Memory Hierarchies in CPU/GPU Architectures

Siegfried Höfinger

VSC Research Center, TU Wien

October 28, 2024

→ https://tinyurl.com/cudafordummies/i/l2/notes-l2.pdf

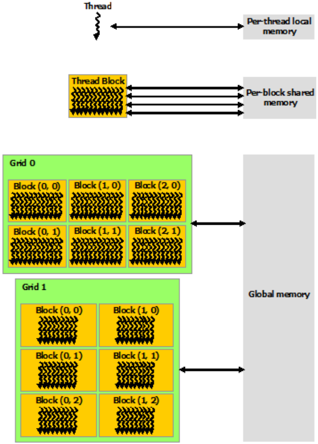
Outline

Memory Hierarchy

Take Home Messages

Memory Hierarchy

CUDA C-Programming Guide, Programming Model

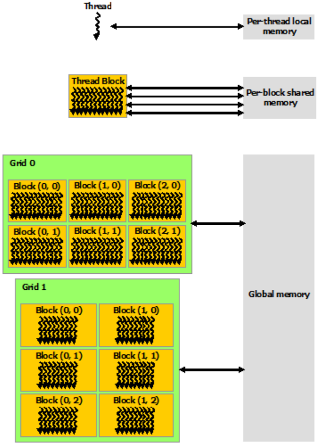


→ https://docs.nvidia.com/cuda/cuda-c-programming-guide

- Threads have access to several memory spaces

Memory Hierarchy

CUDA C-Programming Guide, Programming Model

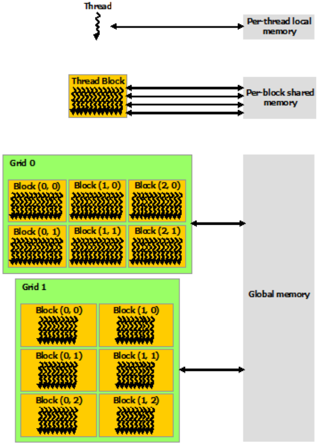


→ https://docs.nvidia.com/cuda/cuda-c-programming-guide

- Threads have access to several memory spaces
- Private local memory for each individual thread (fast)

Memory Hierarchy

CUDA C-Programming Guide, Programming Model

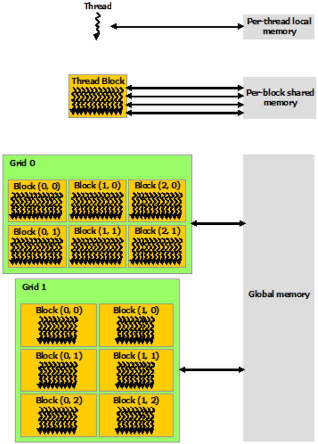


→ https://docs.nvidia.com/cuda/cuda-c-programming-guide

- Threads have access to several memory spaces
- Private local memory for each individual thread (fast)
- Shared memory for all threads within a thread block (pretty fast)

Memory Hierarchy

CUDA C-Programming Guide, Programming Model

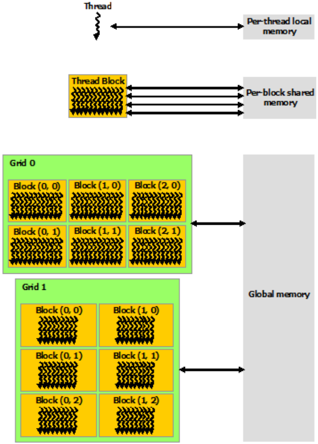


→ https://docs.nvidia.com/cuda/cuda-c-programming-guide

- Threads have access to several memory spaces
- Private local memory for each individual thread (fast)
- Shared memory for all threads within a thread block (pretty fast)
- Global memory accessible to all threads (slow)

Memory Hierarchy

CUDA C-Programming Guide, Programming Model

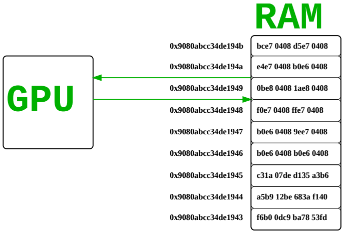


→ https://docs.nvidia.com/cuda/cuda-c-programming-guide

- Threads have access to several memory spaces
- Private local memory for each individual thread (fast)
- Shared memory for all threads within a thread block (pretty fast)
- Global memory accessible to all threads (slow)
- Special ROMs, constant and texture memory

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming



```
0x9080abcc34de194b  bce7 0408 d5e7 0408
0x9080abcc34de194a  e4e7 0408 b0e6 0408
0x9080abcc34de1949  0be8 0408 1ae8 0408
0x9080abcc34de1948  f0e7 0408 ffe7 0408
0x9080abcc34de1947  b0e6 0408 9ee7 0408
0x9080abcc34de1946  b0e6 0408 b0e6 0408
0x9080abcc34de1945  c31a 07de d135 a3b6
0x9080abcc34de1944  a5b9 12be 683a f140
0x9080abcc34de1943  f6b0 0dc9 ba78 53fd
```

Unified Memory:

GPUs of compute capability $\geq$ 6.x and $\geq$ CUDA 8.0

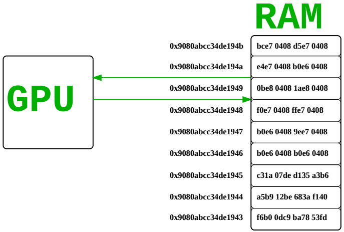
Separate Device/Host Memory:

GPUs of compute capability $<$ 6.x and $<$ CUDA 8.0

→ https://docs.nvidia.com/cuda/cuda-c-programming-guide

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming



0x9080abcc34de194b	bce7 0408 d5e7 0408
0x9080abcc34de194a	e4e7 0408 b0e6 0408
0x9080abcc34de1949	0be8 0408 1ae8 0408
0x9080abcc34de1948	f0e7 0408 ffe7 0408
0x9080abcc34de1947	b0e6 0408 9ee7 0408
0x9080abcc34de1946	b0e6 0408 b0e6 0408
0x9080abcc34de1945	c31a 07de d135 a3b6
0x9080abcc34de1944	a5b9 12be 683a f140
0x9080abcc34de1943	f6b0 0dc9 ba78 53fd

Unified Memory:

GPUs of compute capability $\geq$ 6.x and $\geq$ CUDA 8.0

Separate Device/Host Memory:

GPUs of compute capability $<$ 6.x and $<$ CUDA 8.0

→ https://docs.nvidia.com/cuda/cuda-c-programming-guide

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

- Unified memory (managed) for recent GPUs (Pascal class or newer) greatly simplifies programming4/porting2 the GPU

→ https://docs.nvidia.com/cuda/cuda-c-programming-guide

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

- Unified memory (managed) for recent GPUs (Pascal class or newer) greatly simplifies programming4/porting2 the GPU
- Single common address space accessible from any processor in a system

→ https://docs.nvidia.com/cuda/cuda-c-programming-guide

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

- Unified memory (managed) for recent GPUs (Pascal class or newer) greatly simplifies programming4/porting2 the GPU
- Single common address space accessible from any processor in a system
- Strict distinction into device- and host-memory becoming less important to the programmer

→ https://docs.nvidia.com/cuda/cuda-c-programming-guide

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

- Unified memory (managed) for recent GPUs (Pascal class or newer) greatly simplifies programming4/porting2 the GPU
- Single common address space accessible from any processor in a system
- Strict distinction into device- and host-memory becoming less important to the programmer
- Single-pointer-to-data model, no longer any need to copy forth/back data into/from GPU memory

→ https://docs.nvidia.com/cuda/cuda-c-programming-guide

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

- Unified memory (managed) for recent GPUs (Pascal class or newer) greatly simplifies programming4/porting2 the GPU
- Single common address space accessible from any processor in a system
- Strict distinction into device- and host-memory becoming less important to the programmer
- Single-pointer-to-data model, no longer any need to copy forth/back data into/from GPU memory
- On-demand page migration (hardware supported) — Page Migration Engine

→ https://docs.nvidia.com/cuda/cuda-c-programming-guide

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

- Unified memory (managed) for recent GPUs (Pascal class or newer) greatly simplifies programming4/porting2 the GPU
- Single common address space accessible from any processor in a system
- Strict distinction into device- and host-memory becoming less important to the programmer
- Single-pointer-to-data model, no longer any need to copy forth/back data into/from GPU memory
- On-demand page migration (hardware supported) — Page Migration Engine
- Simply invoked via cudaMallocManaged()

→ https://docs.nvidia.com/cuda/cuda-c-programming-guide

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

Multiple Thread Blocks Matrix Addition

```
__global__void MatAdd( float **A, float **B, float **C )
{
    int i, j;
    i = (blockIdx.x * blockDim.x) + threadIdx.x;
    j = (blockIdx.y * blockDim.y) + threadIdx.y;
    C[i][j] = A[i][j] + B[i][j];
}

int main()
{
    int i;
    dim3 threadsPerBlock, numBlocks;
    float **A, **B, **C;
    // unified memory allocation, B and C analogous
    cudaMallocManaged(&A, N * sizeof(float *));
    for (i = 0; i < N; i++) {
       cudaMallocManaged(&A[i], N * sizeof(float));
    }
    // kernel invocation
    MatAdd <<< numBlocks, threadsPerBlock >>> (A, B, C);
    cudaDeviceSynchronize();
    ...
    cudaFree(A);
}
```

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

Multiple Thread Blocks Matrix Addition

Standard malloc-like allocation on the host

```
__global__void MatAdd( float **A, float **B, float **C )
{
    int i, j;
    i = (blockIdx.x * blockDim.x) + threadIdx.x;
    j = (blockIdx.y * blockDim.y) + threadIdx.y;
    C[i][j] = A[i][j] + B[i][j];
}

int main()
{
    int i;
    dim3 threadsPerBlock, numBlocks;
    float **A, **B, **C;
    // unified memory allocation, B and C analogous
    cudaMallocManaged(&A, N * sizeof(float *));
    for (i = 0; i < N; i++) {
        cudaMallocManaged(&A[i], N * sizeof(float));
    }
    // kernel invocation
    MatAdd <<< numBlocks, threadsPerBlock >>> (A, B, C);
    cudaDeviceSynchronize();
    ...
    cudaFree(A);
}
```

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

Multiple Thread Blocks Matrix Addition

```
__global__void MatAdd( float **A, float **B, float **C )
{
    int i, j;
    i = (blockIdx.x * blockDim.x) + threadIdx.x;
    j = (blockIdx.y * blockDim.y) + threadIdx.y;
    C[i][j] = A[i][j] + B[i][j];
}

int main()
{
    int i;
    dim3 threadsPerBlock, numBlocks;
    float **A, **B, **C;
    // unified memory allocation, B and C analogous
    cudaMallocManaged(&A, N * sizeof(float *));
    for (i = 0; i < N; i++) {
        cudaMallocManaged(&A[i], N * sizeof(float));
    }
    // kernel invocation
    MatAdd <<< numBlocks, threadsPerBlock >>> (A, B, C);
    cudaDeviceSynchronize();
    ...
    cudaFree(A);
}
```

