Modern Hardware Models — How to Exploit Specific Features with CUDA

Siegfried Höfinger

VSC Research Center, TU Wien

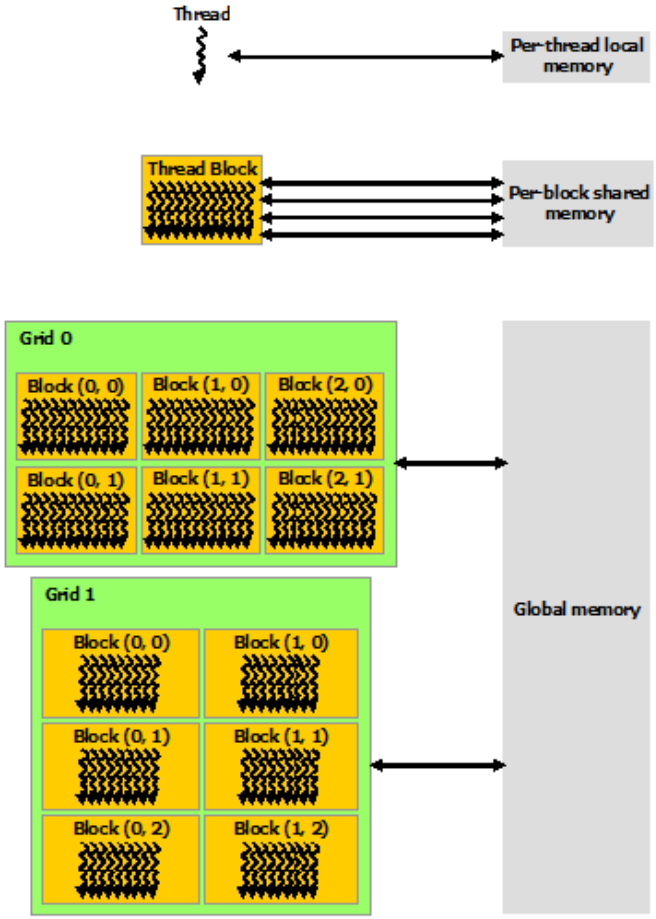
October 1, 2021

→ https://tinyurl.com/cuda4dummies/i/l2/notes-l2.pdf

Outline

Memory Hierarchy

CUDA C-Programming Guide, Programming Model cont.

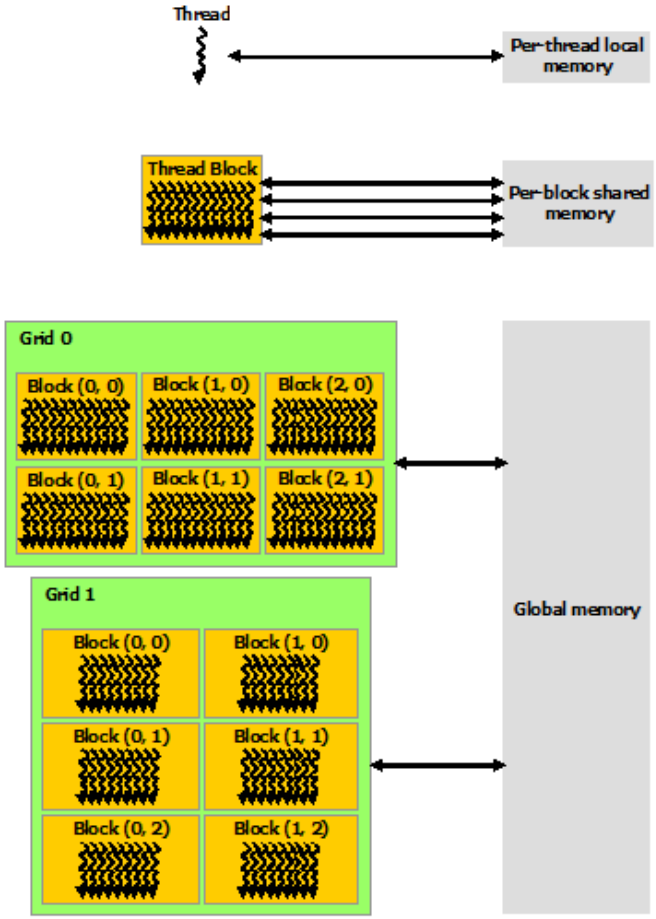


- Threads have access to several memory spaces

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

Memory Hierarchy

CUDA C-Programming Guide, Programming Model cont.

