

CUDA-X86

Michael Wolfe
The Portland Group
www.pgroup.com

CUDA C for GPUs

```

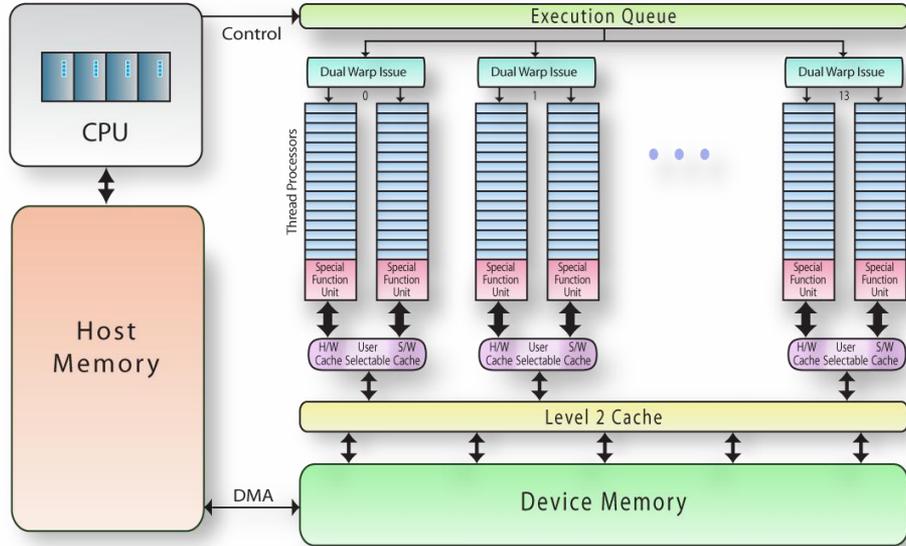
cudaMalloc (&A, bytes) ;
cudaMemcpy (A, data, bytes) ;
...
sgemm<<<dim3 (m/16, n/16) , dim3 (16, 16)>>>
(A, la, B, lb, C, lc) ;
...

```

```

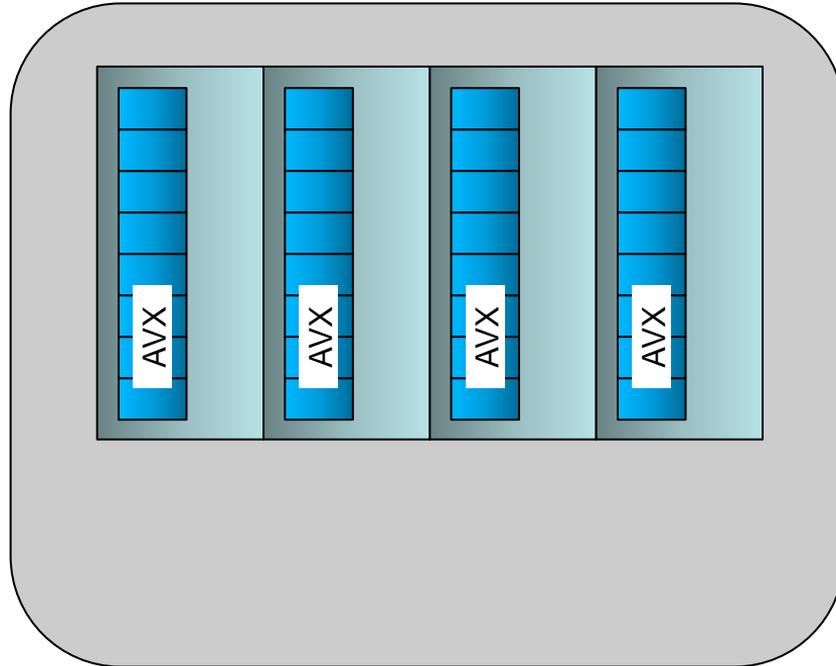
__global__ void sgemm( float *A, int la,
float* B, int lb, float* C, int lc )
{
    int tx=threadIdx.x, ty=threadIdx.y;
    int i = blockIdx.x*16+tx;
    int j = blockIdx.y*16+ty;
    float Cij = C[i+j*lc];
    __shared__ float Ab[16][16];
    __shared__ float Bb[16][16];
    for(int kb=0; kb<lc; kb+=16){
        Ab[tx][ty] = A[i+la*(kb+ty)];
        Bb[tx][ty] = B[kb+tx+lb*(j)];
        __syncthreads();
        for(int k=0; k<16; ++k)
            Cij += Ab[tx][k]*Bb[k][ty];
        __syncthreads();
    }
    C[i+j*lc] = Cij;
}

```



©2010 The Portland Group, Inc.

