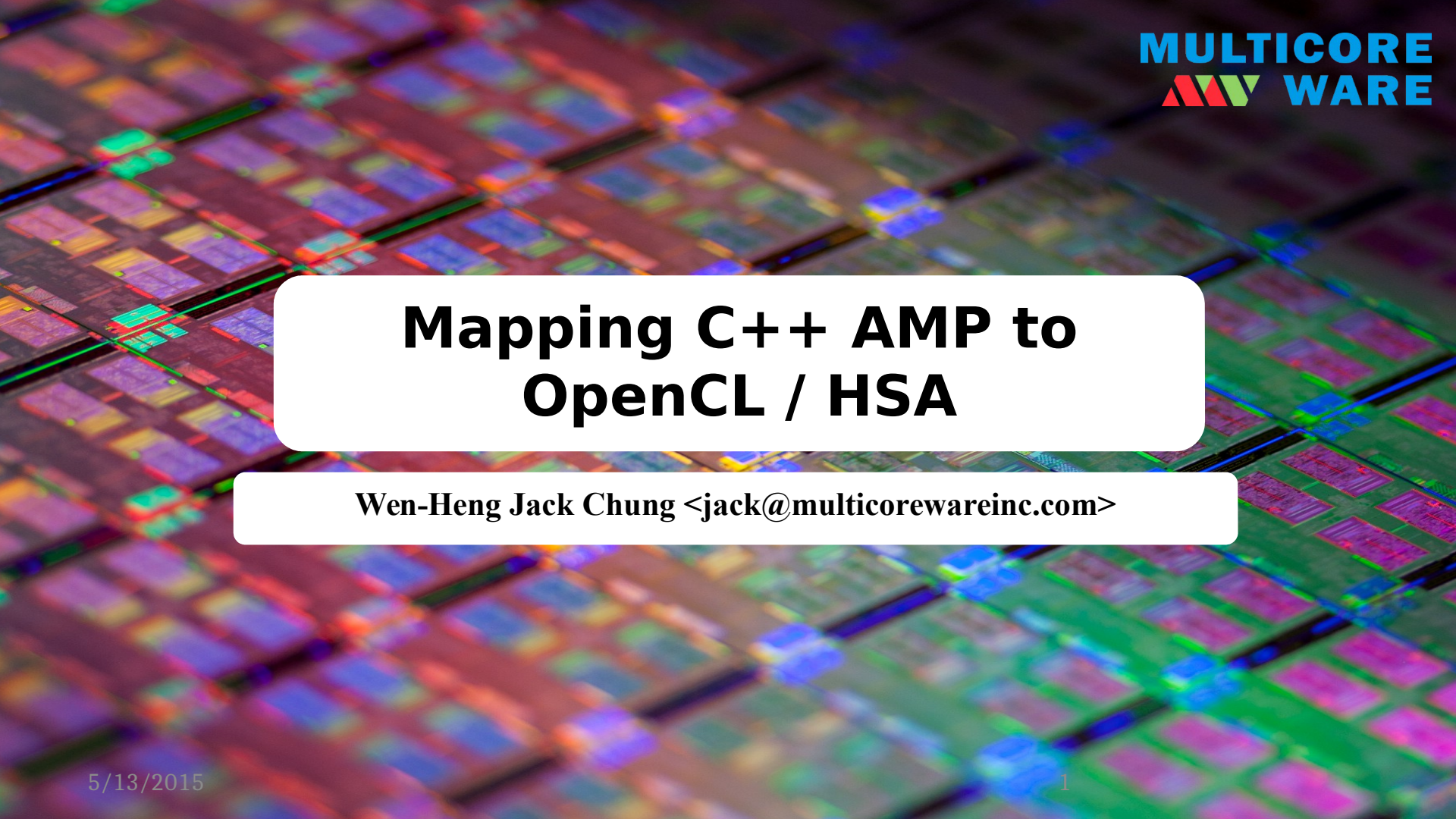
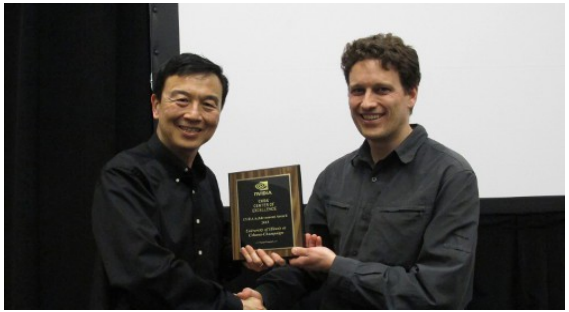


