Harnessing GPU compute with **C++ Accelerated Massive Parallelism**

Daniel Moth
Principal Program Manager
Developer Division
Microsoft Corporation

关于我

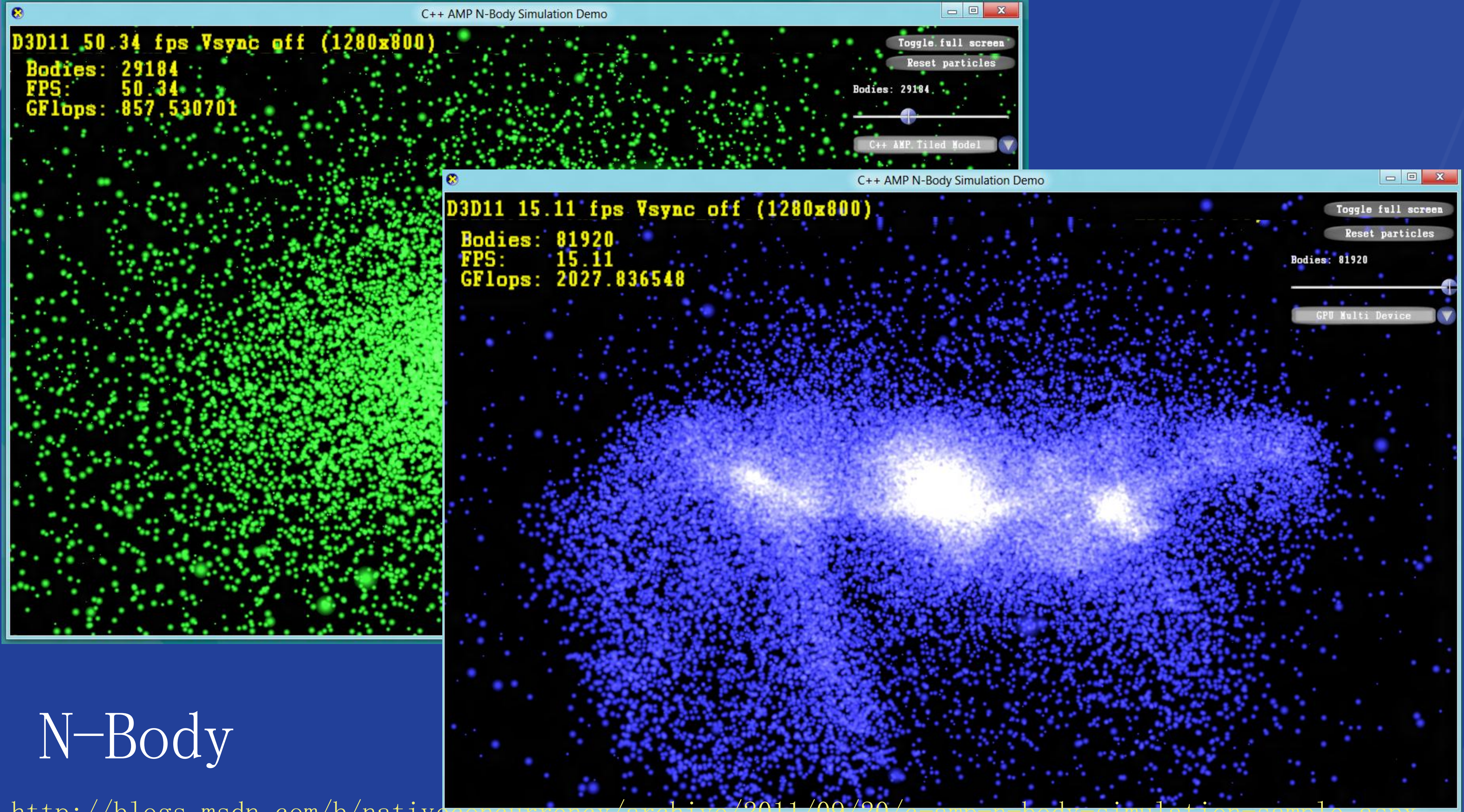
- 施凡
  - @装配脑袋
  - MSRA 微软亚洲研究

- https://github.com/ninputer
  - /AMP-Demo
- http://www.cnblogs.com/ninputer

Agenda

- Context
- Code
- IDE
- Summary

lorem ipsum dolor sit
Consectetuer adipiscing
Sed diam nonummy
Euismod tincidunt laoreet
Dolore magna aliquam
Volutpat. Ut wisi enim ad
Minim veniam nostrud
Exerci tation ullamcorper
Suscipit lobortis nisl ut



C++ AMP N-Body Simulation Demo

```
D3D11 50.34 fps Vsync off (1280x800)
  Bodies: 29184
  FPS:    50.34
  GFlops: 857.530701
```

Bodies: 29184

C++ AMP Tiled Model

C++ AMP N-Body Simulation Demo

```
D3D11 15.11 fps Vsync off (1280x800)
  Bodies: 81920
  FPS:    15.11
  GFlops: 2027.836548
```

Bodies: 81920

GPU Multi Device

N-Body

http://blogs.msdn.com/b/nativeconcurrency/archive/2011/09/20/c-amp-n-body-simulation-sample.aspx

CPUs vs GPUs today

CPU

GPU

- Low memory bandwidth
- Higher power consumption
- Medium level of parallelism
- Shallow execution pipelines
- Random accesses
- Supports general code
- Mainstream programming

- High memory bandwidth
- Lower power consumption
- High level of parallelism
- Deep execution pipelines
- Sequential accesses
- Supports data-parallel code
- Niche programming

images source: AMD

Tomorrow…

- CPUs and GPUs coming closer together…
  - …nothing settled in this space, things still in motion…

- C++ AMP is designed as a mainstream solution not only for today, but also for tomorrow

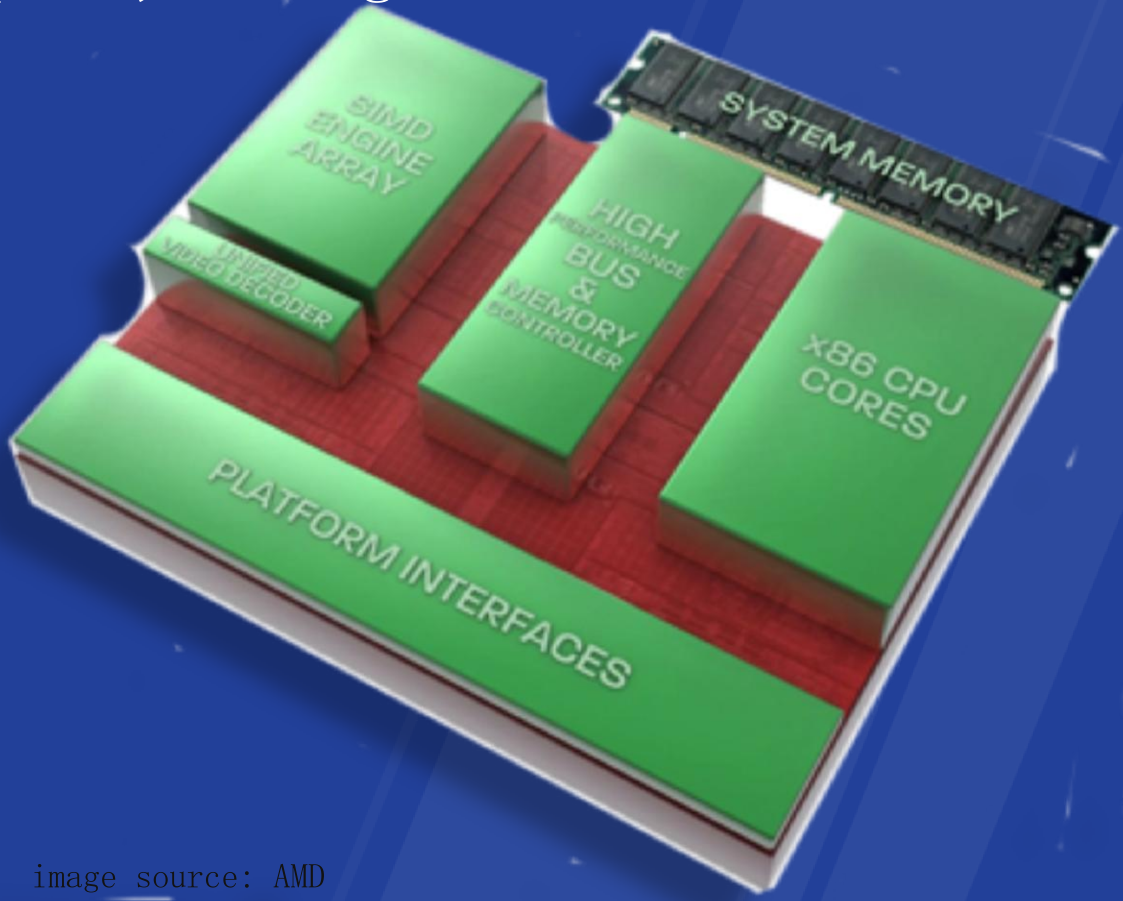


image source: AMD

C++ AMP

- Part of Visual C++
- Visual Studio integration
- STL-like library for multidimensional
- Builds on Direct3D
- An open specification

performance

productivity

portability

http://www.danielmoth.com/Blog/C-Accelerated-Massive-Parallelism.aspx

Agenda checkpoint

- Context ✓
- Code
  - Simple Model
  - Tiled Model
  - (optional) More
- IDE
- Summary

lorem ipsum dolor sit
Consectetuer adipiscing
Sed diam nonummy
Euismod tincidunt laoreet
Dolore magna aliquam
Volutpat. Ut wisi enim ad
Minim veniam nostrud
Exerci tation ullamcorper
Suscipit lobortis nisl ut

Hello World: Array Addition

```
void AddArrays(int n, int * pA, int * pB, int * pSum)
{
   for (int i=0; i<n; i++)
   {
         pSum[i] = pA[i] + pB[i];
   }
}
```

How do we take the serial code on the left that runs on the CPU and convert it to run on an accelerator like the GPU?

