Sai Kashyap Kurella
Akash Mattaparthy

Lab6

Compare the performance of using memcpy, pinned memory, and UVM (also with different hints of UVM). You can simply use the code of the previous assignments (e.g., MatrixMul or VectorAdd) with timing measurements and replace their memory allocation method. Make sure you also include the time spent on memcpy for a fair comparison, since pinned memory and UVM do not need that. Rather, their data transfer overhead is amortized on each memory access or each page miss.
Also, you should test with data sets of different sizes (e.g., 32B, 64B, ..., 2GB) for this experiment.

Sai Kashyap Kurella
Akash Mattaparthy

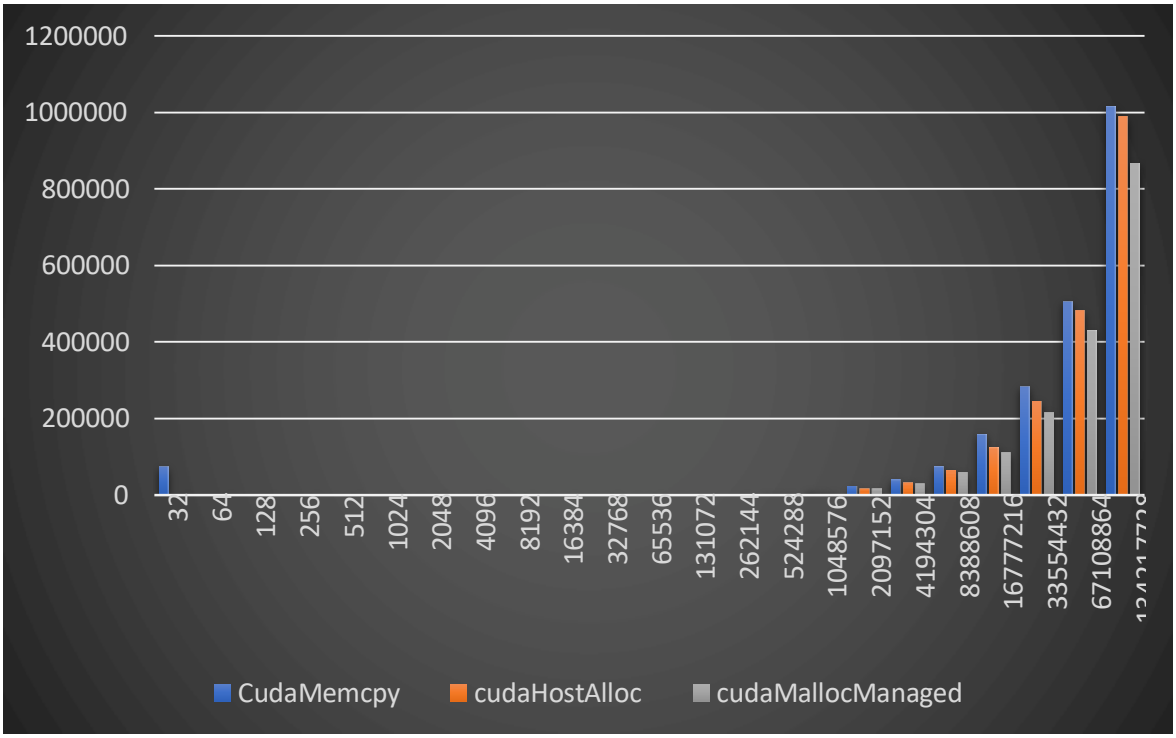
```
#include   <time.h>
#include   <stdio.h>
#include <stdlib.h>

__global__ void VecAdd(const float *A, const float *B, float* C, long unsigned int n) {
        long unsigned int ID = blockIdx.x * blockDim.x + threadIdx.x;

   if (ID < n)
      C[ID] = A[ID] + B[ID];
}

int main (int argc, char *argv[]){
   float *A_h, *B_h, *C_h;
   float *A_d, *B_d, *C_d;
        int blockSize, gridSize;
        long unsigned int VecSize, Limit = (1 << 25); // Limit 32MB. //maximum = 256 MB (32 MB * 4 bytes)

        FILE *fp = fopen("results.csv", "w");
        fprintf(fp,"Method, Size, Time\n");

        printf("\nCudaMemcpy Test\n");
        for(VecSize = 8; VecSize <= Limit; VecSize *= 2){
                clock_t start = clock(); //Starts
                A_h = (float*) malloc( sizeof(float) * VecSize );
                B_h = (float*) malloc( sizeof(float) * VecSize );
                C_h = (float*) malloc( sizeof(float) * VecSize );

                for (long unsigned int i=0; i < VecSize; i++) {
                        A_h[i] = 1.0f;
                        B_h[i] = 2.0f;
                }
                cudaDeviceSynchronize();

                cudaMalloc(&A_d, sizeof(float) * VecSize);
                cudaMalloc(&B_d, sizeof(float) * VecSize);
                cudaMalloc(&C_d, sizeof(float) * VecSize);

                cudaMemcpy(A_d, A_h, sizeof(float) * VecSize, cudaMemcpyHostToDevice);
                cudaMemcpy(B_d, B_h, sizeof(float) * VecSize, cudaMemcpyHostToDevice);

                cudaDeviceSynchronize();

                blockSize = 32;
                gridSize = (int)ceil((float)VecSize/blockSize);

                VecAdd<<<gridSize, blockSize>>>(A_d, B_d, C_d, VecSize);

                cudaMemcpy(C_h, C_d, sizeof(float) * VecSize, cudaMemcpyDeviceToHost);

                cudaDeviceSynchronize();

                free(A_h);
                free(B_h);
                free(C_h);

                cudaFree(A_d);
                cudaFree(B_d);
                cudaFree(C_d);

                cudaDeviceSynchronize();
                clock_t end = clock(); //Ends
```