Standard malloc-like allocation on the host

host pointers directly usable in kernel code

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

Multiple Thread Blocks Matrix Addition

```
__global__void MatAdd( float **A, float **B, float **C )
{
    int i, j;
    i = (blockIdx.x * blockDim.x) + threadIdx.x;
    j = (blockIdx.y * blockDim.y) + threadIdx.y;
    C[i][j] = A[i][j] + B[i][j];
}

int main()
{
    int i;
    dim3 threadsPerBlock, numBlocks;
    float **A, **B, **C;
    // unified memory allocation, B and C analogous
    cudaMallocManaged(&A, N * sizeof(float *));
    for (i = 0; i < N; i++) {
        cudaMallocManaged(&A[i], N * sizeof(float));
    }
    // kernel invocation
    MatAdd <<< numBlocks, threadsPerBlock >>> (A, B, C);
    cudaDeviceSynchronize();
    ...
    cudaFree(A);
}
```

Standard malloc-like allocation on the host

host pointers directly usable in kernel code

ensure proper kernel completion

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

Multiple Thread Blocks Matrix Addition

```
__global__void MatAdd( float **A, float **B, float **C )
{
    int i, j;
    i = (blockIdx.x * blockDim.x) + threadIdx.x;
    j = (blockIdx.y * blockDim.y) + threadIdx.y;
    C[i][j] = A[i][j] + B[i][j];
}

int main()
{
    int i;
    dim3 threadsPerBlock, numBlocks;
    float **A, **B, **C;
    // unified memory allocation, B and C analogous
    cudaMallocManaged(&A, N * sizeof(float *));
    for (i = 0; i < N; i++) {
       cudaMallocManaged(&A[i], N * sizeof(float));
    }
    // kernel invocation
    MatAdd <<< numBlocks, threadsPerBlock >>> (A, B, C);
    cudaDeviceSynchronize();
    ...
    cudaFree(A);
}
```

Standard malloc-like allocation on the host

free memory when done

host pointers directly usable in kernel code

ensure proper kernel completion

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

2nd Form — Managed Global Memory

```
#define ARRAYDIM 268435456

// global managed declaration on GPU
__device__ __managed__ float x[ARRAYDIM], y[ARRAYDIM], z[ARRAYDIM];

__global__ void KrnlDmmy()
{
    int i;
    i = (blockIdx.x * blockDim.x) + threadIdx.x;
    x[i] = y[i] + z[i];
    return;
}

int main()
{
    ...
    // kernel invocation
    KrnlDmmy <<< numBlocks, threadsPerBlock >>> ();
    cudaDeviceSynchronize();
    ...
}
```

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

2nd Form — Managed Global Memory

Global variables directly usable on device & host

```
#define ARRAYDIM 268435456

// global managed declaration on GPU
__device__ __managed__ float x[ARRAYDIM], y[ARRAYDIM], z[ARRAYDIM];

__global__ void KrnlDmmy()
{
    int i;
    i = (blockIdx.x * blockDim.x) + threadIdx.x;
    x[i] = y[i] + z[i];
    return;
}

int main()
{
    ...
   // kernel invocation
   KrnlDmmy <<< numBlocks, threadsPerBlock >>> ();
   cudaDeviceSynchronize();
    ...
}
```

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

2nd Form — Managed Global Memory

Global variables directly usable on device & host

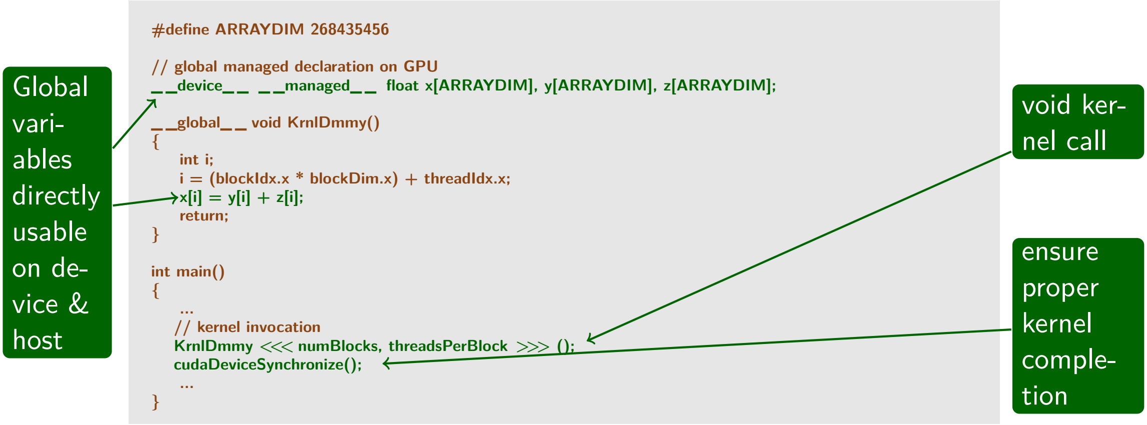
```
#define ARRAYDIM 268435456

// global managed declaration on GPU
__device__ __managed__ float x[ARRAYDIM], y[ARRAYDIM], z[ARRAYDIM];

__global__ void KrnlDmmy()
{
    int i;
    i = (blockIdx.x * blockDim.x) + threadIdx.x;
    x[i] = y[i] + z[i];
    return;
}

int main()
{
    ...
   // kernel invocation
   KrnlDmmy <<< numBlocks, threadsPerBlock >>> ();
   cudaDeviceSynchronize();
    ...
}
```

void kernel call

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

2nd Form — Managed Global Memory



```
#define ARRAYDIM 268435456

// global managed declaration on GPU
__device__ __managed__ float x[ARRAYDIM], y[ARRAYDIM], z[ARRAYDIM];

__global__ void KrnlDmmy()
{
    int i;
    i = (blockIdx.x * blockDim.x) + threadIdx.x;
    x[i] = y[i] + z[i];
    return;
}

int main()
{
    ...
   // kernel invocation
   KrnlDmmy <<< numBlocks, threadsPerBlock >>> ();
   cudaDeviceSynchronize();
    ...
}
```

Global variables directly usable on device & host

void kernel call

ensure proper kernel completion

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

Unified Memory Example Version 1

```
#define ARRAYDIM 268435456

__global__ void KrnlDmmy(float *x, float *y, float *z)
{
    int i;
    i = (blockIdx.x * blockDim.x) + threadIdx.x;
    x[i] = (float) i;
    y[i] = (float) (i + 1);
    z[i] = x[i] + y[i];
    return;
}

int main()
{
    ...
    // unified memory allocation, b and c analogous
    cudaMallocManaged(&a, ARRAYDIM * sizeof(float));
    // kernel invocation
    KrnlDmmy <<< numBlocks, threadsPerBlock >>> (a, b, c);
    cudaDeviceSynchronize();
     ...
}
```

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

Unified Memory Example Version 1

```
#define ARRAYDIM 268435456

__global__ void KrnlDmmy(float *x, float *y, float *z)
{
    int i;
    i = (blockIdx.x * blockDim.x) + threadIdx.x;
    x[i] = (float) i;
    y[i] = (float) (i + 1);
    z[i] = x[i] + y[i];
    return;
}

int main()
{
    ...
    // unified memory allocation, b and c analogous
    cudaMallocManaged(&a, ARRAYDIM * sizeof(float));
    // kernel invocation
    KrnlDmmy <<< numBlocks, threadsPerBlock >>> (a, b, c);
    cudaDeviceSynchronize();
     ...
}
```

Kernel initialization and calculation

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

Unified Memory Example Version 1

```
#define ARRAYDIM 268435456

__global__ void KrnlDmmy(float *x, float *y, float *z)
{
    int i;
    i = (blockIdx.x * blockDim.x) + threadIdx.x;
    x[i] = (float) i;
    y[i] = (float) (i + 1);
    z[i] = x[i] + y[i];
    return;
}

int main()
{
    ...
    // unified memory allocation, b and c analogous
    cudaMallocManaged(&a, ARRAYDIM * sizeof(float));
    // kernel invocation
    KrnlDmmy <<< numBlocks, threadsPerBlock >>> (a, b, c);
    cudaDeviceSynchronize();
     ...
}
```

Kernel initialization and calculation

1 GB arrays

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

```
cuda–zen sh@n3073–009:~$  nvcc ./unified_memory_example_1.cu
cuda–zen sh@n3073–009:~$  ./a.out
cuda–zen sh@n3073–009:~$  nsys nvprof ./a.out

CUDA API Statistics:

Time (%)  Total Time (ns)  Num Calls   Avg (ns)     Med (ns)     Min (ns)   Max (ns)   StdDev (ns)           Name
--------  ---------------  ---------  -----------  -----------  ---------  ---------  -----------  ---------------------
    40.1        225077888          3   75025962.7      25740.0      15769  225036379  129912831.5  cudaMallocManaged
    37.2        208995989          1  208995989.0  208995989.0  208995989  208995989          0.0  cudaDeviceSynchronize
    22.7        127419916          3   42473305.3   45704095.0   35981266   45734555    5622291.6  cudaFree
     0.0            41560          1      41560.0      41560.0      41560      41560          0.0  cudaLaunchKernel

CUDA Kernel Statistics:

Time (%)  Total Time (ns)  Instances   Avg (ns)     Med (ns)     Min (ns)   Max (ns)   StdDev (ns)                  Name
--------  ---------------  ---------  -----------  -----------  ---------  ---------  -----------  -----------------------------------
   100.0        208993223          1  208993223.0  208993223.0  208993223  208993223          0.0  KrnlDmmy(float *, float *, float *)
```

→ https://tinyurl.com/cudafordummies/i/l2/unified_memory_example_1.cu

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

```
cuda-zen sh@n3073-009:~$  nvcc ./unified_memory_example_1.cu
cuda-zen sh@n3073-009:~$  ./a.out
cuda-zen sh@n3073-009:~$  nsys nvprof ./a.out

CUDA API Statistics:

Time (%)  Total Time (ns)  Num Calls   Avg (ns)     Med (ns)     Min (ns)   Max (ns)   StdDev (ns)          Name
--------  ---------------  ---------  -----------  -----------  ---------  ---------  -----------  ---------------------
    40.1        225077888          3   75025962.7      25740.0      15769  225036379  129912831.5  cudaMallocManaged
    37.2        208995989          1  208995989.0  208995989.0  208995989  208995989          0.0  cudaDeviceSynchronize
    22.7        127419916          3   42473305.3   45704095.0   35981266   45734555    5622291.6  cudaFree
     0.0            41560          1      41560.0      41560.0      41560      41560          0.0  cudaLaunchKernel

CUDA Kernel Statistics:

Time (%)  Total Time (ns)  Instances   Avg (ns)     Med (ns)     Min (ns)   Max (ns)   StdDev (ns)                 Name
--------  ---------------  ---------  -----------  -----------  ---------  ---------  -----------  -----------------------------------
   100.0        208993223          1  208993223.0  208993223.0  208993223  208993223          0.0  KrnlDmmy(float *, float *, float *)
```

→ https://tinyurl.com/cudafordummies/i/l2/unified_memory_example_1.cu

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

- Straightforward profiling with nsys nvprof has lost detailed information w.r.2 unified memory analysis, especially numbers of page faults etc.