Hello World: Array Addition

```
void AddArrays(int n, int * pA, int * pB, int * pSum)
{




    for (int i=0; i<n; i++)


    {
        pSum[i] = pA[i] + pB[i];
    }

}
```

```
#include <amp.h>
using namespace concurrency;

void AddArrays(int n, int * pA, int * pB, int * pSum)
{
    array_view<int,1> a(n, pA);
    array_view<int,1> b(n, pB);
    array_view<int,1> sum(n, pSum);

    parallel_for_each(
        sum.extent,
        [=](index<1> i) restrict(amp)
        {
            sum[i] = a[i] + b[i];
        }
     );
}
```

Basic Elements of C++ AMP coding

parallel_for_each: execute the lambda on the accelerator once per thread

```
void AddArrays(int n, int * pA, int * pB, int * pSum)
{
   array_view<int,1> a(n, pA);
   array_view<int,1> b(n, pB);
   array_view<int,1> sum(n, pSum);

   parallel_for_each(
         sum.extent,
         [=](index<1> i) restrict(amp)
         {
               sum[i] = a[i] + b[i];
         }
   );
```

restrict(amp): tells the compiler to check that this code conforms to C++ AMP language restrictions

array_view: wraps the data to operate on the accelerator

extent: the number and shape of threads to execute the lambda

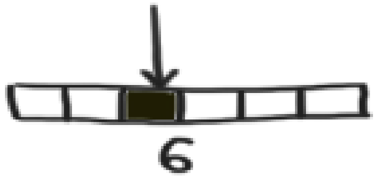
index: the thread ID that is running the lambda, used to index into data

array_view variables captured and associated data copied to accelerator (on demand)

extent<N> and index<N>

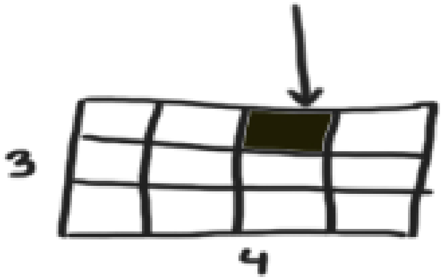
- index<N> - an N-dimensional point
- extent<N> - # of units in each dimension of an N-dim space

index<1> i(2);



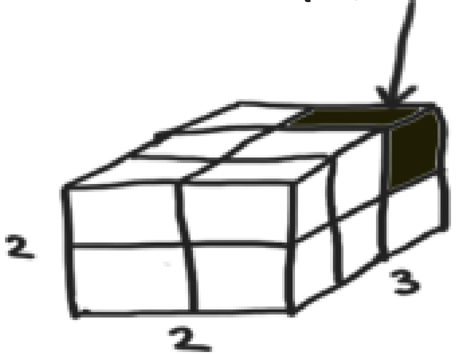
extent<1> e(6);

index<2> i(0,2);



extent<2> e(3,4);

index<3> i(2,0,1);



extent<3> e(3,2,2);

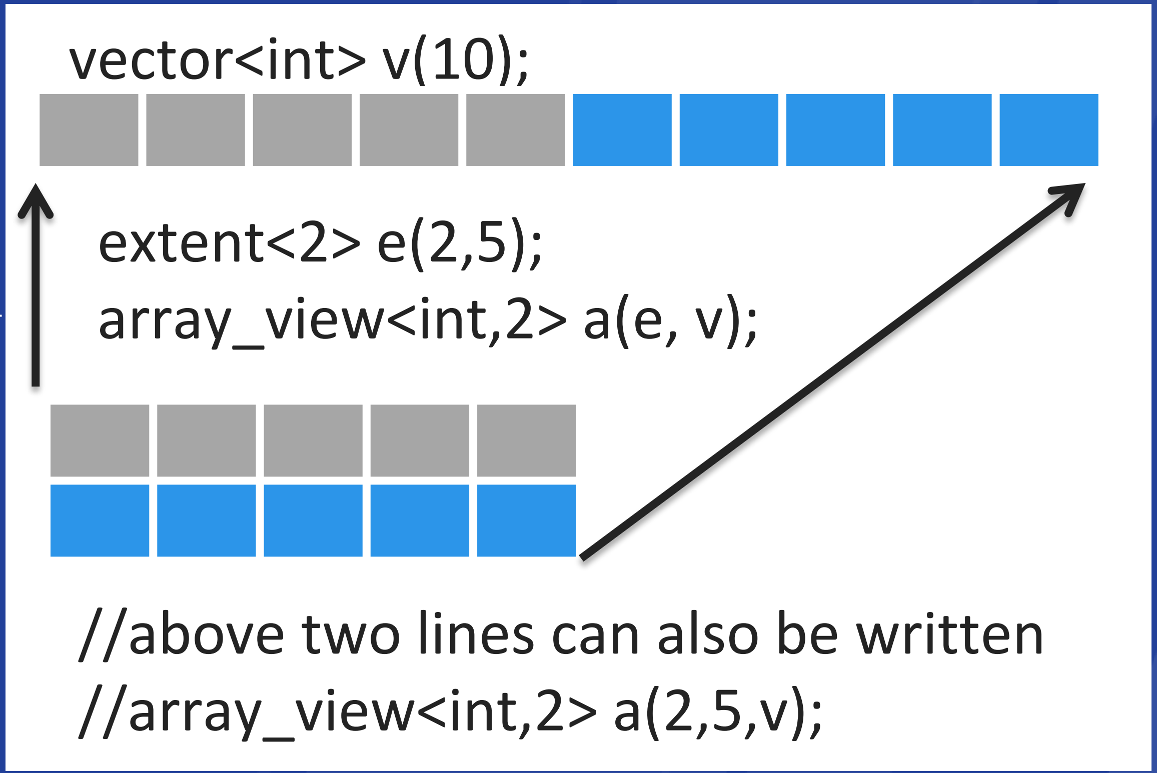
- rank N can be any number <=128

http://www.danielmoth.com/Blog/concurrencyindex-From-Amph.aspx
http://www.danielmoth.com/Blog/concurrencyextent-From-Amph.aspx

array_view<T,N>

- View on existing data on the CPU or GPU
- Dense in least significant dimension
- Of element T and rank N
- Requires extent
- Rectangular
- Access anywhere (implicit sy

```
index<2> i(1,3);

int o = a[i]; // or a[i] = 16;
//or int o = a(1, 3);
```



```
vector<int> v(10);

extent<2> e(2,5);
array_view<int,2> a(e, v);

//above two lines can also be written
//array_view<int,2> a(2,5,v);
```

http://www.danielmoth.com/Blog/array-And-Arrayview-From-Amph.aspx

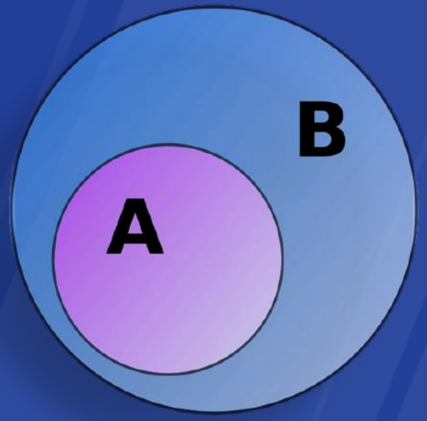
parallel_for_each

- Executes the kernel for each point in the extent
- As-if synchronous in terms of visible side-effects

```
parallel_for_each(
    e,    //e is of type extent<N>
    [ ](index<N> idx) restrict(amp)
    {
        // kernel code
    }
);
```

http://www.danielmoth.com/Blog/parallelforeach-From-Amph-Part-1.aspx

restrict(. . .)



- Applies to functions (including lambdas)

- *restrict(…)* informs the compiler to enforce language restrictions
  - e.g., target-specific restrictions, optimizations, special code-gen

- In 1st release we are only implementing two options
  - *cpu* – the implicit default
  - *amp* – checks that the function conforms to C++ AMP restrictions

http://blogs.msdn.com/b/nativeconcurrency/archive/2011/09/05/restrict-a-key-new-language-feature-introduced-with-c-

restrict(amp) restrictions

- Can only call other *restrict(amp)* functions
- All functions must be inlinable
- Only amp-supported types
  - int, unsigned int, float, double, bool[1]
  - structs & arrays of these types
- Pointers and References
  - Lambdas cannot capture by reference[1], nor capture pointers
  - References and single-indirection pointers supported only as local variables and function arguments

restrict(amp) restrictions

- No
  - recursion
  - ’volatile’
  - virtual functions
  - pointers to functions
  - pointers to member functions
  - pointers in structs
  - pointers to pointers
  - bitfields

- No
  - goto or labeled statements
  - throw, try, catch
  - globals or statics
  - dynamic_cast or typeid
  - asm declarations
  - varargs
  - unsupported types
    - e.g. char, short, long double

http://blogs.msdn.com/b/nativeconcurrency/archive/2011/12/19/restrict-amp-restrictions-part-0-of-n-

Example: restrict overloading

```
double cos( double d );                                        // 1a: cpu code
double cos( double d ) restrict(amp);     // 1b: amp code
double bar( double d ) restrict(cpu,amp);           // 2  : common subset of both

void some_method(array_view<double,2>& c) {
   parallel_for_each( c.extent, [=](index<2> idx) restrict(amp)
   {
     //…
     double d0 = c[idx];
     double d1 = bar(d0);        // ok, bar restrictions include amp
     double d2 = cos(d0);        // ok, chooses amp overload
     //…
   });
}
```

