ECE 277: GPU Programming
Final Project Presentation

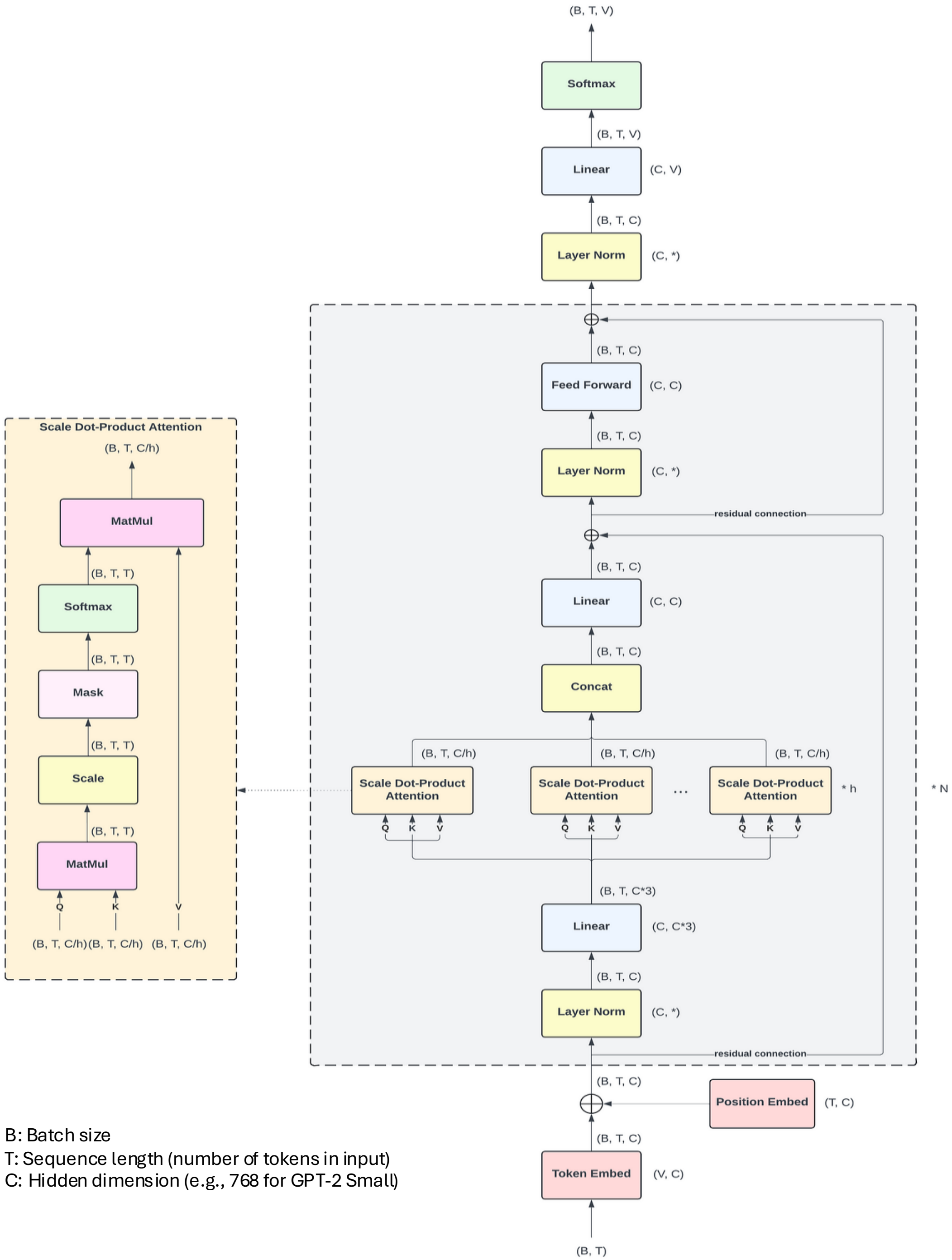
CUDA-Accelerated GPT-2 Inference Optimization

Apeksha Gaonkar

A69027235

Problem statement

- **What is GPT-2?**
  - A transformer-based language model for natural language processing tasks.
  - It is used for text generation, summarization and other NLP tasks
  - GPT-2 inference is computationally expensive operation.
  - Latency issues for real-time applications.
- **Goal**:
  - Accelerate key computations with GPU for real-time performance.
  - Focus on optimizing performance-critical components:
    1. Layer Norm
    2. Softmax
    3. Attention



B: Batch size
T: Sequence length (number of tokens in input)
C: Hidden dimension (e.g., 768 for GPT-2 Small)