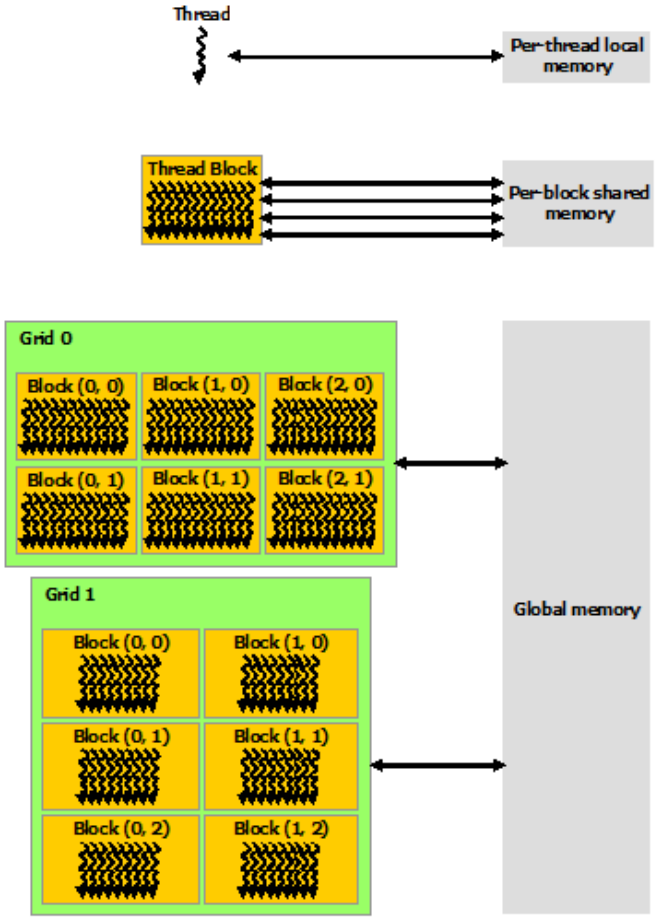


- Threads have access to several memory spaces
- Private local memory for each individual thread (fast)

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

Memory Hierarchy

CUDA C-Programming Guide, Programming Model cont.

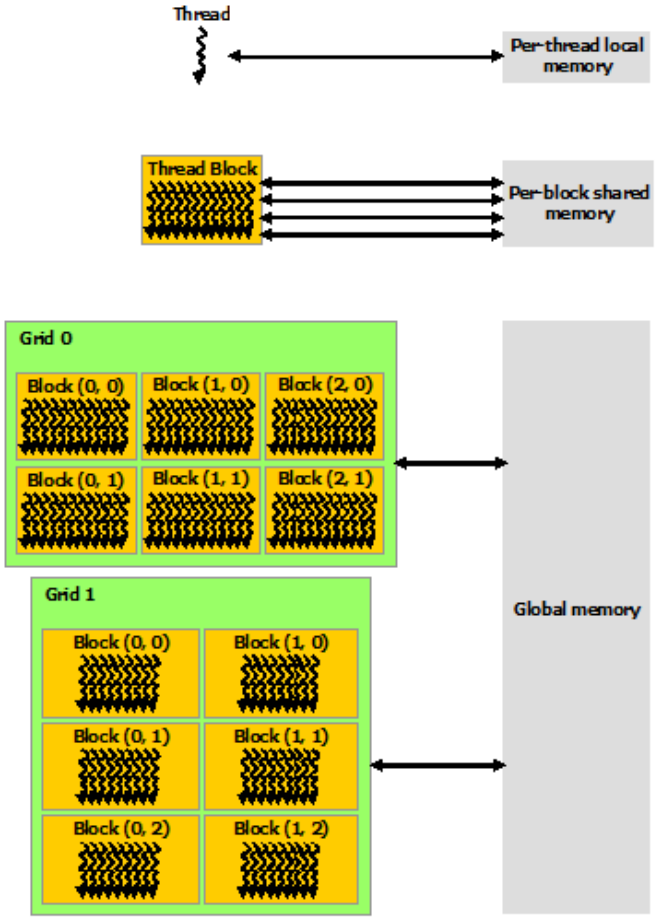


- Threads have access to several memory spaces
- Private local memory for each individual thread (fast)
- Shared memory for all threads within a thread block (pretty fast)

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

Memory Hierarchy

CUDA C-Programming Guide, Programming Model cont.