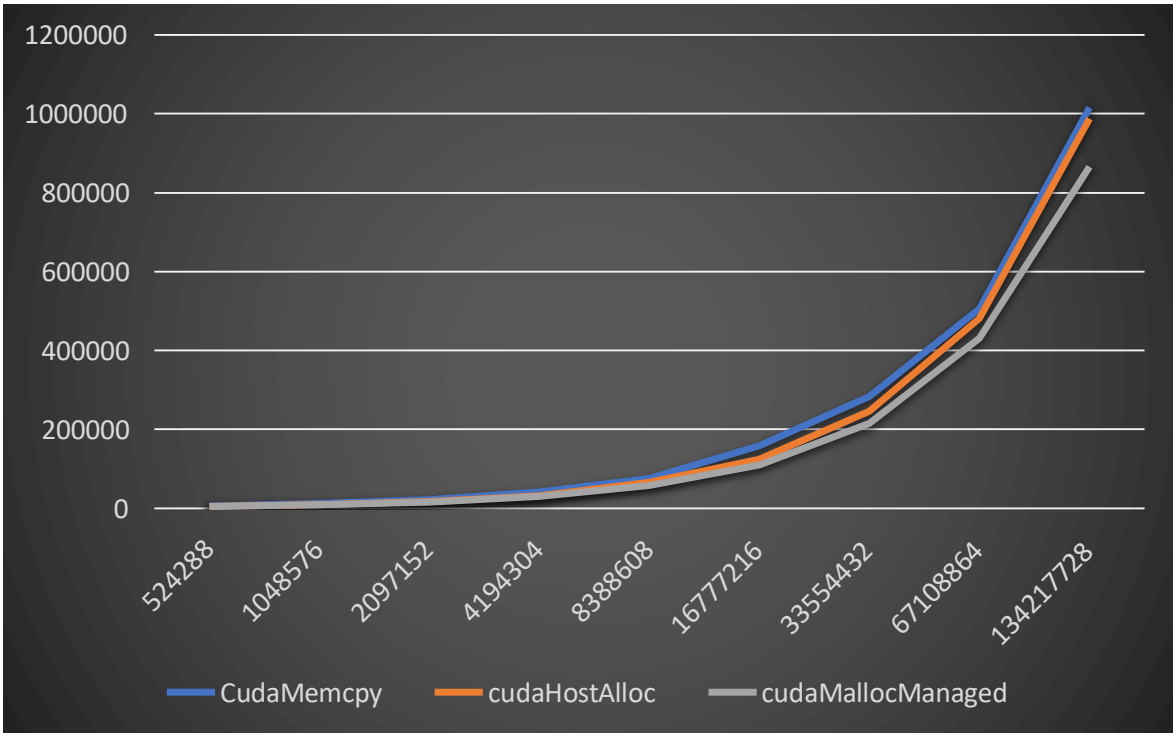
```
printf("\nPinned Memory(cudaHostAlloc) Test\n");

for(VecSize = 8; VecSize <= Limit; VecSize *= 2){

        clock_t start = clock(); //Starts
        cudaDeviceSynchronize();

        cudaHostAlloc(&A_h, sizeof(float) * VecSize, cudaHostAllocDefault);
        cudaHostAlloc(&B_h, sizeof(float) * VecSize, cudaHostAllocDefault);
        cudaHostAlloc(&C_h, sizeof(float) * VecSize, cudaHostAllocDefault);

        for (long unsigned int i=0; i < VecSize; i++) {
                A_h[i] = 1.0f;
                B_h[i] = 2.0f;
        }

        cudaHostGetDevicePointer(&A_d, A_h, 0);
        cudaHostGetDevicePointer(&B_d, B_h, 0);
        cudaHostGetDevicePointer(&C_d, C_h, 0);

        cudaDeviceSynchronize();

        blockSize = 32;
        gridSize = (int)ceil((float)VecSize/blockSize);

        VecAdd<<<gridSize, blockSize>>>(A_d, B_d, C_d, VecSize);

        cudaDeviceSynchronize();

        cudaFreeHost(A_h);
        cudaFreeHost(B_h);
        cudaFreeHost(C_h);

        A_d = NULL;
        B_d = NULL;
        C_d = NULL;

        cudaDeviceSynchronize();
        clock_t end = clock(); //Ends

        //Measure Data
        double elasped_secs = (((double) end - (double) start) / CLOCKS_PER_SEC) * 1000000;
        printf("Size: %ld, Time: %f us\n", sizeof(float) * VecSize, elasped_secs);
        fprintf(fp,"%s, %ld, %f\n", "CudaMemcpy", sizeof(float) * VecSize, elasped_secs);