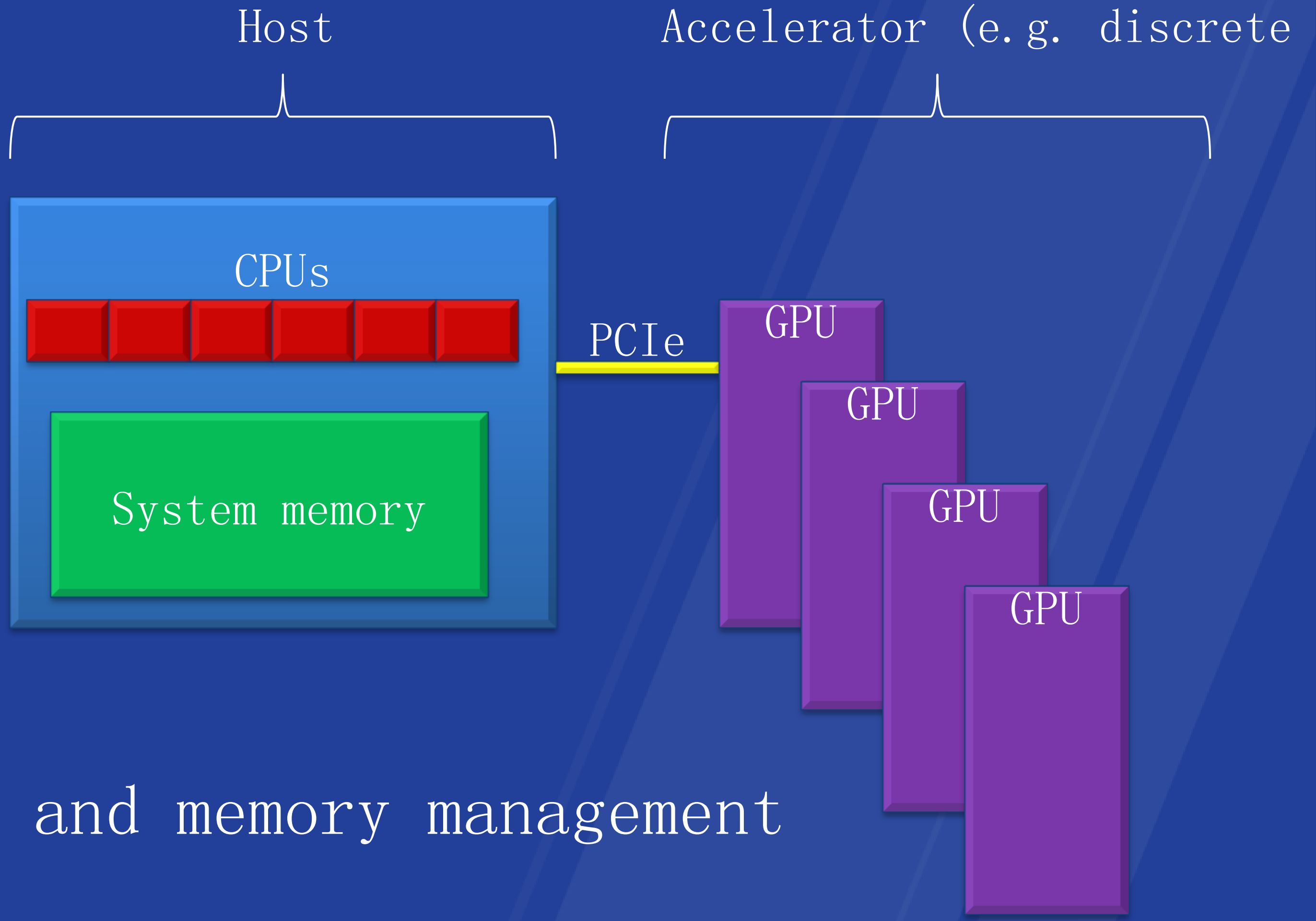
Example: Matrix Multiplication

```
void MatrixMultiplySerial( vector<float>& vC,
    const vector<float>& vA,
    const vector<float>& vB, int M, int N, int W )
{




   for (int row = 0; row < M; row++) {
     for (int col = 0; col < N; col++){
       float sum = 0.0f;
       for(int i = 0; i < W; i++)
         sum += vA[row * W + i] * vB[i * N + col];
       vC[row * N + col] = sum;
     }
   }
}
```

```
void MatrixMultiplyAMP( vector<float>& vC,
    const vector<float>& vA,
    const vector<float>& vB, int M, int N, int W )
{
   array_view<const float,2> a(M,W,vA),b(W,N,vB);
   array_view<float,2> c(M,N,vC);
   c.discard_data();
   parallel_for_each(c.extent,
     [=](index<2> idx) restrict(amp) {
         int row = idx[0]; int col = idx[1];
         float sum = 0.0f;
         for(int i = 0; i < W; i++)
           sum += a(row, i) * b(i, col);
         c[idx] = sum;
     }
   );
}
```

accelerator, accelerator_view

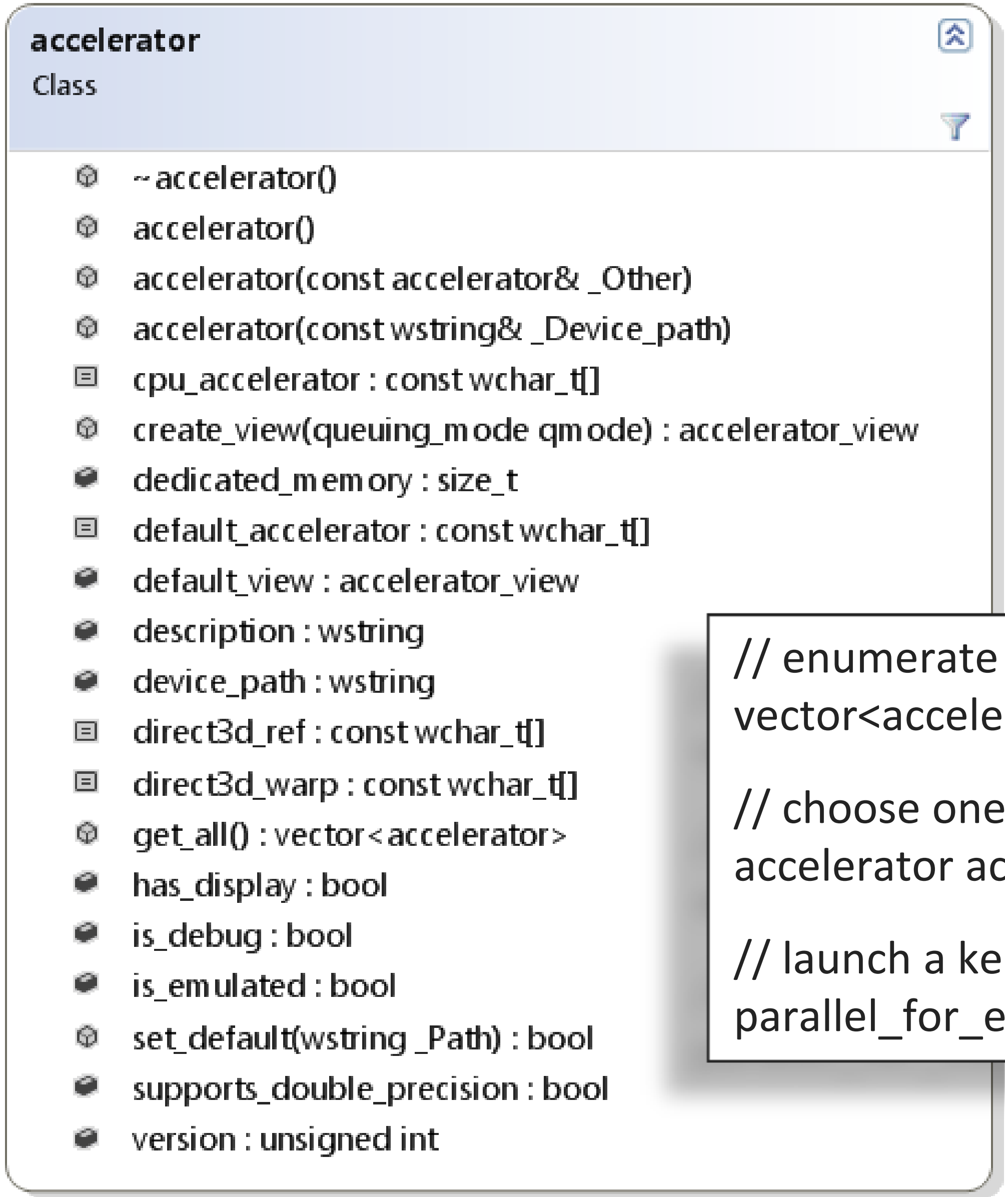
- accelerator
  - e.g. DX11 GPU
  - e.g. WARP, REF
  - e.g. CPU



- accelerator_view
  - a context for scheduling and memory management

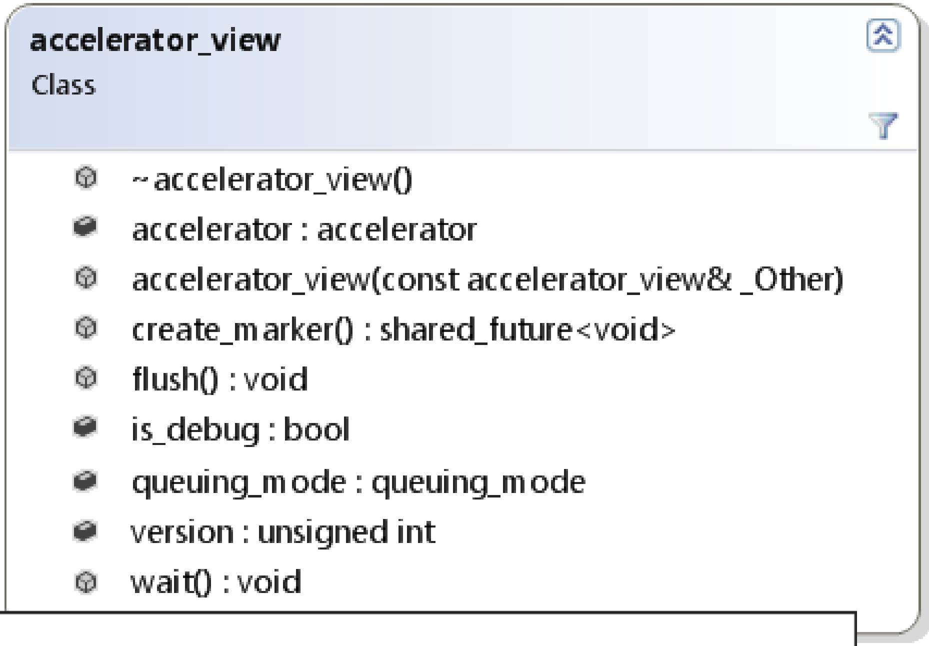
http://www.danielmoth.com/Blog/concurrencyaccelerator.aspx
http://www.danielmoth.com/Blog/concurrencyacceleratorview.aspx