→ https://devblogs.nvidia.com/unified-memory-cuda-beginners

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

- Straightforward profiling with nsys nvprof has lost detailed information w.r.2 unified memory analysis, especially numbers of page faults etc.
- In a real application, the GPU is likely to use the data more often and for more complex computations, so the overhead from page faulting may become negligible; to quantify, we...

→ https://devblogs.nvidia.com/unified-memory-cuda-beginners

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

- Straightforward profiling with nsys nvprof has lost detailed information w.r.2 unified memory analysis, especially numbers of page faults etc.
- In a real application, the GPU is likely to use the data more often and for more complex computations, so the overhead from page faulting may become negligible; to quantify, we...

1. Could separate the actual calculation from the initialization with the help of a second kernel

→ https://devblogs.nvidia.com/unified-memory-cuda-beginners

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

- Straightforward profiling with nsys nvprof has lost detailed information w.r.2 unified memory analysis, especially numbers of page faults etc.
- In a real application, the GPU is likely to use the data more often and for more complex computations, so the overhead from page faulting may become negligible; to quantify, we...

1. Could separate the actual calculation from the initialization with the help of a second kernel
2. Could repeat the kernel doing the calculation many times

→ https://devblogs.nvidia.com/unified-memory-cuda-beginners

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

- Straightforward profiling with nsys nvprof has lost detailed information w.r.2 unified memory analysis, especially numbers of page faults etc.
- In a real application, the GPU is likely to use the data more often and for more complex computations, so the overhead from page faulting may become negligible; to quantify, we...

1. Could separate the actual calculation from the initialization with the help of a second kernel
2. Could repeat the kernel doing the calculation many times
3. Could use unified memory prefetching to explicitly move the data to the GPU

→ https://devblogs.nvidia.com/unified-memory-cuda-beginners

Memory Hierarchy cont.

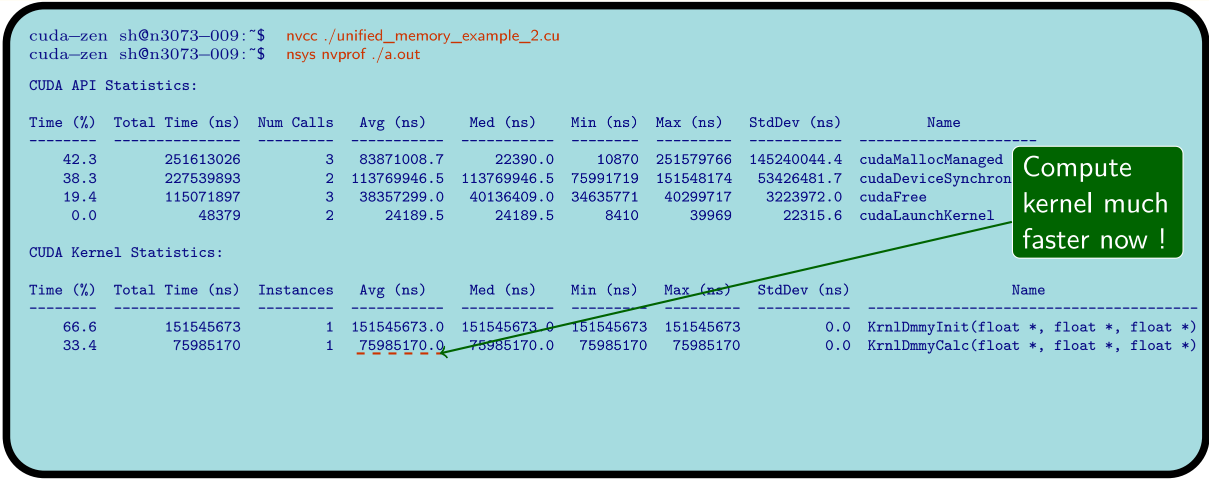
CUDA C-Programming Guide, Unified Memory Programming cont.

```
cuda-zen sh@n3073-009:~$  nvcc ./unified_memory_example_2.cu
cuda-zen sh@n3073-009:~$  nsys nvprof ./a.out

CUDA API Statistics:

Time (%)  Total Time (ns)  Num Calls    Avg (ns)      Med (ns)     Min (ns)   Max (ns)    StdDev (ns)           Name
--------  ---------------  ---------  -----------  -----------  --------  ---------  -----------  ---------------------
    42.3        251613026          3   83871008.7      22390.0     10870  251579766  145240044.4  cudaMallocManaged
    38.3        227539893          2  113769946.5  113769946.5  75991719  151548174   53426481.7  cudaDeviceSynchronize
    19.4        115071897          3   38357299.0   40136409.0  34635771   40299717    3223972.0  cudaFree
     0.0            48379          2      24189.5      24189.5      8410      39969      22315.6  cudaLaunchKernel

CUDA Kernel Statistics:

Time (%)  Total Time (ns)  Instances    Avg (ns)      Med (ns)     Min (ns)   Max (ns)   StdDev (ns)                   Name
--------  ---------------  ---------  -----------  -----------  ---------  ---------  -----------  ----------------------------------------
    66.6        151545673          1  151545673.0  151545673.0  151545673  151545673          0.0  KrnlDmmyInit(float *, float *, float *)
    33.4         75985170          1   75985170.0   75985170.0   75985170   75985170          0.0  KrnlDmmyCalc(float *, float *, float *)
```

→ https://tinyurl.com/cudafordummies/i/l2/unified_memory_example_2.cu

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.



```
cuda-zen sh@n3073-009:~$  nvcc ./unified_memory_example_2.cu
cuda-zen sh@n3073-009:~$  nsys nvprof ./a.out

CUDA API Statistics:

Time (%)  Total Time (ns)  Num Calls    Avg (ns)     Med (ns)    Min (ns)   Max (ns)    StdDev (ns)          Name
--------  ---------------  ---------  -----------  -----------  --------  ---------  -----------  ---------------------
    42.3        251613026          3   83871008.7      22390.0     10870  251579766  145240044.4  cudaMallocManaged
    38.3        227539893          2  113769946.5  113769946.5  75991719  151548174   53426481.7  cudaDeviceSynchron
    19.4        115071897          3   38357299.0   40136409.0  34635771   40299717    3223972.0  cudaFree
     0.0            48379          2      24189.5      24189.5      8410      39969      22315.6  cudaLaunchKernel

CUDA Kernel Statistics:

Time (%)  Total Time (ns)  Instances    Avg (ns)     Med (ns)     Min (ns)    Max (ns)    StdDev (ns)                   Name
--------  ---------------  ---------  -----------  -----------  ---------  ---------  -----------  ------------------------------------
    66.6        151545673          1  151545673.0  151545673.0  151545673  151545673          0.0  KrnlDmmyInit(float *, float *, float *)
    33.4         75985170          1   75985170.0   75985170.0   75985170   75985170          0.0  KrnlDmmyCalc(float *, float *, float *)
```

Compute kernel much faster now !

→ https://tinyurl.com/cudafordummies/i/l2/unified_memory_example_2.cu

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

```
cuda-zen sh@n3073-009:~$  nvcc ./unified_memory_example_2.cu
cuda-zen sh@n3073-009:~$  nsys nvprof ./a.out

CUDA API Statistics:

Time (%)  Total Time                                                  Min (ns)   Max (ns)    StdDev (ns)            Name
--------  ----------                                                  --------  ---------  -----------  ---------------------
    42.3        2516                                                     10870  251579766  145240044.4  cudaMallocManaged
    38.3        2275                                                  75991719  151548174   53426481.7  cudaDeviceSynchron
    19.4        1150                                                  34635771   40299717    3223972.0  cudaFree
     0.0                                                                  8410      39969      22315.6  cudaLaunchKernel

CUDA Kernel Statisti

Time (%)  Total Time (ns)  Instances   Avg (ns)     Med (ns)     Min (ns)   Max (ns)   StdDev (ns)                 Name
--------  ---------------  ---------  -----------  -----------  ---------  ---------  -----------  ------------------------------------
    66.6        151545673          1  151545673.0  151545673.0  151545673  151545673          0.0  KrnlDmmyInit(float *, float *, float *)
    33.4         75985170          1   75985170.0   75985170.0   75985170   75985170          0.0  KrnlDmmyCalc(float *, float *, float *)
```

Still very low effective memory bandwith,

$$\frac{\frac{3\times 268435456\times 4}{10^9}}{75985170\times 10^{-9}} = 42.39\,GB/s$$

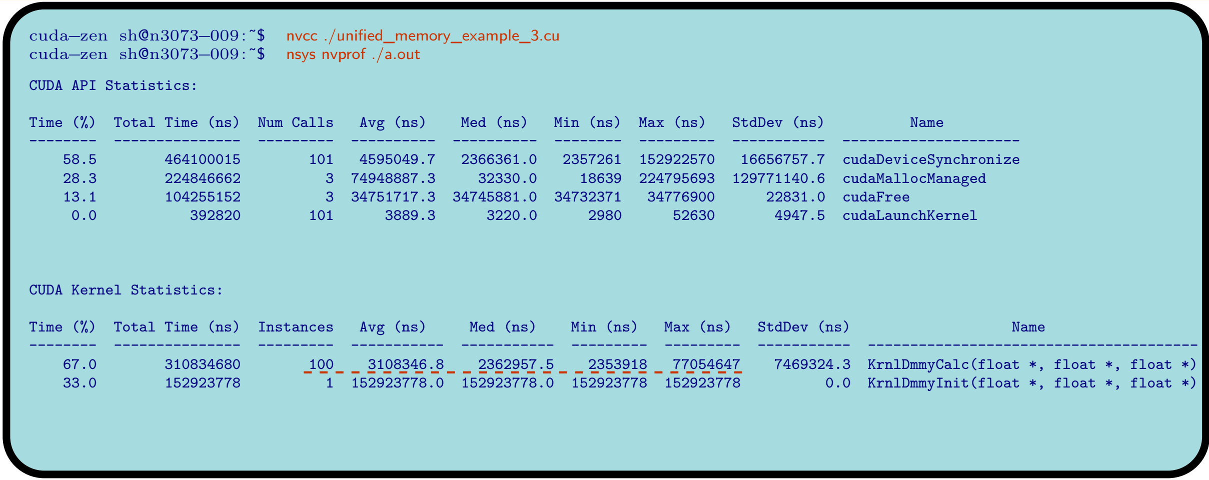
from theoretical $1600\,GB/s$

Compute kernel much faster now !

