication is too high

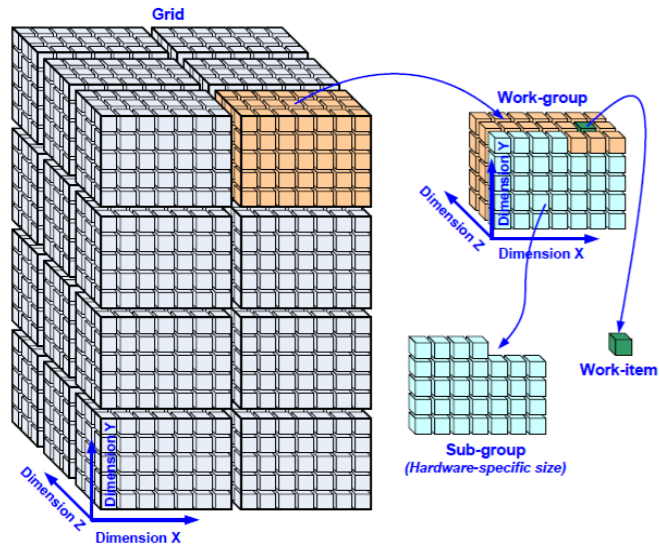
Contributions of the Paper

1. Definition of **Scoped Synchronization**
2. Sequential Consistency for Heterogeneous-race-free (**SC for HRF**) memory model
3. HRF-direct
4. HRF-indirect

Scopes



Scopes



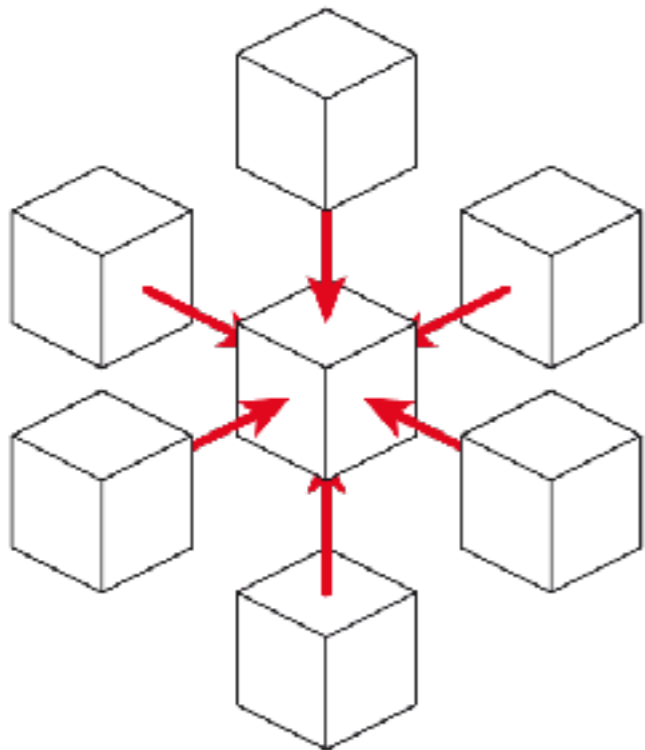
OpenCL	CUDA	Memory	Process Units
work-item	thread	local memory	stream processors (SPs)
sub-group (wavefront)	warp		
work-group	block (cooperating thread array "CTA")	shared memory	multiprocessor (MP)
device (NDRange)	grid	global memory	GPU

HRF-direct: Basic Scope Synchronization

- Requires communicating work-items to synchronize using the exact **same scope**
- Choose the **minimal** scope for synchronization

An Example of HRF-direct Algorithm

```
For t=0; t < nsteps; t+= group_step_size:  
  group_step_size times do:  
    <stencil calculation on shared grid>  
    BarrierSync(work-group scope)  
    BarrierSync(device scope)  
    <exchange work-group boundary conditions>
```



“Stencil codes are a class of **iterative kernels** which update array elements according to some **fixed pattern**, called stencil.” -wikipedia

Definition of HRF-direct

Definitions for a Sequentially Consistent Candidate Execution

Program Order (\overrightarrow{po}): $op1 \overrightarrow{po} op2$ iff both are from the same work-item or thread and $op1$ completes before $op2$.

