Memory Hierarchies in CPU/GPU Architectures

Siegfried Höfinger

ASC Research Center, TU Wien

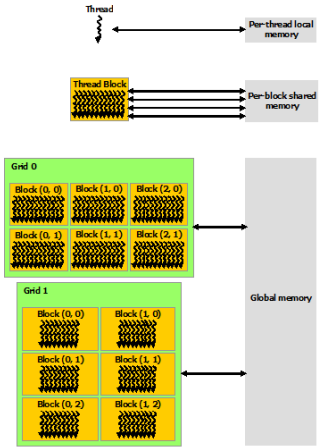
October 20, 2025

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

Outline

Memory Hierarchy

CUDA C-Programming Guide, Programming Model

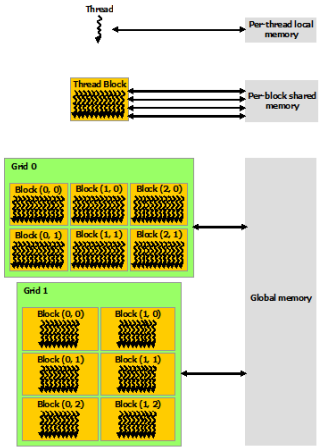


- CUDA threads have access to several memory spaces

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

Memory Hierarchy

CUDA C-Programming Guide, Programming Model

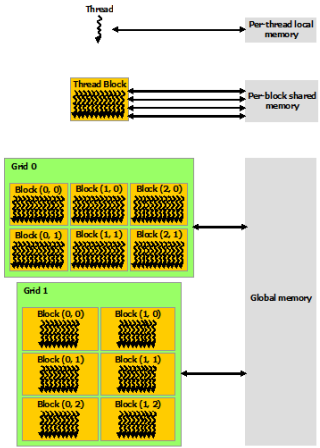


- CUDA threads have access to several memory spaces
- Private local memory for each individual thread (small but very fast)

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

Memory Hierarchy

CUDA C-Programming Guide, Programming Model

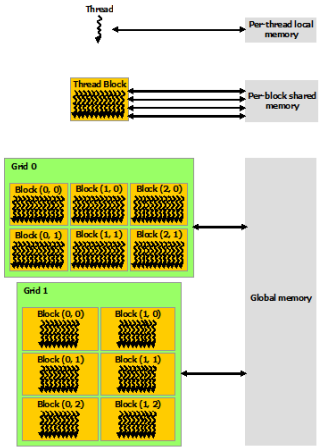


- CUDA threads have access to several memory spaces
- Private local memory for each individual thread (small but very fast)
- Shared memory for all threads within a thread block (pretty fast)

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

Memory Hierarchy

CUDA C-Programming Guide, Programming Model

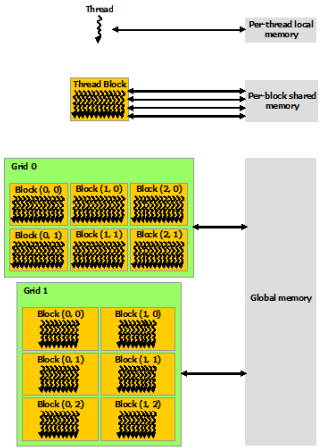


- CUDA threads have access to several memory spaces
- Private local memory for each individual thread (small but very fast)
- Shared memory for all threads within a thread block (pretty fast)
- Global memory accessible by all threads (slow)

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

Memory Hierarchy

CUDA C-Programming Guide, Programming Model

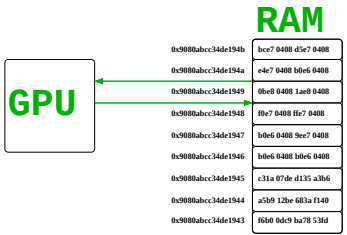


- CUDA threads have access to several memory spaces
- Private local memory for each individual thread (small but very fast)
- Shared memory for all threads within a thread block (pretty fast)
- Global memory accessible by all threads (slow)
- Special ROMs, constant and texture memory