CUDA C for Multi-core



```
cudaMalloc (&A, bytes) ;
cudaMemcpy (A, data, bytes) ;
...
sgemm<<<dim3 (m/16, n/16) , dim3 (16, 16)>>>
    (A, la, B, lb, C, lc) ;
...

__global__ void sgemm( float *A, int la,
    float* B, int lb, float* C, int lc )
{
    int tx=threadIdx.x, ty=threadIdx.y;
    int i = blockIdx.x*16+tx;
    int j = blockIdx.y*16+ty;
    float Cij = C[i+j*lc];
    __shared__ float Ab[16][16];
    __shared__ float Bb[16][16];
    for(int kb=0; kb<lc; kb+=16){
        Ab[tx][ty] = A[i+la*(kb+ty)];
        Bb[tx][ty] = B[kb+tx+lb*(j)];
        __syncthreads();
        for(int k=0; k<16; ++k)
            Cij += Ab[tx][k]*Bb[k][ty];
        __syncthreads();
    }
    C[i+j*lc] = Cij;
}
```

PGI[®]

- C, C++, Fortran Compilers
- Optimizing, Vectorizing, Parallelizing
- Graphical debugger, profiler
- AMD and Intel, 32-bit and 64-bit
- PGI Unified Binary
- Linux, Apple OS/X, Microsoft Windows
- PGI Visual Fortran in Visual Studio
- CUDA Fortran
- PGI Accelerator Programming Model

The screenshot shows the PGI website in a Mozilla Firefox browser window. The browser title is "The Portland Group - Mozilla Firefox" and the address bar shows "http://www.pgroup.com/". The website features a navigation menu with links for Technology, Products, Services, Downloads, Support, Licensing, Resources, Purchase, and Company. A large banner image of a snow-capped mountain is displayed with the text "PGI 2010 PGI Accelerator™ x64+NVIDIA Compilers". Below the banner, there are sections for "Multicore Optimizing Parallel Compilers" and "Join PGI at SC10". The "Multicore Optimizing Parallel Compilers" section includes three product boxes: "PGI Workstation", "PGI CDK", and "PGI Visual Fortran". The "Join PGI at SC10" section mentions a booth at SC10 and provides links for PGI News, including a plan to develop a CUDA C compiler for x86, a tutorial on GPU programming, and the availability of PGI 2010 version 10.9. The website also has a footer with sections for "Optimizing Performance", "Purchasing Information", and "Support Options".

C Matrix Sum

```
nbytes = N*M*sizeof(float);
A = (float*)malloc( nbytes );
B = (float*)malloc( nbytes );
.../* fill A and B */
MyMadd( A, B, N, M );
.../* continue using A and B */
free( A );
free( B );
....
```

```
void MyMadd( float* A, float* B, int N, int M ){
    int i,j;
    for( j = 0; j < N; ++j )
        for( i = 0; i < M; ++i )
            A[i+j*M] += B[i+j*M];
}
```

CUDA C Matrix Sum

```
cudaMalloc((void**)&dA, nbytes );
cudaMalloc((void**)&dB, nbytes );
cudaMemcpy( dA, A, nbytes, cudaMemcpyHostToDevice);
cudaMemcpy( dB, B, nbytes, cudaMemcpyHostToDevice);
/* ... other stuff ... */
GPUMadd<<< dim3(N/64,M/4,1), dim3(64,4,1) >>>
    (dA, dB, N, M );
/* ... more uses of dA, dB ... */
cudaFree( dA );
cudaFree( dB );
....
__global__ void GPUMadd(float* A,float* B,int N,int M){
    int  j = threadIdx.y + blockIdx.y*blockDim.y;
    int  i = threadIdx.x + blockIdx.x*blockDim.x;
    A[i+j*M] += B[i+j*M];
}
```