Scoped Synchronization Order ($\overrightarrow{so_S}$): Release $rel1$ appears before acquire $acq2$ in $\overrightarrow{so_S}$ iff both are performed with respect to scope S , and $rel1$ completes before $acq2$.

Heterogeneous-happens-before-direct ($\overrightarrow{hhb.d}$): The union of the irreflexive transitive closures of all scope synchronization orders with program order:

$$\bigcup_{S \in \mathcal{S}} (\overrightarrow{po} \cup \overrightarrow{so_S})^+$$

where \mathcal{S} is the set of all scopes in an execution. Note that in the above equation, the closure applies only to the inner union and is not applied to the outer union.

Heterogeneous Race: A heterogeneous race occurs iff a pair of operations $op1$ and $op2$ that are either an ordinary or a synchronization conflict are unordered in $\overrightarrow{hhb.d}$.

Heterogeneous-race-free Execution: An execution is heterogeneous-race-free iff there are no heterogeneous races.

Definition of HRF-direct

Program and Model Definitions

Heterogeneous-Race-Free Program: A program is heterogeneous-race-free iff all possible sequentially consistent executions of the program are heterogeneous-race-free.

Sequential Consistency for Heterogeneous-race-free-direct (HRF-direct): A system obeys the HRF-direct memory model iff all actual executions of a heterogeneous-race-free program are sequentially consistent.

Difference between HRF-direct and DRF

- HRF-direct considers synchronization operations performed at **different scopes** to be **conflicting**
- HRF-direct creates a separate happens-before order relative to **each scope**

Subtleties of synchronization order

```
__global atomic<int> A = {0};  
__global atomic<int> B = {0};  
// __global means "not in a scratchpad" in OpenCL
```

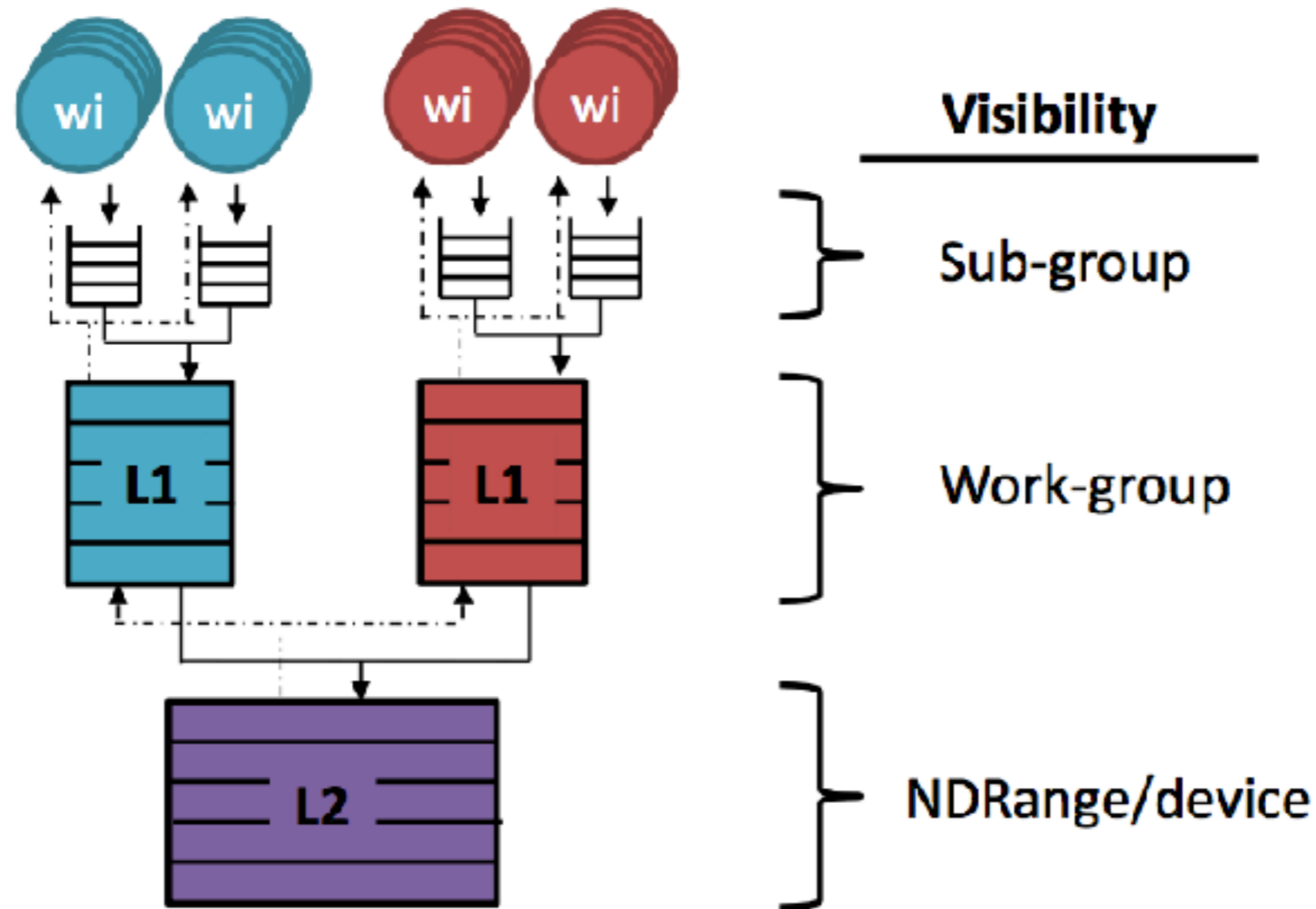
Work-item wi1

```
A.store(1, memory_order_seq_cst,  
        memory_scope_work_group);  
B.load(memory_order_seq_cst,  
        memory_scope_device);
```