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

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming



Unified Memory:

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

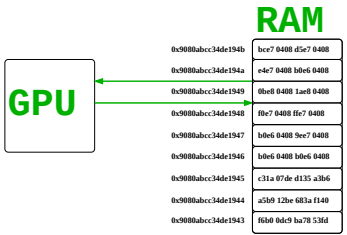
Separate Device/Host Memory:

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



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 is getting irrelevant 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 is getting irrelevant 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 is getting irrelevant 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 is getting irrelevant 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

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);
}
```

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);
}
```

Multiple Thread Blocks Matrix Addition

Standard malloc-like allocation on the host

host pointers directly usable in kernel code

```
__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);
}
```

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

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

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();
    ...
}
```

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();
    ...
}
```

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

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

ensure proper kernel completion

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();
     ...
}
```

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

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@n3073-004:~$  nvcc ./unified_memory_example_1.cu
cuda-zen sh@n3073-004:~$  ./a.out
cuda-zen sh@n3073-004:~$  nsys nvprof ./a.out

[4/7] Executing 'cudaapisum' stats report

 Time (%)  Total Time (ns)  Num Calls   Avg (ns)     Med (ns)    Min (ns)   Max (ns)   StdDev (ns)          Name
 --------  ---------------  ---------  -----------  -----------  ---------  ---------  -----------  ----------------------
     39.5        238225943          1  238225943.0  238225943.0  238225943  238225943          0.0  cudaDeviceSynchronize
     38.9        234132874          3   78044291.3      43351.0      22130  234067393  135119970.0  cudaMallocManaged
     21.5        129631998          3   43210666.0   37602092.0   37551931   54477975    9757808.1  cudaFree
      0.1           417977          1     417977.0     417977.0     417977     417977          0.0  cudaLaunchKernel

[5/7] Executing 'gpukernsum' stats report

Time (%)  Total Time (ns)  Instances  Avg (ns)    Med/Min/Max StdDev (ns)    GridXYZ     BlockXYZ              Name
--------  ---------------  ---------  ----------  ----------- -----------  -----------  ---------  ----------------------------------
   100.0        238223963          1 238223963.0  238223963.0          0.0  1048576 1 1   256  1  1  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@n3073-004:~$ nvcc ./unified_memory_example_1.cu
cuda-zen sh@n3073-004:~$ ./a.out
cuda-zen sh@n3073-004:~$ nsys nvprof ./a.out

[4/7] Executing 'cudaapisum' stats report

 Time (%)  Total Time (ns)  Num Calls   Avg (ns)     Med (ns)    Min (ns)   Max (ns)   StdDev (ns)          Name
 --------  ---------------  ---------  -----------  -----------  ---------  ---------  -----------  ----------------------
     39.5        238225943          1  238225943.0  238225943.0  238225943  238225943          0.0  cudaDeviceSynchronize
     38.9        234132874          3   78044291.3      43351.0      22130  234067393  135119970.0  cudaMallocManaged
     21.5        129631998          3   43210666.0   37602092.0   37551931   54477975    9757808.1  cudaFree
      0.1           417977          1     417977.0     417977.0     417977     417977          0.0  cudaLaunchKernel

[5/7] Executing 'gpukernsum' stats report

Time (%)  Total Time (ns)  Instances  Avg (ns)    Med/Min/Max StdDev (ns)    GridXYZ     BlockXYZ              Name
--------  ---------------  ---------  ----------  ----------- -----------  ----------  ----------  ----------------------------------
   100.0        238223963          1  238223963.0  238223963.0          0.0  1048576 1 1   256  1  1  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 on the command line, however, suppressed information about unified memory, especially numbers of page faults etc.

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