Parallel Structure of the Kernel

thread block 0,0

0,0	A[i+j*M] += B[i+j*M]
0,1	A[i+j*M] += B[i+j*M]
0,2	A[i+j*M] += B[i+j*M]
0,4	A[i+j*M] += B[i+j*M]
0,5	A[i+j*M] += B[i+j*M]
0,6	A[i+j*M] += B[i+j*M]
0,7	A[i+j*M] += B[i+j*M]
0,8	A[i+j*M] += B[i+j*M]
0,9	A[i+j*M] += B[i+j*M]

●
●
●

thread block 1,0

0,0	A[i+j*M] += B[i+j*M]
0,1	A[i+j*M] += B[i+j*M]
0,2	A[i+j*M] += B[i+j*M]
0,4	A[i+j*M] += B[i+j*M]
0,5	A[i+j*M] += B[i+j*M]
0,6	A[i+j*M] += B[i+j*M]
0,7	A[i+j*M] += B[i+j*M]
0,8	A[i+j*M] += B[i+j*M]
0,9	A[i+j*M] += B[i+j*M]

●
●
●

thread block N/64,M/4

0,0	A[i+j*M] += B[i+j*M]
0,1	A[i+j*M] += B[i+j*M]
0,2	A[i+j*M] += B[i+j*M]
0,4	A[i+j*M] += B[i+j*M]
0,5	A[i+j*M] += B[i+j*M]
0,6	A[i+j*M] += B[i+j*M]
0,7	A[i+j*M] += B[i+j*M]
0,8	A[i+j*M] += B[i+j*M]
0,9	A[i+j*M] += B[i+j*M]

● ● ●

●
●
●

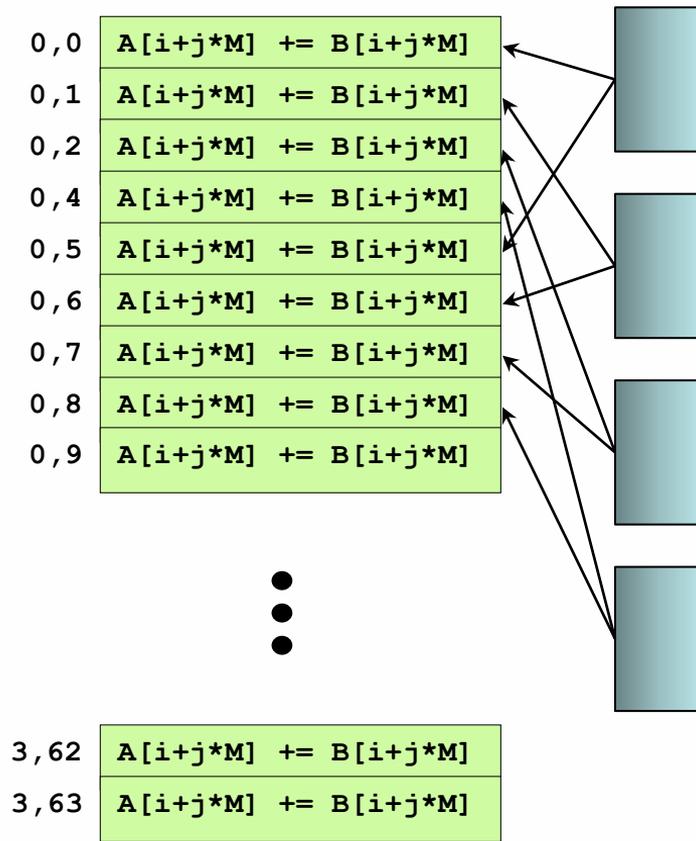
3,62	A[i+j*M] += B[i+j*M]
3,63	A[i+j*M] += B[i+j*M]

3,62	A[i+j*M] += B[i+j*M]
3,63	A[i+j*M] += B[i+j*M]

3,62	A[i+j*M] += B[i+j*M]
3,63	A[i+j*M] += B[i+j*M]