Example: accelerator



accelerator
Class

- ~accelerator()
- accelerator()
- accelerator(const accelerator& _Other)
- accelerator(const wstring& _Device_path)
- cpu_accelerator : const wchar_t[]
- create_view(queuing_mode qmode) : accelerator_view
- dedicated_memory : size_t
- default_accelerator : const wchar_t[]
- default_view : accelerator_view
- description : wstring
- device_path : wstring
- direct3d_ref : const wchar_t[]
- direct3d_warp : const wchar_t[]
- get_all() : vector<accelerator>
- has_display : bool
- is_debug : bool
- is_emulated : bool
- set_default(wstring _Path) : bool
- supports_double_precision : bool
- version : unsigned int



accelerator_view
Class

- ~accelerator_view()
- accelerator : accelerator
- accelerator_view(const accelerator_view& _Other)
- create_marker() : shared_future<void>
- flush() : void
- is_debug : bool
- queuing_mode : queuing_mode
- version : unsigned int
- wait() : void

```
// enumerate all accelerators
vector<accelerator> accs = accelerator::get_all();

// choose one based on your criteria
accelerator acc = accs[0];

// launch a kernel on it
parallel_for_each(acc.default_view, my_extent, [=]…);
```

array<T,N>

- Multi-dimensional array of rank N with element T
- Container whose storage lives on a specific accelerator
- Capture by reference [&] in the lambda
- Explicit copy
- Nearly identical interface to array_view<T,N>

```
vector<int> v(8 * 12);
extent<2> e(8,12);
accelerator acc = …
array<int,2> a(e, acc.default_view);
copy_async(v.begin(), v.end(), a);
```

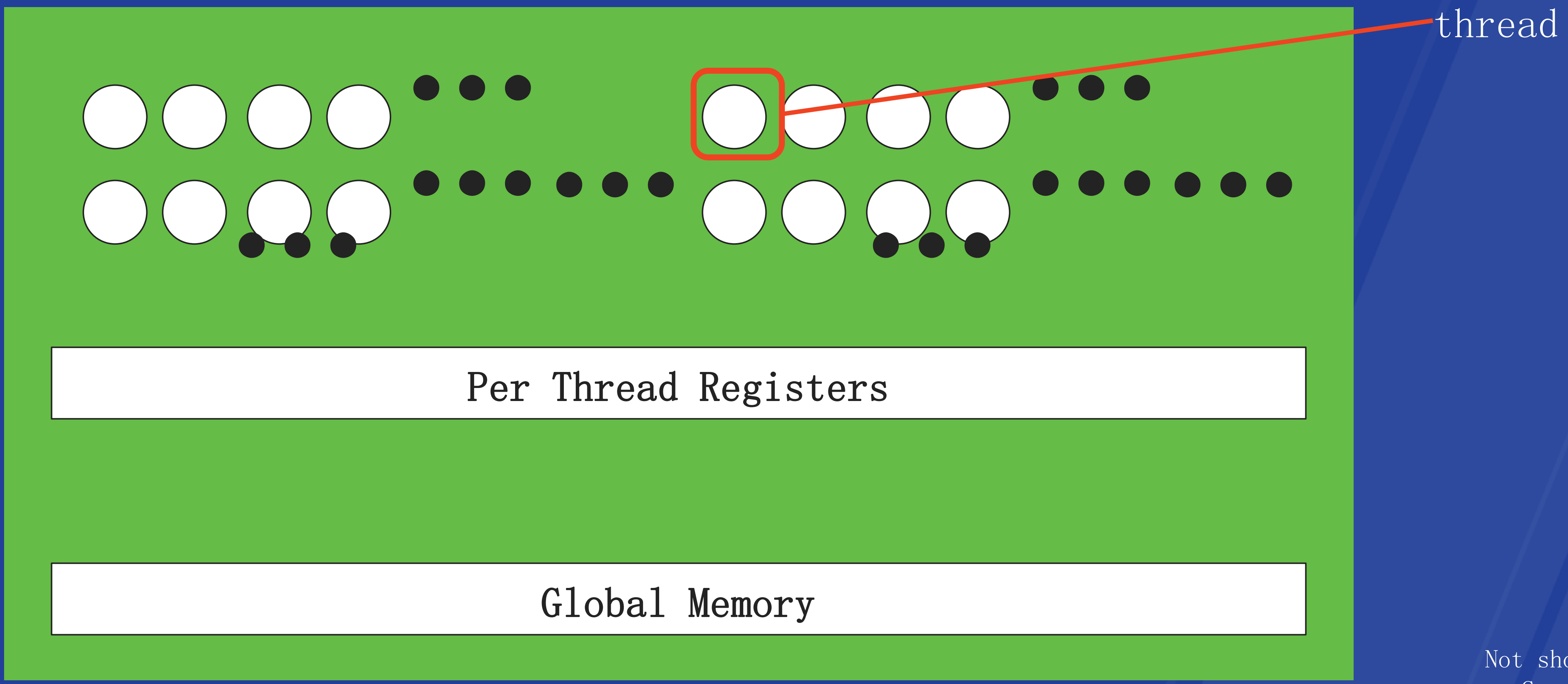
```
parallel_for_each(e, [&](index<2> idx) restrict(amp)
{
    a[idx] += 1;
});
copy(a, v.begin());
```

http://www.danielmoth.com/Blog/array-And-Arrayview-From-Amph.aspx

C++ AMP at a Glance (so far)

- restrict(amp, cpu)
- parallel_for_each
- class accelerator_view
- class accelerator
- class extent<N>
- class index<N>
- class array_view<T,N>
- class array<T,N>

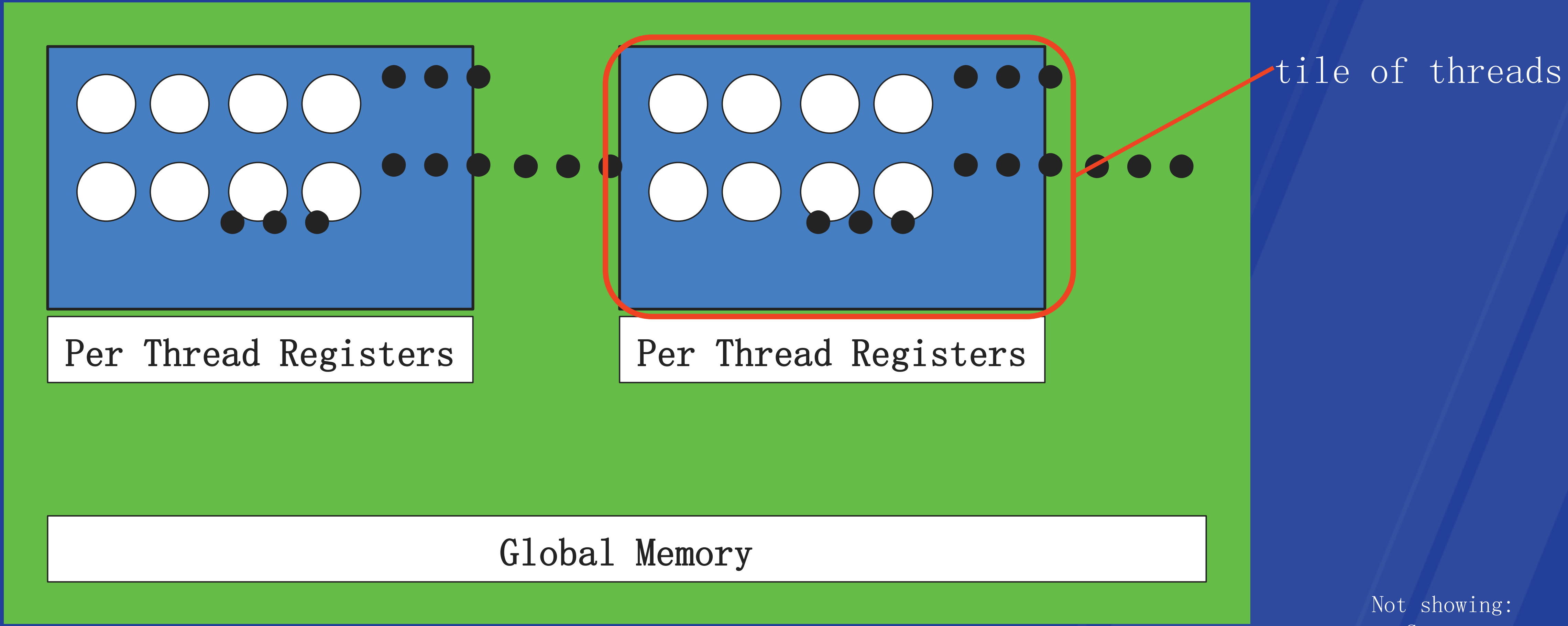
Hardware from a Developer Perspective



Not showing:

- Constant memory
- Memory controllers
- Schedulers
- Other caches
- Multi-GPU case