Work-item wi2

```
B.store(1, memory_order_seq_cst,  
        memory_scope_device);  
  
// A.load forms a race when  
// workgroup of wi1 ≠ workgroup of wi2  
A.load(memory_order_seq_cst,  
        memory_scope_work_group);
```

Hardware Implementation for HRF-direct



Limitations of HRF-direct

- Difficult to optimize applications with **irregular parallelism** in which the producer and consumer may not be known a priori.
- HRF-direct is overly **conservative** for current GPU memory systems

HRF-indirect: Adding Transitivity

- Build on HRF-direct model by permitting **transitive chain** of synchronization in **different scopes**

HRF-indirect: An Example

```

__global atomic<int> A = {0};
__global atomic<int> B = {0};
__global int X;
// __global means "not in a scratchpad" in OpenCL

```

Work-item wi1 -- in work-group wgX

```

11: X = 1;
12: A.store(1, memory_order_seq_cst, // release
        memory_scope_work_group);

```

Work-item wi2 -- in work-group wgX

```

21: while(!A.load(memory_order_seq_cst, // acquire
        memory_scope_work_group));
22: int R2 = X; // R2 will get value 1
23: B.store(1, memory_order_seq_cst, // release
        memory_scope_device);

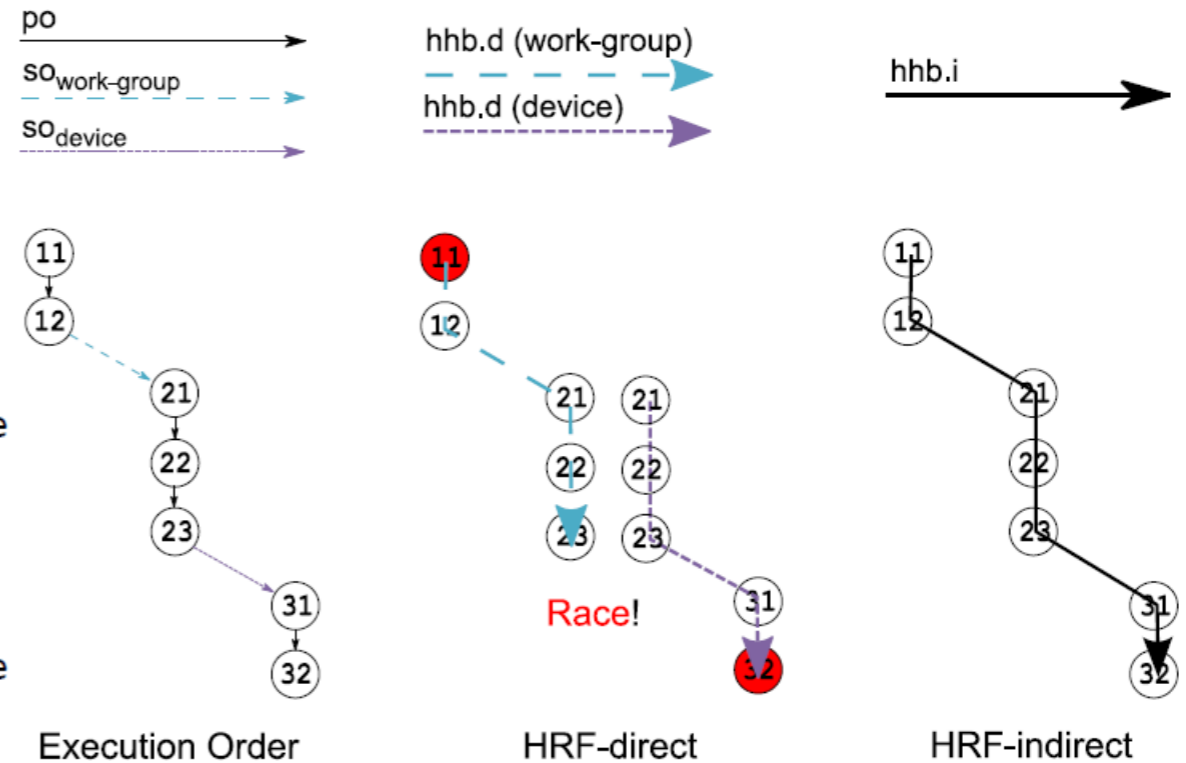
```

