softmax 的 cuda 优化

何文昊

August 2024

1 优化策略概览

这个项目需要对 softmax 这一深度学习中常用的函数进行 cuda 层面的并行优化。

一般来说，softmax 的计算逻辑可以分为下面三个步骤：

1. 对于固定的某行，需要遍历当前行的所有元素获取最大值；
2. 获得最大值以后，需要计算数值和；
3. 指数变换缩放，得到最终的数值结果。

按照 softmax 的计算过程，我尝试了以下几种优化方式，最终性能均优于 baseline，且不同的并行策略在不同数据规模下各有优势 (由于 softmax 的计算特性，网格和线程块只使用一维的):

1. 使用不同并行规模: 1 个 warp 处理 1 行，1 个 warp 处理 2 行，1 个 warp 处理 4 行
2. 使用 `__shfl_down_sync()` 和 `__shfl_sync()` 原语进行向量规约，在 1 个 warp 处理 1 行时可以使数据储存在寄存器内 (之前是存在共享内存里)，优化访存。
3. 利用 `__shfl_down_sync()` 和 `__shfl_sync()`的同步特性, 减少耗时较多的 syncthreads()
4. 利用 float4 对每一种并行策略进行访存优化
5. 减少 CPU 和 GPU 之间来回搬数据的次数和 malloc 次数，在所有策略执行之前先 malloc 将数据搬到 GPU 上

2 优化方法实现

下面提供每一种策略的核函数实现

2.1 每个 warp 处理一行数据

2.1.1 普通版本

```
__global__ void softmax_one_warp_one_row(float *input, float *output, int M, int N)
{
    int row = blockIdx.x; // 获取当前处理的行索引
    float val = -__FLT_MAX__;

    // 找到当前行的最大值
    for (int i = threadIdx.x; i < N; i += BLOCK_DIM_WARP)
    {
        val = max(val, input[row * N + i]); // 每个线程处理一行中的一部分数据，找出局部
    }
    __syncthreads(); // 同步线程

    // 使用warp内的__shfl_down_sync原语进行归约，找出全局最大值
    for (int offset = BLOCK_DIM_WARP / 2; offset > 0; offset /= 2)
    {
        val = max(val, __shfl_down_sync(0xffffffff, val, offset)); // 向量规约
    }
    float globalMax = __shfl_sync(0xffffffff, val, 0); // 在寄存器里获取全局最大值

    val = 0.0f; // 赋值为0，因为val在后面会用到
    // 计算指数和
    for (int i = threadIdx.x; i < N; i += BLOCK_DIM_WARP)
    {
        val += __expf(input[row * N + i] - globalMax); // 计算指数并累加
    }
    __syncthreads();
    // 计算全局指数和
    for (int offset = BLOCK_DIM_WARP / 2; offset > 0; offset /= 2)
    {
        val += __shfl_down_sync(0xffffffff, val, offset); /
    }
    float globalSum = __shfl_sync(0xffffffff, val, 0); // 在寄存器里获取全局指数和

    for (int i = threadIdx.x; i < N; i += BLOCK_DIM_WARP)
    {
        output[row * N + i] = __expf(input[row * N + i] - globalMax) / globalSum;
    }
}
```