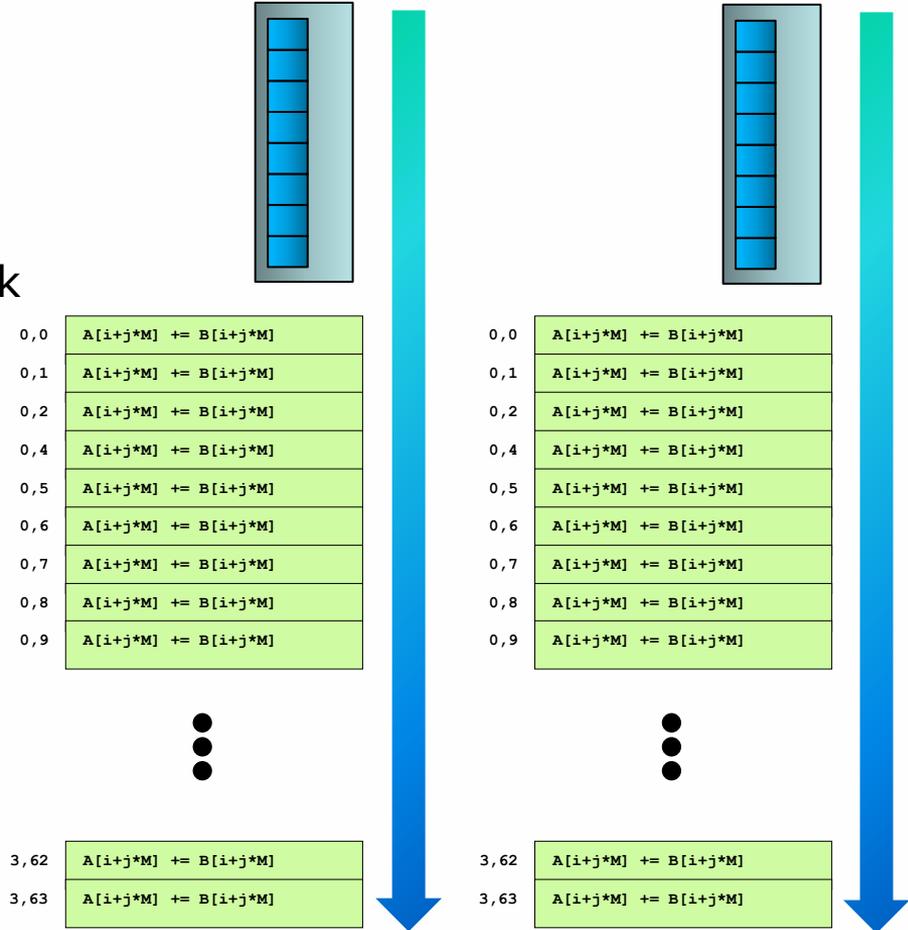
Phase 1: Emulation

- Each CUDA thread becomes a task
- x86 threads share the tasks
- `__syncthreads()` becomes task barrier
- Full functionality
- Full debugger support
- Parallelism between threads in block
- Blocks executed sequentially
- CUDA Fortran emulation mode



Phase 2: Optimization

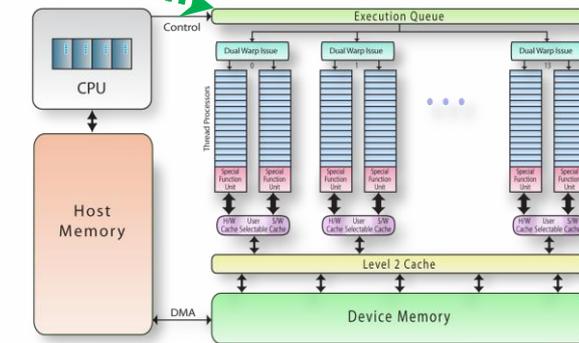
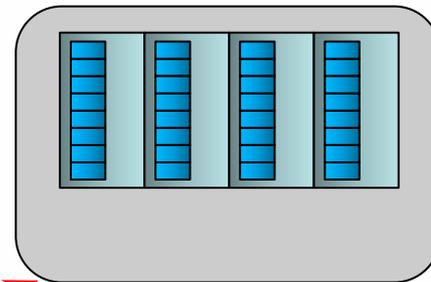
- Each CUDA thread block becomes a task
- One X86 thread executes each task
- Vectorize the thread indices
- `__shared__` becomes local to task
- `__syncthreads()` comes for free
- Parallelism between thread blocks