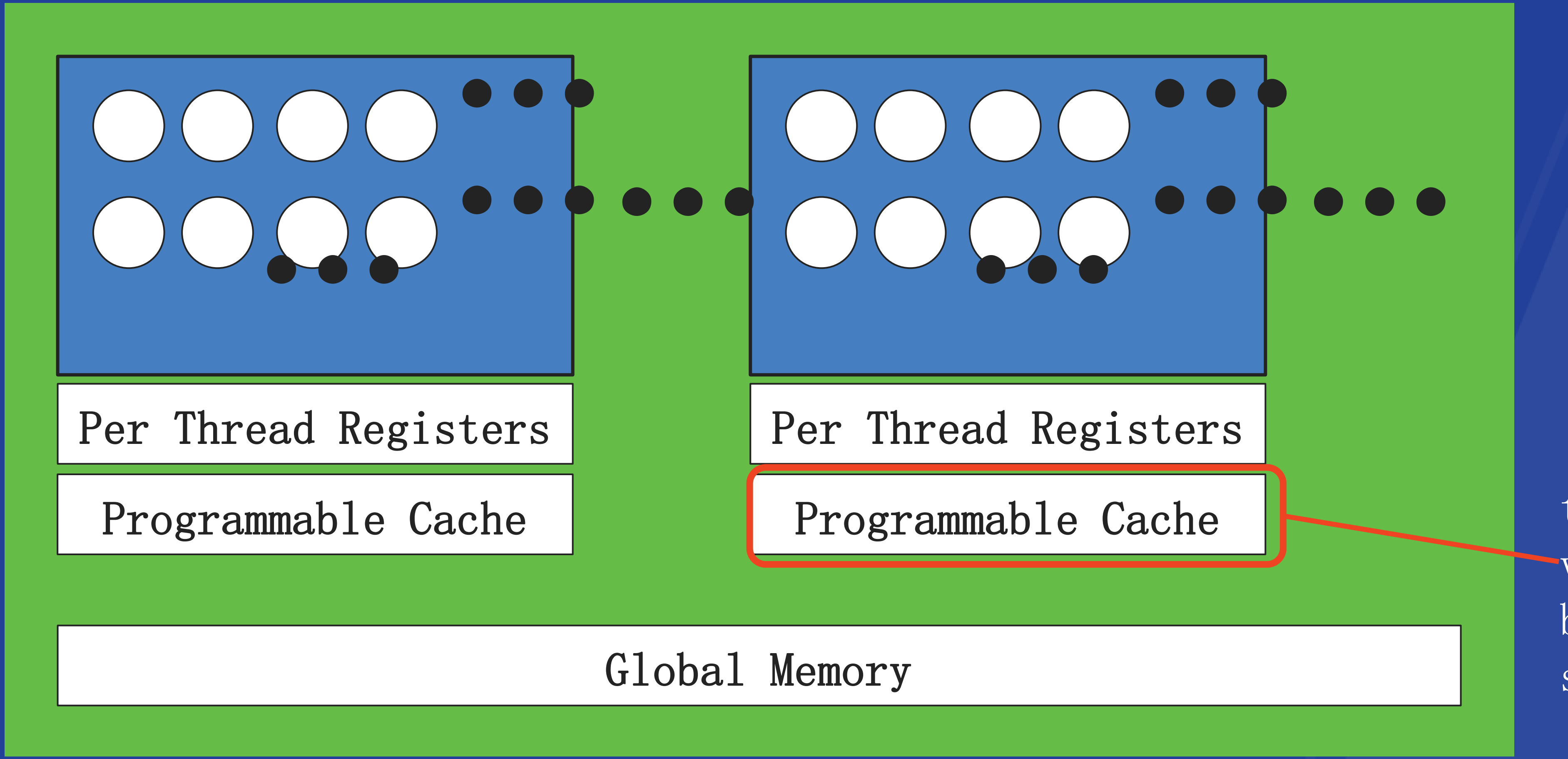
Hardware from a Developer Perspective



Not showing:

- Constant memory
- Memory controllers
- Schedulers
- Other caches
- Multi-GPU case

Hardware from a Developer Perspective



tile_static variables shared by threads in the same tile

Not showing:

- Constant memory
- Memory controllers
- Schedulers
- Other caches
- Multi-GPU case

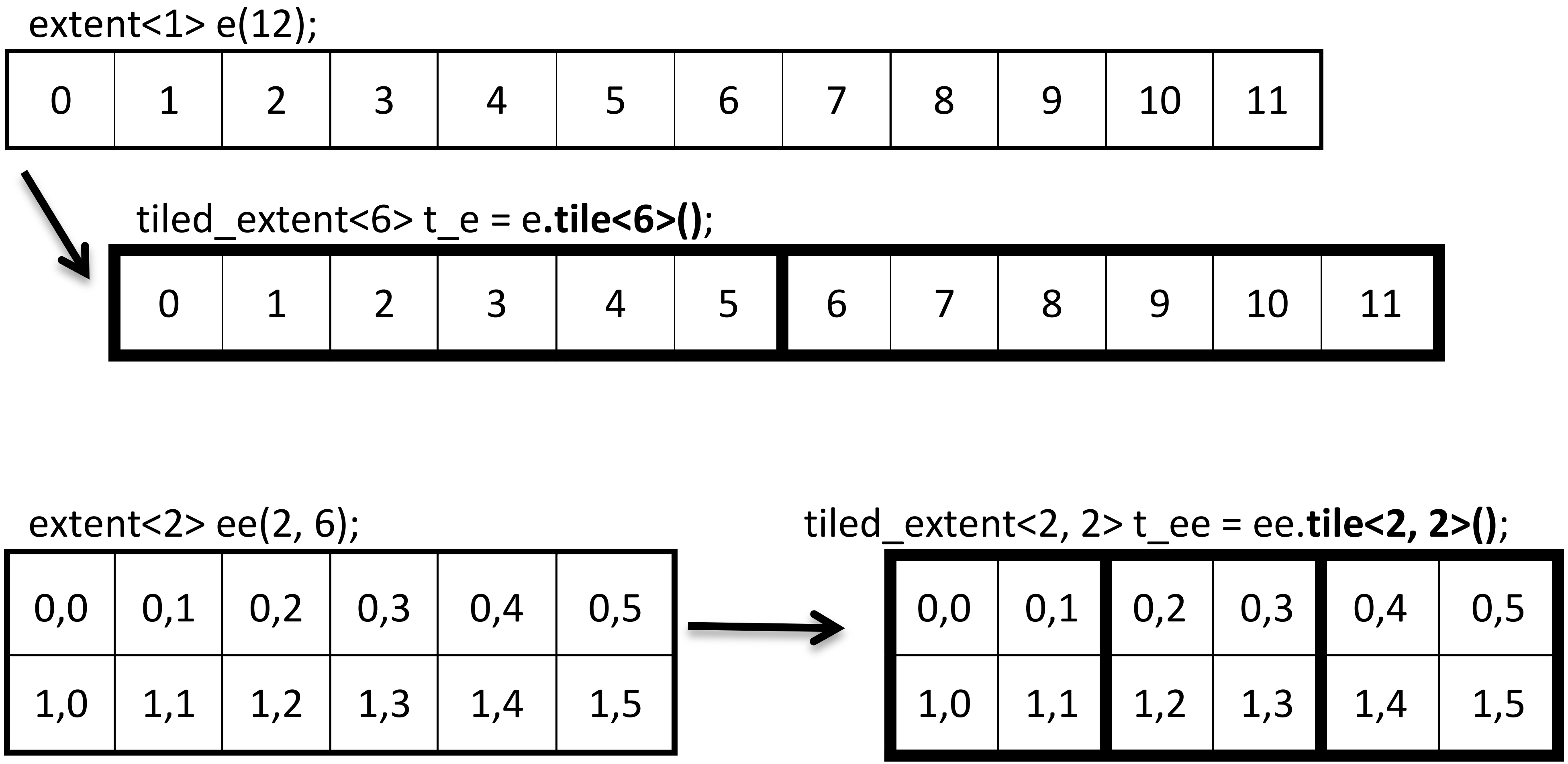
parallel_for_each: tiled overload

- Schedule threads in tiles
  - Gain ability to use tile static

```
array_view<int,1> data(12, my_data);

parallel_for_each(data.extent,
   [=] (index<1> idx) restrict(amp)
   { … });


parallel_for_each(data.extent.tile<6>(),
   [=] (tiled_index<6> t_idx) restrict(amp)
   { … });
```

- parallel_for_each overload for tiles accepts
  - tiled_extent<D0> or tiled_extent<D0, D1> or tiled_extent<D0, D1, D2>
  - a lambda which accepts
    - tiled_index<D0> or tiled_index<D0, D1> or tiled_index<D0, D1, D2>

tiled_extent (from extent)



extent<1> e(12);

0	1	2	3	4	5	6	7	8	9	10	11

tiled_extent<6> t_e = e.**tile<6>()**;

0	1	2	3	4	5	6	7	8	9	10	11

extent<2> ee(2, 6);

0,0	0,1	0,2	0,3	0,4	0,5
1,0	1,1	1,2	1,3	1,4	1,5

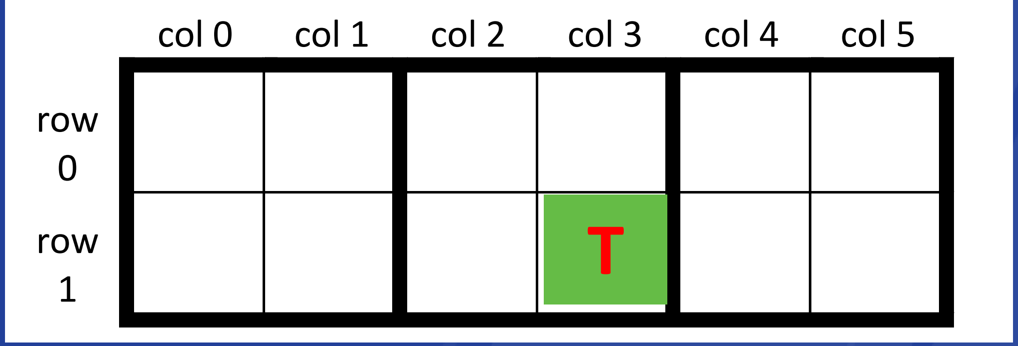
tiled_extent<2, 2> t_ee = ee.**tile<2, 2>()**;

0,0	0,1	0,2	0,3	0,4	0,5
1,0	1,1	1,2	1,3	1,4	1,5

tiled_index

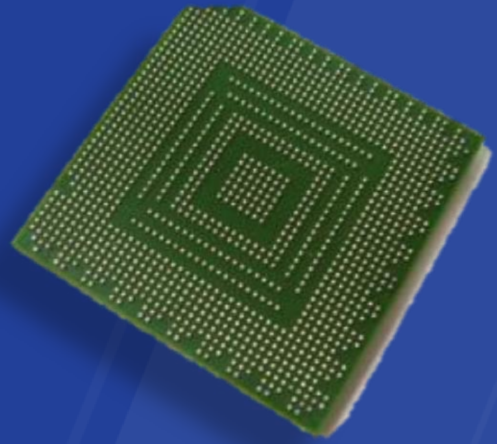
- Given

```
array_view<int,2> data(2, 6, p_my_data);
parallel_for_each(
    data.extent.tile<2,2>(),
    [=] (tiled_index<2,2> t_idx)… { … });
```



- When the lambda is executed T
  - t_idx.global // index<2> (1,3)
  - t_idx.local // index<2> (1,1)
  - t_idx.tile // index<2> (0,1)
  - t_idx.tile_origin // index<2> (0,2)

tile_static



- The tile_static storage class
  - Second addition to the C++ language
  - Reflects hardware memory hierarchy