→ https://tinyurl.com/cudafordummies/i/l2/unified_memory_example_2.cu

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.



```
cuda-zen sh@n3073-009:~$  nvcc ./unified_memory_example_3.cu
cuda-zen sh@n3073-009:~$  nsys nvprof ./a.out

CUDA API Statistics:

Time (%)  Total Time (ns)  Num Calls   Avg (ns)    Med (ns)    Min (ns)   Max (ns)   StdDev (ns)          Name
--------  ---------------  ---------  ----------  ----------  --------  ---------  -----------  ---------------------
    58.5        464100015        101   4595049.7   2366361.0   2357261  152922570   16656757.7  cudaDeviceSynchronize
    28.3        224846662          3  74948887.3     32330.0     18639  224795693  129771140.6  cudaMallocManaged
    13.1        104255152          3  34751717.3  34745881.0  34732371   34776900      22831.0  cudaFree
     0.0           392820        101      3889.3      3220.0      2980      52630       4947.5  cudaLaunchKernel



CUDA Kernel Statistics:

Time (%)  Total Time (ns)  Instances   Avg (ns)     Med (ns)     Min (ns)   Max (ns)   StdDev (ns)                   Name
--------  ---------------  ---------  -----------  -----------  ---------  ---------  -----------  ---------------------------------------
    67.0        310834680        100    3108346.8    2362957.5    2353918   77054647    7469324.3  KrnlDmmyCalc(float *, float *, float *)
    33.0        152923778          1  152923778.0  152923778.0  152923778  152923778          0.0  KrnlDmmyInit(float *, float *, float *)
```

→ https://tinyurl.com/cudafordummies/i/l2/unified_memory_example_3.cu

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

```
cuda−zen sh@n3073−009:~$  nvcc ./unified_memory_example_3.cu
cuda−zen sh@n3073−009:~$  nsys nvprof ./a.out

CUDA API Statistics:

Time (%)  Total Time (ns)  Num Calls    Avg (ns)    Med (ns)   Min (ns)   Max (ns)   StdDev (ns)   ---
--------  ---------------  ---------  ----------  ----------  --------  ---------  -----------   ---
    58.5        464100015        101   4595049.7   2366361.0   2357261  152922570   16656757.7   cud
    28.3        224846662          3  74948887.3     32330.0     18639  224795693  129771140.6   cud
    13.1        104255152          3  34751717.3  34745881.0  34732371   34776900      22831.0   cud
     0.0           392820        101      3889.3      3220.0      2980      52630       4947.5   cud


CUDA Kernel Statistics:

Time (%)  Total Time (ns)  Instances    Avg (ns)     Med (ns)    Min (ns)   Max (ns)   StdDev (ns)                     Name
--------  ---------------  ---------  -----------  -----------  ---------  ---------  -----------  ---------------------------------------
    67.0        310834680        100    3108346.8    2362957.5    2353918   77054647    7469324.3  KrnlDmmyCalc(float *, float *, float *)
    33.0        152923778          1  152923778.0  152923778.0  152923778  152923778          0.0  KrnlDmmyInit(float *, float *, float *)
```

100× greatly improves effective memory bandwidth, 1036 GB/s and compute performance

→ https://tinyurl.com/cudafordummies/i/l2/unified_memory_example_3.cu

Memory Hierarchy cont.

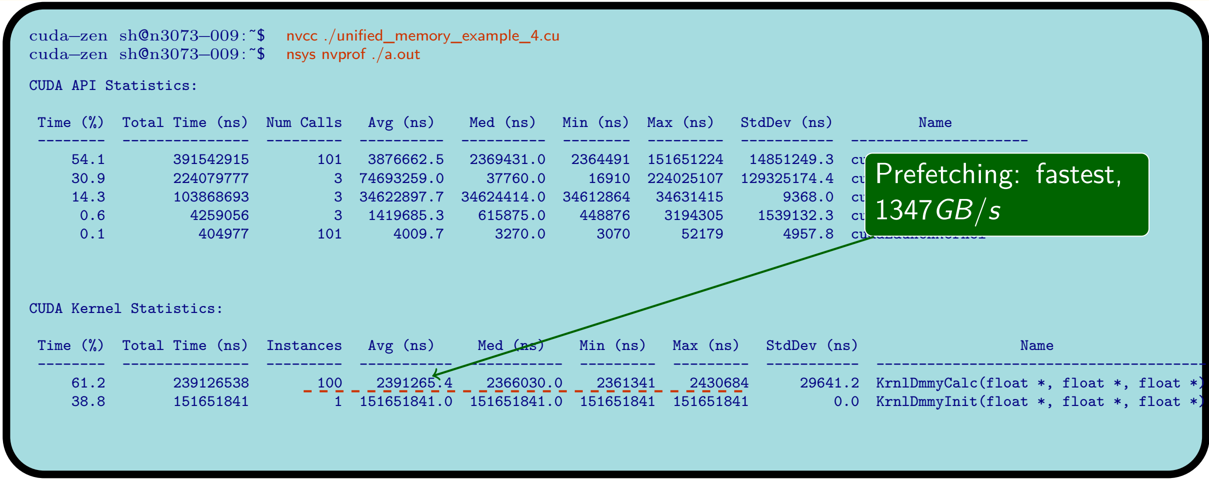
CUDA C-Programming Guide, Unified Memory Programming cont.

```
cuda-zen sh@n3073-009:~$  nvcc ./unified_memory_example_4.cu
cuda-zen sh@n3073-009:~$  nsys nvprof ./a.out

CUDA API Statistics:

 Time (%)  Total Time (ns)  Num Calls    Avg (ns)     Med (ns)    Min (ns)   Max (ns)   StdDev (ns)           Name
 --------  ---------------  ---------  ----------  ----------  --------  ---------  -----------  ---------------------
     54.1        391542915        101   3876662.5   2369431.0   2364491  151651224   14851249.3  cudaDeviceSynchronize
     30.9        224079777          3  74693259.0     37760.0     16910  224025107  129325174.4  cudaMallocManaged
     14.3        103868693          3  34622897.7  34624414.0  34612864   34631415       9368.0  cudaFree
      0.6          4259056          3   1419685.3    615875.0    448876    3194305    1539132.3  cudaMemPrefetchAsync
      0.1           404977        101      4009.7      3270.0      3070      52179       4957.8  cudaLaunchKernel



CUDA Kernel Statistics:

 Time (%)  Total Time (ns)  Instances    Avg (ns)     Med (ns)     Min (ns)   Max (ns)   StdDev (ns)                   Name
 --------  ---------------  ---------  -----------  -----------  ---------  ---------  -----------  ----------------------------------------
     61.2        239126538        100    2391265.4    2366030.0    2361341    2430684      29641.2  KrnlDmmyCalc(float *, float *, float *)
     38.8        151651841          1  151651841.0  151651841.0  151651841  151651841          0.0  KrnlDmmyInit(float *, float *, float *)
```

→ https://tinyurl.com/cudafordummies/i/l2/unified_memory_example_4.cu

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.



```
cuda-zen sh@n3073-009:~$  nvcc ./unified_memory_example_4.cu
cuda-zen sh@n3073-009:~$  nsys nvprof ./a.out

CUDA API Statistics:

 Time (%)  Total Time (ns)  Num Calls    Avg (ns)     Med (ns)    Min (ns)   Max (ns)    StdDev (ns)           Name
 --------  ---------------  ---------  -----------  -----------  ---------  ---------  ------------  --------------------
     54.1        391542915        101    3876662.5    2369431.0    2364491  151651224    14851249.3  cu
     30.9        224079777          3   74693259.0      37760.0      16910  224025107   129325174.4  cu
     14.3        103868693          3   34622897.7   34624414.0   34612864   34631415        9368.0  cu
      0.6          4259056          3    1419685.3     615875.0     448876    3194305     1539132.3  cu
      0.1           404977        101       4009.7       3270.0       3070      52179        4957.8  cu


CUDA Kernel Statistics:

 Time (%)  Total Time (ns)  Instances    Avg (ns)     Med (ns)     Min (ns)    Max (ns)   StdDev (ns)                  Name
 --------  ---------------  ---------  -----------  -----------  ----------  ----------  -----------  ----------------------------------------
     61.2        239126538        100    2391265.4    2366030.0     2361341     2430684      29641.2  KrnlDmmyCalc(float *, float *, float *)
     38.8        151651841          1  151651841.0  151651841.0   151651841   151651841          0.0  KrnlDmmyInit(float *, float *, float *)
```

Prefetching: fastest, $1347\,GB/s$

→ https://tinyurl.com/cudafordummies/i/l2/unified_memory_example_4.cu

Memory Hierarchy cont.