- Straightforward profiling with nsys nvprof on the command line, however, suppressed information about unified memory, 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 on the command line, however, suppressed information about unified memory, 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 on the command line, however, suppressed information about unified memory, 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 on the command line, however, suppressed information about unified memory, 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@n3073–004:~$ nvcc ./unified_memory_example_2.cu
cuda–zen sh@n3073–004:~$ nsys nvprof ./a.out

[4/7] Executing 'cudaapisum' stats report

 Time (%)  Total Time (ns)  Num Calls   Avg (ns)     Med (ns)    Min (ns)  Max (ns)   StdDev (ns)           Name
 --------  ---------------  ---------  -----------  -----------  --------  ---------  -----------  ----------------------
     43.0        262687803          2  131343901.5  131343901.5  80061098  182626705   72524836.2  cudaDeviceSynchronize
     36.3        221685218          3   73895072.7      37931.0     23340  221623947  127936958.3  cudaMallocManaged
     19.5        118880186          3   39626728.7   38862579.0  37921894   42095713    2189322.9  cudaFree
      1.2          7069296          2    3534648.0    3534648.0     26820    7042476    4960817.9  cudaLaunchKernel

[5/7] Executing 'gpukernsum' stats report

 Time (%)  Total Time (ns)  Instances   Avg (ns)     Med/Min/Max   StdDev (ns)   GridXYZ   BlockXYZ                    Name
 --------  ---------------  ---------  -----------  -----------  -----------  ----------  --------  ----------------------------------------
     69.5        182625001          1  182625001.0  182625001.0          0.0  1048576 1 1  256  1 1  KrnlDmmyInit(float *, float *, float *)
     30.5         80056695          1   80056695.0   80056695.0          0.0  1048576 1 1  256  1 1  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@n3073–004:~$  nvcc ./unified_memory_example_2.cu
cuda–zen sh@n3073–004:~$  nsys nvprof ./a.out

[4/7] Executing 'cudaapisum' stats report

 Time (%)  Total Time (ns)  Num Calls   Avg (ns)     Med (ns)    Min (ns)  Max (ns)   StdDev (ns)          Name
 --------  ---------------  ---------  -----------  -----------  --------  ---------  -----------  ----------------------
     43.0        262687803          2  131343901.5  131343901.5  80061098  182626705   72524836.2  cudaDeviceSynchro
     36.3        221685218          3   73895072.7      37931.0     23340  221623947  127936958.3  cudaMallocManaged
     19.5        118880186          3   39626728.7   38862579.0  37921894   42095713    2189322.9  cudaFree
      1.2          7069296          2    3534648.0    3534648.0     26820    7042476    4960817.9  cudaLaunchKernel

[5/7] Executing 'gpukernsum' stats report

 Time (%)  Total Time (ns)  Instances   Avg (ns)     Med/Min/Max   StdDev (ns)   GridXYZ   BlockXYZ                  Name
 --------  ---------------  ---------  -----------  -----------  -----------  ----------  --------  ----------------------------------------
     69.5        182625001          1  182625001.0  182625001.0          0.0  1048576 1 1  256 1 1  KrnlDmmyInit(float *, float *, float *)
     30.5         80056695          1   80056695.0   80056695.0          0.0  1048576 1 1  256 1 1  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@n3073–004:~$ nvcc ./unified_memory_example_2.cu
cuda–zen sh@n3073–004:~$ nsys nvprof ./a.out

[4/7] Executing 'cudaapisum' stats report

 Time (%)  Total Tim                                          Min (ns)  Max (ns)   StdDev (ns)           Name
 --------  ---------                                          --------  ---------  -----------  ---------------------
     43.0       262                                          80061098  182626705   72524836.2  cudaDeviceSynchro
     36.3       221                                             23340  221623947  127936958.3  cudaMallocManaged
     19.5       118                                          37921894   42095713    2189322.9  cudaFree
      1.2         7                                             26820    7042476    4960817.9  cudaLaunchKernel

[5/7] Executing 'gpu

 Time (%)  Total Time (ns)  Instances   Avg (ns)     Med/Min/Max   StdDev (ns)   GridXYZ   BlockXYZ                 Name
 --------  ---------------  ---------  ----------  -------------  -----------  ----------  -------  ----------------------------------
     69.5        182625001          1  182625001.0  182625001.0          0.0  1048576 1 1  256  1 1  KrnlDmmyInit(float *, float *, float *)
     30.5         80056695          1   80056695.0   80056695.0          0.0  1048576 1 1  256  1 1  KrnlDmmyCalc(float *, float *, float *)
```