Work-item wi3 -- in work-group wgY (Note! not wgX)

```

31: while(!B.load(memory_order_seq_cst, // acquire
        memory_scope_device));
32: int R3 = X; // R3 will get value 1

```



- In HRF-direct, 11 and 32 form a **race**, because $X=1$ might not propagate from cache (work group) to memory (device)

HRF-indirect: Note & Advantages

- Each synchronization pair may use the **same scope**
- 1. Producers and consumers aren't aware of each other, which may **accelerate**
- 2. Programmers have more **flexible** choice of scope
- HRF-direct is **compatible** with HRF-indirect
 - Race-free in HRF-direct is also race-free in HRF-indirect

HRF-indirect: Formal Definition

Formal Definition of HRF-indirect

The structure of HRF-indirect is identical to the HRF-direct definition but with a different happens-before relation:

Heterogeneous-happens-before-indirect ($\overrightarrow{\text{hhb.i}}$): The ir-reflexive transitive closure of all scope synchronization orders and program order:

$$\left(\overrightarrow{po} \cup \bigcup_{\forall S \in \mathcal{S}} \overrightarrow{so_S} \right)^+$$

where \mathcal{S} represents the set of all scopes in an execution.

HRF-indirect: System Optimizations

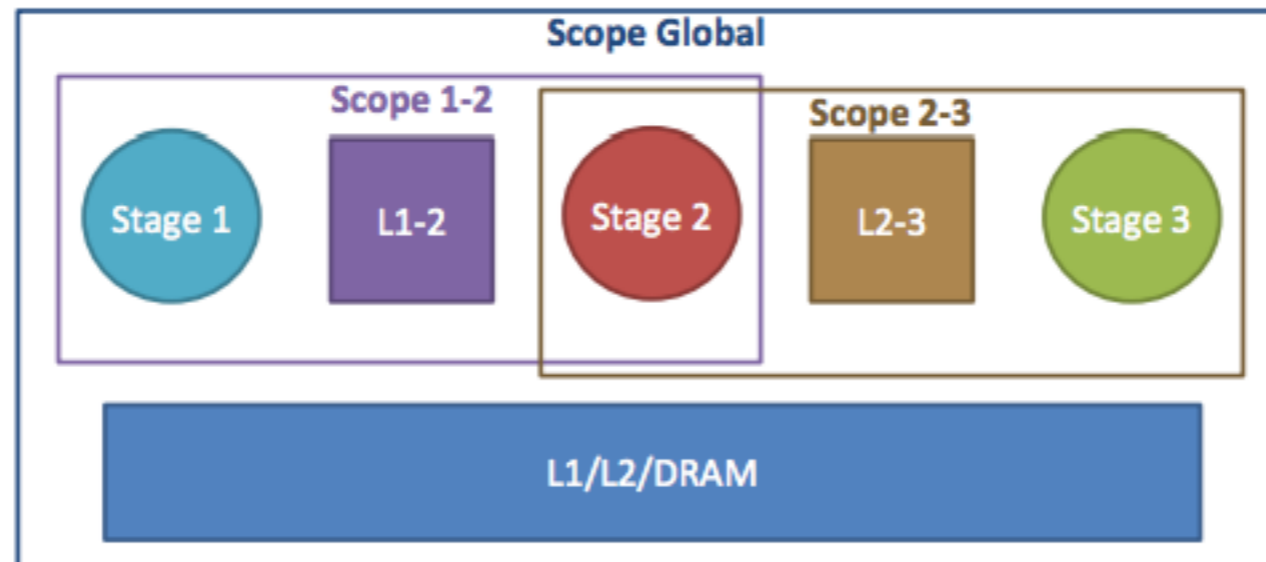


Figure 8. Architecture and scope layout of a possible programmable pipeline.

- Build a **cache** between the stages that can be controlled with an HRF scope.

Memory Model Comparisons

- **DRF**: Both HRF-direct and HRF-indirect are compatible
 - DRF with system scope sync -> adjusting scope -> HRF-direct
- **OpenCL 2.0**: More restrictive than HRF-direct
 - SC for HRF when (memory_order_seq_cst) is used
 - Use **either** device scope or system scope
 - Intermix of different scoped atomics is **disallowed**
- **CUDA**: Compatible with HRF-direct
 - **__threadfence** acts in block, device, or system scope.
 - Not sure about scope transitivity for HRF-indirect

Future HRF Models: Scope Inclusion

Sync when the scope of one includes the scope of the other

```
__global atomic<int> A = {0};  
__global atomic<int> B = {0};  
// __global means "not in a scratchpad" in OpenCL
```

Work-item wi1 - in workgroup wgX

```
A.store(1, memory_order_seq_cst,  
        memory_scope_device);  
B.load(memory_order_seq_cst,  
        memory_scope_device);
```

Work-item wi2 - in same workgroup wgX

```
B.store(1, memory_order_seq_cst,  
        memory_scope_work_group);  
A.load(memory_order_seq_cst,  
        memory_scope_work_group);
```