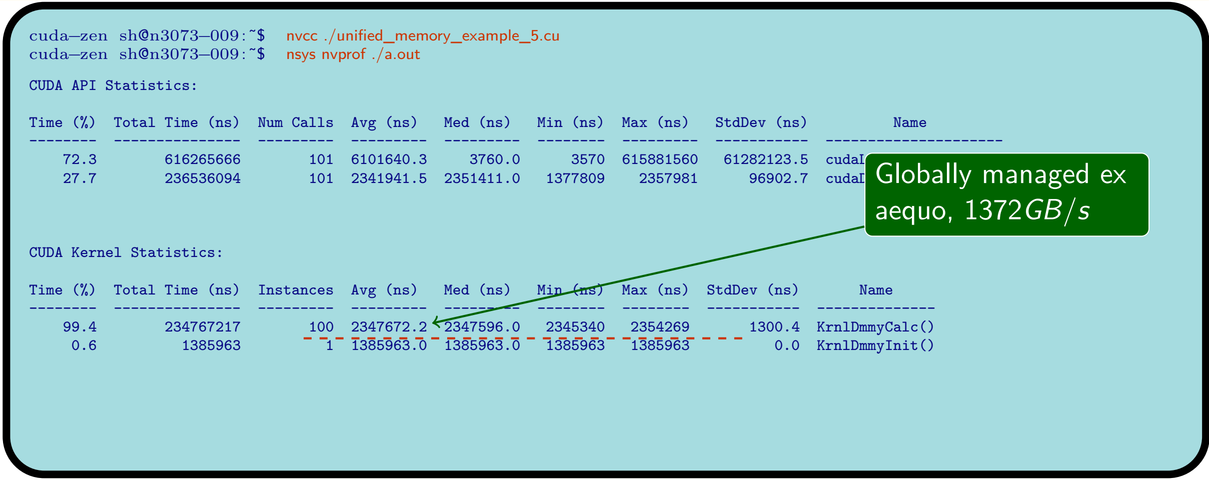
CUDA C-Programming Guide, Unified Memory Programming cont.

```
cuda-zen sh@n3073-009:~$  nvcc ./unified_memory_example_5.cu
cuda-zen sh@n3073-009:~$  nsys nvprof ./a.out

CUDA API Statistics:

Time (%)  Total Time (ns)  Num Calls  Avg (ns)   Med (ns)   Min (ns)  Max (ns)   StdDev (ns)          Name
--------  ---------------  ---------  ---------  ---------  --------  ---------  -----------  ---------------------
    72.3        616265666        101  6101640.3     3760.0      3570  615881560   61282123.5  cudaLaunchKernel
    27.7        236536094        101  2341941.5  2351411.0   1377809    2357981      96902.7  cudaDeviceSynchronize



CUDA Kernel Statistics:

Time (%)  Total Time (ns)  Instances  Avg (ns)   Med (ns)   Min (ns)  Max (ns)  StdDev (ns)       Name
--------  ---------------  ---------  ---------  ---------  --------  --------  -----------  --------------
    99.4        234767217        100  2347672.2  2347596.0   2345340   2354269       1300.4  KrnlDmmyCalc()
     0.6          1385963          1  1385963.0  1385963.0   1385963   1385963          0.0  KrnlDmmyInit()
```

→ https://tinyurl.com/cudafordummies/i/l2/unified_memory_example_5.cu

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.



```
cuda-zen sh@n3073-009:~$  nvcc ./unified_memory_example_5.cu
cuda-zen sh@n3073-009:~$  nsys nvprof ./a.out

CUDA API Statistics:

Time (%)  Total Time (ns)  Num Calls  Avg (ns)   Med (ns)   Min (ns)  Max (ns)   StdDev (ns)          Name
--------  ---------------  ---------  ---------  ---------  --------  ---------  -----------  ---------------------
    72.3        616265666        101  6101640.3     3760.0      3570  615881560   61282123.5  cudaL
    27.7        236536094        101  2341941.5  2351411.0   1377809    2357981      96902.7  cudaD


CUDA Kernel Statistics:

Time (%)  Total Time (ns)  Instances  Avg (ns)   Med (ns)   Min (ns)  Max (ns)  StdDev (ns)       Name
--------  ---------------  ---------  ---------  ---------  --------  --------  -----------  --------------
    99.4        234767217        100  2347672.2  2347596.0   2345340   2354269       1300.4  KrnlDmmyCalc()
     0.6          1385963          1  1385963.0  1385963.0   1385963   1385963          0.0  KrnlDmmyInit()
```

Globally managed ex aequo, 1372GB/s

→ https://tinyurl.com/cudafordummies/i/l2/unified_memory_example_5.cu

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

GPU Memory Oversubscription

```
#define DBLEARRAY24GB 3221225472

__global__ void KrnlDmmy(double *a, double *b, double *c)
{
    unsigned int i;
    i = (blockIdx.x * blockDim.x) + threadIdx.x;
    a[i] = (double) i;
    b[i] = (double) 3221225472 - i;
    c[i] = a[i] + b[i];
    return;
}

int main()
{
    ...
    // unified memory allocation a[], b[], c[]
    cudaMallocManaged(&a, DBLEARRAY24GB * sizeof(double));
    // kernel invocation
    KrnlDmmy <<< numBlocks, threadsPerBlock >>> (a, b, c);
    cudaDeviceSynchronize();
    check_array(c);
    cudaFree(c); cudaFree(b); cudaFree(a);
    return(0);
}
```

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

GPU Memory Oversubscription

```
#define DBLEARRAY24GB 3221225472

__global__ void KrnlDmmy(double *a, double *b, double *c)
{
    unsigned int i;
    i = (blockIdx.x * blockDim.x) + threadIdx.x;
    a[i] = (double) i;
    b[i] = (double) 3221225472 - i;
    c[i] = a[i] + b[i];
    return;
}

int main()
{
    ...
    // unified memory allocation a[], b[], c[]
    cudaMallocManaged(&a, DBLEARRAY24GB * sizeof(double));
    // kernel invocation
    KrnlDmmy <<< numBlocks, threadsPerBlock >>> (a, b, c);
    cudaDeviceSynchronize();
    check_array(c);
    cudaFree(c); cudaFree(b); cudaFree(a);
    return(0);
}
```

24 GB array(s)

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

GPU Memory Oversubscription

```
#define DBLEARRAY24GB 3221225472

__global__ void KrnlDmmy(double *a, double *b, double *c)
{
    unsigned int i;
    i = (blockIdx.x * blockDim.x) + threadIdx.x;
    a[i] = (double) i;
    b[i] = (double) 3221225472 - i;
    c[i] = a[i] + b[i];
    return;
}

int main()
{
    ...
    // unified memory allocation a[], b[], c[]
    cudaMallocManaged(&a, DBLEARRAY24GB * sizeof(double));
    // kernel invocation
    KrnlDmmy <<< numBlocks, threadsPerBlock >>> (a, b, c);
    cudaDeviceSynchronize();
    check_array(c);
    cudaFree(c); cudaFree(b); cudaFree(a);
    return(0);
}
```

Kernel calculation (40 GB onboard)

24 GB array(s)

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

GPU Memory Oversubscription

```
#define DBLEARRAY24GB 3221225472

__global__ void KrnlDmmy(double *a, double *b, double *c)
{
    unsigned int i;
    i = (blockIdx.x * blockDim.x) + threadIdx.x;
    a[i] = (double) i;
    b[i] = (double) 3221225472 - i;
    c[i] = a[i] + b[i];
    return;
}

int main()
{
    ...
    // unified memory allocation a[], b[], c[]
    cudaMallocManaged(&a, DBLEARRAY24GB * sizeof(double));
    // kernel invocation
    KrnlDmmy <<< numBlocks, threadsPerBlock >>> (a, b, c);
    cudaDeviceSynchronize();
    check_array(c);
    cudaFree(c); cudaFree(b); cudaFree(a);
    return(0);
}
```

Kernel calculation (40 GB onboard)

24 GB array(s)

Simple correctness check

Memory Hierarchy cont.

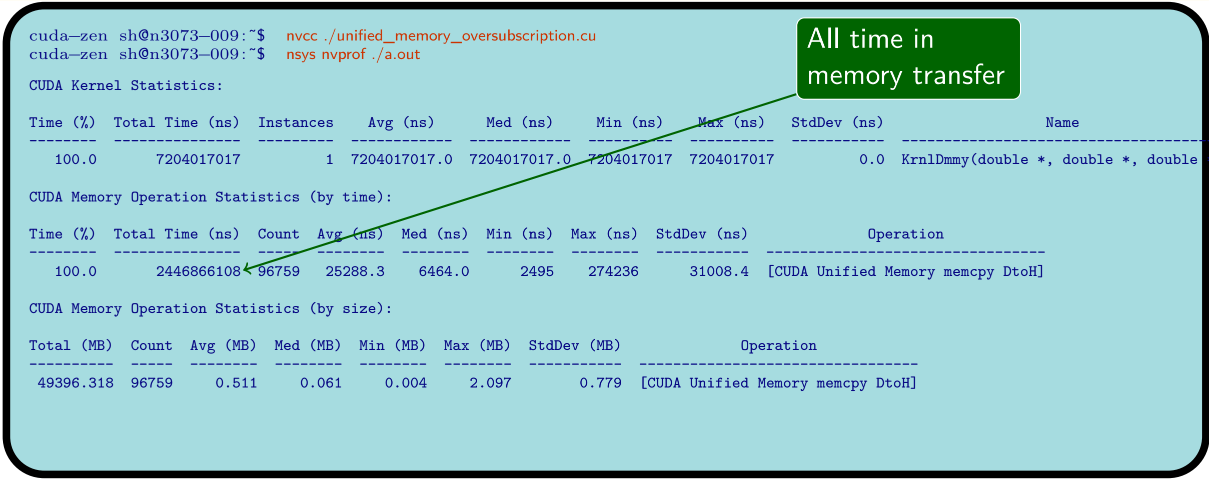
CUDA C-Programming Guide, Unified Memory Programming cont.

```
cuda-zen sh@n3073-009:~$  nvcc ./unified_memory_oversubscription.cu
cuda-zen sh@n3073-009:~$  nsys nvprof ./a.out

CUDA Kernel Statistics:

Time (%)  Total Time (ns)  Instances    Avg (ns)      Med (ns)      Min (ns)    Max (ns)    StdDev (ns)                  Name
--------  ---------------  ---------  ------------  ------------  ----------  ----------  -----------  ------------------------------------
   100.0       7204017017          1  7204017017.0  7204017017.0  7204017017  7204017017          0.0  KrnlDmmy(double *, double *, double

CUDA Memory Operation Statistics (by time):

Time (%)  Total Time (ns)  Count  Avg (ns)  Med (ns)  Min (ns)  Max (ns)  StdDev (ns)              Operation
--------  ---------------  -----  --------  --------  --------  --------  -----------  ---------------------------------
   100.0       2446866108  96759   25288.3    6464.0      2495    274236      31008.4  [CUDA Unified Memory memcpy DtoH]

CUDA Memory Operation Statistics (by size):

Total (MB)  Count  Avg (MB)  Med (MB)  Min (MB)  Max (MB)  StdDev (MB)              Operation
----------  -----  --------  --------  --------  --------  -----------  ---------------------------------
 49396.318  96759     0.511     0.061     0.004     2.097        0.779  [CUDA Unified Memory memcpy DtoH]
```

→ https://tinyurl.com/cudafordummies/i/l2/unified_memory_oversubscription.cu

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.