Still very low effective memory bandwith,

$$\frac{\frac{3\times 268435456\times 4}{10^9}}{80056695\times 10^{-9}} = 40.24 GB/s$$

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@n3073–004:~$ nvcc ./unified_memory_example_3.cu
cuda–zen sh@n3073–004:~$ nsys nvprof ./a.out

[4/7] Executing 'cudaapisum' stats report

 Time (%)  Total Time (ns)  Num Calls   Avg (ns)    Med (ns)   Min (ns)  Max (ns)   StdDev (ns)           Name
 --------  ---------------  ---------  ----------  ----------  --------  ---------  -----------  ----------------------
     59.4        502130428        101   4971588.4   2366751.0   2362901  180230574   19454564.9  cudaDeviceSynchronize
     26.9        227483529          3  75827843.0     40981.0     23201  227419347  131282093.8  cudaMallocManaged
     13.5        114443761          3  38147920.3  38122817.0  38117096   38203848      48519.2  cudaFree
      0.1           893207        101      8843.6      3850.0      3650     457938      45211.6  cudaLaunchKernel



[5/7] Executing 'gpukernsum' stats report

 Time (%) Total Time (ns) Instances  Avg (ns)     Med (ns)    Min (ns)  Max (ns)  StdDev (ns)   GridXYZ   BlockXYZ                        Name
 -------- --------------- --------- ----------- ----------- --------- --------- ----------- ---------- ---------- ----------------------------
     64.1       321527279       100  3215272.8   2363150.5   2359535  85412082   8302785.2 1048576 1 1  256 1 1 KrnlDmmyCalc(float *, floa
     35.9       180201758         1 180201758.0 180201758.0 180201758 180201758         0.0 1048576 1 1  256 1 1 KrnlDmmyInit(float *, floa
```

→ 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@n3073–004:~$ nvcc ./unified_memory_example_3.cu
cuda–zen sh@n3073–004:~$ nsys nvprof ./a.out

[4/7] Executing 'cudaapisum' stats report

 Time (%)  Total Time (ns)  Num Calls   Avg (ns)    Med (ns)   Min (ns)   Max (ns)   StdDev (ns)
 --------  ---------------  ---------  ----------  ----------  --------  ---------  -----------
     59.4        502130428        101   4971588.4   2366751.0   2362901  180230574   19454564.9  cu
     26.9        227483529          3  75827843.0     40981.0     23201  227419347  131282093.8  cu
     13.5        114443761          3  38147920.3  38122817.0  38117096   38203848      48519.2  cu
      0.1           893207        101      8843.6      3850.0      3650     457938      45211.6  cu



[5/7] Executing 'gpukernsum' stats report

 Time (%) Total Time (ns) Instances  Avg (ns)     Med (ns)    Min (ns)   Max (ns)  StdDev (ns)   GridXYZ   BlockXYZ                    Name
 -------- --------------- --------- ----------- ----------- ---------- ---------- ----------- ---------- ---------- ----------------------------
     64.1       321527279       100  3215272.8   2363150.5    2359535   85412082   8302785.2 1048576 1 1  256 1 1 KrnlDmmyCalc(float *, floa
     35.9       180201758         1 180201758.0 180201758.0 180201758  180201758         0.0 1048576 1 1  256 1 1 KrnlDmmyInit(float *, floa
```

100× greatly improves effective memory bandwidth, $1002 GB/s$ 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@n3073–004:~$ nvcc ./unified_memory_example_4.cu
cuda–zen sh@n3073–004:~$ nsys nvprof ./a.out

