CPU GPU Data Transfer Bandwidth using CUDA

Ali K. Rahimian

November 30, 2025

1 Introduction

This project measures the effective bandwidth between host and device memory using CUDA `cudaMemcpy` transfers. For a range of array sizes, host to device (H2D) and device to host (D2H) copies are timed using CUDA events and the achieved GB/s is reported.

Two host memory types are compared:

- Pageable host buffers allocated with `malloc`.
- Pinned (page locked) host buffers allocated with `cudaHostAlloc`.

Transfers are run for sizes from 1 MB up to 1 GB, and for each size four measurements are obtained: H2D pageable, D2H pageable, H2D pinned, and D2H pinned. The goal is to quantify the impact of host memory type and transfer direction on sustained bandwidth.

2 Methodology

For each transfer size:

1. Allocate a host buffer (pageable or pinned) and a device buffer with `cudaMalloc`.
2. Initialize the host buffer with dummy floating point data.
3. Create CUDA events, perform a warm up copy, then time `num_iters` repeated `cudaMemcpy` calls.
4. Compute effective bandwidth as

$$\text{Bandwidth (GB/s)} = \frac{\text{bytes} \times \text{num\_iters}}{\text{time (s)} \times 10^9}.$$

Array sizes are

$$1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 \text{ MB},$$

and each configuration is repeated with the same number of iterations.

3 Results

3.1 Tables and measurements

Table 1 summarizes the measured bandwidths for all sizes and configurations.

Size (MB)	Pageable		Pinned	
	H2D	D2H	H2D	D2H
1	3.846	2.118	2.506	4.864
2	11.652	6.564	9.892	8.520
4	13.352	7.090	10.056	8.928
8	19.567	12.200	22.272	15.912
16	20.608	12.727	23.510	15.352
32	19.644	13.076	12.188	15.112
64	8.120	9.064	15.961	15.407
128	5.152	13.316	16.913	15.363
256	4.525	13.428	17.027	15.364
512	4.494	13.397	17.120	15.197
1024	13.679	13.396	17.392	15.021

Table 1: Measured H2D and D2H bandwidth (GB/s) for pageable and pinned host memory across different transfer sizes.

3.2 Bandwidth vs array size plot

The CSV output from the executable is postprocessed with a short Python script using `matplotlib` to generate a bandwidth versus size plot with four curves (H2D/D2H and pageable/pinned). A representative plot is shown in Figure 1.

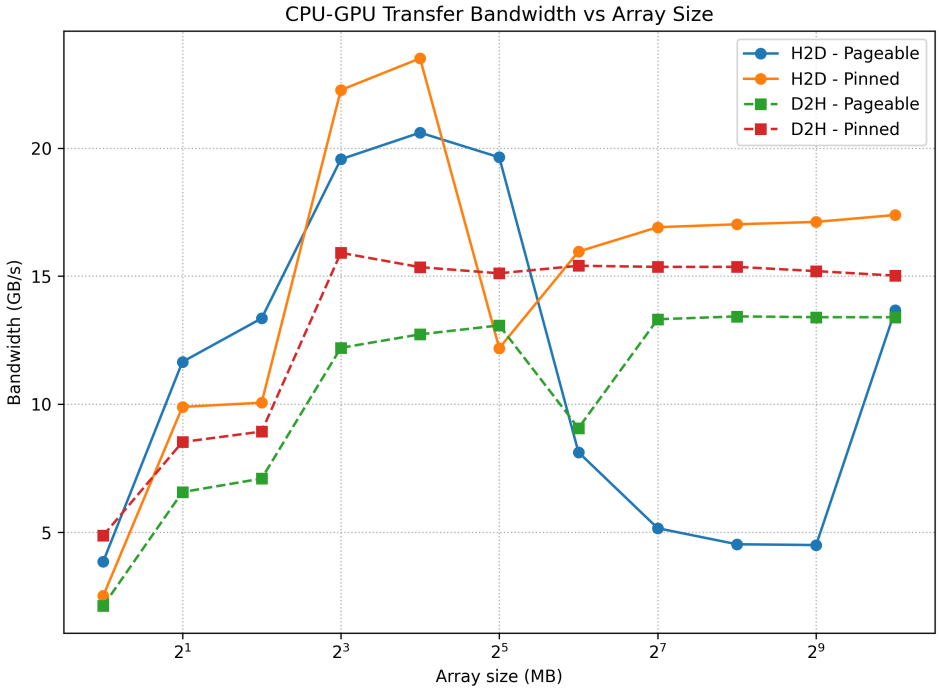

Figure 1: CPU GPU transfer bandwidth vs array size for H2D and D2H, using pageable and pinned host memory.

4 Discussion

Key observations:

- Small transfers (1 to 4 MB) have low and noisy effective bandwidth due to fixed overheads

dominating the timing.

- For medium sizes (8 to 32 MB), bandwidth rises toward a plateau that reflects the interconnect and memory subsystem throughput.
- For large sizes (64 MB and above), pinned memory clearly outperforms pageable memory, especially for H2D.