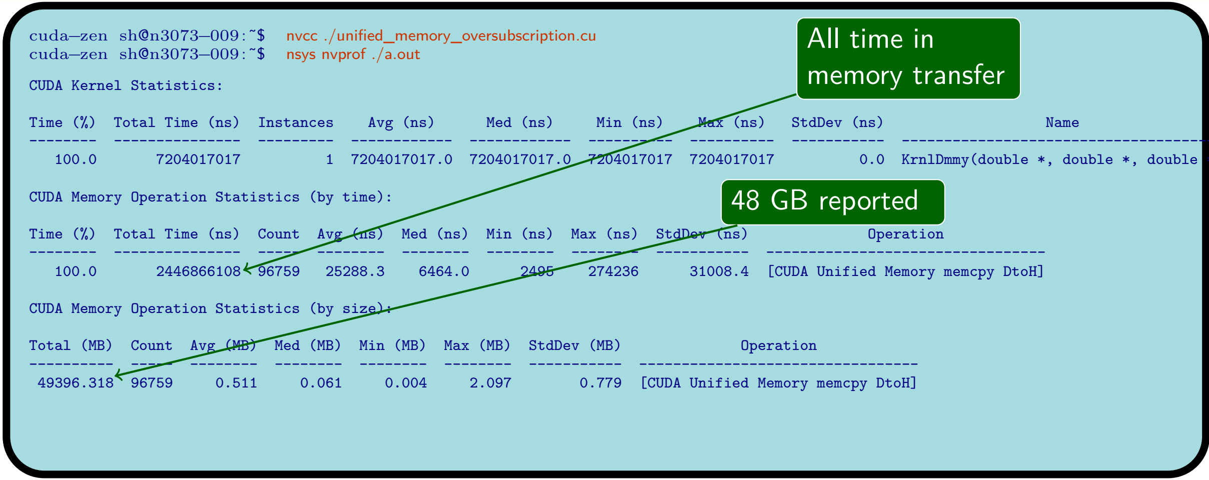
```
cuda−zen sh@n3073−009:~$  nvcc ./unified_memory_oversubscription.cu
cuda−zen sh@n3073−009:~$  nsys nvprof ./a.out

CUDA Kernel Statistics:

Time (%)  Total Time (ns)  Instances    Avg (ns)       Med (ns)      Min (ns)    Max (ns)    StdDev (ns)                Name
--------  ---------------  ---------  ------------  ------------  ----------  ----------  -----------  ----------------------------------
   100.0       7204017017          1  7204017017.0  7204017017.0  7204017017  7204017017          0.0  KrnlDmmy(double *, double *, double

CUDA Memory Operation Statistics (by time):

Time (%)  Total Time (ns)  Count   Avg (ns)  Med (ns)  Min (ns)  Max (ns)  StdDev (ns)            Operation
--------  ---------------  -----  --------  --------  --------  --------  -----------  ---------------------------------
   100.0       2446866108  96759   25288.3    6464.0      2495    274236      31008.4  [CUDA Unified Memory memcpy DtoH]

CUDA Memory Operation Statistics (by size):

Total (MB)  Count  Avg (MB)  Med (MB)  Min (MB)  Max (MB)  StdDev (MB)            Operation
----------  -----  --------  --------  --------  --------  -----------  ---------------------------------
 49396.318  96759     0.511     0.061     0.004     2.097        0.779  [CUDA Unified Memory memcpy DtoH]
```

All time in memory transfer

→ https://tinyurl.com/cudafordummies/i/l2/unified_memory_oversubscription.cu

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.



```
cuda-zen sh@n3073-009:~$  nvcc ./unified_memory_oversubscription.cu
cuda-zen sh@n3073-009:~$  nsys nvprof ./a.out

CUDA Kernel Statistics:

Time (%)  Total Time (ns)  Instances    Avg (ns)      Med (ns)      Min (ns)    Max (ns)    StdDev (ns)                  Name
--------  ---------------  ---------  ------------  ------------  ----------  ----------  -----------  ------------------------------------
   100.0       7204017017          1  7204017017.0  7204017017.0  7204017017  7204017017          0.0  KrnlDmmy(double *, double *, double

CUDA Memory Operation Statistics (by time):

Time (%)  Total Time (ns)  Count  Avg (ns)  Med (ns)  Min (ns)  Max (ns)  StdDev (ns)           Operation
--------  ---------------  -----  --------  --------  --------  --------  -----------  ---------------------------------
   100.0       2446866108  96759   25288.3    6464.0      2495    274236      31008.4  [CUDA Unified Memory memcpy DtoH]

CUDA Memory Operation Statistics (by size):

Total (MB)  Count  Avg (MB)  Med (MB)  Min (MB)  Max (MB)  StdDev (MB)           Operation
----------  -----  --------  --------  --------  --------  -----------  ---------------------------------
 49396.318  96759     0.511     0.061     0.004     2.097        0.779  [CUDA Unified Memory memcpy DtoH]
```

All time in memory transfer

48 GB reported

→ https://tinyurl.com/cudafordummies/i/l2/unified_memory_oversubscription.cu

Memory Hierarchy cont.

CUDA C-Programming Guide, Programming Model cont.

How did this work in the days before CUDA managed unified memory...

→ https://docs.nvidia.com/cuda/cuda-c-programming-guide

Memory Hierarchy cont.

CUDA C-Programming Guide, Device Memory

On pre-Pascal-class GPUs a strict distinction is made between host- and device-memory. Kernels operate out of device-memory, so the CUDA runtime provides functions to allocate, deallocate, and copy data into device-memory as well as transfer data between host-memory and device-memory. A typical sequence of operations is:

1. Declare and allocate arrays in host- and device-memory
2. Initialize host data
3. Transfer data from the host to the device
4. Execute one or more kernels
5. Transfer back results from the device to the host

→ https://docs.nvidia.com/cuda/cuda-c-programming-guide
→ https://devblogs.nvidia.com/easy-introduction-cuda-c-and-c

Memory Hierarchy cont.

CUDA C-Programming Guide, Device Memory cont.

```
#define ARRAYDIM 268435456
__global__ void KrnlDmmy(float *x, float *y, float *z)
{
    int i;
    i = (blockIdx.x * blockDim.x) + threadIdx.x;
    z[i] = x[i] + y[i];
    return;
}

int main()
{
    // host memory allocation, hb, hc analogous
    ha = (float *) malloc(ARRAYDIM * sizeof(float));
    // device memory allocation, db, dc analogous
    cudaMalloc(&da, ARRAYDIM * sizeof(float));
    // host to device memory transfer
    cudaMemcpy(da, ha, ARRAYDIM * sizeof(float), cudaMemcpyHostToDevice);
    // kernel invocation
    KrnlDmmy <<< numBlocks, threadsPerBlock >>> (da, db, dc);
    // device to host memory transfer
    cudaMemcpy(hc, dc, ARRAYDIM * sizeof(float), cudaMemcpyDeviceToHost);
    // free device/host memory
    cudaFree(da);
    free(ha);
}
```

Memory Hierarchy cont.

CUDA C-Programming Guide, Device Memory cont.

```
#define ARRAYDIM 268435456
__global__ void KrnlDmmy(float *x, float *y, float *z)
{
   int i;
   i = (blockIdx.x * blockDim.x) + threadIdx.x;
   z[i] = x[i] + y[i];
   return;
}

int main()
{
   // host memory allocation, hb, hc analogous
   ha = (float *) malloc(ARRAYDIM * sizeof(float));
   // device memory allocation, db, dc analogous
   cudaMalloc(&da, ARRAYDIM * sizeof(float));
   // host to device memory transfer
   cudaMemcpy(da, ha, ARRAYDIM * sizeof(float), cudaMemcpyHostToDevice);
   // kernel invocation
   KrnlDmmy <<< numBlocks, threadsPerBlock >>> (da, db, dc);
   // device to host memory transfer
   cudaMemcpy(hc, dc, ARRAYDIM * sizeof(float), cudaMemcpyDeviceToHost);
   // free device/host memory
   cudaFree(da);
   free(ha);
}
```

Kernel memory allocation & set up

Memory Hierarchy cont.

CUDA C-Programming Guide, Device Memory cont.

```
#define ARRAYDIM 268435456
__global__ void KrnlDmmy(float *x, float *y, float *z)
{
   int i;
   i = (blockIdx.x * blockDim.x) + threadIdx.x;
   z[i] = x[i] + y[i];
   return;
}

int main()
{
   // host memory allocation, hb, hc analogous
   ha = (float *) malloc(ARRAYDIM * sizeof(float));
   // device memory allocation, db, dc analogous
   cudaMalloc(&da, ARRAYDIM * sizeof(float));
   // host to device memory transfer
   cudaMemcpy(da, ha, ARRAYDIM * sizeof(float), cudaMemcpyHostToDevice);
   // kernel invocation
   KrnlDmmy <<< numBlocks, threadsPerBlock >>> (da, db, dc);
   // device to host memory transfer
   cudaMemcpy(hc, dc, ARRAYDIM * sizeof(float), cudaMemcpyDeviceToHost);
   // free device/host memory
   cudaFree(da);
   free(ha);
}
```

Kernel memory allocation & set up

Stnd kernel call

Memory Hierarchy cont.

CUDA C-Programming Guide, Device Memory cont.

```
#define ARRAYDIM 268435456
__global__ void KrnlDmmy(float *x, float *y, float *z)
{
    int i;
    i = (blockIdx.x * blockDim.x) + threadIdx.x;
    z[i] = x[i] + y[i];
    return;
}

int main()
{
    // host memory allocation, hb, hc analogous
    ha = (float *) malloc(ARRAYDIM * sizeof(float));
    // device memory allocation, db, dc analogous
    cudaMalloc(&da, ARRAYDIM * sizeof(float));
    // host to device memory transfer
    cudaMemcpy(da, ha, ARRAYDIM * sizeof(float), cudaMemcpyHostToDevice);
    // kernel invocation
    KrnlDmmy <<< numBlocks, threadsPerBlock >>> (da, db, dc);
    // device to host memory transfer
    cudaMemcpy(hc, dc, ARRAYDIM * sizeof(float), cudaMemcpyDeviceToHost);
    // free device/host memory
    cudaFree(da);
    free(ha);
}
```

Kernel memory allocation & set up

Stnd kernel call

Device memory back-transfer

Memory Hierarchy cont.

CUDA C-Programming Guide, Device Memory cont.

- Special forms cudaMallocPitch() and cudaMalloc3D() for 2D and 3D arrays
- Optimized for best performance when accessing via pointers
- Also good for device copies cudaMemcpy2D() and cudaMemcpy3D()
- Returned pitch (or stride) must be used to access array elements
- Optimally padded to meet alignment requirements
- Additional types of global memory, e.g.
  __device__ float *devPointer

→ https://docs.nvidia.com/cuda/cuda-c-programming-guide

Memory Hierarchy cont.