Mapping C++ AMP to OpenCL / HSA

Wen-Heng Jack Chung <jack@multicorewareinc.com>

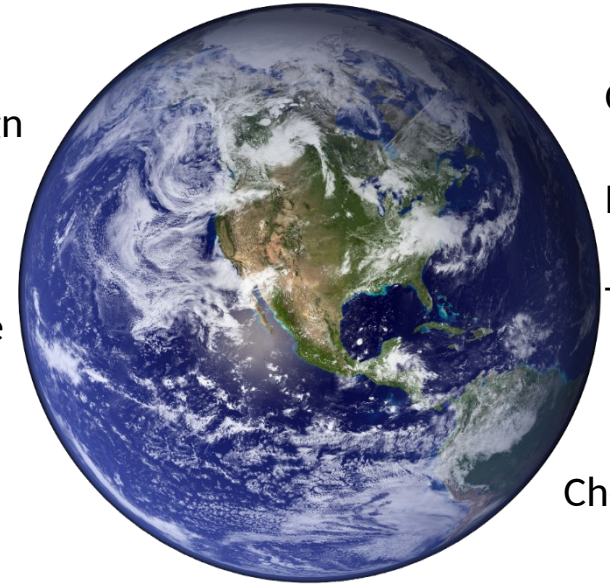
MulticoreWare

- Founded in 2009
- Largest Independent OpenCL Team



Dr. Wen Mei-Hwu, MCW CTO and PI for the UIUC Blue Waters Supercomputer accepts the Second Annual Achievement Award at GTC 2013

Locations



Agenda

- C++ AMP
- Kalmar : C++ AMP Implementation on OpenCL
- Case Studies
- Kalmar on HSA
- Toward Heterogeneous C++

C++ AMP Programming Model

```
void MultiplyWithAMP(int* aMatrix, int* bMatrix, int *productMatrix) {
   array_view<int, 2> a(3, 2, aMatrix);
   array_view<int, 2> b(2, 3, bMatrix);
   array_view<int, 2> product(3, 3, productMatrix);
   parallel_for_each(
      product.extent,
      [=](index<2> idx) restrict(amp) {
         int row = idx[0];
         int col = idx[1];
         for (int inner = 0; inner < a.get_extent()[1]; ++inner)
            product[idx] += a(row, inner) * b(inner, col);
      }
   );
   product.synchronize();
}
```

C++ AMP Programming Model

```
void MultiplyWithAMP(int* aMatrix, int* bMatrix, int *productMatrix) {
   array_view<int, 2> a(3, 2, aMatrix);
   array_view<int, 2> b(2, 3, bMatrix);
   array_view<int, 2> product(3, 3, productMatrix);
   parallel_for_each(
      product.extent,
      [=](index<2> idx) restrict(amp) {
         int row = idx[0];
         int col = idx[1];
         for (int inner = 0; inner < a.get_extent()[1]; ++inner)
            product[idx] += a(row, inner) * b(inner, col);
      }
   );
   product.synchronize();
}
```

GPU data modeled as data container objects

C++ AMP Programming Model

```
void MultiplyWithAMP(int* aMatrix, int* bMatrix, int *productMatrix) {
   array_view<int, 2> a(3, 2, aMatrix);
   array_view<int, 2> b(2, 3, bMatrix);
   array_view<int, 2> product(3, 3, productMatrix);
   parallel_for_each(
      product.extent,
      [=](index<2> idx) restrict(amp) {
         int row = idx[0];
         int col = idx[1];
         for (int inner = 0; inner < a.get_extent()[1]; ++inner)
            product[idx] += a(row, inner) * b(inner, col);
      }
   );
   product.synchronize();
}
```

