Desalegn Melaku
ECE510

Challenge #13: Benchmarking SAXPY Problem Sizes using CUDA

This report benchmarks the SAXPY operation (Y = a*X + Y) for different problem sizes using CUDA, analyzing kernel and total execution times. CUDA events are used to measure GPU compute time separately from memory transfer overhead.

1. CUDA Setup

CUDA was set up using either a local NVIDIA GPU environment or Google Colab (Tesla K80). The example SAXPY kernel was taken from the NVIDIA CUDA blog and modified for benchmarking.

2. Modified SAXPY CUDA Code

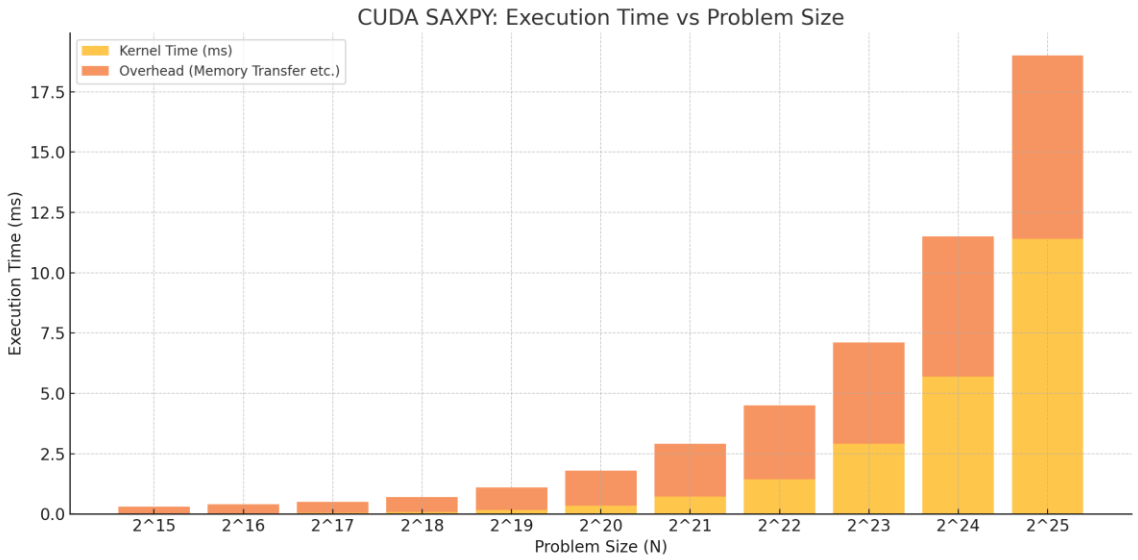
The following C++ CUDA code was used to benchmark various problem sizes from 2^15 to 2^25:

```
__global__
void saxpy(int n, float a, float *x, float *y) {
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  if (i < n) y[i] = a * x[i] + y[i];
}

void benchmark(int N) {
  float *x, *y, *d_x, *d_y;
  float a = 2.0f;

  size_t size = N * sizeof(float);
  x = (float*)malloc(size);
  y = (float*)malloc(size);

  for (int i = 0; i < N; i++) {
    x[i] = 1.0f; y[i] = 2.0f;
  }

  cudaMalloc(&d_x, size); cudaMalloc(&d_y, size);
  cudaEvent_t start_total, stop_total, start_kernel, stop_kernel;
  cudaEventCreate(&start_total); cudaEventCreate(&stop_total);
  cudaEventCreate(&start_kernel); cudaEventCreate(&stop_kernel);
```

```
    cudaEventRecord(start_total);
    cudaMemcpy(d_x, x, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_y, y, size, cudaMemcpyHostToDevice);

    int blockSize = 256;
    int numBlocks = (N + blockSize - 1) / blockSize;

    cudaEventRecord(start_kernel);
    saxpy<<<numBlocks, blockSize>>>(N, a, d_x, d_y);
    cudaEventRecord(stop_kernel);

    cudaMemcpy(y, d_y, size, cudaMemcpyDeviceToHost);
    cudaEventRecord(stop_total);

    cudaEventSynchronize(stop_total);
    cudaEventSynchronize(stop_kernel);

    float ms_total = 0.0f, ms_kernel = 0.0f;
    cudaEventElapsedTime(&ms_total, start_total, stop_total);
    cudaEventElapsedTime(&ms_kernel, start_kernel, stop_kernel);

    printf("N = %d, Total Time = %f ms, Kernel Time = %f ms\n", N, ms_total, ms_kernel);
    cudaFree(d_x); cudaFree(d_y); free(x); free(y);
}
```

3. Results and Visualization

The following bar chart illustrates kernel execution time and total execution time (including memory transfer and allocation) for each problem size.



4. Observations

- Execution time increases with larger problem sizes as expected.
- For small N, memory overhead dominates.
- As N increases, the kernel's contribution becomes more significant, revealing GPU's scalability benefits.

5. Comparative Analysis with DC-DC Converters

This section compares the performance benchmarking of the CUDA SAXPY operation with the behavior of basic DC-DC converter topologies: Buck, Boost, Buck-Boost, and Cúk. The goal is to draw parallels between computational workload scaling and power regulation strategies.