CUDA C-Programming Guide, Shared Memory

Matrix-Matrix Multiplication (simplest relationships):

$$\underbrace{\begin{pmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,N} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,N} \\ \vdots & \vdots & \vdots & \vdots \\ A_{k,1} & A_{k,2} & \cdots & A_{k,N} \\ \vdots & \vdots & \vdots & \vdots \\ A_{N,1} & A_{N,2} & \cdots & A_{N,N} \end{pmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix} B_{1,1} & B_{1,2} & \cdots & B_{1,N} \\ B_{2,1} & B_{2,2} & \cdots & B_{2,N} \\ \vdots & \vdots & \vdots & \vdots \\ B_{k,1} & B_{k,2} & \cdots & B_{k,N} \\ \vdots & \vdots & \vdots & \vdots \\ B_{N,1} & B_{N,2} & \cdots & B_{N,N} \end{pmatrix}}_{\mathbf{B}} = \underbrace{\begin{pmatrix} C_{1,1} & C_{1,2} & \cdots & C_{1,N} \\ C_{2,1} & C_{2,2} & \cdots & C_{2,N} \\ \vdots & \vdots & \vdots & \vdots \\ C_{k,1} & C_{k,2} & \cdots & C_{k,N} \\ \vdots & \vdots & \vdots & \vdots \\ C_{N,1} & C_{N,2} & \cdots & C_{N,N} \end{pmatrix}}_{\mathbf{C}}$$

→ https://docs.nvidia.com/cuda/cuda-c-programming-guide

Memory Hierarchy cont.

CUDA C-Programming Guide, Shared Memory cont.

Matrix Matrix Multiplication Version 1

```
#define N 4800
__global__ void KrnlMMM(float **A, float **B, float **C)
{
    int i, j, k;
    float tmpC;
    i = ((blockIdx.x * blockDim.x) + threadIdx.x);
    j = ((blockIdx.y * blockDim.y) + threadIdx.y);
    tmpC = (float) 0;
    for (k=0; k<N; k++) {
        tmpC += A[i][k] * B[k][j];
    }
    C[i][j] = tmpC;
    return;
}

int main()
{
    threadsPerBlock.x = 16;
    threadsPerBlock.y = 16;
    numBlocks.x = N / threadsPerBlock.x;
    numBlocks.y = N / threadsPerBlock.y;
    KrnlMMM <<< numBlocks, threadsPerBlock >>> (A, B, C);
    ...
}
```

Memory Hierarchy cont.

CUDA C-Programming Guide, Shared Memory cont.

Matrix Matrix Multiplication Version 1

```
#define N 4800
__global__ void KrnlMMM(float **A, float **B, float **C)
{
    int i, j, k;
    float tmpC;
    i = ((blockIdx.x * blockDim.x) + threadIdx.x);
    j = ((blockIdx.y * blockDim.y) + threadIdx.y);
    tmpC = (float) 0;
    for (k=0; k<N; k++) {
        tmpC += A[i][k] * B[k][j];
    }
    C[i][j] = tmpC;
    return;
}

int main()
{
    threadsPerBlock.x = 16;
    threadsPerBlock.y = 16;
    numBlocks.x = N / threadsPerBlock.x;
    numBlocks.y = N / threadsPerBlock.y;
    KrnlMMM <<< numBlocks, threadsPerBlock >>> (A, B, C);
    ...
}
```

Each thread computes its specific C[i][j]

Memory Hierarchy cont.

CUDA C-Programming Guide, Shared Memory cont.

Matrix Matrix Multiplication Version 1

Far-from-optimal memory access !

```
#define N 4800
__global__ void KrnlMMM(float **A, float **B, float **C)
{
    int i, j, k;
    float tmpC;
    i = ((blockIdx.x * blockDim.x) + threadIdx.x);
    j = ((blockIdx.y * blockDim.y) + threadIdx.y);
    tmpC = (float) 0;
    for (k=0; k<N; k++) {
        tmpC += A[i][k] * B[k][j];
    }
    C[i][j] = tmpC;
    return;
}

int main()
{
    threadsPerBlock.x = 16;
    threadsPerBlock.y = 16;
    numBlocks.x = N / threadsPerBlock.x;
    numBlocks.y = N / threadsPerBlock.y;
    KrnlMMM <<< numBlocks, threadsPerBlock >>> (A, B, C);
    ...
}
```

Each thread computes its specific C[i][j]

Memory Hierarchy cont.

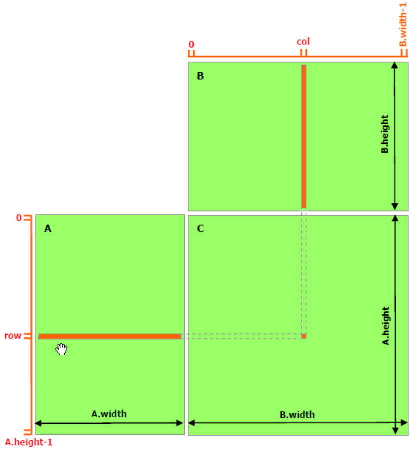
CUDA C-Programming Guide, Shared Memory cont.

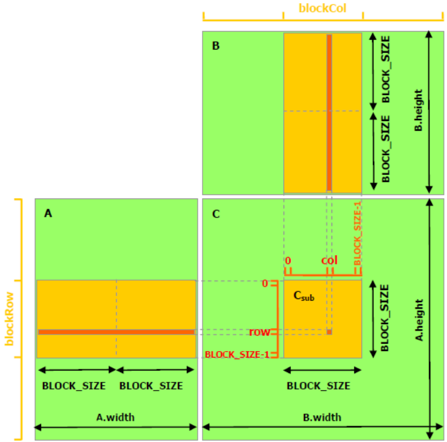
```
cuda—zen sh@n3073—009:˜$  nvcc ./mmm_example_2.cu
cuda—zen sh@n3073—009:˜$  nsys nvprof ./a.out

CUDA Kernel Statistics:

Time (%)  Total Time (ns)  Instances   Avg (ns)     Med (ns)     Min (ns)   Max (ns)   StdDev (ns)                   Name
--------  ---------------  ---------  -----------  -----------  ---------  ---------  -----------  -------------------------------------
   100.0        539762592          1  539762592.0  539762592.0  539762592  539762592          0.0  KrnlMMM(float **, float **, float **)

CUDA Memory Operation Statistics (by time):

Time (%)  Total Time (ns)  Count  Avg (ns)  Med (ns)  Min (ns)  Max (ns)  StdDev (ns)            Operation
--------  ---------------  -----  --------  --------  --------  --------  -----------  ---------------------------------
    67.8         31663299   5629    5625.0    4191.0      3071    200253       5744.5  [CUDA Unified Memory memcpy HtoD]
    32.2         15010965   2388    6286.0    2528.0      1727    197852      10260.2  [CUDA Unified Memory memcpy DtoH]

CUDA Memory Operation Statistics (by size):

Total (MB)  Count  Avg (MB)  Med (MB)  Min (MB)  Max (MB)  StdDev (MB)            Operation
----------  -----  --------  --------  --------  --------  -----------  ---------------------------------
   283.116   2388     0.119     0.025     0.004     1.036        0.249  [CUDA Unified Memory memcpy DtoH]
   283.116   5629     0.050     0.025     0.004     0.987        0.097  [CUDA Unified Memory memcpy HtoD]
```

→ https://tinyurl.com/cudafordummies/i/l2/mmm_example_2.cu

Memory Hierarchy cont.

CUDA C-Programming Guide, Shared Memory cont.





→ https://docs.nvidia.com/cuda/cuda-c-programming-guide

Memory Hierarchy cont.

CUDA C-Programming Guide, Shared Memory cont.

Matrix Matrix Multiplication Version 2

```
#define N 4800
#define BS 16
__global__ void KrnlMMM(float **A, float **B, float **C)
{
    int i, j, k, Blckk;
    __shared__ float BlckA[BS][BS], BlckB[BS][BS], BlckC[BS][BS];
    float tmpC;
    i = threadIdx.y;
    j = threadIdx.x;
    BlckC[i][j] = (float) 0;
    for (Blckk = 0; Blckk < (N/BS); Blckk++) {
        BlckA[i][j] = A[(blockIdx.y*BS)+i][(Blckk*BS)+j];
        BlckB[i][j] = B[(Blckk*BS)+i][(blockIdx.x*BS)+j];
        __syncthreads();
        tmpC = (float) 0;
        for (k = 0; k<BS; k++) {
            tmpC += BlckA[i][k] * BlckB[k][j];
        }
        BlckC[i][j] += tmpC;
        __syncthreads();
    }
    C[(blockIdx.y*BS+i][(blockIdx.x*BS)+j] = BlckC[i][j];
    __syncthreads();
    return;
}
```

Memory Hierarchy cont.

CUDA C-Programming Guide, Shared Memory cont.

Matrix Matrix Multiplication Version 2

Declaration of shared arrays

```
#define N 4800
#define BS 16
__global__ void KrnlMMM(float **A, float **B, float **C)
{
   int i, j, k, Blckk;
   __shared__ float BlckA[BS][BS], BlckB[BS][BS], BlckC[BS][BS];
   float tmpC;
   i = threadIdx.y;
   j = threadIdx.x;
   BlckC[i][j] = (float) 0;
   for (Blckk = 0; Blckk < (N/BS); Blckk++) {
      BlckA[i][j] = A[(blockIdx.y*BS)+i][(Blckk*BS)+j];
      BlckB[i][j] = B[(Blckk*BS)+i][(blockIdx.x*BS)+j];
      __syncthreads();
      tmpC = (float) 0;
      for (k = 0; k<BS; k++) {
         tmpC += BlckA[i][k] * BlckB[k][j];
      }
      BlckC[i][j] += tmpC;
      __syncthreads();
   }
   C[(blockIdx.y*BS+i][(blockIdx.x*BS)+j] = BlckC[i][j];
   __syncthreads();
   return;
}
```

Memory Hierarchy cont.

CUDA C-Programming Guide, Shared Memory cont.

Matrix Matrix Multiplication Version 2

Declaration of shared arrays

Initialization by entire threadblock

```
#define N 4800
#define BS 16
__global__ void KrnlMMM(float **A, float **B, float **C)
{
   int i, j, k, Blckk;
   __shared__ float BlckA[BS][BS], BlckB[BS][BS], BlckC[BS][BS];
   float tmpC;
   i = threadIdx.y;
   j = threadIdx.x;
   BlckC[i][j] = (float) 0;
   for (Blckk = 0; Blckk < (N/BS); Blckk++) {
      BlckA[i][j] = A[(blockIdx.y*BS)+i][(Blckk*BS)+j];
      BlckB[i][j] = B[(Blckk*BS)+i][(blockIdx.x*BS)+j];
      __syncthreads();
      tmpC = (float) 0;
      for (k = 0; k<BS; k++) {
         tmpC += BlckA[i][k] * BlckB[k][j];
      }
      BlckC[i][j] += tmpC;
      __syncthreads();
   }
   C[(blockIdx.y*BS+i][(blockIdx.x*BS)+j] = BlckC[i][j];
   __syncthreads();
   return;
}
```

Memory Hierarchy cont.