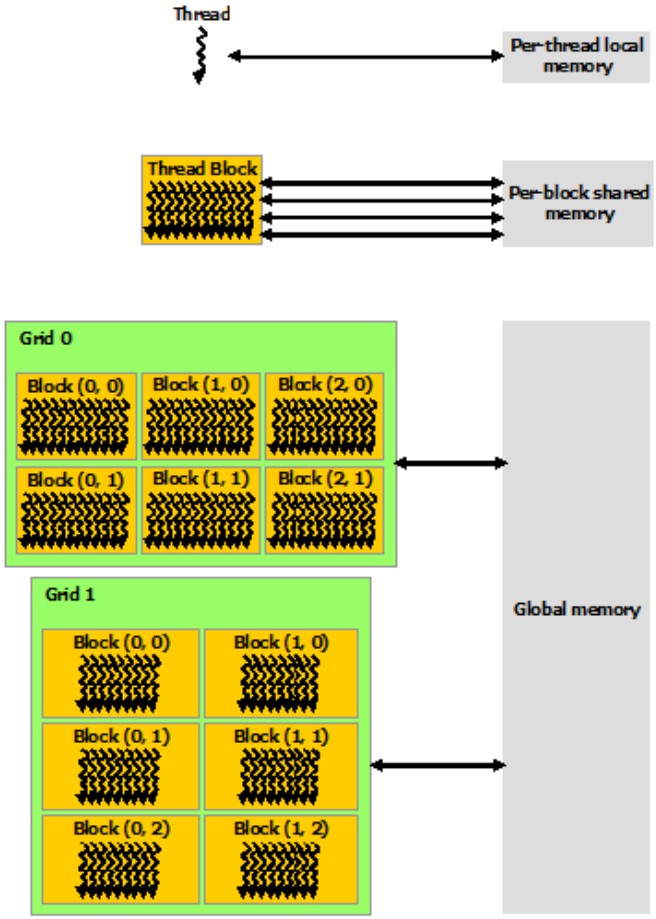


- Threads have access to several memory spaces
- Private local memory for each individual thread (fast)
- Shared memory for all threads within a thread block (pretty fast)
- Global memory accessible to all threads (slow)

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

Memory Hierarchy

CUDA C-Programming Guide, Programming Model cont.

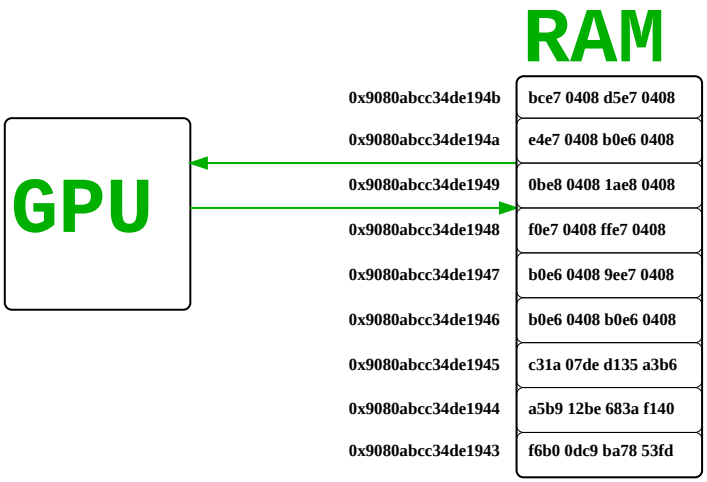


- Threads have access to several memory spaces
- Private local memory for each individual thread (fast)
- Shared memory for all threads within a thread block (pretty fast)
- Global memory accessible to all threads (slow)
- Special ROMs, constant and texture memory

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

Memory Hierarchy

CUDA C-Programming Guide, Unified Memory Programming



Unified Memory:

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

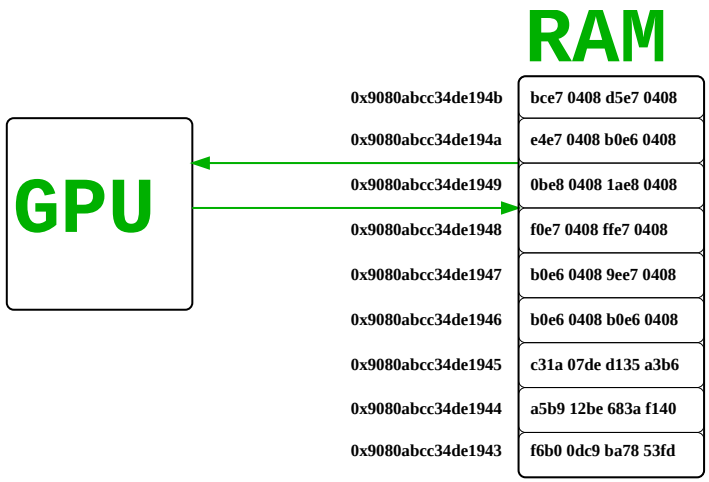
Separate Device/Host Memory:

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

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

Memory Hierarchy

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

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

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

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

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

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

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

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);
}
```

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

Memory Hierarchy

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);
}
```

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

Memory Hierarchy

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

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

Memory Hierarchy

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

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

Memory Hierarchy

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

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

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();
    ...
}
```

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

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();
    ...
}
```

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

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

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

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

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

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();
     ...
}
```

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

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

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

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

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

Memory Hierarchy

CUDA C-Programming Guide, Unified Memory Programming cont.

```
[sh@n566-009]$ nvcc ./unified_memory_example_1.cu
[sh@n566-009]$ ./a.out
[sh@n566-009]$ nsys nvprof ./a.out

Generating CUDA API Statistics...
CUDA API Statistics (nanoseconds)

Time(%)   Total Time    Calls      Average     Minimum      Maximum  Name
-------  -----------  -------  -----------  ----------  -----------  ----------------------
   60.4    569670516        3  189890172.0       13074    569636072  cudaMallocManaged
   28.1    265384460        1  265384460.0   265384460    265384460  cudaDeviceSynchronize
   11.5    108355541        3   36118513.7    35997508     36355106  cudaFree
    0.0        51967        1      51967.0       51967        51967  cudaLaunchKernel

Generating CUDA Kernel Statistics...
CUDA Kernel Statistics (nanoseconds)

Time(%)  Total Time   Instances      Average      Minimum      Maximum  Name
-------  ----------  ----------  -----------  -----------  -----------  ----------------------
  100.0   265363613           1  265363613.0    265363613    265363613  KrnlDmmy(float*, float*, float*)
```

→ https://tinyurl.com/cuda4dummies/i/l2/unified_memory_example_1.cu

Memory Hierarchy

CUDA C-Programming Guide, Unified Memory Programming cont.

```
[sh@n566-009]$ nvcc ./unified_memory_example_1.cu
[sh@n566-009]$ ./a.out
[sh@n566-009]$ nsys nvprof ./a.out

Generating CUDA API Statistics...
CUDA API Statistics (nanoseconds)

Time(%)  Total Time    Calls      Average     Minimum      Maximum  Name
-------  ----------  -------  -----------  ----------  -----------  ----------------------
   60.4   569670516        3  189890172.0       13074    569636072  cudaMallocManaged
   28.1   265384460        1  265384460.0   265384460    265384460  cudaDeviceSynchronize
   11.5   108355541        3   36118513.7    35997508     36355106  cudaFree
    0.0       51967        1      51967.0       51967        51967  cudaLaunchKernel

Generating CUDA Kernel Statistics...
CUDA Kernel Statistics (nanoseconds)

Time(%)  Total Time   Instances      Average      Minimum      Maximum  Name
-------  ----------  ----------  -----------  -----------  -----------  ----------------------
  100.0   265363613           1  265363613.0    265363613    265363613  KrnlDmmy(float*, float*, float*)
```

→ https://tinyurl.com/cuda4dummies/i/l2/unified_memory_example_1.cu

Memory Hierarchy

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

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

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

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

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

CUDA C-Programming Guide, Unified Memory Programming cont.