Execution interface; marking an implicitly parallel region for GPU execution

C++ AMP Programming Model

```
void MultiplyWithAMP(int* aMatrix, int* bMatrix, int *productMatrix) {
  array_view<int, 2> a(3, 2, aMatrix);
  array_view<int, 2> b(2, 3, bMatrix);
  array_view<int, 2> product(3, 3, productMatrix);
  parallel_for_each(
    product.extent,
    [=](index<2> idx) restrict(amp) {
      int row = idx[0];
      int col = idx[1];
      for (int inner = 0; inner < a.get_extent()[1]; ++inner)
        product[idx] += a(row, inner) * b(inner, col);
    }
  );
  product.synchronize();
}
```

Kernels modeled as lambdas; arguments are implicitly modeled as captured variables

C++ AMP Programming Model

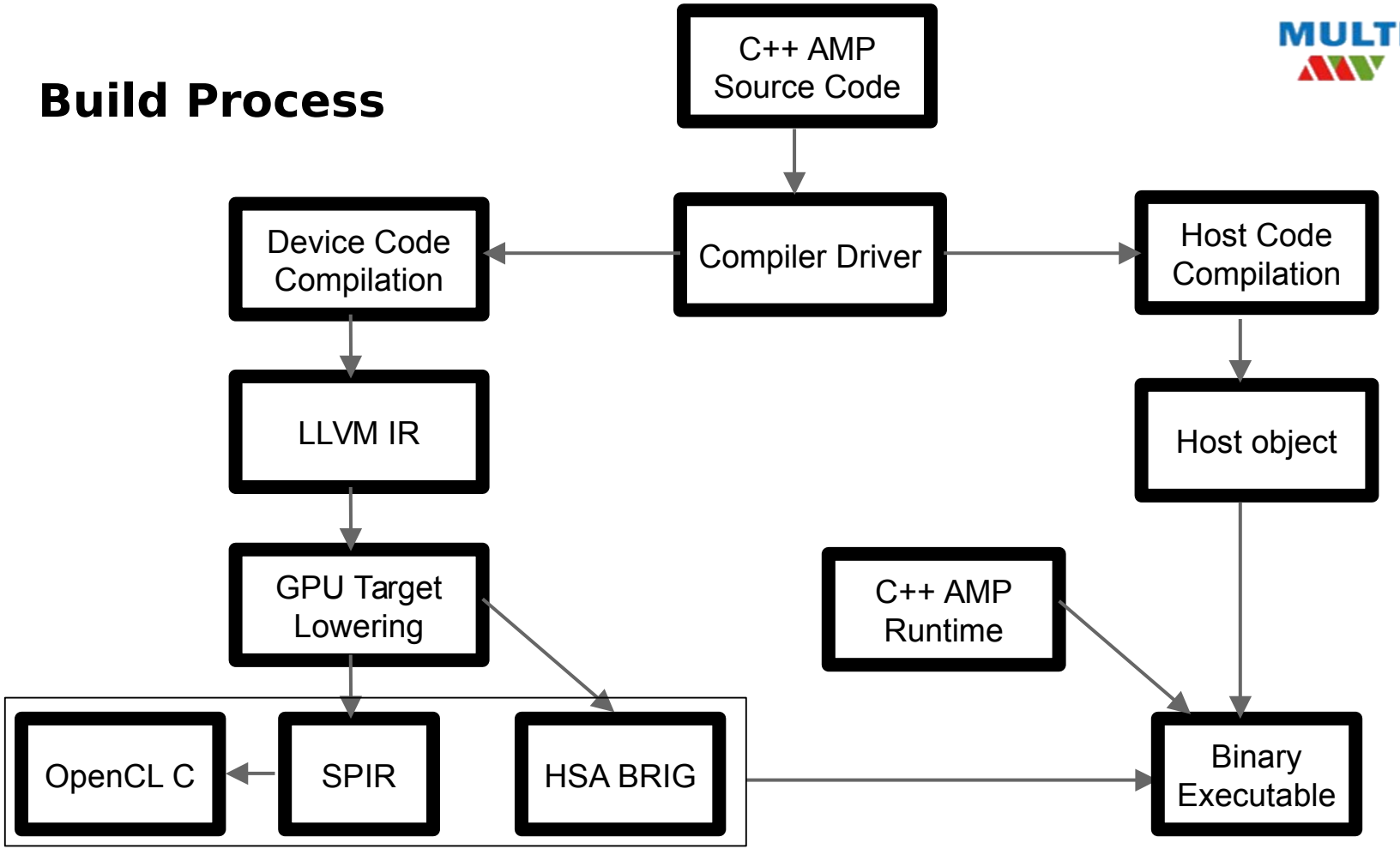
```
void MultiplyWithAMP(int* aMatrix, int* bMatrix, int *productMatrix) {
   array_view<int, 2> a(3, 2, aMatrix);
   array_view<int, 2> b(2, 3, bMatrix);
   array_view<int, 2> product(3, 3, productMatrix);
   parallel_for_each(
      product.extent,
      [=](index<2> idx) restrict(amp) {
         int row = idx[0];
         int col = idx[1];
         for (int inner = 0; inner < a.get_extent()[1]; ++inner)
            product[idx] += a(row, inner) * b(inner, col);
      }
   );
   product.synchronize();
}
```