CUDA C-Programming Guide, Shared Memory cont.

Matrix Matrix Multiplication Version 2

```
#define N 4800
#define BS 16
__global__ void KrnlMMM(float **A, float **B, float **C)
{
   int i, j, k, Blckk;
   __shared__ float BlckA[BS][BS], BlckB[BS][BS], BlckC[BS][BS];
   float tmpC;
   i = threadIdx.y;
   j = threadIdx.x;
   BlckC[i][j] = (float) 0;
   for (Blckk = 0; Blckk < (N/BS); Blckk++) {
      BlckA[i][j] = A[(blockIdx.y*BS)+i][(Blckk*BS)+j];
      BlckB[i][j] = B[(Blckk*BS)+i][(blockIdx.x*BS)+j];
      __syncthreads();
      tmpC = (float) 0;
      for (k = 0; k<BS; k++) {
         tmpC += BlckA[i][k] * BlckB[k][j];
      }
      BlckC[i][j] += tmpC;
      __syncthreads();
   }
   C[(blockIdx.y*BS+i][(blockIdx.x*BS)+j] = BlckC[i][j];
   __syncthreads();
   return;
}
```

Declaration of shared arrays

Initialization by entire threadblock

Loop combining A/B blocks in a dot-product like fashion

Memory Hierarchy cont.

CUDA C-Programming Guide, Shared Memory cont.

Matrix Matrix Multiplication Version 2

Declaration of shared arrays

Initialization by entire threadblock

Copy into shared memory arrays

```
#define N 4800
#define BS 16
__global__ void KrnlMMM(float **A, float **B, float **C)
{
   int i, j, k, Blckk;
   __shared__ float BlckA[BS][BS], BlckB[BS][BS], BlckC[BS][BS];
   float tmpC;
   i = threadIdx.y;
   j = threadIdx.x;
   BlckC[i][j] = (float) 0;
   for (Blckk = 0; Blckk < (N/BS); Blckk++) {
      BlckA[i][j] = A[(blockIdx.y*BS)+i][(Blckk*BS)+j];
      BlckB[i][j] = B[(Blckk*BS)+i][(blockIdx.x*BS)+j];
      __syncthreads();
      tmpC = (float) 0;
      for (k = 0; k<BS; k++) {
         tmpC += BlckA[i][k] * BlckB[k][j];
      }
      BlckC[i][j] += tmpC;
      __syncthreads();
   }
   C[(blockIdx.y*BS+i][(blockIdx.x*BS)+j] = BlckC[i][j];
   __syncthreads();
   return;
}
```

Loop combining A/B blocks in a dot-product like fashion

Memory Hierarchy cont.

CUDA C-Programming Guide, Shared Memory cont.

Matrix Matrix Multiplication Version 2

Declaration of shared arrays

Initialization by entire threadblock

Copy into shared memory arrays

```
#define N 4800
#define BS 16
__global__ void KrnlMMM(float **A, float **B, float **C)
{
    int i, j, k, Blckk;
    __shared__ float BlckA[BS][BS], BlckB[BS][BS], BlckC[BS][BS];
    float tmpC;
    i = threadIdx.y;
    j = threadIdx.x;
    BlckC[i][j] = (float) 0;
    for (Blckk = 0; Blckk < (N/BS); Blckk++) {
        BlckA[i][j] = A[(blockIdx.y*BS)+i][(Blckk*BS)+j];
        BlckB[i][j] = B[(Blckk*BS)+i][(blockIdx.x*BS)+j];
        __syncthreads();
        tmpC = (float) 0;
        for (k = 0; k<BS; k++) {
            tmpC += BlckA[i][k] * BlckB[k][j];
        }
        BlckC[i][j] += tmpC;
        __syncthreads();
    }
    C[(blockIdx.y*BS+i][(blockIdx.x*BS)+j] = BlckC[i][j];
    __syncthreads();
    return;
}
```

Loop combining A/B blocks in a dot-product like fashion

Thread-wise update of BlckC

Memory Hierarchy cont.

CUDA C-Programming Guide, Shared Memory cont.

Matrix Matrix Multiplication Version 2

```
#define N 4800
#define BS 16
__global__ void KrnlMMM(float **A, float **B, float **C)
{
   int i, j, k, Blckk;
   __shared__ float BlckA[BS][BS], BlckB[BS][BS], BlckC[BS][BS];
   float tmpC;
   i = threadIdx.y;
   j = threadIdx.x;
   BlckC[i][j] = (float) 0;
   for (Blckk = 0; Blckk < (N/BS); Blckk++) {
      BlckA[i][j] = A[(blockIdx.y*BS)+i][(Blckk*BS)+j];
      BlckB[i][j] = B[(Blckk*BS)+i][(blockIdx.x*BS)+j];
      __syncthreads();
      tmpC = (float) 0;
      for (k = 0; k<BS; k++) {
         tmpC += BlckA[i][k] * BlckB[k][j];
      }
      BlckC[i][j] += tmpC;
      __syncthreads();
   }
   C[(blockIdx.y*BS+i][(blockIdx.x*BS)+j] = BlckC[i][j];
   __syncthreads();
   return;
}
```

Declaration of shared arrays

Initialization by entire threadblock

Copy into shared memory arrays

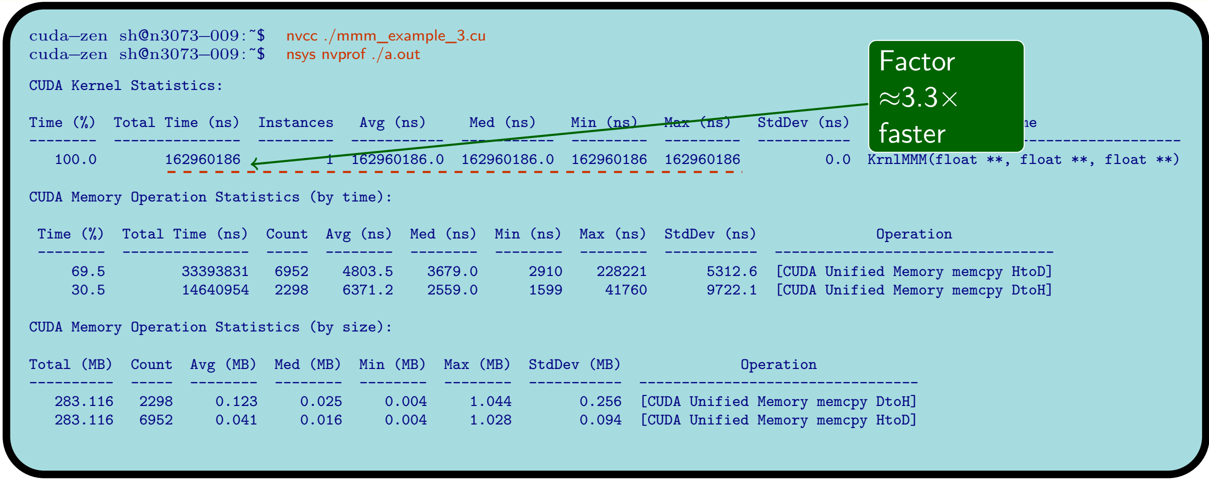
Loop combining A/B blocks in a dot-product like fashion

Thread-wise update of BlckC

Back-copy from shared to global memory

Memory Hierarchy cont.

CUDA C-Programming Guide, Shared Memory cont.



```
cuda-zen sh@n3073-009:~$  nvcc ./mmm_example_3.cu
cuda-zen sh@n3073-009:~$  nsys nvprof ./a.out

CUDA Kernel Statistics:

Time (%)  Total Time (ns)  Instances   Avg (ns)      Med (ns)     Min (ns)   Max (ns)   StdDev (ns)                          ne
--------  ---------------  ---------  -----------  -----------  ---------  ---------  -----------  ------------------------------------
   100.0        162960186          1  162960186.0  162960186.0  162960186  162960186          0.0  KrnlMMM(float **, float **, float **)

CUDA Memory Operation Statistics (by time):

 Time (%)  Total Time (ns)  Count  Avg (ns)  Med (ns)  Min (ns)  Max (ns)  StdDev (ns)              Operation
 --------  ---------------  -----  --------  --------  --------  --------  -----------  ---------------------------------
     69.5         33393831   6952    4803.5    3679.0      2910    228221       5312.6  [CUDA Unified Memory memcpy HtoD]
     30.5         14640954   2298    6371.2    2559.0      1599     41760       9722.1  [CUDA Unified Memory memcpy DtoH]

CUDA Memory Operation Statistics (by size):

Total (MB)  Count  Avg (MB)  Med (MB)  Min (MB)  Max (MB)  StdDev (MB)              Operation
----------  -----  --------  --------  --------  --------  -----------  ---------------------------------
   283.116   2298     0.123     0.025     0.004     1.044        0.256  [CUDA Unified Memory memcpy DtoH]
   283.116   6952     0.041     0.016     0.004     1.028        0.094  [CUDA Unified Memory memcpy HtoD]
```

Factor ≈3.3× faster

→ https://tinyurl.com/cudafordummies/i/l2/mmm_example_3.cu

Memory Hierarchy cont.

CUDA C-Programming Guide, Local Memory, Registers

- Local memory for certain automatic variables
- Access to local memory is similar to global memory, i.e. high latency and low bandwidth
- Organized such that consecutive 32-bit words are accessed by consecutive thread IDs
- Analyzable with Nsight
- Fastest memory is registers, L1, for small, statically indexed arrays, e.g. A[16]

→ https://docs.nvidia.com/cuda/cuda-c-programming-guide

→ https://stackoverflow.com/questions/10297067/in-a-cuda-kernel-how-do-i-store-an-array-in-local-thread-memory

Take Home Messages

- Unified memory — a big improvement

Take Home Messages

- Unified memory — a big improvement
- Single address space also facilitates easier handling of 2D/3D arrays on the GPU

Take Home Messages

- Unified memory — a big improvement
- Single address space also facilitates easier handling of 2D/3D arrays on the GPU
- Straightforward profiling with nsys nvprof

Take Home Messages

- Unified memory — a big improvement
- Single address space also facilitates easier handling of 2D/3D arrays on the GPU
- Straightforward profiling with nsys nvprof
- Page migration engine facilitates seamless data transfer for array sizes exceeding largely the amount of RAM available on the GPU

Take Home Messages

- Unified memory — a big improvement
- Single address space also facilitates easier handling of 2D/3D arrays on the GPU
- Straightforward profiling with nsys nvprof
- Page migration engine facilitates seamless data transfer for array sizes exceeding largely the amount of RAM available on the GPU
- Using shared memory greatly improves kernel performance