1.Layer Norm

A technique that normalizes the inputs across the feature dimension within a layer to stabilize training and improve convergence.

$$\text{LayerNorm}[x] = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta,$$

Each thread processes a subset of the array.

```
__global__
void layernorm_kernel(float* out, float* x, float* w, float* b, int C){
int idx = threadIdx.x; //Thread indexing
float mean = 0;
__shared__ float s_mean[256]; //Shared memory allocation
__shared__ float s_var[256]; //Shared memory allocation
s_mean[idx] = 0.0f;
s_var[idx] = 0.0f;
__syncthreads();

for(int i = idx;i<C;i+=blockDim.x){
s_mean[idx] += x[i];
}
__syncthreads();
if(idx == 0){
float m = 0;
for(int i = 0;i<blockDim.x;i++){
m += s_mean[i];
}
m /= C;
s_mean[0] = m;
}
__syncthreads();
```

Each thread accumulates partial sums for mean and stores them in shared memory(**s_mean**).

These partial sums are reduced to compute the overall mean.

```
mean = s_mean[0];
for(int i = idx;i<C;i+=blockDim.x){
float diff = x[i] - mean;
s_var[idx] += diff * diff;
}
__syncthreads();
if(idx == 0){
float v = 0;
for(int i = 0;i<blockDim.x;i++){
v += s_var[i];
}
v /= C;
s_var[0] = v;
}
__syncthreads();
float var = s_var[0];
float scale = 1.0 / sqrt(var + 1e-6);
for(int i = idx;i<C;i+=blockDim.x){
out[i] = (x[i] - mean) * scale * w[i] + b[i];
}
}

void layernorm_gpu(float* out, float* x, float* w, float* b, int C){
int numThreads = 256;
int block = 1;
layernorm_kernel<<<block,numThreads>>>(out,x,w,b,C);
}
```

Each thread accumulates partial variance for variance and stores them in shared memory(**s_var**).

These partial variance are reduced to compute the overall variance.

Each thread normalizes its portion of x, applies scaling (w), and adds the bias (b), writing the result to out.

File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Search (Ctrl+Q) Final

Debug x64 Local Windows Debugger Live Share

attention.cu layernorm.cu softmax.cu

attention (Global Scope) isequal(float * a, float * b, int n)

```
using namespace std;

void rand_init(float* x, int n) {
    for (int i = 0; i < n; i++) {
        x[i] = (float)rand() / RAND_MAX;
    }
}

void isequal(float* a, float* b, int n) {
    float maxval = -INFINITY;
    for (int i = 0; i < n; i++) {
        maxval = fmaxf(maxval, fmaxf(a[i], b[i]));
    }
    float eps = 1e-5;

    for (int i = 0; i < n; i++) {
        if (fabs(a[i] - b[i]) > eps * (maxval + 1)) {
            cout << "Mismatches" << endl;
            for (int j = i; j < min(n, i + 10); j++) {
                cout << a[j] << " " << b[j] << endl;
            }
            return;
        }
    }
    cout << "Results match " << endl;
    for (int i = 0; i < 4; i++) {
        cout << a[i] << " " << b[i] << endl;
    }
}

void softmax(float* x, int N) {
    float max = x[0];
    for (int i = 1; i < N; i++) {
        if (x[i] > max) {
            max = x[i];
        }
    }
    float sum = 0;
```

100 % 1 0 Ln: 31 Ch: 1 SPC LF

Output

Show output from: Build

```
1>Done building project "softmax.vcxproj".
2>Done building project "layernorm.vcxproj".
3>attention.vcxproj -> H:\FinalProject_\build\Src\gpt2\Debug\attention.exe
3>Done building project "attention.vcxproj".
4>------ Skipped Build: Project: ALL_BUILD, Configuration: Debug x64 ------
4>Project not selected to build for this solution configuration
========== Build: 3 succeeded, 0 failed, 1 up-to-date, 1 skipped ==========
```

Solution Explorer

Search Solution Explorer (Ctrl+;)

- Solution 'Final' (5 of 5 projects)
  - ALL_BUILD
  - attention
  - layernorm
  - softmax
    - References
    - External Dependencies
    - CMake Rules
    - Object Files
    - Source Files
      - softmax.cu
    - CMakeLists.txt
  - ZERO_CHECK

Solution Explorer Git Changes

Properties

attention Project Properties

Misc	
(Name)	attention
Project Dependencies	
Project File	H:\FinalProject_\build\Src\gpt2\atte
Root Namespace	

(Name)
Specifies the project name.