- Within the tiled parallel_for_each lambda we can use
  - tile_static for local variables
    - indicates that the variable is allocated in fast cache memory
      - i.e. shared by each thread in a tile of threads
    - only applicable in restrict(amp) functions

http://blogs.msdn.com/b/nativeconcurrency/archive/2012/01/11/restrict-amp-restrictions-part-10-of-n-tile-static.aspx

tile_static storage class

0,0	0,1	0,2	0,3	0,4	0,5
1,0	1,1	1,2	1,3	1,4	1,5

```
static const int TS = 2;
array_view<int, 2> av(2, 6, my_vector);
parallel_for_each(av.extent.tile<TS,TS>(),
[=](tiled_index<TS,TS> t_idx) restrict(amp)
{



                    imagine the code here



});
int sum = av(0,0) + av(0,2) + av(0,4); //the three tile_origins
```

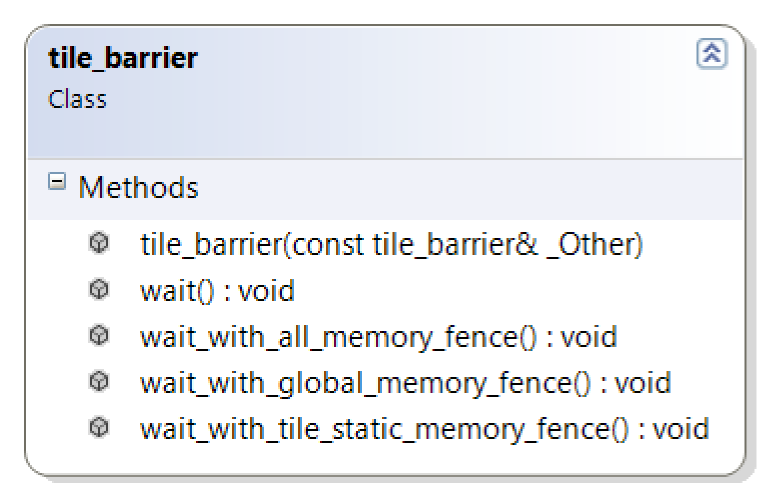
tile_static storage class

0,0	0,1	0,2	0,3	0,4	0,5
1,0	1,1	1,2	1,3	1,4	1,5

```
static const int TS = 2;
array_view<int, 2> av(2, 6, my_vector);
parallel_for_each(av.extent.tile<TS,TS>(),
[=](tiled_index<TS,TS> t_idx) restrict(amp)
{
    tile_static int t[TS][TS];
    t[t_idx.local[0]][t_idx.local[1]] = av[t_idx.global];

    if (t_idx.local == index<2>(0,0)) {
       int temp = t[0][0] + t[0][1] + t[1][0] + t[1][1];
       av[t_idx.tile_origin] = temp;
    }
});
int sum = av(0,0) + av(0,2) + av(0,4); //the three tile_origins
```

tile_barrier

- class tile_barrier
  - synchronize all threads within a tile
  - e.g. t_idx.barrier.wait();



tile_barrier
Class

⊟ Methods

- tile_barrier(const tile_barrier& _Other)
- wait() : void
- wait_with_all_memory_fence() : void
- wait_with_global_memory_fence() : void
- wait_with_tile_static_memory_fence() : void

- Plus
  - Fences (without barriers)
    - all_memory_fence, global_memory_fence, tile_static_memory_fence
  - Atomics
    - atomic_exchange, atomic_compare_exchange, atomic_fetch_*

http://blogs.msdn.com/b/nativeconcurrency/archive/2011/12/24/tile-barrier-in-c-amp.aspx
http://blogs.msdn.com/b/nativeconcurrency/archive/2012/01/04/c-amp-s-atomic-operations.aspx

tile_barrier class

0,0	0,1	0,2	0,3	0,4	0,5
1,0	1,1	1,2	1,3	1,4	1,5

```
static const int TS = 2;
array_view<int, 2> av(2, 6, my_vector);
parallel_for_each(av.extent.tile<TS,TS>(),
[=](tiled_index<TS,TS> t_idx) restrict(amp)
{
    tile_static int t[TS][TS];
    t[t_idx.local[0]][t_idx.local[1]] = av[t_idx.global];
    t_idx.barrier.wait();
    if (t_idx.local == index<2>(0,0)) {
       int temp = t[0][0] + t[0][1] + t[1][0] + t[1][1];
       av[t_idx.tile_origin] = temp;
    }
});
int sum = av(0,0) + av(0,2) + av(0,4); //the three tile_origins
```

Example: Matrix Multiplication (tiled) -

```
void MatrixMultSimple(vector<float>& vC, const
vector<float>& vA, const vector<float>& vB, int M, int N,
int W )
{

   array_view<const float,2> a(M, W, vA), b(W, N, vB);
   array_view<float,2> c(M,N,vC);  c.discard_data();
   parallel_for_each(c.extent,
      [=] (index<2> idx) restrict(amp)
      {
         int row = idx[0];
         int col = idx[1];

         float sum = 0.0f;
         for(int k = 0; k < W; k++)
            sum += a(row, k) * b(k, col);

         c[idx] = sum;
      } );
}
```

```
void MatrixMultTiled(vector<float>& vC, const
vector<float>& vA, const vector<float>& vB, int M, int
N, int W )
{
   static const int TS = 16;
   array_view<const float,2> a(M, W, vA), b(W, N, vB);
   array_view<float,2> c(M,N,vC);  c.discard_data();
   parallel_for_each(c.extent.tile< TS, TS >(),
      [=] (tiled_index< TS, TS> t_idx) restrict(amp)
      {
         int row = t_idx.global[0];
         int col =   t_idx.global[1];

         float sum = 0.0f;
         for(int k = 0; k < W; k++)
            sum += a(row, k) * b(k, col);

         c[t_idx.global] = sum;
      } );
}
```

Example: Matrix Multiplication (tiled) -

```
void MatrixMultSimple(vector<float>& vC, const vector<float>& vA,
const vector<float>& vB, int M, int N, int W )
{
   static const int TS = 16;
   array_view<const float,2> a(M, W, vA), b(W, N, vB);
   array_view<float,2> c(M,N,vC);  c.discard_data();
   parallel_for_each(c.extent.tile< TS, TS >(),
      [=] (tiled_index< TS, TS> t_idx) restrict(amp)  {
         int row = t_idx.global[0]; int col = t_idx.global[1];

         float sum = 0.0f;





         for(int k = 0; k < W; k++)
            sum += a(row, k) * b(k, col);


         c[t_idx.global] = sum;
      } );
}
```

```
void MatrixMultTiled(vector<float>& vC, const vector<float>& vA,
const vector<float>& vB, int M, int N, int W )
{
   static const int TS = 16;
   array_view<const float,2> a(M, W, vA), b(W, N, vB);
   array_view<float,2> c(M,N,vC);  c.discard_data();
   parallel_for_each(c.extent.tile< TS, TS >(),
      [=] (tiled  index< TS, TS> t  idx) restrict(amp)  {






               imagine the code here







         c[t_idx.global] = sum;
      } );
}
```

Example: Matrix Multiplication (tiled) -

```
void MatrixMultSimple(vector<float>& vC, const vector<float>& vA,
const vector<float>& vB, int M, int N, int W )
{
   static const int TS = 16;
   array_view<const float,2> a(M, W, vA), b(W, N, vB);
   array_view<float,2> c(M,N,vC);  c.discard_data();
   parallel_for_each(c.extent.tile< TS, TS >(),
      [=] (tiled_index< TS, TS> t_idx) restrict(amp)  {
         int row = t_idx.global[0]; int col = t_idx.global[1];

         float sum = 0.0f;





         for(int k = 0; k < W; k++)
            sum += a(row, k) * b(k, col);


         c[t_idx.global] = sum;
      } );
}
```

```
void MatrixMultTiled(vector<float>& vC, const vector<float>& vA,
const vector<float>& vB, int M, int N, int W )
{
   static const int TS = 16;
   array_view<const float,2> a(M, W, vA), b(W, N, vB);
   array_view<float,2> c(M,N,vC);  c.discard_data();
   parallel_for_each(c.extent.tile< TS, TS >(),
      [=] (tiled_index< TS, TS> t_idx) restrict(amp)  {
         int row = t_idx.local[0]; int col = t_idx.local[1];
         tile_static float locA[TS][TS], locB[TS][TS];
         float sum = 0.0f;
         for (int i = 0; i < W; i += TS) {
            locA[row][col] = a(t_idx.global[0], col + i);
            locB[row][col] = b(row + i, t_idx.global[1]);
            t_idx.barrier.wait();

            for (int k = 0; k < TS; k++)
               sum += locA[row][k] * locB[k][col];
            t_idx.barrier.wait();
          }
         c[t_idx.global] = sum;
      } );
}
```

Phase 1

Phase 2

C++ AMP at a Glance

- restrict(amp, cpu)
- parallel_for_each
- class accelerator_view
- class accelerator
- class extent<N>
- class index<N>
- class array_view<T,N>
- class array<T,N>

- tile_static storage class
- class tiled_extent< , , >
- class tiled_index< , , >
- class tile_barrier

Error handling

- Some APIs can throw
  - e.g. parallel_for_each

- Exceptions
  - concurrency::runtime_exception
  - concurrency::out_of_memory
  - concurrency::unsupported_feature
  - concurrency::invalid_compute_domain
  - concurrency::accelerator_view_removed

```
/* Trying to use REF emulator on a
machine that does not have it installed,
throws runtime_exception */
 try
 {
   accelerator a(accelerator::direct3d_ref);
 }
 catch(runtime_exception& ex)
{
  std::cout << ex.what() << std::endl;
}
```

http://blogs.msdn.com/b/nativeconcurrency/archive/2012/01/27/c-amp-runtime-exceptions.aspx

<amp_math.h>

- concurrency::fast_math
  - Wrap HLSL intrinsics
  - 35 functions
  - Single-precision only
  - Sacrifice accuracy for sp
- concurrency::precise_mat
  - 68 functions
  - Require full double preci
    - even for single precision

```
#include <amp.h>
#include <amp_math.h>
using namespace concurrency;
using namespace concurrency::fast_math;
// using namespace concurrency::precise_math;
int main() {
    float a = 2.2f, b = 3.5f;
    float result = pow(a,b);
    std::vector<float> v(1);
    array_view<float> av(1,v);
    parallel_for_each(av.extent, [=](index<1> idx)
restrict(amp)
    {
        av[idx] = pow(a,b);
    });
}
```

http://blogs.msdn.com/b/nativeconcurrency/archive/2012/02/08/math-library-for-c-amp.aspx