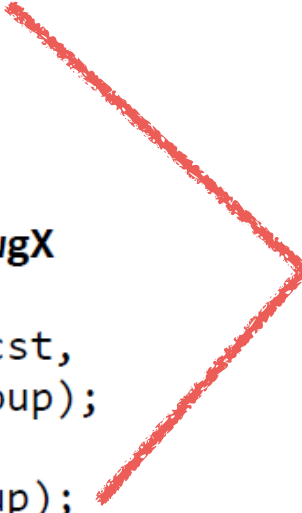


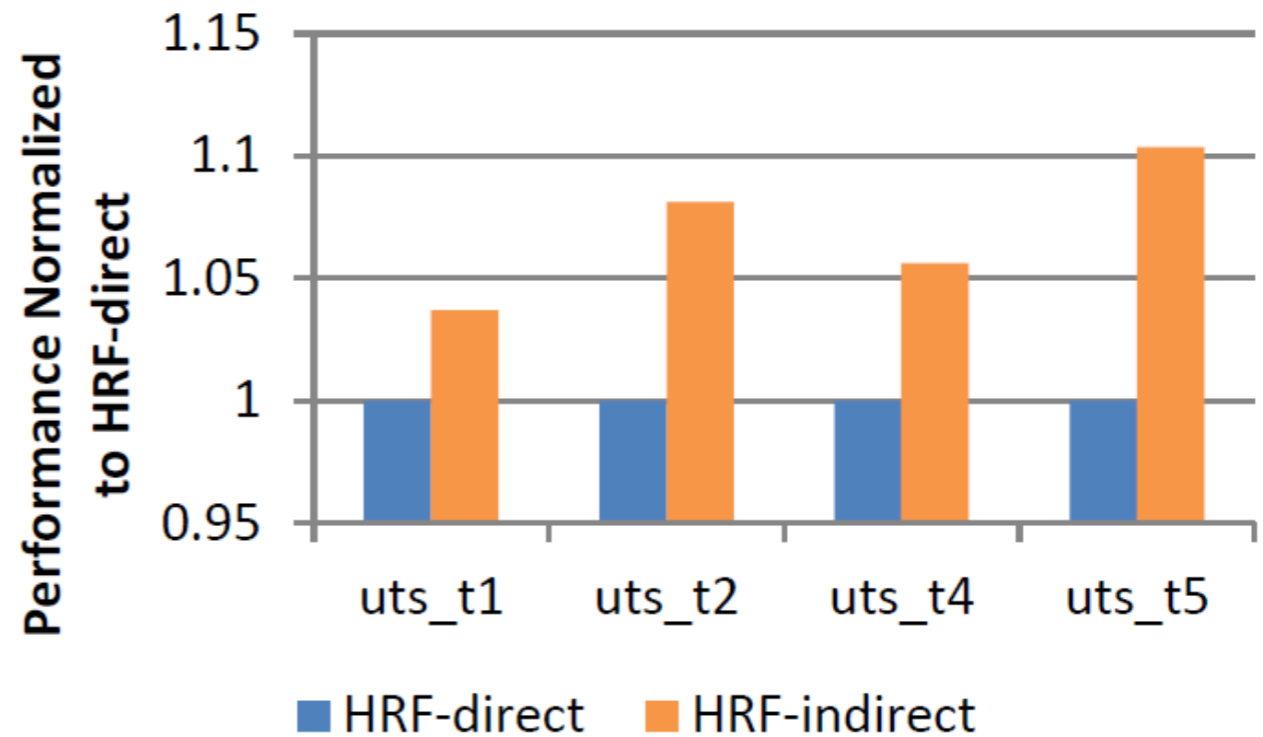