2.Softmax

$$\mathrm{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$

Each thread processes a subset of the array.

```
__global__ void softmax_kernel(float* x, int N) {
int idx = threadIdx.x; //Thread indexing
__shared__ float smax[1024]; //Shared memory allocation
__shared__ float ssum[1024]; //Shared memory allocation
smax[idx] = -FLT_MAX;
ssum[idx] = 0.0f;
__syncthreads();

for (int i = idx; i < N; i += blockDim.x) {
smax[idx] = fmaxf(smax[idx], x[i]);
}
__syncthreads();
if (idx == 0) {
float maxval = -FLT_MAX;
for (int i = 0; i < blockDim.x; i++) {
maxval = fmaxf(maxval, smax[i]);
}
smax[0] = maxval;
}
__syncthreads();
```

Each thread computes a partial maximum using **strided access** within the loop. And stores them in shared memory(**s_max**).

Reduction in shared memory for the global maximum.

```
float maxval = smax[0];
float local_sum = 0.0f;
for (int i = idx; i < N; i += blockDim.x) {
x[i] = expf(x[i] - maxval);
local_sum += x[i];
}
ssum[idx] = local_sum;
__syncthreads();
if (idx == 0) {
float sum = 0.0f;
for (int i = 0; i < blockDim.x; i++) {
sum += ssum[i];
}
ssum[0] = sum;
}
__syncthreads();
float sum = ssum[0];
for (int i = idx; i < N; i += blockDim.x) {
x[i] /= sum;
}
}

void softmax_gpu(float* x, int N) {
int numThreads = 1024;
softmax_kernel<<<1, numThreads>>>(x, N);
}
```

Threads compute exp(x[i] - maxval) for their segment. Local sums stored in shared memory.

Thread 0 accumulates partial sums into the global sum.

Threads divide their computed exponentials by the global sum.

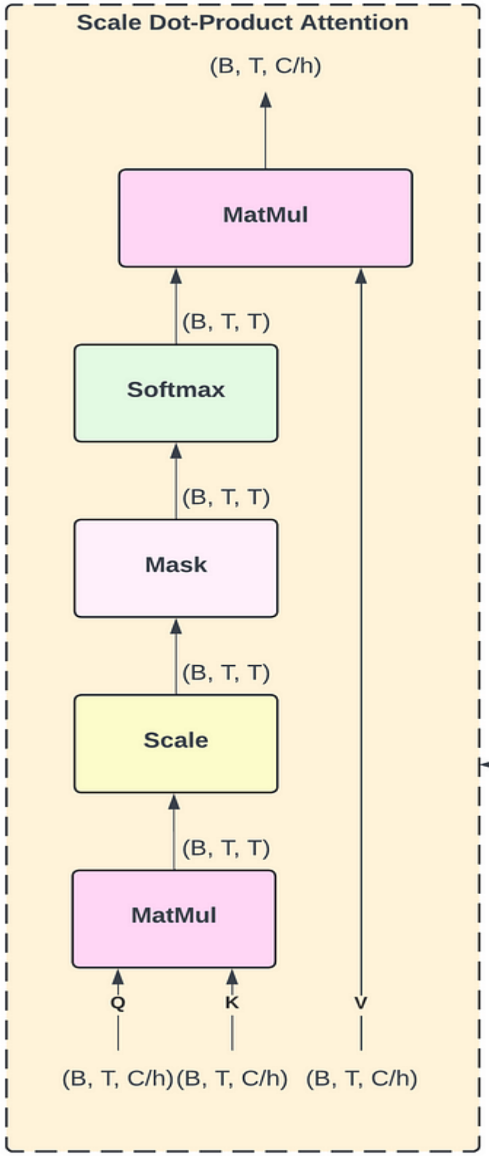
```
using namespace std;

void rand_init(float* x, int n) {
    for (int i = 0; i < n; i++) {
        x[i] = (float)rand() / RAND_MAX;
    }
}

void isequal(float* a, float* b, int n) {
    float maxval = -INFINITY;
    for (int i = 0; i < n; i++) {
        maxval = fmaxf(maxval, fmaxf(a[i], b[i]));
    }
    float eps = 1e-5;

    for (int i = 0; i < n; i++) {
        if (fabs(a[i] - b[i]) > eps * (maxval + 1)) {
            cout << "Mismatches" << endl;
            for (int j = i; j < min(n, i + 10); j++) {
                cout << a[j] << " " << b[j] << endl;
            }
            return;
        }
    }
    cout << "Results match " << endl;
    for (int i = 0; i < 4; i++) {
        cout << a[i] << " " << b[i] << endl;
    }
}

void softmax(float* x, int N) {
    float max = x[0];
    for (int i = 1; i < N; i++) {
        if (x[i] > max) {
            max = x[i];
        }
    }
    float sum = 0;
```

```
1>Done building project "softmax.vcxproj".
2>Done building project "layernorm.vcxproj".
3>attention.vcxproj -> H:\FinalProject_\build\Src\gpt2\Debug\attention.exe
3>Done building project "attention.vcxproj".
4>------ Skipped Build: Project: ALL_BUILD, Configuration: Debug x64 ------
4>Project not selected to build for this solution configuration
========== Build: 3 succeeded, 0 failed, 1 up-to-date, 1 skipped ==========
```