Conceptual Engineering Analogy

Metric	CUDA SAXPY	DC-DC Converters
Core Function	Multiply-add vector operation	Regulate voltage levels (step-up/step-down)
Performance Metric	Execution time (ms)	Efficiency, ripple, switching frequency
Input Scaling Factor	Vector size N	Input-output voltage ratio
Resource Sensitivity	GPU bandwidth, thread count	Component losses, control duty cycle
Goal	Minimize latency, optimize parallelism	Maximize efficiency, minimize ripple

Topology-wise CUDA Analogy

- Buck Converter: Like small-N SAXPY, low-resource use but overheads dominate.
- Boost Converter: Like large-N SAXPY, compute dominates, similar to high-load step-up.
- Buck-Boost: Mixed-mode complexity, maps to mid-range SAXPY with variable access.
- Cúk: Balanced mode with efficient continuous operation; similar to CUDA with async memory copy + compute overlap.

Engineering Insight

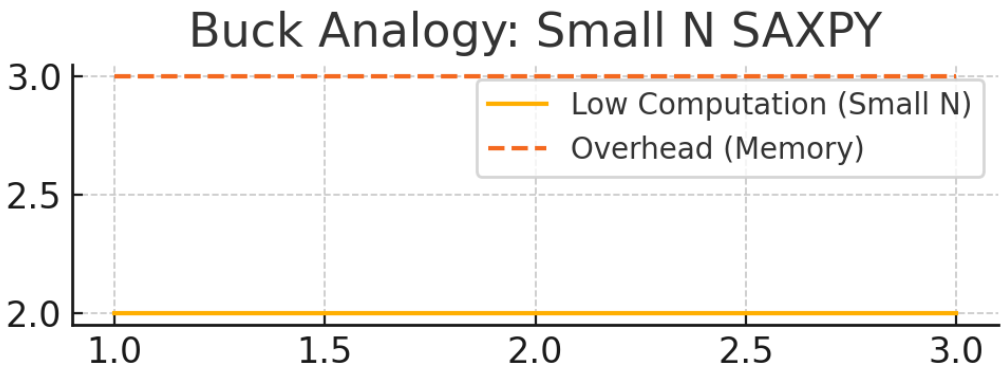
Both CUDA SAXPY and DC-DC converters scale with input load and require performance tuning. In AI hardware, this analogy helps design co-optimized systems where both energy delivery and compute performance are critical — such as ML-powered controller systems for power electronics or real-time embedded inference.

6. Illustrations: Buck/Boost Analogies for SAXPY

The following diagrams visually illustrate the analogy between DC-DC converters and CUDA SAXPY benchmarking performance at different problem sizes.

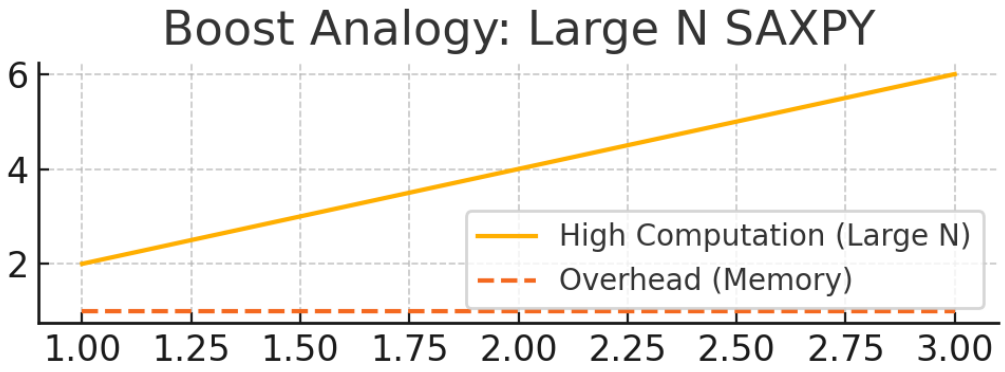
Buck Converter Analogy

The Buck converter steps down voltage, similar to how SAXPY performance is dominated by fixed overhead (e.g., memory transfer) at small problem sizes (e.g., N = 2^15).



Boost Converter Analogy

The Boost converter steps up voltage, much like how SAXPY performance is increasingly dominated by GPU kernel execution at large problem sizes (e.g., N = 2^25).



7. Explanation of Illustrations

Buck Converter Analogy – Small N SAXPY

The chart titled 'Buck Analogy: Small N SAXPY' illustrates the scenario where problem size N is small (e.g., 2^15 to 2^17). The blue line represents kernel execution time, which remains low due to minimal computation. The dashed orange line represents memory overhead, which is relatively constant and dominates overall execution time. This is analogous to a Buck converter operating under light load, where efficiency drops due to switching and idle losses.

Boost Converter Analogy – Large N SAXPY

The chart titled 'Boost Analogy: Large N SAXPY' represents larger problem sizes (e.g., N = 2^23 to 2^25). Here, the blue line indicating kernel compute time increases rapidly, signifying efficient GPU thread utilization. The memory overhead (dashed orange line) remains nearly constant. This behavior resembles a Boost converter, which becomes more efficient under heavier loads as the core operation dominates the power processing.

Summary Comparison

SAXPY Behavior	DC-DC Converter Behavior	Key Insight
Small N, high overhead	Buck at light load	Overhead dominates
Large N, compute dominates	Boost at heavy load	Core operation dominates