```
[sh@n566-009]$  nvcc ./unified_memory_example_2.cu
[sh@n566-009]$  nsys nvprof ./a.out

Generating CUDA API Statistics...
CUDA API Statistics (nanoseconds)

Time(%)   Total Time    Calls      Average      Minimum      Maximum  Name
-------  -----------  -------  -----------  -----------  -----------  ----------------------------------
   50.3    269573096        2  134786548.0     99348239    170224857  cudaDeviceSynchronize
   29.5    157880723        3   52626907.7        24115    157817345  cudaMallocManaged
   20.2    108148966        3   36049655.3     36012242     36097392  cudaFree
    0.0        91190        2      45595.0        10329        80861  cudaLaunchKernel

Generating CUDA Kernel Statistics...
CUDA Kernel Statistics (nanoseconds)

Time(%)   Total Time   Instances   Average      Minimum      Maximum  Name
-------  -----------  ----------  -----------  -----------  -----------  ----------------------------------
   63.1    170226303           1  170226303.0    170226303    170226303  KrnlDmmyInit(float*, float*, float*
   36.9     99341634           1   99341634.0     99341634     99341634  KrnlDmmyCalc(float*, float*, float*
```

→ https://tinyurl.com/cuda4dummies/i/l2/unified_memory_example_2.cu

Memory Hierarchy

CUDA C-Programming Guide, Unified Memory Programming cont.

```
[sh@n566-009]$  nvcc ./unified_memory_example_2.cu
[sh@n566-009]$  nsys nvprof ./a.out

Generating CUDA API Statistics...
CUDA API Statistics (nanoseconds)

Time(%)   Total Time    Calls      Average      Minimum      Maximum  Name
-------  -----------  -------  -----------  -----------  -----------  ----------------------------------
   50.3    269573096        2  134786548.0     99348239    170224857  cudaDeviceS
   29.5    157880723        3   52626907.7        24115    157817345  cudaMallocM
   20.2    108148966        3   36049655.3     36012242     36097392  cudaFree
    0.0        91190        2      45595.0        10329        80861  cudaLaunchK

Generating CUDA Kernel Statistics...
CUDA Kernel Statistics (nanoseconds)

Time(%)   Total Time   Instances   Average      Minimum      Maximum  Name
-------  -----------  ---------  -----------  -----------  -----------  ----------------------------------
   63.1    170226303          1  170226303.0    170226303    170226303  KrnlDmmyInit(float*, float*, float*
   36.9     99341634          1   99341634.0     99341634     99341634  KrnlDmmyCalc(float*, float*, float*
```

Compute kernel much faster now !

→ https://tinyurl.com/cuda4dummies/i/l2/unified_memory_example_2.cu

Memory Hierarchy

CUDA C-Programming Guide, Unified Memory Programming cont.

```
[sh@n566-009]$ nvcc ./unified_memory_example_2.cu
[sh@n566-009]$ nsys nvprof ./a.out

Generating CUDA API Statistics...
CUDA API Statistics (nanoseconds)

Time(%)   Total Time     Calls       Average      Minimum      Maximum  Name
-------  -----------  -------  -----------  -----------  -----------  ----------------------------------
                                                99348239    170224857  cudaDeviceS
                                                   24115    157817345  cudaMallocM
                                                36012242     36097392  cudaFree
                                                   10329        80861  cudaLaunchK

                                                 Minimum      Maximum  Name
-------  -----------  -------  -----------  -----------  -----------  ----------------------------------
   63.1    170226303        1  170226303.0    170226303    170226303  KrnlDmmyInit(float*, float*, float*
   36.9     99341634        1   99341634.0     99341634     99341634  KrnlDmmyCalc(float*, float*, float*
```

Still very low effective memory bandwith,

$$\frac{\frac{3 \times 268435456 \times 4}{10^9}}{99341634 \times 10^{-9}} = 32.4 GB/s$$

from theoretical $696 GB/s$

Compute kernel much faster now !

→ https://tinyurl.com/cuda4dummies/i/l2/unified_memory_example_2.cu

Memory Hierarchy

CUDA C-Programming Guide, Unified Memory Programming cont.