3. Attention Block

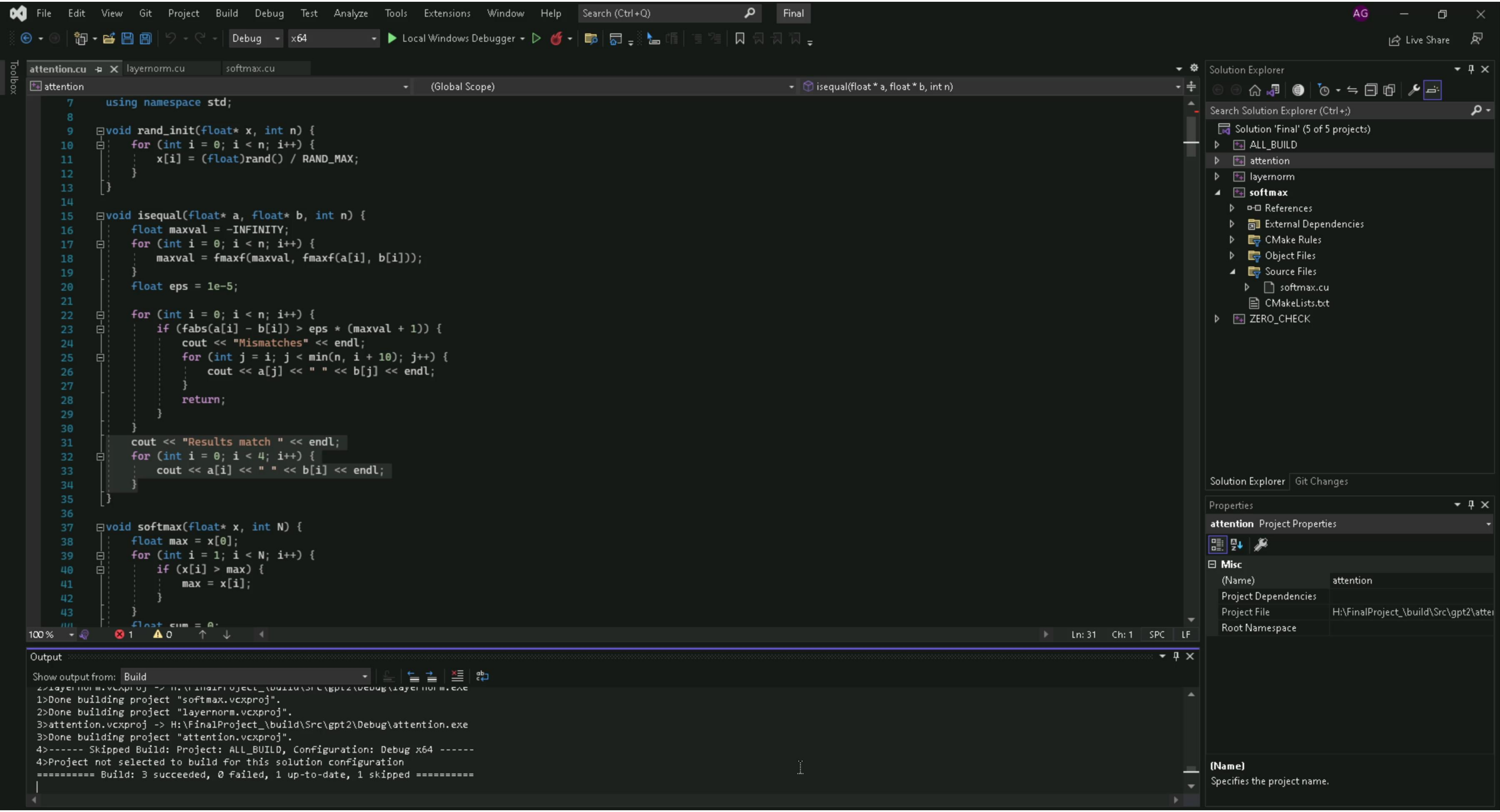
- The attention mechanism helps models focus on the most relevant parts of the input sequence when generating output.