2.1.2 float4 优化版本

```
    __global__ void softmax_one_warp_one_row_float4(float *input, float *output, int M,
{
    int row_id = blockIdx.x; // 获取当前处理的行索引
    int lane_id = threadIdx.x % 32; // 获取线程在warp中的位置

    float4 val4 = make_float4(-__FLT_MAX__, -__FLT_MAX__, -__FLT_MAX__, -__FLT_MAX__);

    for (int i = lane_id * 4; i < N; i += 32 * 4)
    {
        float4 data = *reinterpret_cast<float4*>(&input[row_id * N + i]); // 使用float4
        // 比较4个值并获取局部最大值
        val4.x = max(val4.x, data.x);
        val4.y = max(val4.y, data.y);
        val4.z = max(val4.z, data.z);
        val4.w = max(val4.w, data.w);
    }
    float max_val = max(max(val4.x, val4.y), max(val4.z, val4.w)); // 计算局部最大值
```

```
for (int offset = 16; offset > 0; offset /= 2)
{
    max_val = max(max_val, __shfl_down_sync(0xffffffff, max_val, offset));
}

__shared__ float warpMax; // 共享内存用于存储全局最大值
if (lane_id == 0)
{
    warpMax = max_val; // 仅warp内第一个线程写入全局最大值
}
__syncthreads();

float4 sum4 = make_float4(0.0f, 0.0f, 0.0f, 0.0f);

for (int i = lane_id * 4; i < N; i += 32 * 4)
{
    float4 data = *reinterpret_cast<float4*>(&input[row_id * N + i]);
    // 计算指数和
    sum4.x += __expf(data.x - warpMax);
    sum4.y += __expf(data.y - warpMax);
    sum4.z += __expf(data.z - warpMax);
    sum4.w += __expf(data.w - warpMax);
}

float sum_val = sum4.x + sum4.y + sum4.z + sum4.w; // 计算局部指数和
for (int offset = 16; offset > 0; offset /= 2)
{
    sum_val += __shfl_down_sync(0xffffffff, sum_val, offset);
}

__shared__ float warpSum;
if (lane_id == 0)
{
        warpSum = sum_val;
    }
    __syncthreads();

    for (int i = lane_id * 4; i < N; i += 32 * 4)
    {
        float4 data = *reinterpret_cast<float4*>(&input[row_id * N + i]);
        data.x = __expf(data.x - warpMax) / warpSum;
        data.y = __expf(data.y - warpMax) / warpSum;
        data.z = __expf(data.z - warpMax) / warpSum;
        data.w = __expf(data.w - warpMax) / warpSum;
        *reinterpret_cast<float4*>(&output[row_id * N + i]) = data;
    }
}
```

2.2 每个 warp 处理两行数据

2.2.1 普通版本

```
__global__ void softmax_one_warp_two_row(float *input, float *output, int M, int N)
{
    constexpr int size = 2; // 每个warp处理的行数
    constexpr int group_size = BLOCK_DIM_WARP / size; // 每个warp内的线程数
    int row_group_id = blockIdx.x; // 获取当前处理的行组索引
    int warp_group_id = threadIdx.x / group_size; // 获取线程在warp组内的索引
    int id_in_warp = threadIdx.x % group_size; // 获取线程在组内的位置
    int gap = group_size; // 每个线程处理的步长

    __shared__ float globalMax[size]; // 共享内存用于存储每行的最大值
    __shared__ float globalSum[size]; // 共享内存用于存储每行的指数和

    float val = -__FLT_MAX__;

    for (int i = id_in_warp; i < N; i += gap)
    {
```

```
    val = max(val, input[row_group_id * size * N + warp_group_id * N + i]);
}

// 16个线程进行同步规约，找到每行的全局最大值
for (int offset = gap / 2; offset > 0; offset /= 2)
{
    val = max(val, __shfl_down_sync(0xffff, val, offset, group_size));
}
if (id_in_warp == 0)
{
    globalMax[warp_group_id] = val;
}
__syncthreads();

val = 0.0f;

for (int i = id_in_warp; i < N; i += gap)
{
    val += __expf(input[row_group_id * size * N + warp_group_id * N + i] - globalMa
}

for (int offset = gap / 2; offset > 0; offset /= 2)
{
    val += __shfl_down_sync(0xffff, val, offset, group_size);
}

if (id_in_warp == 0)
{
    globalSum[warp_group_id] = val;
}
__syncthreads();

for (int i = id_in_warp; i < N; i += gap)
```

```
    {
        output[row_group_id * size * N + warp_group_id * N + i] = __expf(input[row_grou
    }
}
```