Synchronize GPU data back to host

Kalmar

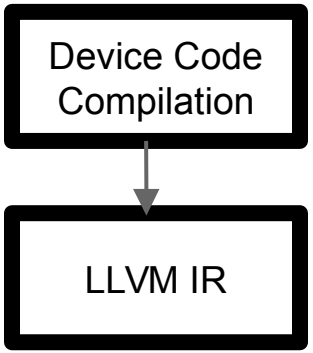
- An Open Source implementation contributed by MulticoreWare
- Based on Clang & LLVM 3.3 and 3.5
- Lower C++ AMP codes to:
  - OpenCL SPIR
  - OpenCL C
  - HSA HSAIL / BRIG
  - x86-64 (CPU fallback)
- Compatible with all major OpenCL stacks, and HSA
- https://bitbucket.org/multicoreware/cppamp-driver-ng/

Build Process



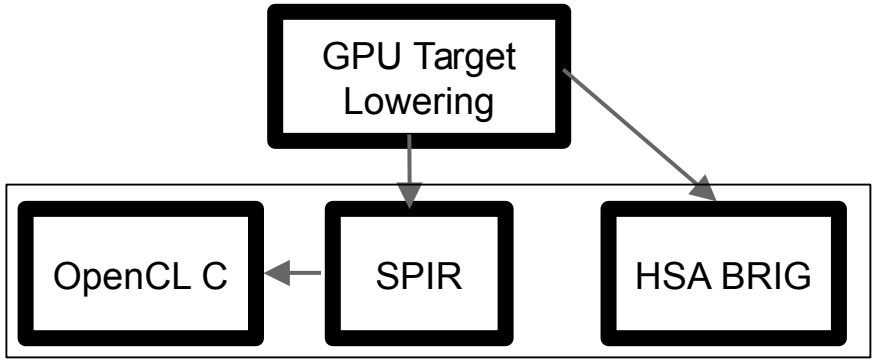
Tasks for Device Code Compilation

- Enforce additional rules checks for device codes
  - Restriction specifier
  - C++ data types unsupported on GPU
  - C++ syntax unsupported on GPU
- Turn captured variables into OpenCL arguments
  - Members of captured objects will be flattened into a list of primitive arguments
- Construct kernel function prototype
  - Kernel names would be mangled per Itanium ABI
- Deserialization: emit device codes to reconstruct OpenCL arguments into captured objects on device
- Populate work-item index into kernel



Tasks for GPU Target Lowering

- Deduce memory address spaces for kernel arguments
- Link with platform-specific built-in functions
  - Atomic functions
  - Floating point functions
  - Work-item indexing
- Inlining and optimization
- Produce SPIR binary
- Produce OpenCL C
  - Modify C Backend from LLVM to emit OpenCL C from SPIR
- Produce HSAIL BRIG
  - Depend on HSAIL Compiler from HSA Foundation



