A LOOK AT THE OPENCL 2.0 EXECUTION MODEL

Benedict R. Gaster / @cuberoo_

UWE BRISTOL
University of the West of England

bettertogether

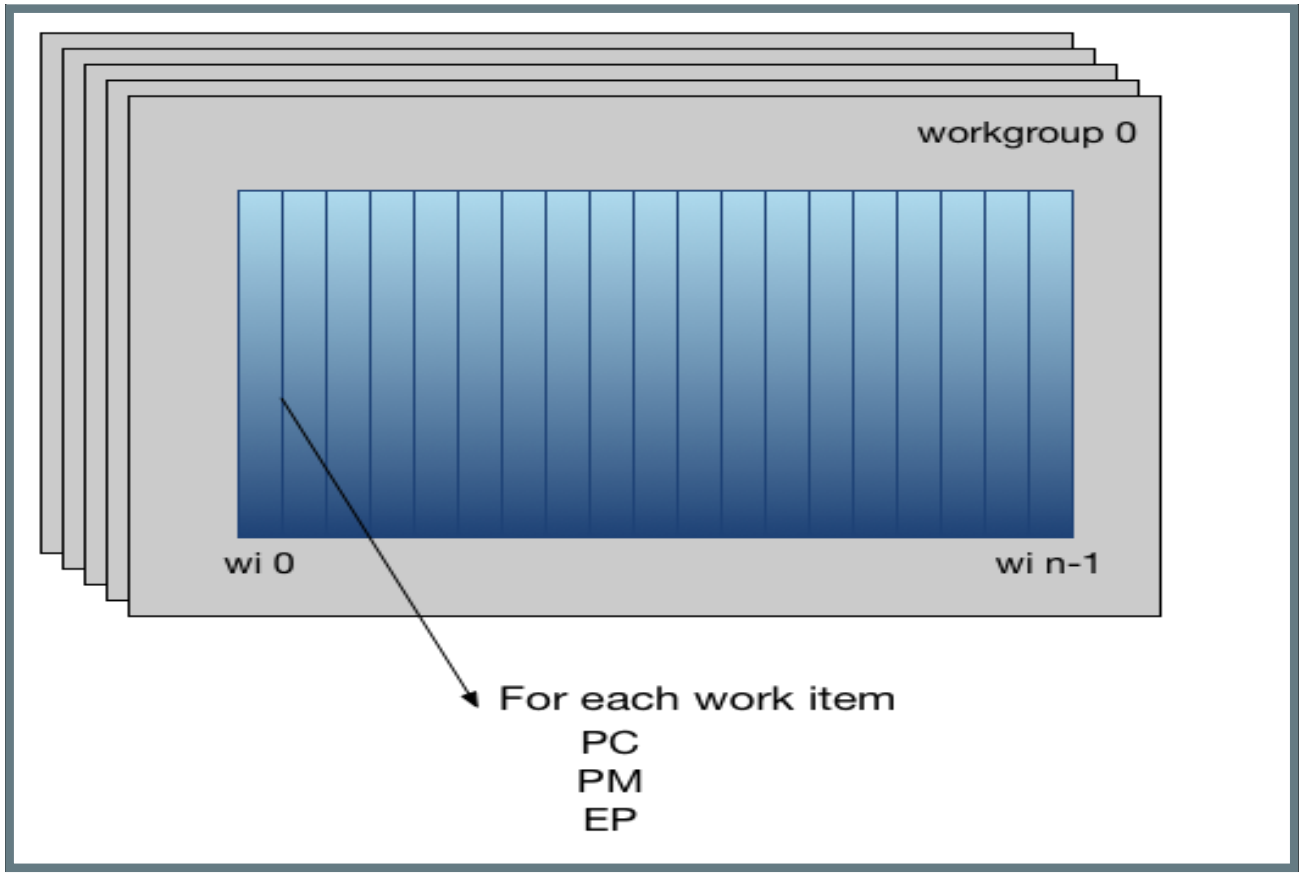
OPENCL 1.X EXECUTION MODEL

I'm going to assume we all know the 1.x NDRange model

OPENCL 2.0 SUBSUMES 1.X EXECUTION MODEL

Of course, OpenCL 2.0 supports this data-parallel model

OPENCL 2.0 EXECUTION MODEL (DATA PARALLEL MODE)