From the data:

- H2D pinned reaches about 16 to 17 GB/s for large sizes.
- D2H pinned is slightly lower, roughly 15 to 16 GB/s.
- Pageable D2H stabilizes near 13 GB/s, while pageable H2D is lower and more variable.

The advantage of pinned memory is consistent with the elimination of an internal staging copy and direct DMA access to page locked host buffers. The mild asymmetry between H2D and D2H reflects hardware and driver implementation details. For performance critical code, large transfers and pinned memory are preferred; many small transfers should be batched.

5 Source Code

5.1 cuda_utils.cuh

```cpp
#ifndef CUDA_UTILS_CUH
#define CUDA_UTILS_CUH

#include <cstdio>
#include <cstdlib>
#include <cuda_runtime.h>

#define CHECK_CUDA(call) \
    do { \
        cudaError_t err = (call); \
        if (err != cudaSuccess) { \
            fprintf(stderr, "CUDA error at %s:%d: %s\n", \
                    __FILE__, __LINE__, cudaGetErrorString(err)); \
            std::exit(EXIT_FAILURE); \
        } \
    } while (0)

#endif // CUDA_UTILS_CUH
```

5.2 bandwidth.h

```cpp
#ifndef BANDWIDTH_H
#define BANDWIDTH_H

#include <cstddef>
#include <vector>

struct BandwidthResult {
    double sizeMB; // Array size in MB
    double h2d_pageable; // GB/s
    double d2h_pageable; // GB/s
    double h2d_pinned; // GB/s
    double d2h_pinned; // GB/s
};

/**
 * Run bandwidth tests for a list of sizes.
 *
 * @param sizes Array sizes in bytes.
 * @param num_iters Number of repetitions per measurement.
 * @param results Output vector of BandwidthResult, one per size.
 */
void run_bandwidth_tests(const std::vector<size_t>& sizes,
                         int num_iters,
                         std::vector<BandwidthResult>& results);

#endif // BANDWIDTH_H
```

5.3 bandwidth.cu

```cpp
#include "bandwidth.h"
```

```cpp
#include "cuda_utils.cuh"

#include <cstdlib>

// Measure bandwidth for a single direction and memory pairing
static double measure_bandwidth_single_direction(
    void* dst,
    const void* src,
    size_t bytes,
    cudaMemcpyKind kind,
    int num_iters)
{
    cudaEvent_t start, stop;
    CHECK_CUDA(cudaEventCreate(&start));
    CHECK_CUDA(cudaEventCreate(&stop));

    // Warm up to avoid first use overhead
    CHECK_CUDA(cudaMemcpy(dst, src, bytes, kind));

    CHECK_CUDA(cudaEventRecord(start, 0));
    for (int i = 0; i < num_iters; ++i) {
        CHECK_CUDA(cudaMemcpy(dst, src, bytes, kind));
    }
    CHECK_CUDA(cudaEventRecord(stop, 0));
    CHECK_CUDA(cudaEventSynchronize(stop));

    float ms = 0.0f;
    CHECK_CUDA(cudaEventElapsedTime(&ms, start, stop));

    CHECK_CUDA(cudaEventDestroy(start));
    CHECK_CUDA(cudaEventDestroy(stop));

    double seconds = ms / 1000.0;
    double total_bytes = static_cast<double>(bytes) * num_iters;

    // Bandwidth in GB/s
    double gb_per_s = total_bytes / (seconds * 1.0e9);
    return gb_per_s;
}

// Run test for one size, filling in the 4 bandwidth numbers
static void run_test_for_size(size_t bytes,
                              int num_iters,
                              double& h2d_pageable,
                              double& d2h_pageable,
                              double& h2d_pinned,
                              double& d2h_pinned)
{
    // ----------------------------
    // Pageable host memory
    // ----------------------------
    {
        // Host allocation (pageable)
        float* h_buf = static_cast<float*>(std::malloc(bytes));
        if (!h_buf) {
            std::fprintf(stderr,
                         "Failed to allocate pageable host memory (%zu bytes)\n",
                         bytes);
```

```
            std::exit(EXIT_FAILURE);
        }

        // Initialize host data
        size_t n = bytes / sizeof(float);
        for (size_t i = 0; i < n; ++i) {
            h_buf[i] = static_cast<float>(i);
        }

        // Device allocation
        float* d_buf = nullptr;
        CHECK_CUDA(cudaMalloc(&d_buf, bytes));

        // Measure H2D pageable
        h2d_pageable = measure_bandwidth_single_direction(
            d_buf, h_buf, bytes, cudaMemcpyHostToDevice, num_iters);

        // Measure D2H pageable
        d2h_pageable = measure_bandwidth_single_direction(
            h_buf, d_buf, bytes, cudaMemcpyDeviceToHost, num_iters);

        CHECK_CUDA(cudaFree(d_buf));
        std::free(h_buf);
    }

    // ----------------------------
    // Pinned host memory
    // ----------------------------
    {
        // Host allocation (pinned)
        float* h_buf = nullptr;
        CHECK_CUDA(cudaHostAlloc(reinterpret_cast<void**>(&h_buf),
                                 bytes,
                                 cudaHostAllocDefault));

        // Initialize host data
        size_t n = bytes / sizeof(float);
        for (size_t i = 0; i < n; ++i) {
            h_buf[i] = static_cast<float>(i);
        }

        // Device allocation
        float* d_buf = nullptr;
        CHECK_CUDA(cudaMalloc(&d_buf, bytes));

        // Measure H2D pinned
        h2d_pinned = measure_bandwidth_single_direction(
            d_buf, h_buf, bytes, cudaMemcpyHostToDevice, num_iters);

        // Measure D2H pinned
        d2h_pinned = measure_bandwidth_single_direction(
            h_buf, d_buf, bytes, cudaMemcpyDeviceToHost, num_iters);

        CHECK_CUDA(cudaFree(d_buf));
        CHECK_CUDA(cudaFreeHost(h_buf));
    }
}