```
[sh@n566-009]$ nvcc ./unified_memory_example_3.cu
[sh@n566-009]$ nsys nvprof ./a.out

Generating CUDA API Statistics...
CUDA API Statistics (nanoseconds)

Time(%)   Total Time    Calls      Average      Minimum      Maximum  Name
-------  -----------  -------  -----------  -----------  -----------  ----------------------
   76.1    825438454      101    8172658.0      5529452    173083071  cudaDeviceSynchronize
   14.3    155191348        3   51730449.3        26900    155098254  cudaMallocManaged
    9.5    103193254        3   34397751.3     34318346     34447969  cudaFree
    0.1       682853      101       6760.9         4398       111619  cudaLaunchKernel

Generating CUDA Kernel Statistics...
CUDA Kernel Statistics (nanoseconds)

Time(%)   Total Time  Instances    Average      Minimum      Maximum  Name
-------  -----------  ---------  -----------  -----------  -----------  ----------------------
   79.0    651943469        100   6519434.7      5524687    102454875  KrnlDmmyCalc(float*, float*, float*
   21.0    173112476          1  173112476.0    173112476    173112476  KrnlDmmyInit(float*, float*, float*
```

→ https://tinyurl.com/cuda4dummies/i/l2/unified_memory_example_3.cu

Memory Hierarchy

CUDA C-Programming Guide, Unified Memory Programming cont.

```
[sh@n566-009]$ nvcc ./unified_memory_example_3.cu
[sh@n566-009]$ nsys nvprof ./a.out

Generating CUDA API Statistics...
CUDA API Statistics (nanoseconds)

Time(%)   Total Time    Calls      Average      Minimum
-------  -----------  -------  -----------  -----------
   76.1    825438454      101    8172658.0      5529452
   14.3    155191348        3   51730449.3        26900
    9.5    103193254        3   34397751.3     34318346
    0.1       682853      101       6760.9         4398

Generating CUDA Kernel Statistics...
CUDA Kernel Statistics (nanoseconds)

Time(%)   Total Time  Instances    Average      Minimum      Maximum  Name
-------  -----------  ---------  -----------  -----------  -----------  ----------------------
   79.0    651943469        100   6519434.7      5524687    102454875  KrnlDmmyCalc(float*, float*, float*
   21.0    173112476          1  173112476.0    173112476    173112476  KrnlDmmyInit(float*, float*, float*
```

100× greatly improves effective memory bandwith, $494 GB/s$ and compute performance

→ https://tinyurl.com/cuda4dummies/i/l2/unified_memory_example_3.cu

Memory Hierarchy

CUDA C-Programming Guide, Unified Memory Programming cont.

```
[sh@n566-009]$ nvcc ./unified_memory_example_4.cu
[sh@n566-009]$ nsys nvprof ./a.out

Generating CUDA API Statistics...
CUDA API Statistics (nanoseconds)

Time(%)   Total Time    Calls      Average      Minimum      Maximum  Name
-------  -----------  -------  -----------  -----------  -----------  ----------------------
   73.0    727072166      101    7198734.3      5536076    171690752  cudaDeviceSynchronize
   15.5    154488022        3   51496007.3        28112    154380362  cudaMallocManaged
   10.8    107270172        3   35756724.0     35124321     36291645  cudaFree
    0.6      6085422        3    2028474.0       480839      4987409  cudaMemPrefetchAsync
    0.1       605713      101       5997.2         4629        86201  cudaLaunchKernel

Generating CUDA Kernel Statistics...
CUDA Kernel Statistics (nanoseconds)

Time(%)   Total Time  Instances    Average      Minimum      Maximum  Name
-------  -----------  ---------  -----------  -----------  -----------  ----------------------
   76.4    554979343        100    5549793.4      5531120      5569200  KrnlDmmyCalc(float*, float*, float*
   23.6    171691198          1  171691198.0    171691198    171691198  KrnlDmmyInit(float*, float*, float*
```

→ https://tinyurl.com/cuda4dummies/i/l2/unified_memory_example_4.cu

Memory Hierarchy

CUDA C-Programming Guide, Unified Memory Programming cont.