Conceptual Compilation of Device Code

C++ AMP

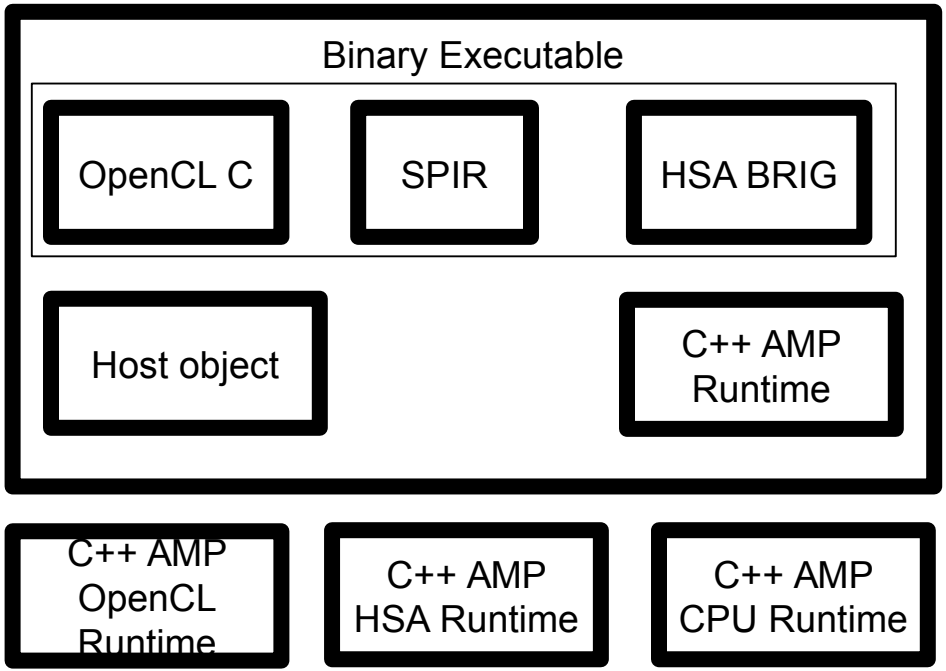
```
[=](index<2> idx) restrict(amp) {
  int row = idx[0];
  int col = idx[1];
  for (int inner = 0; inner < a.get_extent()[1]; ++inner)
    product[idx] += a(row, inner) * b(inner, col);
}
```

OpenCL C

```
__kernel void _Z6kernelPiiS_iiS_ii(__global int* vP, int rowP, int colP,
                                   __global int* vA, int rowA, int colA,
                                   __global int* vB, int rowB, int colB){
  int row = get_global_id(0);
  int col = get_global_id(1);
  for (int inner = 0; inner < colA; ++inner)
    vP[row * colP + col] += vA[row * colA + inner] * vB[inner * colB + col];
}
```

Tasks for C++ AMP Runtime

- Lazily determine which platform-dependent runtime is available
- Dynamically load and delegate runtime calls to platform-dependent C++ AMP runtimes



Case Study: SGEMM in BLAS

					clBLAS	C++ AMP	
					OpenCL 1.2	SPIR 1.2	
M	N	K	TransA	TransB	Avg Time(ms)	Avg Time(ms)	Factor Improvement
3025	48	1	1	0	0.255	0.062	412.97%
729	128	1	1	0	0.197	0.034	575.80%
169	1921	1	1	0	0.391	0.138	283.89%
363	48	3025	1	0	1.162	0.956	121.63%
1728	128	169	1	0	0.856	0.262	326.05%
1152	192	169	1	0	0.884	0.259	341.85%
4096	1000	8	0	1	0.816	0.674	121.01%
3025	363	48	0	1	0.657	0.360	182.36%
169	1152	192	0	1	0.597	0.291	204.94%
169	1728	128	0	1	0.439	0.292	150.22%
4096	8	4096	0	0	2.301	1.808	127.27%
1000	100	1	0	0	0.031	0.037	84.67%
3025	48	363	0	0	1.106	0.424	260.92%
169	192	1152	0	0	2.334	0.455	512.66%
169	128	1728	0	0	3.335	0.615	542.68%
169	128	1	0	0	0.060	0.020	300.10%
18432	100	1	1	0	1.787	0.241	741.16%
100	18432	1	0	1	0.330	0.316	104.40%

Configuration:

CPU: Intel i5-4440@3.10GHz
GPU: AMD FirePro W8100

Case Study: Neural Network

Configuration A:

CPU: Intel i5-4440@3.10GHz
GPU: Nvidia GeForce GTX 780Ti

Neural Network package: Torch7
GPU backend: cuTorch + cuNN

Training data set: AlexNet

Training Epochs: 100

Time: 1,013 secs

Configuration B:

CPU: Intel i5-4440@3.10GHz
GPU: AMD FirePro W8100

Neural Network package: Torch7
GPU backend: C++ AMP implementation

Training data set: AlexNet

Training Epochs: 100

Time: 700 secs

Matrix Multiplication on HSA

```
void MultiplyWithSVM(int* aMatrix, int* bMatrix, int *productMatrix) {
    [redacted]
    parallel_for_each(
        extent<2>(3, 3),
        [&](index<2> idx) restrict(amp) {
            int row = idx[0];
            int col = idx[1];
            for (int inner = 0; inner < 2; inner++)
                productMatrix[row * 3 + col] += aMatrix[row * 2 + inner] *
                                                bMatrix[inner * 3 + col];
        }
    );
    [redacted]
}
```

array_view instances are no longer needed

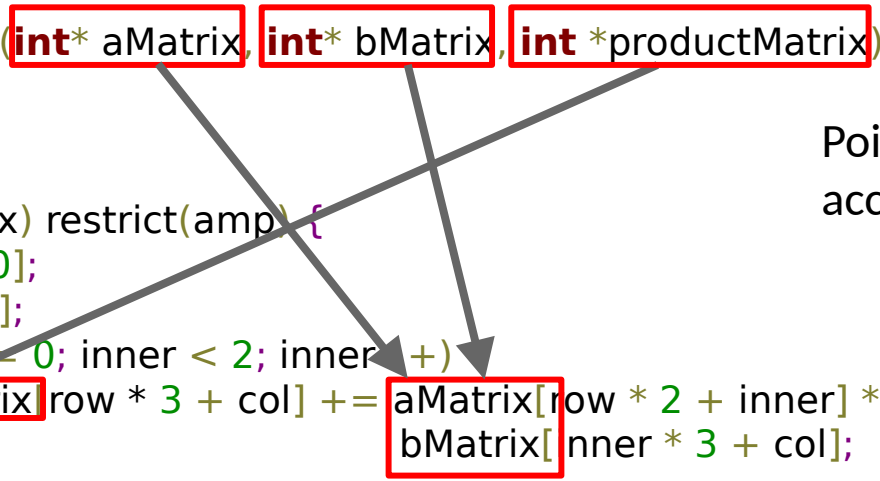
Synchronization back to host is no longer necessary

Matrix Multiplication on HSA

```
void MultiplyWithSVM(int* aMatrix, int* bMatrix, int *productMatrix) {

    parallel_for_each(
        extent<2>(3, 3),
        [&](index<2> idx) restrict(amp) {
            int row = idx[0];
            int col = idx[1];
            for (int inner = 0; inner < 2; inner++)
                productMatrix[row * 3 + col] += aMatrix[row * 2 + inner] *
                                                bMatrix[inner * 3 + col];
        }
    );

}
```



Pointers on host can be accessed in GPU kernels

Platform Atomics on HSA

- atomic objects in <atomic> can be shared between CPU and GPU via SVM
- Lock-free synchronization between CPU and GPU

GPU

```
// send command request to CPU
cmd.store(val, std::memory_order_release);

// while until CPU returns
while (result.load(std::memory_order_acquire));
```

CPU

```
while (true) {
    // fetch command request from GPU
    val = cmd.load(std::memory_order_acquire);

    // do something according to val
    // omitted
    ret = ....;

    // store results to GPU
    result.store(ret, std::memory_order_release);
}
```

Towards Heterogeneous C++

- Kalmar infrastructure allow one compiler frontend to accept various GPU programming paradigms:
  - C++ AMP
  - C++17 proposals for parallelism (ex: N4167, N4409, N4494)
    - Being implemented. Available in Q3 '15.
- Allow more C++ syntax within GPU kernels
- OpenCL 2.0 & SPIR-V support
- Upstream our work


MULTICORE
WARE