[4/7] Executing 'cudaapisum' stats report

 Time (%)  Total Time (ns)  Num Calls   Avg (ns)    Med (ns)   Min (ns)  Max (ns)   StdDev (ns)           Name
 --------  ---------------  ---------  ----------  ----------  --------  ---------  -----------  ----------------------
     55.1        422794342        101   4186082.6   2430620.0   2362338  181284947   17798470.6  cudaDeviceSynchronize
     29.2        224184902          3  74728300.7     38891.0     23080  224122931  129379545.3  cudaMallocManaged
     14.8        113834964          3  37944988.0  37947335.0  37937445   37950184       6685.9  cudaFree
      0.8          6191521          3   2063840.3   1278401.0    960186    3952934    1643721.8  cudaMemPrefetchAsync
      0.1           891612        101      8827.8      3800.0      3600     474048      46837.1  cudaLaunchKernel



[5/7] Executing 'gpukernsum' stats report

 Time (%)  Total Time (ns)  Instances  Avg (ns)    Med (ns)   Min (ns)  Max (ns)  StdDev (ns)     GridXYZ     BlockXYZ                 Name
 --------  ---------------  ---------  ----------  ----------  --------  --------  -----------  -----------  -----------  ----------------------
     57.0        240257784        100   2402577.8   2426684.5   2358605   2455405      32694.9  1048576 1 1  256 1 1  KrnlDmmyCalc(float *, flo
     43.0        181291027          1  181291027.0  181291027.0  181291027  181291027          0.0  1048576 1 1  256 1 1  KrnlDmmyInit(float *, flo
```

→ 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@n3073–004:~$  nvcc ./unified_memory_example_4.cu
cuda–zen sh@n3073–004:~$  nsys nvprof ./a.out

[4/7] Executing 'cudaapisum' stats report

 Time (%)  Total Time (ns)  Num Calls   Avg (ns)    Med (ns)   Min (ns)  Max (ns)   StdDev (ns)          Name
 --------  ---------------  ---------  ----------  ----------  --------  ---------  -----------  ----------------------
     55.1        422794342        101   4186082.6   2430620.0   2362338  181284947   17798470.6  cu
     29.2        224184902          3  74728300.7     38891.0     23080  224122931  129379545.3  cu
     14.8        113834964          3  37944988.0  37947335.0  37937445   37950184       6685.9  cu
      0.8          6191521          3   2063840.3   1278401.0    960186    3952934    1643721.8  cu
      0.1           891612        101      8827.8      3800.0      3600     474048      46837.1  cu

[5/7] Executing 'gpukernsum' stats report

 Time (%)  Total Time (ns)  Instances  Avg (ns)     Med (ns)    Min (ns)  Max (ns)  StdDev (ns)     GridXYZ     BlockXYZ          Name
 --------  ---------------  ---------  ----------  ----------  ---------  --------  -----------  -----------  ---------  ----------------------
     57.0        240257784        100   2402577.8   2426684.5    2358605   2455405      32694.9  1048576 1 1  256 1 1  KrnlDmmyCalc(float *, flo
     43.0        181291027          1  181291027.0  181291027.0  181291027  181291027        0.0  1048576 1 1  256 1 1  KrnlDmmyInit(float *, flo
```

Prefetching: fastest, $1341\,GB/s$

→ 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@n3073–004:~$ nvcc ./unified_memory_example_5.cu
cuda–zen sh@n3073–004:~$ nsys nvprof ./a.out

[4/7] Executing 'cudaapisum' stats report

 Time (%)  Total Time (ns)  Num Calls  Avg (ns)   Med (ns)   Min (ns)  Max (ns)   StdDev (ns)           Name
 --------  ---------------  ---------  ---------  ---------  --------  ---------  -----------  ----------------------
     65.0        439291852        101  4349424.3     3800.0      3590  438878684   43669650.2  cudaLaunchKernel
     35.0        236640315        101  2342973.4  2352459.0   1377152    2358178      97072.8  cudaDeviceSynchronize