2.2.2 float4 优化版本

```
__global__ void softmax_one_warp_two_row_float4(float *input, float *output, int M, int
{
    constexpr int size = 2;
    constexpr int group_size = BLOCK_DIM_WARP / size;
    int row_group_id = blockIdx.x;
    int warp_group_id = threadIdx.x / group_size;
    int id_in_warp = threadIdx.x % group_size;
    int gap = group_size;

    __shared__ float globalMax[size];
    __shared__ float globalSum[size];

    float4 val4 = make_float4(-__FLT_MAX__, -__FLT_MAX__, -__FLT_MAX__, -__FLT_MAX__);
    for (int i = id_in_warp * 4; i < N; i += gap * 4)
    {
        float4 data = *reinterpret_cast<float4 *>(&input[row_group_id * size * N + warp
        val4.x = max(val4.x, data.x);
        val4.y = max(val4.y, data.y);
        val4.z = max(val4.z, data.z);
        val4.w = max(val4.w, data.w);
    }
    float val = max(max(val4.x, val4.y), max(val4.z, val4.w));
    for (int offset = gap / 2; offset > 0; offset /= 2)
    {
        val = max(val, __shfl_down_sync(0xffffffff, val, offset, group_size));
    }
    if (id_in_warp == 0)
```

```
{
    globalMax[warp_group_id] = val;
}
__syncthreads();

float4 sum4 = make_float4(0.0f, 0.0f, 0.0f, 0.0f);
for (int i = id_in_warp * 4; i < N; i += gap * 4)
{
    float4 data = *reinterpret_cast<float4 *>(&input[row_group_id * size * N + warp
    sum4.x += __expf(data.x - globalMax[warp_group_id]);
    sum4.y += __expf(data.y - globalMax[warp_group_id]);
    sum4.z += __expf(data.z - globalMax[warp_group_id]);
    sum4.w += __expf(data.w - globalMax[warp_group_id]);
}
float sum = sum4.x + sum4.y + sum4.z + sum4.w;
for (int offset = gap / 2; offset > 0; offset /= 2)
{
    sum += __shfl_down_sync(0xffffffff, sum, offset, group_size);
}
if (id_in_warp == 0)
{
    globalSum[warp_group_id] = sum;
}
__syncthreads();

for (int i = id_in_warp * 4; i < N; i += gap * 4)
{
    float4 data = *reinterpret_cast<float4 *>(&input[row_group_id * size * N + warp
    data.x = __expf(data.x - globalMax[warp_group_id]) / globalSum[warp_group_id];
    data.y = __expf(data.y - globalMax[warp_group_id]) / globalSum[warp_group_id];
    data.z = __expf(data.z - globalMax[warp_group_id]) / globalSum[warp_group_id];
    data.w = __expf(data.w - globalMax[warp_group_id]) / globalSum[warp_group_id];
    *reinterpret_cast<float4 *>(&output[row_group_id * size * N + warp_group_id * N
    }
}
```

2.3 每个 warp 处理四行数据

2.3.1 普通版本

```
__global__ void softmax_one_warp_four_row(float *input, float *output, int M, int N)
{
    constexpr int size = 4;
    constexpr int group_size = BLOCK_DIM_WARP / size;
    int row_group_id = blockIdx.x;
    int warp_group_id = threadIdx.x / group_size;
    int id_in_warp = threadIdx.x % group_size;
    int gap = group_size;

    __shared__ float globalMax[size];
    __shared__ float globalSum[size];

    float val = -__FLT_MAX__;
    for (int i = id_in_warp; i < N; i += gap)
    {
        val = max(val, input[row_group_id * size * N + warp_group_id * N + i]);
    }
    for (int offset = gap / 2; offset > 0; offset /= 2)
    {
        val = max(val, __shfl_down_sync(0xff, val, offset, 8));
    }
    if (id_in_warp == 0)
    {
        globalMax[warp_group_id] = val;
    }
    __syncthreads();

    val = 0.0f;
    for (int i = id_in_warp; i < N; i += gap)
    {
        val += __expf(input[row_group_id * size * N + warp_group_id * N + i] - globalMa
    }
    for (int offset = gap / 2; offset > 0; offset /= 2)
    {
        val += __shfl_down_sync(0xff, val, offset, 8);
    }
    if (id_in_warp == 0)
    {
        globalSum[warp_group_id] = val;
    }
    __syncthreads();

    for (int i = id_in_warp; i < N; i += gap)
    {
        output[row_group_id * 4 * N + warp_group_id * N + i] = __expf(input[row_group_i
    }
}
```