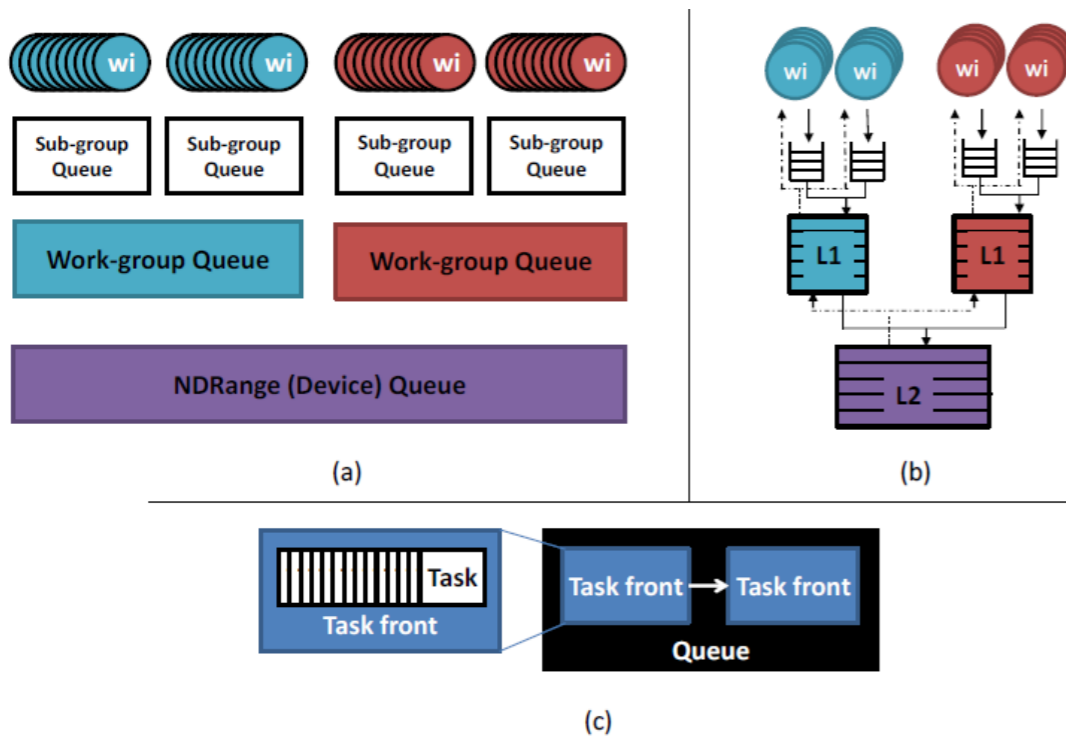
Figure 7. An example of scope inclusion.