<amp_graphics.h>, concurrency::graphics

- norm/unorm scalar type
- Short vector types (int_3, float_4, norm_2, etc.)
  - Swizzle expressions: myvec.yzx = int_3(1, 2, 3);

- Textures - efficient access to 1d, 2d
  - Element type is scalar or SVT of rank 1,
  - DX limitations apply
  - Different encodings supported
  - Interop with DX texture resources

http://blogs.msdn.com/b/nativeconcurrency/archive/2012/01/25/concurrency-graphics-in-c-amp.aspx

DirectX Integration, concurrency::direct3d

- Weave C++ AMP code and data with DX-based applications

C++ AMP type	DirectX type	C++ AMP interop API
array	ID3D11Buffer	*get_buffer, make_array
texture	ID3D11Texture1D/2D/3D	*get_texture, make_texture
accelerator_view	ID3D11Device	*get_device, create_accelerator_view

- Example: **n-body simulation**
  - Populates an array of particular positions using parallel_for_each
  - Then read it in a vertex shader

http://blogs.msdn.com/b/nativeconcurrency/archive/2011/12/29/interoperability-between-direct-3d-and-c-amp.aspx

http://blogs.msdn.com/b/nativeconcurrency/archive/2012/02/24/direct3d-namespace-and-hlsl-intrinsics-in-c-amp.aspx

Agenda checkpoint

- Context ✓
- Code ✓
- IDE
- Summary

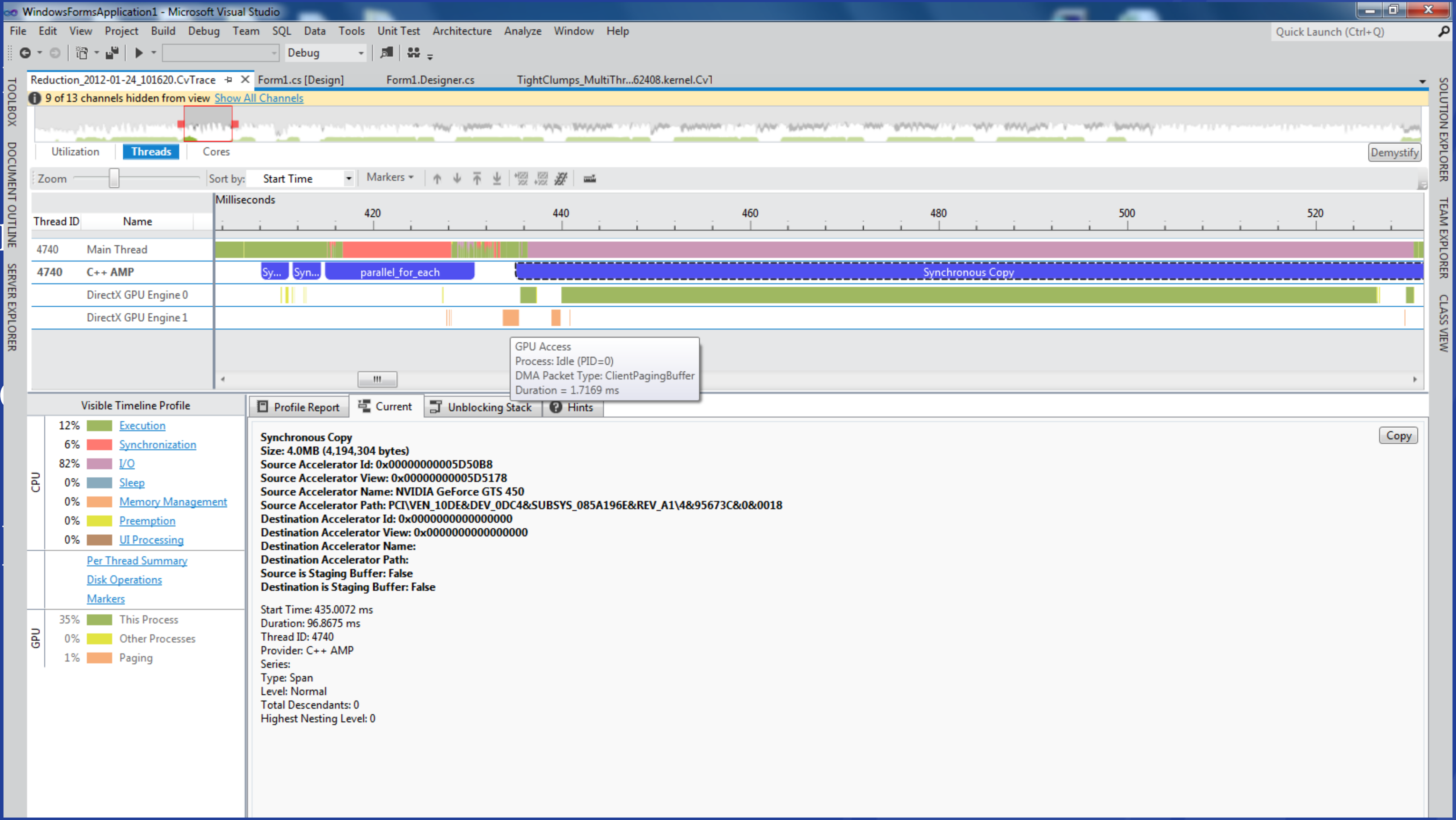
lorem ipsum dolor sit
Consectetuer adipiscing
Sed diam nonummy
Euismod tincidunt laoreet
Dolore magna aliquam
Volutpat. Ut wisi enim ad
Minim veniam nostrud
Exerci tation ullamcorper
Suscipit lobortis nisl ut

Visual Studio 2012

- Organize
- Edit
- Design
- Build
- Browse
- Debug
- Profile

Visual Studio 2012

- Organi
- Edit
- Design
- Build
- Browse
- Debug
- Profil



Thread ID	Name
4740	Main Thread
4740	C++ AMP
	DirectX GPU Engine 0
	DirectX GPU Engine 1

GPU Access
Process: Idle (PID=0)
DMA Packet Type: ClientPagingBuffer
Duration = 1.7169 ms

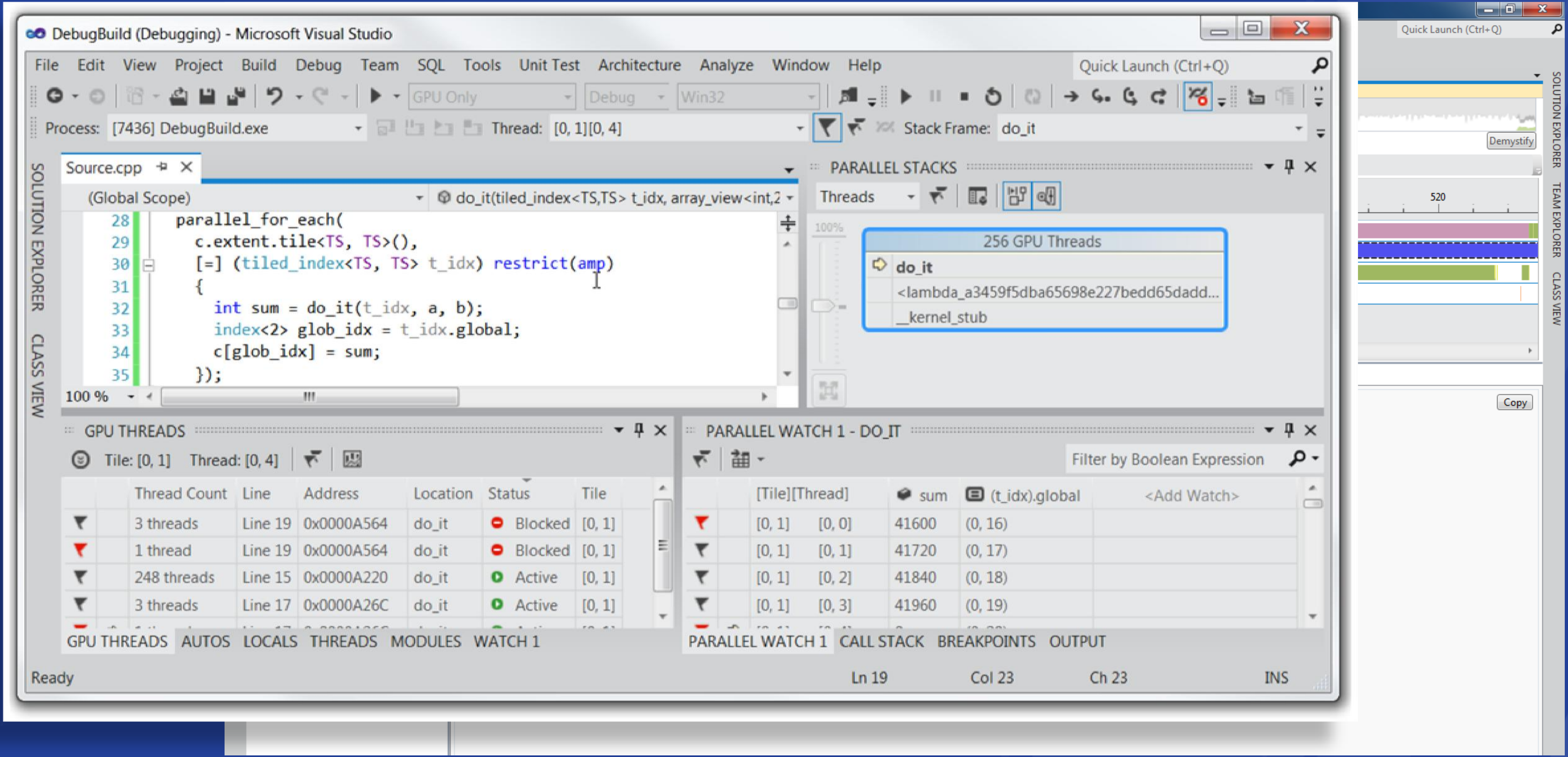
Visible Timeline Profile

CPU	12%	Execution
	6%	Synchronization
	82%	I/O
	0%	Sleep
	0%	Memory Management
	0%	Preemption
	0%	UI Processing
		Per Thread Summary
		Disk Operations
		Markers
GPU	35%	This Process
	0%	Other Processes
	1%	Paging