```

```cpp
void run_bandwidth_tests(const std::vector<size_t>& sizes,
                         int num_iters,
                         std::vector<BandwidthResult>& results)
{
    results.clear();
    results.reserve(sizes.size());

    for (size_t bytes : sizes) {
        double h2d_pageable = 0.0;
        double d2h_pageable = 0.0;
        double h2d_pinned = 0.0;
        double d2h_pinned = 0.0;

        run_test_for_size(bytes, num_iters,
                          h2d_pageable, d2h_pageable,
                          h2d_pinned, d2h_pinned);

        BandwidthResult r;
        r.sizeMB = static_cast<double>(bytes) / (1024.0 * 1024.0);
        r.h2d_pageable = h2d_pageable;
        r.d2h_pageable = d2h_pageable;
        r.h2d_pinned = h2d_pinned;
        r.d2h_pinned = d2h_pinned;

        results.push_back(r);
    }
}
```

5.4 main.cu

```cpp
#include <cstdio>
#include <vector>

#include "cuda_utils.cuh"
#include "bandwidth.h"

int main(int argc, char** argv)
{
    // Use device 0 by default
    CHECK_CUDA(cudaSetDevice(0));

    // Sizes from 1 MB to 1 GB in powers of 2:
    // 1, 2, 4, ..., 1024 MB
    const int num_sizes = 11;
    std::vector<size_t> sizes;
    sizes.reserve(num_sizes);

    for (int i = 0; i < num_sizes; ++i) {
        size_t bytes = static_cast<size_t>(1) << (20 + i); // 2^(20+i)
        sizes.push_back(bytes);
    }

    // Number of repetitions per measurement
    int num_iters = 10;

    std::vector<BandwidthResult> results;
    run_bandwidth_tests(sizes, num_iters, results);
```

```
28
29      // CSV header
30      std::printf("size_MB,h2d_pageable_GBs,d2h_pageable_GBs,"
31                  "h2d_pinned_GBs,d2h_pinned_GBs\n");
32
33      for (const auto& r : results) {
34          std::printf("%.0f,%.3f,%.3f,%.3f,%.3f\n",
35                      r.sizeMB,
36                      r.h2d_pageable,
37                      r.d2h_pageable,
38                      r.h2d_pinned,
39                      r.d2h_pinned);
40      }
41
42      CHECK_CUDA(cudaDeviceReset());
43      return 0;
44  }
```

5.5 Makefile

```
1  NVCC := nvcc
2  NVCC_FLAGS := -O2
3
4  SRCS := main.cu bandwidth.cu
5  HDRS := bandwidth.h cuda_utils.cuh
6
7  TARGET := bandwidth
8
9  $(TARGET): $(SRCS) $(HDRS)
10   $(NVCC) $(NVCC_FLAGS) $(SRCS) -o $(TARGET)
11
12 clean:
13   rm -f $(TARGET) bandwidth.csv bandwidth_plot.png
```

5.6 plot_bandwidth.py

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  # Load CSV data produced by ./bandwidth > bandwidth.csv
5  data = np.loadtxt('bandwidth.csv', delimiter=',', skiprows=1)
6
7  size_mb = data[:, 0]
8  h2d_pageable = data[:, 1]
9  d2h_pageable = data[:, 2]
10 h2d_pinned = data[:, 3]
11 d2h_pinned = data[:, 4]
12
13 plt.figure(figsize=(8, 6))
14
15 # H2D curves
16 plt.plot(size_mb, h2d_pageable, 'o-', label='H2D - Pageable')
17 plt.plot(size_mb, h2d_pinned, 'o-', label='H2D - Pinned')
18
19 # D2H curves
```

```
plt.plot(size_mb, d2h_pageable, 's--', label='D2H - Pageable')
plt.plot(size_mb, d2h_pinned, 's--', label='D2H - Pinned')

plt.xscale('log', base=2)
plt.xlabel('Array size (MB)')
plt.ylabel('Bandwidth (GB/s)')
plt.title('CPU-GPU Transfer Bandwidth vs Array Size')
plt.grid(True, which='both', linestyle=':')
plt.legend()
plt.tight_layout()
plt.savefig('bandwidth_plot.png', dpi=300)
plt.show()
```