EACH WORKITEM

- PC - Program counter
- PM - Private memory (i.e. registers)
- EP - Execution predicate (is enabled)

EACH WORKITEM

- PC - Program counter
  - conceptually same for each workitem
- PM - Private memory (i.e. registers)
- EP - Execution predicate (is enabled)

SYNCHRONIZING COMMUNICATION

```
kernel foo(...) {
  local int l[WORK_GROUP_SIZE_PLUS_ONE];
  l[WORK_GROUP_SIZE_PLUS_ONE-1] = 0;
  barrier(CLK_LOCAL_MEM_FENCE );

  l[get_local_id(0)] = f(...);

  barrier(CLK_LOCAL_MEM_FENCE );

  int v = l[get_local_id(0) + 1];

}
```

DIVERGENCE CAN BE BAD...

```
kernel foo(...) {
  local int l[WORK_GROUP_SIZE_PLUS_ONE];
  l[WORK_GROUP_SIZE_PLUS_ONE-1] = 0;
  barrier(CLK_LOCAL_MEM_FENCE );)

  l[get_local_id(0)] = f(...);

  if (b) {
    barrier(CLK_LOCAL_MEM_FENCE );
  }

  int v = l[get_local_id(0) + 1];

}
```

EVEN WORSE...

```
kernel foo(...) {
  local int l[WORK_GROUP_SIZE_PLUS_ONE];
  l[WORK_GROUP_SIZE_PLUS_ONE-1] = 0;
  barrier(CLK_LOCAL_MEM_FENCE );)

   bar(l,...); does bar contain a barrier?

  int v = l[get_local_id(0) + 1];

}
```