Phase 3: Unification

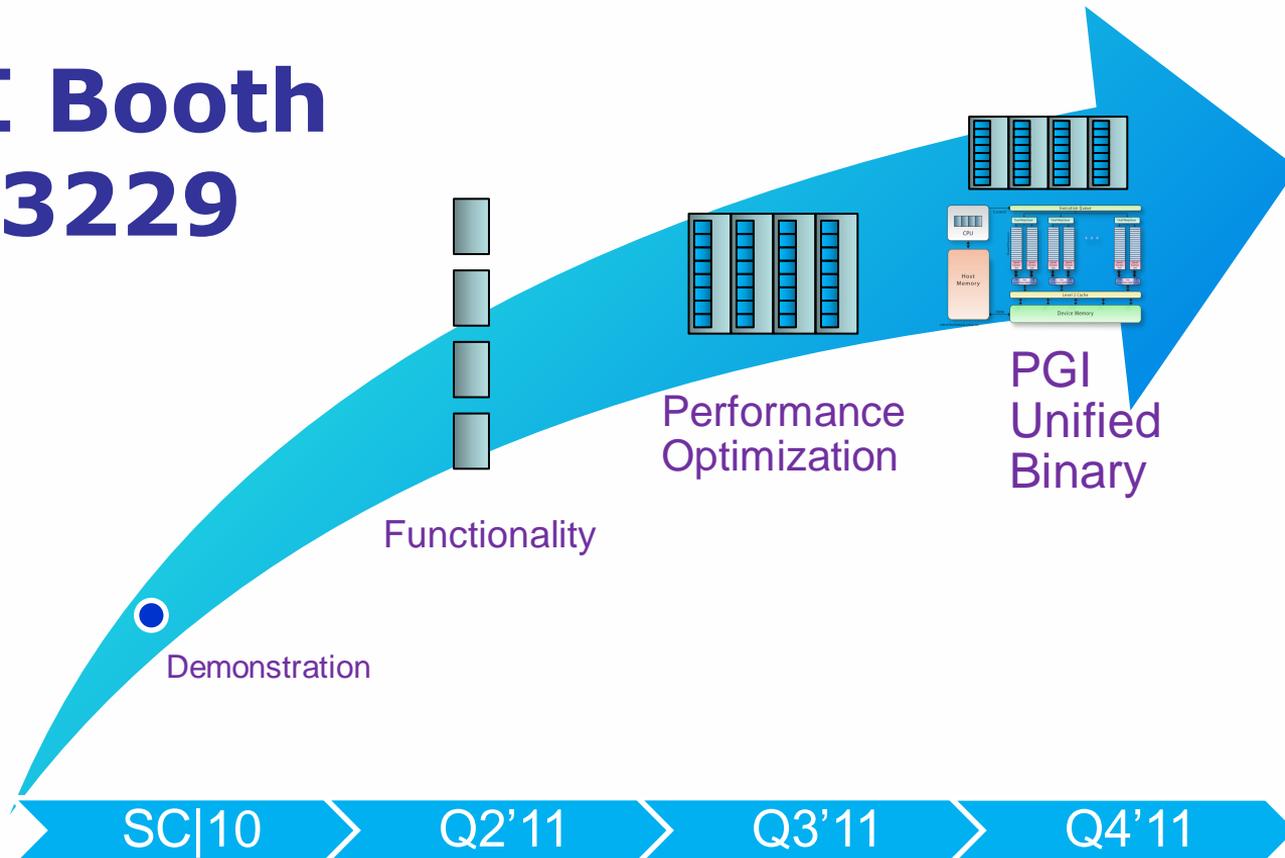
- PGI Unified Binary technology
- Generate both x86 host code AND NVIDIA device code for kernels
- Choose at runtime which code to run

```
0,0  A[i+j*M] += B[i+j*M]
0,1  A[i+j*M] += B[i+j*M]
0,2  A[i+j*M] += B[i+j*M]
0,4  A[i+j*M] += B[i+j*M]
0,5  A[i+j*M] += B[i+j*M]
0,6  A[i+j*M] += B[i+j*M]
0,7  A[i+j*M] += B[i+j*M]
0,8  A[i+j*M] += B[i+j*M]
0,9  A[i+j*M] += B[i+j*M]
...
3,62 A[i+j*M] += B[i+j*M]
3,63 A[i+j*M] += B[i+j*M]
```



©2010 The Portland Group, Inc.

PGI Booth #3229



More information at www.pgroup.com/cuda_x86.htm