Synchronous Copy
Size: 4.0MB (4,194,304 bytes)
Source Accelerator Id: 0x00000000005D50B8
Source Accelerator View: 0x00000000005D5178
Source Accelerator Name: NVIDIA GeForce GTS 450
Source Accelerator Path: PCI\VEN_10DE&DEV_0DC4&SUBSYS_085A196E&REV_A1\4&95673C&0&0018
Destination Accelerator Id: 0x0000000000000000
Destination Accelerator View: 0x0000000000000000
Destination Accelerator Name:
Destination Accelerator Path:
Source is Staging Buffer: False
Destination is Staging Buffer: False

Start Time: 435.0072 ms
Duration: 96.8675 ms
Thread ID: 4740
Provider: C++ AMP
Series:
Type: Span
Level: Normal
Total Descendants: 0
Highest Nesting Level: 0

Visual Studio 2012



DebugBuild (Debugging) - Microsoft Visual Studio

Process: [7436] DebugBuild.exe

Thread: [0, 1][0, 4]

Stack Frame: do_it

Source.cpp

(Global Scope)

do_it(tiled_index<TS,TS> t_idx, array_view<int,2

```
parallel_for_each(
  c.extent.tile<TS, TS>(),
  [=] (tiled_index<TS, TS> t_idx) restrict(amp)
  {
    int sum = do_it(t_idx, a, b);
    index<2> glob_idx = t_idx.global;
    c[glob_idx] = sum;
  });
```

100 %

PARALLEL STACKS

Threads

256 GPU Threads

- do_it
- <lambda_a3459f5dba65698e227bedd65dadd...
- __kernel_stub

GPU THREADS

Tile: [0, 1]   Thread: [0, 4]

Thread Count	Line	Address	Location	Status	Tile
3 threads	Line 19	0x0000A564	do_it	Blocked	[0, 1]
1 thread	Line 19	0x0000A564	do_it	Blocked	[0, 1]
248 threads	Line 15	0x0000A220	do_it	Active	[0, 1]
3 threads	Line 17	0x0000A26C	do_it	Active	[0, 1]

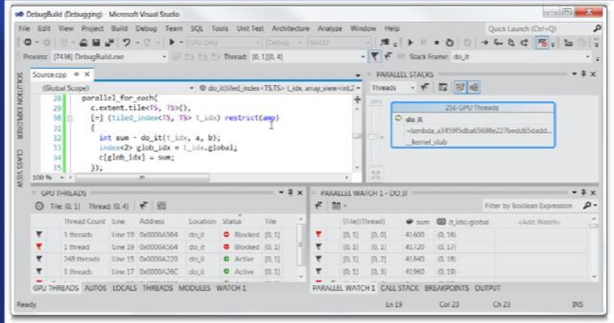
PARALLEL WATCH 1 - DO_IT

Filter by Boolean Expression

[Tile]	[Thread]	sum	(t_idx).global	<Add Watch>
[0, 1]	[0, 0]	41600	(0, 16)	
[0, 1]	[0, 1]	41720	(0, 17)	
[0, 1]	[0, 2]	41840	(0, 18)	
[0, 1]	[0, 3]	41960	(0, 19)	

Ready   Ln 19   Col 23   Ch 23   INS

C++ AMP Parallel Debugger



```
parallel_for_each(
  c.extent.tile<TS, TS>(),
  [=] (tiled_index<TS, TS> t_idx) restrict(amp)
  {
    int sum = do_it(t_idx, a, b);
    index<2> glob_idx = t_idx.global;
    c[glob_idx] = sum;
  });
```

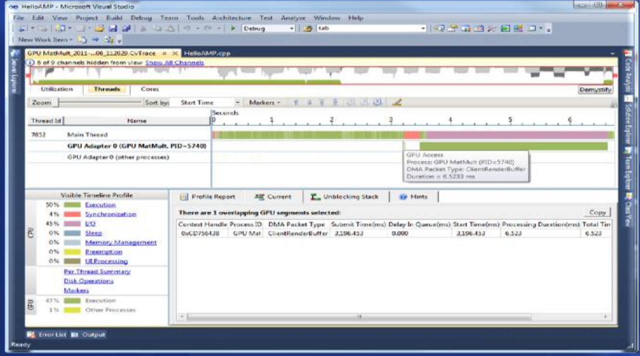
Thread Count	Line	Address	Location	Status	Tile
3 threads	Line 19	0x0000A564	do_it	Blocked	[0, 1]
1 thread	Line 19	0x0000A564	do_it	Blocked	[0, 1]
248 threads	Line 15	0x0000A220	do_it	Active	[0, 1]
3 threads	Line 17	0x0000A26C	do_it	Active	[0, 1]

[Tile]	[Thread]	sum	(t_idx).global	<Add Watch>
[0, 1]	[0, 0]	41600	(0, 16)	
[0, 1]	[0, 1]	41720	(0, 17)	
[0, 1]	[0, 2]	41840	(0, 18)	
[0, 1]	[0, 3]	41960	(0, 19)	

- Well known Visual Studio debugging features
  - Launch (incl. remote), Attach, Break, Stepping, Breakpoints, DataTips
  - Toolwindows
    - Processes, Debug Output, Modules, Disassembly, Call Stack, Memory, Registers, Locals, Watch, Quick Watch
- New features (for both CPU and GPU)
  - Parallel Stacks window, Parallel Watch window, Barrier
- New GPU-specific
  - Emulator, GPU Threads window, race detection
- concurrency::direct3d_printf, _errorf, _abort

http://channel9.msdn.com/Events/BUILD/BUILD2011/TOOL-802T *(51:54-59:16)*

Concurrency Visualizer for GPU


HelloAMP - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Architecture Test Analyze Window Help
Debug
tab
New Work Item
GPU MatMult_2011-...06_112029.CvTrace
HelloAMP.cpp
Utilization
Threads
Cores
Demystify
Zoom
Sort by: Start Time
Markers
Seconds
0
1
2
3
4
5
6
Error List
Output
Ready

6 of 9 channels hidden from view Show All Channels

Thread Id	Name
7852	Main Thread
	GPU Adapter 0 (GPU MatMult, PID=5740)
	GPU Adapter 0 (other processes)

GPU Access
Process: GPU MatMult (PID=5740)
DMA Packet Type: ClientRenderBuffer
Duration = 6.5233 ms

Visible Timeline Profile

CPU	50%	🟩 Execution
	4%	🟥 Synchronization
	45%	I/O
	0%	Sleep
	0%	Memory Management
	0%	Preemption
	0%	UI Processing
		Per Thread Summary
		Disk Operations
		Markers
GPU	47%	🟩 Execution
	1%	Other Processes

Profile Report | Current | Unblocking Stack | Hints

There are 1 overlapping GPU segments selected: Copy

Context Handle	Process ID	DMA Packet Type	Submit Time(ms)	Delay In Queue(ms)	Start Time(ms)	Processing Duration(ms)	Total Tim
0xCD756438	GPU Mat	ClientRenderBuffer	3,196.453	0.000	3,196.453	6.523	6.523

- Direct3D-centric
  - Supports any library/programming model built on it
  - C++ AMP specific events

- Integrated GPU and CPU view

- Goal is to analyze high-level performance metrics
  - Memory copy overheads
  - Synchronization overheads across CPU/GPU
  - GPU activity and contention with other processes

http://blogs.msdn.com/b/nativeconcurrency/archive/2012/03/09/analyzing-c-amp-code-with-the-concurrency-visualizer.aspx

Agenda checkpoint

- Context ✓
- Code ✓
- IDE ✓
- Summary

lorem ipsum dolor sit
Consectetuer adipiscing
Sed diam nonummy
Euismod tincidunt laoreet
Dolore magna aliquam
Volutpat. Ut wisi enim ad
Minim veniam nostrud
Exerci tation ullamcorper
Suscipit lobortis nisl ut

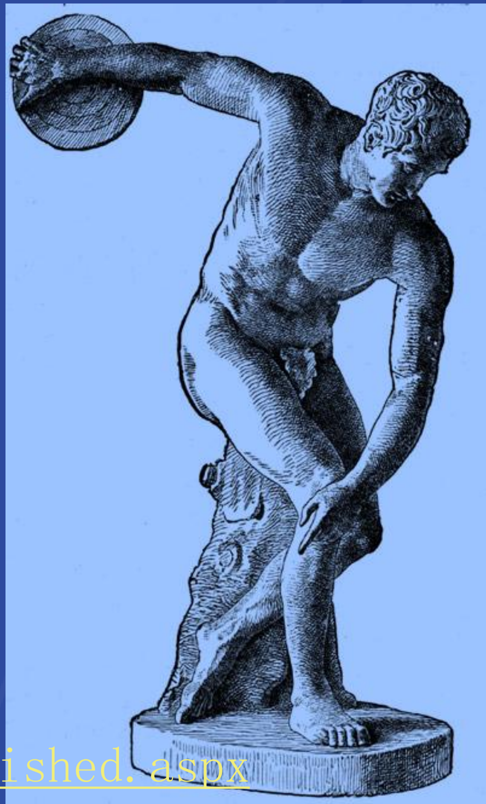
Learn C++ AMP

- book http://www.gregcons.com/cppamp/
- training http://www.acceleware.com/cpp-amp-training
- videos http://channel9.msdn.com/Tags/c++-accelerated-massive-parallelism
- articles http://blogs.msdn.com/b/nativeconcurrency/archive/2012/04/05/c-amp-articles-in-msdn-magazine-april-issue.aspx
- samples http://blogs.msdn.com/b/nativeconcurrency/archive/2012/01/30/c-amp-sample-projects-for-download.aspx
- guides http://blogs.msdn.com/b/nativeconcurrency/archive/2012/04/11/c-amp-for-the-cuda-programmer.aspx
- spec http://blogs.msdn.com/b/nativeconcurrency/archive/2012/02/03/c-amp-open-spec-published.aspx
- forum http://social.msdn.microsoft.com/Forums/en/parallelcppnative/threads

http://blogs.msdn.com/nativeconcurrency/

Summary

- Democratization of parallel hardware programmability
  - Performance for the mainstream
  - Hardware abstraction platform
  - High-level abstractions in modern C++ (*not* C)
  - Future proof, minimal, data-parallel API
  - An open specification
  - State-of-the-art Visual Studio IDE



http://blogs.msdn.com/b/nativeconcurrency/archive/2012/02/03/c-amp-open-spec-published.aspx

Microsoft®