```
[sh@n566-009]$ nvcc ./unified_memory_example_4.cu
[sh@n566-009]$ nsys nvprof ./a.out

Generating CUDA API Statistics...
CUDA API Statistics (nanoseconds)

Time(%)   Total Time    Calls      Average      Minimum      Maximum  Name
-------  -----------  -------  -----------  -----------  -----------  ----------------------
   73.0    727072166      101    7198734.3      5536076
   15.5    154488022        3   51496007.3        28112
   10.8    107270172        3   35756724.0     35124321
    0.6      6085422        3    2028474.0       480839
    0.1       605713      101       5997.2         4629

Generating CUDA Kernel Statistics...
CUDA Kernel Statistics (nanoseconds)

Time(%)   Total Time  Instances     Average      Minimum      Maximum  Name
-------  -----------  ---------  -----------  -----------  -----------  ----------------------
   76.4    554979343        100    5549793.4      5531120      5569200  KrnlDmmyCalc(float*, float*, float*
   23.6    171691198          1  171691198.0    171691198    171691198  KrnlDmmyInit(float*, float*, float*
```

Prefetching: fastest, $580 GB/s$

→ https://tinyurl.com/cuda4dummies/i/l2/unified_memory_example_4.cu

Memory Hierarchy

CUDA C-Programming Guide, Unified Memory Programming cont.

```
[sh@n566-009]$  nvcc ./unified_memory_example_5.cu
[sh@n566-009]$  nsys nvprof ./a.out

Generating CUDA API Statistics...
CUDA API Statistics (nanoseconds)

Time(%)   Total Time    Calls      Average      Minimum      Maximum  Name
-------  -----------  -------  -----------  -----------  -----------  ---------------------
   53.5    555080565      101    5495847.2      3737742      5540924  cudaDeviceSynchronize
   46.5    482839783      101    4780591.9         4418    482323256  cudaLaunchKernel

Generating CUDA Kernel Statistics...
CUDA Kernel Statistics (nanoseconds)

Time(%)   Total Time  Instances    Average      Minimum      Maximum  Name
-------  -----------  ---------  -----------  -----------  -----------  ----------------------
   99.3    550947384        100  5509473.8    5481552      5537648  KrnlDmmyCalc()
    0.7      3741621          1  3741621.0    3741621      3741621  KrnlDmmyInit()
```

→ https://tinyurl.com/cuda4dummies/i/l2/unified_memory_example_5.cu

Memory Hierarchy

CUDA C-Programming Guide, Unified Memory Programming cont.

```
[sh@n566-009]$  nvcc ./unified_memory_example_5.cu
[sh@n566-009]$  nsys nvprof ./a.out

Generating CUDA API Statistics...
CUDA API Statistics (nanoseconds)

Time(%)   Total Time    Calls      Average      Minimum      Maximum  Name
-------  -----------  -------  -----------  -----------  -----------  ---------------------
   53.5    555080565      101    5495847.2      3737742
   46.5    482839783      101    4780591.9         4418

Generating CUDA Kernel Statistics...
CUDA Kernel Statistics (nanoseconds)

Time(%)   Total Time  Instances    Average      Minimum      Maximum  Name
-------  -----------  ---------  -----------  -----------  -----------  ---------------------
   99.3    550947384        100    5509473.8      5481552      5537648  KrnlDmmyCalc()
    0.7      3741621          1    3741621.0      3741621      3741621  KrnlDmmyInit()
```

Globally managed ex aequo, $584\,GB/s$

→ https://tinyurl.com/cuda4dummies/i/l2/unified_memory_example_5.cu

Memory Hierarchy

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);
}
```

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

Memory Hierarchy

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)

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

Memory Hierarchy

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 (45 GB onboard)

24 GB array(s)

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

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 (45 GB onboard)

Simple correctness check

24 GB array(s)

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

Memory Hierarchy

CUDA C-Programming Guide, Unified Memory Programming cont.

```
[sh@n566-009]$  nvcc ./unified_memory_oversubscription.cu
[sh@n566-009]$  nsys nvprof ./a.out

Generating CUDA Kernel Statistics...
CUDA Kernel Statistics (nanoseconds)

Time(%)   Total Time  Instances      Average      Minimum      Maximum  Name
-------  -----------  ---------  -------------  -----------  -----------  ----------------------
  100.0  11030553978          1  11030553978.0  11030553978  11030553978  KrnlDmmy(double*, double*, double

Generating CUDA Memory Operation Statistics...
CUDA Memory Operation Statistics (nanoseconds)

Time(%)   Total Time  Operations      Average      Minimum      Maximum  Name
-------  -----------  ----------  -------------  -----------  -----------  ----------------------
  100.0   1017604519      104895         9701.2          278        83573  [CUDA Unified Memory memcpy DtoH]


CUDA Memory Operation Statistics (KiB)

           Total   Operations        Average      Minimum           Maximum  Name
---------------  -----------  -------------  -----------  ----------------  -------------------
    44535808.000       104895        424.575        4.000          2048.000  [CUDA Unified Memory memcpy
```