[5/7] Executing 'gpukernsum' stats report

 Time (%) Total Time (ns) Instances Avg (ns)  Med (ns)  Min (ns) Max (ns) StdDev (ns)   GridXYZ   BlockXYZ         Name
 -------- --------------- --------- --------- --------- -------- -------- ----------- ----------- -------- --------------
     99.4       234905893       100 2349058.9 2348844.5  2345838  2355021      1351.8 1048576 1 1 256  1 1 KrnlDmmyCalc()
      0.6         1385416         1 1385416.0 1385416.0  1385416  1385416         0.0 1048576 1 1 256  1 1 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@n3073-004:~$  nvcc ./unified_memory_example_5.cu
cuda-zen sh@n3073-004:~$  nsys nvprof ./a.out

[4/7] Executing 'cudaapisum' stats report

 Time (%)  Total Time (ns)  Num Calls  Avg (ns)   Med (ns)   Min (ns)  Max (ns)   StdDev (ns)           Name
 --------  ---------------  ---------  ---------  ---------  --------  ---------  -----------  ----------------------
     65.0        439291852        101  4349424.3     3800.0      3590  438878684   43669650.2  cuda
     35.0        236640315        101  2342973.4  2352459.0   1377152    2358178      97072.8  cuda

[5/7] Executing 'gpukernsum' stats report

 Time (%) Total Time (ns) Instances Avg (ns)  Med (ns)  Min (ns) Max (ns) StdDev (ns)   GridXYZ   BlockXYZ         Name
 -------- --------------- --------- --------- --------- -------- -------- ----------- ---------- -------- --------------
     99.4       234905893       100 2349058.9 2348844.5  2345838  2355021      1351.8 1048576 1 1 256  1 1 KrnlDmmyCalc()
      0.6         1385416         1 1385416.0 1385416.0  1385416  1385416         0.0 1048576 1 1 256  1 1 KrnlDmmyInit()
```

Globally managed ex aequo, $1371 GB/s$

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

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);
}
```

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)

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)

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)

Simple correctness check

24 GB array(s)

Memory Hierarchy cont.