Experiments

- Unbalanced Tree Search (uts) Tasks
- Each tree contains 125K Nodes
- gem5 simulator
- 3%-10% speed-ups

Table 1. Configuration Parameters

Parameter	Value
# Compute Units	8
# SIMD Units / Compute Unit	4
L1 cache	32Kb, 16-way
L2 cache	2MB, 16-way



Conclusion

- Introduced **SC for HRF**, a class of memory models to define behavior of scoped sync in heterogeneous systems
- Explained why DRF is **insufficient** for heterogeneous systems
- Proposed two initial SC for HRF models
- **HRF-direct** requires sync in the same scope. (**present**)
- **HRF-indirect** can sync transitively using different scopes and achieve higher performance. (**future**)

Thank you!

**Any Questions?
(We actually have got A LOT!)**



Semantic Questions

- **Q: In the context of the paper, since scope is mentioned as either being device or work-group, does the term "scope" imply global/shared memory of the data?**
- **A: Please see our slide #8.**

Semantic Questions

- **Q: Even though I understand the intuition of hrf-direct vs. hrf-indirect the formalized definitions don't really make sense to me, can we go over these?**
- **A: We've gone over them in our presentation. :)**

Semantic Questions

- **Q: Are programmers expected to add in transitivity in HRF-indirect?**
- **A: No, it should be supported by hardware and/or compiler.
Please refer to slide #21.**

Semantic Questions

- **Q: In Section 5.2 (Scope Inclusion) the paper talks about one scope being a subset of another. What is the difference between this and overlapping scopes? Related to this, how does scope inclusion actually work based on the example in Figure 7.**
- **A: They are the same (subset = overlapping). We've talked about Figure 7 in our presentation.**

Semantic Questions

- **Q: Figure (7) in the paper describes a scheme of scope inclusion that is almost identical to Figure (5), which is when programming with atomics leads to racey execution. I understand how Figure 5 causes a race but I don't fully understand the scope inclusion method that is proposed to fix this problem. Could you please elaborate on that?**
- **A: We've talked about Figure 7 in our presentation.**

Semantic Questions

- **Q: Considering the case put forth by this paper that a program composed entirely of atomics can have data races, what does "atomics" really mean in this context?**
- **A: Atomics means the operations that occupy the resource exclusively, so that data race will not happen (in traditional multi-thread CPU programs).**

Semantic Questions

- **Q: The paper talks about data races even using only atomics; Is this because of store/load caching?**
- **A: Partially yes. We could put it in another way that in GPU/heterogeneous systems, threads/wi's in different blocks/wg's do not communicate with each other and have their own cache. Therefore, the atomic operation is not across blocks.**

Semantic Questions

- **Q: Page 6, section 3.3 Subtleties of Synchronization. As per the example in Fig 5, we are given 2 atomic synchronization operations A and B. B's scope is device and A's scope is work group. Since B's scope is device, how is B going to have a race ? A's scope is work group. Shouldn't only A have possibility of a race if $wi1$ and $wi2$ belong to different work groups ?**
- **A: B is not going to have a race; A is. If $wi1$ and $wi2$ are not in the same work-group, their memory scopes are different. That being said, A.load in $wi2$ may not load what is written from A.store in $wi1$ for their communication has to go through device scope first.**

Semantic Questions

- **Q: Page 6, section 3.3 Subtleties of Synchronization. As per the example in Fig 5, we are given 2 atomic synchronization operations A and B. B's scope is device and A's scope is work group. Since B's scope is device, how is B going to have a race ? A's scope is work group. Shouldn't only A have possibility of a race if $wi1$ and $wi2$ belong to different work groups ?**
- **A: B is not going to have a race; A is. If $wi1$ and $wi2$ are not in the same work-group, their memory scopes are different. That being said, A.load in $wi2$ may not load what is written from A.store in $wi1$ for their communication has to go through device scope first.**