1. **Head Size Calculation**:
   - Compute head_size = C / NH to divide the total number of features (C) evenly across the attention heads (NH).
2. **Key and Value Cache Population**:
   - The fill_cache kernel reorganizes keys and values from the qkv tensor into key_cache and value_cache.
   - This cache is reshaped to ensure memory coalescing for subsequent attention calculations.
3. **Attention Score Computation**: *att = q * K^T*
   - The compute_att_kernel computes the dot product of the query vector (q) with the key matrix (K^T) for each head.
   - Threads are parallelized across the time steps (T) and attention heads (NH).
4. **Softmax Normalization**:
   - The softmax_kernel normalizes the attention scores in-place to ensure they sum to 1.
   - Employs warp-level reductions (warpReduceMax and warpReduceSum) for efficient summation and normalization.
5. **Weighted Sum (Attention * Value)**:
   - The compute_out_kernel computes the weighted sum of the values (V) based on the normalized attention scores (att) to produce the output tensor (out).



Cuda optimization techniques used

- **Memory Coalescing**
  - **Fill Cache Kernel:** Rearranges keys and values into a layout that enables coalesced memory access during attention computation.
  - Access patterns are aligned for efficient global memory reads and writes.
- **Parallelism**
  - Each CUDA block handles one attention head.
  - Threads within a block handle computations for time steps (TTT) or head size (HSHSHS).
- **Warp-Level Primitives**
  - **Softmax Kernel:** Utilizes __shfl_sync for warp-level reductions to compute max and sum values efficiently.
  - Reduces latency for operations like summation and normalization.
- **Shared Memory Usage**
  - **Softmax Kernel:** Uses shared memory for intermediate storage of max and sum values.
  - Reduces global memory transactions, improving speed.
- **Loop Unrolling**
  - **Dot Product and Weighted Sum Kernels:** Loops over head size and sequence positions are unrolled to maximize instruction-level parallelism.
- **Thread Count Management**
  - Caps threads per block at 1024 to comply with hardware constraints.
  - Dynamically adjusts the number of threads for sequence positions (pos) and head size (HS).



File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
Final
Debug
x64
Local Windows Debugger
Live Share
attention.cu
layernorm.cu
softmax.cu
attention
(Global Scope)
isequal(float * a, float * b, int n)
using namespace std;

void rand_init(float* x, int n) {
    for (int i = 0; i < n; i++) {
        x[i] = (float)rand() / RAND_MAX;
    }
}

void isequal(float* a, float* b, int n) {
    float maxval = -INFINITY;
    for (int i = 0; i < n; i++) {
        maxval = fmaxf(maxval, fmaxf(a[i], b[i]));
    }
    float eps = 1e-5;

    for (int i = 0; i < n; i++) {
        if (fabs(a[i] - b[i]) > eps * (maxval + 1)) {
            cout << "Mismatches" << endl;
            for (int j = i; j < min(n, i + 10); j++) {
                cout << a[j] << " " << b[j] << endl;
            }
            return;
        }
    }
    cout << "Results match " << endl;
    for (int i = 0; i < 4; i++) {
        cout << a[i] << " " << b[i] << endl;
    }
}

void softmax(float* x, int N) {
    float max = x[0];
    for (int i = 1; i < N; i++) {
        if (x[i] > max) {
            max = x[i];
        }
    }
100 %
Ln: 31 Ch: 1 SPC LF
Output
Show output from: Build
1>Done building project "softmax.vcxproj".
2>Done building project "layernorm.vcxproj".
3>attention.vcxproj -> H:\FinalProject_\build\Src\gpt2\Debug\attention.exe
3>Done building project "attention.vcxproj".
4>------ Skipped Build: Project: ALL_BUILD, Configuration: Debug x64 ------
4>Project not selected to build for this solution configuration
========== Build: 3 succeeded, 0 failed, 1 up-to-date, 1 skipped ==========
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Final' (5 of 5 projects)
ALL_BUILD
attention
layernorm
softmax
References
External Dependencies
CMake Rules
Object Files
Source Files
softmax.cu
CMakeLists.txt
ZERO_CHECK
Solution Explorer Git Changes
Properties
attention Project Properties
Misc
(Name) attention
Project Dependencies
Project File H:\FinalProject_\build\Src\gpt2\atter
Root Namespace
(Name)
Specifies the project name.

Performance Bench Marking

	Layer-Norm	Softmax	Attention
CPU	0.0072ms	2.1551ms	5.4209ms
GPU	0.3502ms	1.6969ms	1.4694ms

Thank you!

Thank you!