CUDA C-Programming Guide, Unified Memory Programming cont.

```
cuda-zen sh@n3073-004:~$ nvcc ./unified_memory_oversubscription.cu
cuda-zen sh@n3073-004:~$ nsys nvprof ./a.out

[5/7] Executing 'gpukernsum' stats report

 Time (%)  Total Time (ns)  Instances    Avg (ns)      Med (ns)     Min (ns)    Max (ns)   StdDev (ns)       GridXYZ          BlockXYZ
 --------  ---------------  ---------  ------------  ------------  ----------  ----------  -----------  -----------------  --------------
    100.0       6274870199          1  6274870199.0  6274870199.0  6274870199  6274870199          0.0  6291456    1    1  512    1    1

[6/7] Executing 'gpumemtimesum' stats report

 Time (%)  Total Time (ns)  Count  Avg (ns)  Med (ns)  Min (ns)  Max (ns)  StdDev (ns)              Operation
 --------  ---------------  -----  --------  --------  --------  --------  -----------  ---------------------------------
    100.0       5909981859  96714   61107.8    6656.0      2495    556067      97099.5  [CUDA Unified Memory memcpy DtoH]

[7/7] Executing 'gpumemsizesum' stats report

 Total (MB)  Count  Avg (MB)  Med (MB)  Min (MB)  Max (MB)  StdDev (MB)              Operation
 ----------  -----  --------  --------  --------  --------  -----------  ---------------------------------
  49417.290  96714     0.511     0.061     0.004     2.097        0.780  [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@n3073–004:~$ nvcc ./unified_memory_oversubscription.cu
cuda–zen sh@n3073–004:~$ nsys nvprof ./a.out
[5/7] Executing 'gpukernsum' stats report

 Time (%)  Total Time (ns)  Instances    Avg (ns)      Med (ns)     Min (ns)    Max (ns)  StdDev (ns)       GridXYZ          BlockXYZ
--------  ---------------  ---------  ------------  ------------  ----------  ----------  -----------  ----------------  --------------
    100.0       6274870199          1  6274870199.0  6274870199.0  6274870199  6274870199          0.0  6291456    1    1  512    1    1

[6/7] Executing 'gpumemtimesum' stats report

 Time (%)  Total Time (ns)  Count  Avg (ns)  Med (ns)  Min (ns)  Max (ns)  StdDev (ns)             Operation
--------  ---------------  -----  --------  --------  --------  --------  -----------  --------------------------------
    100.0       5909981859  96714   61107.8    6656.0      2495    556067      97099.5  [CUDA Unified Memory memcpy DtoH]

[7/7] Executing 'gpumemsizesum' stats report

 Total (MB)  Count  Avg (MB)  Med (MB)  Min (MB)  Max (MB)  StdDev (MB)             Operation
----------  -----  --------  --------  --------  --------  -----------  --------------------------------
  49417.290  96714     0.511     0.061     0.004     2.097        0.780  [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@n3073–004:~$ nvcc ./unified_memory_oversubscription.cu
cuda–zen sh@n3073–004:~$ nsys nvprof ./a.out
[5/7] Executing 'gpukernsum' stats report

 Time (%)  Total Time (ns)  Instances    Avg (ns)      Med (ns)     Min (ns)    Max (ns)  StdDev (ns)       GridXYZ          BlockXYZ
--------  ---------------  ---------  ------------  ------------  ----------  ----------  -----------  ----------------  --------------
    100.0       6274870199          1  6274870199.0  6274870199.0  6274870199  6274870199          0.0  6291456    1    1  512    1    1

[6/7] Executing 'gpumemtimesum' stats report

 Time (%)  Total Time (ns)  Count  Avg (ns)  Med (ns)  Min (ns)  Max (ns)  StdDev (ns)             Operation
--------  ---------------  -----  --------  --------  --------  --------  -----------  --------------------------------
    100.0       5909981859  96714   61107.8    6656.0      2495    556067      97099.5  [CUDA Unified Memory memcpy DtoH]

[7/7] Executing 'gpumemsizesum' stats report

 Total (MB)  Count  Avg (MB)  Med (MB)  Min (MB)  Max (MB)  StdDev (MB)             Operation
----------  -----  --------  --------  --------  --------  -----------  --------------------------------
  49417.290  96714     0.511     0.061     0.004     2.097        0.780  [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

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

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);
    ...
}
```

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]

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);
    ...
}
```

Far-from-optimal memory access !

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

Memory Hierarchy cont.

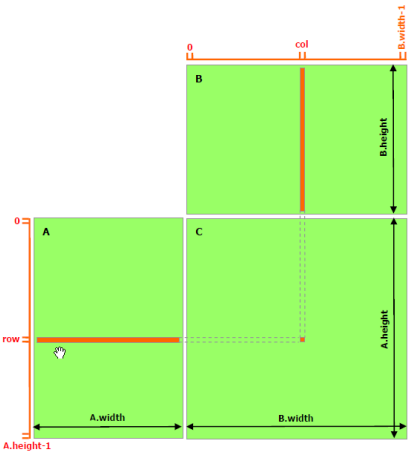
CUDA C-Programming Guide, Shared Memory cont.

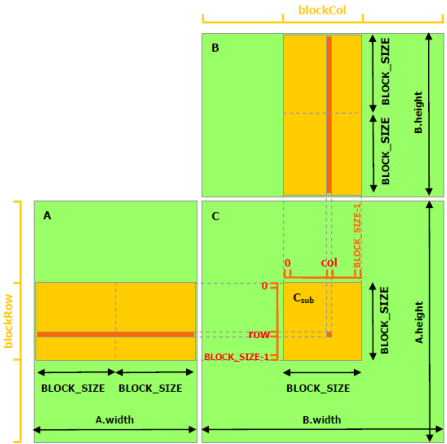
```
cuda–zen sh@n3073–004:~$  nvcc ./mmm_example_2.cu
cuda–zen sh@n3073–004:~$  nsys nvprof ./a.out