→ https://tinyurl.com/cuda4dummies/i/l2/unified_memory_oversubscription.cu

Memory Hierarchy

CUDA C-Programming Guide, Unified Memory Programming cont.

```
[sh@n566-009]$ nvcc ./unified_memory_oversubscription.cu
[sh@n566-009]$ nsys nvprof ./a.out

Generating CUDA Kernel Statistics...
CUDA Kernel Statistics (nanoseconds)

Time(%)   Total Time  Instances      Average      Minimum      Maximum  Name
-------  -----------  ---------  -------------  -----------  -----------  ----------------------
  100.0  11030553978          1  11030553978.0  11030553978  11030553978  KrnlDmmy(double*, double*, double

Generating CUDA Memory Operation Statistics...
CUDA Memory Operation Statistics (nanoseconds)

Time(%)   Total Time  Operations      Average      Minimum      Maximum  Name
-------  -----------  ----------  -------------  -----------  -----------  ----------------------
  100.0   1017604519      104895         9701.2          278        83573  [CUDA Unified Memory memcpy DtoH]


CUDA Memory Operation Statistics (KiB)

           Total   Operations        Average      Minimum           Maximum  Name
---------------  -----------  -------------  -----------  ----------------  -------------------
    44535808.000       104895        424.575        4.000          2048.000  [CUDA Unified Memory memcpy
```

All time in memory transfer

→ https://tinyurl.com/cuda4dummies/i/l2/unified_memory_oversubscription.cu

Memory Hierarchy

CUDA C-Programming Guide, Unified Memory Programming cont.

```
[sh@n566-009]$ nvcc ./unified_memory_oversubscription.cu
[sh@n566-009]$ nsys nvprof ./a.out

Generating CUDA Kernel Statistics...
CUDA Kernel Statistics (nanoseconds)

Time(%)   Total Time  Instances      Average      Minimum      Maximum  Name
-------  -----------  ---------  -------------  -----------  -----------  ----------------------
  100.0  11030553978          1  11030553978.0  11030553978  11030553978  KrnlDmmy(double*, double*, double

Generating CUDA Memory Operation Statistics...
CUDA Memory Operation Statistics (nanoseconds)

Time(%)   Total Time  Operations      Average      Minimum
-------  -----------  ----------  -------------  -----------  -----------  ----------------------
  100.0   1017604519      104895         9701.2          278        83573  [CUDA Unified Memory memcpy DtoH]


CUDA Memory Operation Statistics (KiB)

           Total   Operations        Average      Minimum           Maximum  Name
---------------  ------------  -------------  -----------  ----------------  --------------------
    44535808.000        104895        424.575        4.000          2048.000  [CUDA Unified Memory memcpy
```

All time in memory transfer

45 GB reported

→ https://tinyurl.com/cuda4dummies/i/l2/unified_memory_oversubscription.cu

Memory Hierarchy

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

CUDA C-Programming Guide, Unified Memory Programming cont.

- On pre-Pascal-class GPUs cudaMallocManaged() behaves like cudaMalloc()
- On pre-Pascal-class GPUs managed allocations are automatically visible to all GPUs in a system via peer-to-peer capabilities
- Managed memory is allocated in GPU memory as long as all GPUs have peer-to-peer support enabled, otherwise the driver will migrate all managed allocations to the CPU/host memory ("zero-copy" memory)
- P2P GPUs will experience PCIe bandwidth restrictions

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

Memory Hierarchy

CUDA C-Programming Guide, Device Memory

On pre-Pascal-class GPUs a 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 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 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

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);
    ...
}
```

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

Memory Hierarchy

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

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

Memory Hierarchy

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

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

Memory Hierarchy

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

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

Memory Hierarchy

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

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-threa

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