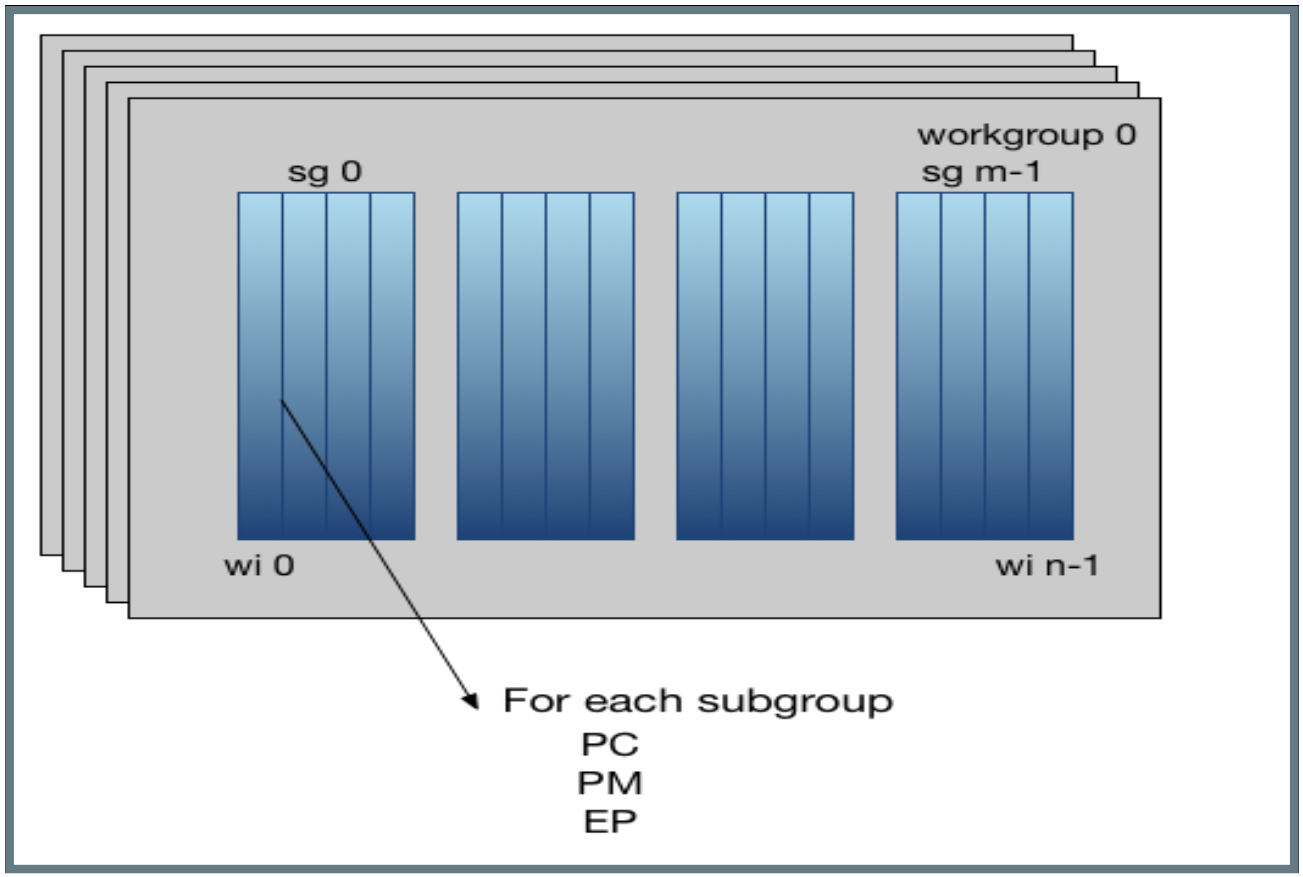
OPENCL 2.0 CAN HELP

- An extended excution model
- An actual memory model

SUBGROUPS

- A "SIMD" hardware thread
- Extension since OpenCL 2.0

OPENCL 2.0 EXECUTION MODEL+ (VECTOR PARALLEL MODE)



EACH SUBGROUP

- PC - A single program counter
- PM - Private memory for each workitem
- EP - A vector Execution Predicate (one mask for each workitem)

OPENCL 2.0 PROVIDES

Independent forward progress between each subgroup

OPENCL 2.0 PROVIDES

How does this help with our barrier problem(s)?

BARRIER OBJECTS

What if we made barriers first class?

FIRST CLASS BARRIER OBJECTS

- Regain composibility
- Subsets of workitems could communicate

BUT...

- OpenCL 2.0 does not have first class barriers!
- HSA has fbarriers, which amount to the same thing!

BARRIER OBJECTS

Well that is disappointing!

BARRIER OBJECTS

Could we define our own?

REQUIREMENTS FOR BARRIER OBJECTS

- Supports synchronized communication between subsets of workitems
- Foward progress between communicating workitems

IMPLEMENTING REQUIREMENTS FOR BARRIER OBJECTS

- Supports synchronized communication between workitems
  - OpenCL 2.0's memory model provides what we need
- Foward progress between communicating workitems
  - Subgroups provide independent forward progress

BARRIER OBJECTS API

```
// create a barrier
barrier_t create_barrier(int num_subgroups, barrier_t bobj);

// take part and wait
void wait(barrier_t, memory_order mo, memory_scope);

// take part but do not wait
void arrive(barrier_t, memory_order, memory_scope scope);
```

BARRIER OBJECTS IMPLEMENTATION

- Fairly straightforward using a noiton of sense, to enable reuse
- Must use sub_group_barrier internally for workitems within subgroup
- Replies on relaxed atomics for memory consistency of barrier object

OUR EXAMPLE AGAIN...

```
kernel foo(...) {
  local int l[WORK_GROUP_SIZE_PLUS_ONE];
  l[WORK_GROUP_SIZE_PLUS_ONE-1] = 0;

  barrier_t b(get_num_subgroups());

  barrier(CLK_LOCAL_MEM_FENCE );

  bar(l,b); //now we know it (likely) uses a barrier?

  int v = l[get_local_id(0) + 1];

}
```

CONCLUSION

- Impossible to do justice to OpenCL 2.0 execution model in 15 minutes!
- Subgroups provide an important new design point
  - Forward progress adds the ability to reason about producer/consumer
  - Barrier objects address concerns with composibility
  - Expose underlying hardware to enable portable optimizations that were already being done in non-portable ways (not covered here)
- Have not mentioned "nested parallelism", you may wonder why?