Cache Questions

- **Q: How would one make potentially shared data uncachable, and along the same lines, how does one determine something is “potentially” shared? (pg 430, paragraph 2)**
- **A: Let’s say two work items from different work-groups synchronize. They have their own cache and share the same memory. In this way, the shared data cannot be cached; otherwise, it will form data race. It’s very hard to tell if some data is “potentially” shared. One way is to let programmers specify it; the other is to let compiler figure it out. This is why we need HRF memory models.**

Cache Questions

- **Q: What in the architecture of a GPU allows us to synchronize across separate scopes? Is it based on the fact that when we load a new scope, we flush the cache, effectively synchronizing across scopes?**
- **A: Yes.**

Cache Questions

- **Q: Could you please provide an example for the 3 step implementation of HRF-direct from section 3.2 paragraph 2. I can't clearly visualise the invalidation of "stale data from caches/memory that are not visible to all threads in the scope of synchronization".**
- **A: Please see slide #15.**

Cache Questions

- **Q: As per Section 6 of the paper, HRF-direct optimization can selectively flush to cache while consistency scope prevents HRF-indirect from doing so. Why is it not possible to use directory based coherence like CPUs with HRF-indirect? i.e line-wise or block-wise flushing as compared to flushing whole cache?**
- **A: We don't know. Let's ask Joe! :)**

Optimization Questions

- **Q: Are there programs that are race-free in HRF-direct, for which HRF-indirect cannot exploit as much parallelism as HRF-direct?**
- **A: Yes, HRF-indirect does not benefit those programs with regular parallel. For example, the matrix transpose program in class.**

Optimization Questions

- **Q: The HRF- indirect version is said to provide better performance than the direct method. Is it possible that as the complexities of system increases, HRF-direct may lead to better performance as it is a more simplified version?**
- **A: Yes, I believe it's possible.**

Optimization Questions

- **Q: Other than the flush optimization mentioned, what other hardware optimizations could be limited by the HRF-indirect model?**
- **A: We haven't come out one yet... Let's ask Joe! :)**

Optimization Questions

- **Q: If my understanding is correct, currently HRF-direct has higher possibilities for optimization, but in the future this will flip and HRF-indirect will optimize better? Can we talk about why?**
- **A: In the authors' viewpoint, HRF-indirect is superior to HRF-direct since it doesn't require synchronize in the same scope. The example given in the HRF-indirect section explains this situation.**

Optimization Questions

- **Q: HRF-direct is mentioned to be a popular choice in existing GPU architectures. The paper also describes synchronization using polling. Can HRF-direct be optimized further by the use of event driven synchronization? Will the additional time taken by a kernel level interrupt be preferable to the current power wastage by a blocking, polling loop over multiple work items in numerous computational units?**
- **A: The paper does not mention pooling, so let's ask Joe. :)**

Optimization Questions

- **Q: Compare the overhead of synchronization on the GPU at a system-scope level in HRF - direct versus the synchronization overhead on the CPU at a process level in SC DRF models. Would it be more beneficial to tailor our work loads toward the CPU or GPU if synchronization is required? What about when compared to HRF - indirect?**
- **A: There's no clear answer to this question. It all depends on the program you're writing and how parallel it is. But keeping the HRF-direct in mind (HRF-indirect is not supported by hardware yet) would help you judge whether you should run your program entirely on GPUs or partially on CPUs and on GPUs.**

CUDA Questions

- **Q: How does this work relate to the CUDA and GPU that we are using in this class? I suspect that it does not have SC for HRF. Is that right? How do we get rid of unexpected behavior then?**
- **A: We've talked about this. And unexpected behavior is the topic of the next paper (and we'll present it too!)**

etc. Questions

- **Q: What is a stencil calculation? (Figure 3)**
- **A: Wiki is your second best friend! ;)**
https://en.wikipedia.org/wiki/Stencil_code

etc. Questions

- **Q: What exactly do the authors mean by the terms irregular and regular parallelism that are used throughout the paper?**
- **A: Irregular parallelism means the threads in a program have very different workloads. That is, their memory access and execution time may vary a lot. On the other hand, regular parallelism means every thread in a program has balanced workload. For example, matrix multiplication has regular parallelism.**