}
```

```
        printf("\nUnified Virtual Memory(cudaMallocManaged) Test\n");
        for(VecSize = 8; VecSize <= Limit; VecSize *= 2){
                clock_t start = clock(); //Starts
                cudaDeviceSynchronize();

                cudaMallocManaged(&A_h, sizeof(float) * VecSize);
                cudaMallocManaged(&B_h, sizeof(float) * VecSize);
                cudaMallocManaged(&C_h, sizeof(float) * VecSize);

                for (long unsigned int i=0; i < VecSize; i++) {
                        A_h[i] = 1.0f;
                        B_h[i] = 2.0f;
                }

                cudaDeviceSynchronize();

                blockSize = 32;
                gridSize = (int)ceil((float)VecSize/blockSize);

                VecAdd<<<gridSize, blockSize>>>(A_h, B_h, C_h, VecSize);

                cudaDeviceSynchronize();

                cudaFree(A_h);
                cudaFree(B_h);
                cudaFree(C_h);

                cudaDeviceSynchronize();
                clock_t end = clock(); //Ends

                //Measure Data
                double elasped_secs = (((double) end - (double) start) / CLOCKS_PER_SEC) * 1000000;
                printf("Size: %ld, Time: %f us\n", sizeof(float) * VecSize, elasped_secs);
                fprintf(fp,"%s, %ld, %f\n", "cudaMallocManaged", sizeof(float) * VecSize, elasped_secs);
        }

        fclose(fp);

    return 0;
}
```