[5/7] Executing 'gpukernsum' stats report

Time (%) Total Time (ns) Instances  Avg (ns)    Med (ns)   Min (ns)  Max (ns)  StdDev (ns)   GridXYZ    BlockXYZ              Name
-------- --------------- --------- ----------- ---------- --------- --------- ----------- ---------- -------- ----------------------
   100.0       583635239         1 583635239.0 583635239.0 583635239 583635239         0.0 300  300  1  16  16 1 KrnlMMM(float **, float **

[6/7] Executing 'gpumemtimesum' stats report

Time (%)  Total Time (ns)  Count  Avg (ns)  Med (ns)  Min (ns)  Max (ns)  StdDev (ns)             Operation
--------  ---------------  -----  --------  --------  --------  --------  -----------  --------------------------------
    51.3         93431969   2432   38417.8    5263.0      1727    363138      85310.1  [CUDA Unified Memory memcpy DtoH]
    48.7         88846629   5887   15092.0    7039.0      3166    317537      24377.6  [CUDA Unified Memory memcpy HtoD]

[7/7] Executing 'gpumemsizesum' stats report

Total (MB)  Count  Avg (MB)  Med (MB)  Min (MB)  Max (MB)  StdDev (MB)             Operation
----------  -----  --------  --------  --------  --------  -----------  --------------------------------
   283.116   2432     0.116     0.025     0.004     1.032        0.246  [CUDA Unified Memory memcpy DtoH]
   283.116   5887     0.048     0.025     0.004     1.024        0.091  [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

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;
}
```

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;
}
```

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;
}
```

Matrix Matrix Multiplication Version 2

Declaration of shared arrays

Initialization by entire threadblock

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

```
#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;
}
```

Matrix Matrix Multiplication Version 2

Declaration of shared arrays

Initialization by entire threadblock

Copy into shared memory arrays

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

```
#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;
}
```

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

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

Back-copy from shared to global memory

Memory Hierarchy cont.

CUDA C-Programming Guide, Shared Memory cont.

```
cuda–zen sh@n3073–004:~$ nvcc ./mmm_example_3.cu
cuda–zen sh@n3073–004:~$ nsys nvprof ./a.out

[5/7] Executing 'gpukernsum' stats report

Time (%) Total Time (ns) Instances  Avg (ns)    Med (ns)   Min (ns)  Max (ns)  StdDev (ns)   GridXY       BlockXYZ          Name
-------- --------------- --------- ----------- ----------- --------- --------- ----------- ------- ------------ ----------------------
   100.0       203907614         1 203907614.0 203907614.0 203907614 203907614         0.0 300  300  1 16 16  1  KrnlMMM(float **, float **

[6/7] Executing 'gpumemtimesum' stats report

Time (%)  Total Time (ns)  Count  Avg (ns)  Med (ns)  Min (ns)  Max (ns)  StdDev (ns)             Operation
--------  ---------------  -----  --------  --------  --------  --------  -----------  --------------------------------
    54.3        105519535   6791   15538.1    8831.0      3712    337602      28616.3  [CUDA Unified Memory memcpy HtoD]
    45.7         88804386   2385   37234.5    4736.0      1535    545476      83237.8  [CUDA Unified Memory memcpy DtoH]

[7/7] Executing 'gpumemsizesum' stats report

Total (MB)  Count  Avg (MB)  Med (MB)  Min (MB)  Max (MB)  StdDev (MB)             Operation
----------  -----  --------  --------  --------  --------  -----------  --------------------------------
   283.116   2385     0.119     0.025     0.004     1.028        0.247  [CUDA Unified Memory memcpy DtoH]
   283.116   6791     0.042     0.016     0.004     1.036        0.096  [CUDA Unified Memory memcpy HtoD]
```

Factor ≈2.9× 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 on the CLI

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 on the CLI
- 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 on the CLI
- 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