Sai Kashyap Kurella
Akash Mattaparthy

Observation:

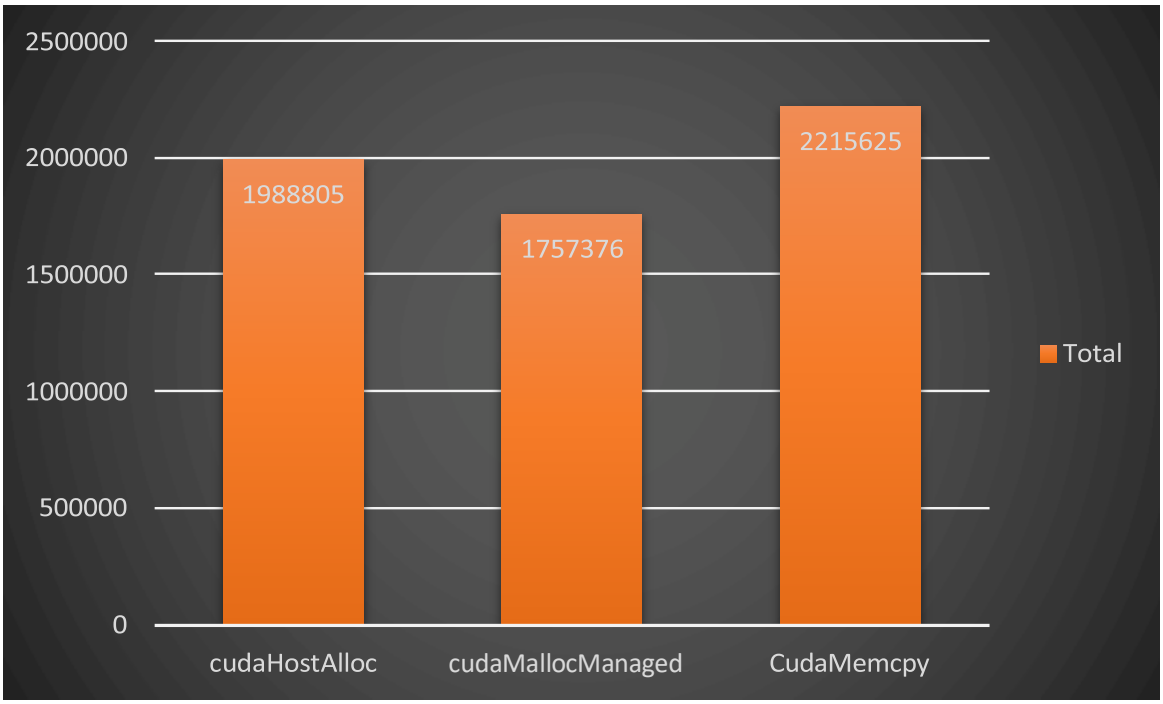




Output Table

Size (bytes)	Size	CudaMemcpy (Memcpy Method)	cudaHostAlloc (Pinned Memory Method)	cudaMallocManaged (Unified Virtual Memory Method)
32	32 Bytes	73669	1429	1129
64	64 Bytes	1413	1161	1016
128	128 Bytes	1198	1112	964
256	256 Bytes	1344	1113	871
512	512 Bytes	1129	1095	954
1024	1KB	1136	1090	958
2048	2 KB	1158	1104	933
4096	4 KB	1182	1168	889
8192	8 KB	1325	1229	975
16384	16 KB	1360	1287	1191
32768	32 KB	1645	1370	1235
65536	64 MB	1848	1519	1520
131072	128 KB	1952	1967	2027
262144	256 KB	2984	2837	2802
524288	512 KB	6170	5055	4698
1048576	1 MB	12412	10026	8822
2097152	2 MB	22419	16944	15810
4194304	4 MB	41976	32801	30352
8388608	8 MB	75276	65556	57874
16777216	16 MB	158646	124417	109865
33554432	32 MB	283186	245548	215942
67108864	64 MB	506180	481251	430665
134217728	128 MB	1016017	987726	865884

Sai Kashyap Kurella
Akash Mattaparthy



- For the fair comparison we have taken the overall time period of memcpy method, pinned memory method, Unified Virtual Memory method.
- As per the Output, we can see that time taken for memory allocation and then free them takes higher time in memcpy method then in pinned Memory method (cudaHostAlloc) and then in Unified memory method (cudaMallocManaged).
- For this observation, the graph is plotted between size of allocation from 32 bytes to 128 Megabytes and different methods for allocation.