2.3.2 float4 优化版本

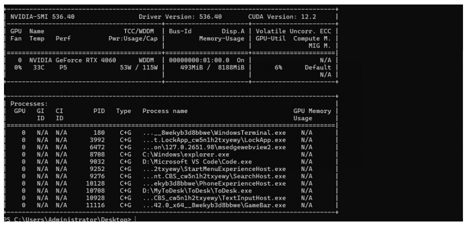
```
__global__ void softmax_one_warp_four_row_float4(float *input, float *output, int M, in
{
    constexpr int size = 4;
    constexpr int group_size = BLOCK_DIM_WARP / size;
    int row_group_id = blockIdx.x;
    int warp_group_id = threadIdx.x / group_size;
    int id_in_warp = threadIdx.x % group_size;
    int gap = group_size;

    __shared__ float globalMax[size];
    __shared__ float globalSum[size];
```

```
float4 val4 = make_float4(-__FLT_MAX__, -__FLT_MAX__, -__FLT_MAX__, -__FLT_MAX__);
for (int i = id_in_warp * 4; i < N; i += gap * 4)
{
    float4 data = *reinterpret_cast<float4 *>(&input[row_group_id * size * N + warp
    val4.x = max(val4.x, data.x);
    val4.y = max(val4.y, data.y);
    val4.z = max(val4.z, data.z);
    val4.w = max(val4.w, data.w);
}
float val = max(max(val4.x, val4.y), max(val4.z, val4.w));
for (int offset = gap / 2; offset > 0; offset /= 2)
{
    val = max(val, __shfl_down_sync(0xffffffff, val, offset, group_size));
}
if (id_in_warp == 0)
{
    globalMax[warp_group_id] = val;
}
__syncthreads();

float4 sum4 = make_float4(0.0f, 0.0f, 0.0f, 0.0f);
for (int i = id_in_warp * 4; i < N; i += gap * 4)
{
    float4 data = *reinterpret_cast<float4 *>(&input[row_group_id * size * N + warp
    sum4.x += __expf(data.x - globalMax[warp_group_id]);
    sum4.y += __expf(data.y - globalMax[warp_group_id]);
    sum4.z += __expf(data.z - globalMax[warp_group_id]);
    sum4.w += __expf(data.w - globalMax[warp_group_id]);
}
float sum = sum4.x + sum4.y + sum4.z + sum4.w;
for (int offset = gap / 2; offset > 0; offset /= 2)
{
    sum += __shfl_down_sync(0xffffffff, sum, offset, group_size);
    }
    if (id_in_warp == 0)
    {
        globalSum[warp_group_id] = sum;
    }
    __syncthreads();

    for (int i = id_in_warp * 4; i < N; i += gap * 4)
    {
        float4 data = *reinterpret_cast<float4 *>(&input[row_group_id * size * N + warp
        data.x = __expf(data.x - globalMax[warp_group_id]) / globalSum[warp_group_id];
        data.y = __expf(data.y - globalMax[warp_group_id]) / globalSum[warp_group_id];
        data.z = __expf(data.z - globalMax[warp_group_id]) / globalSum[warp_group_id];
        data.w = __expf(data.w - globalMax[warp_group_id]) / globalSum[warp_group_id];
        *reinterpret_cast<float4 *>(&output[row_group_id * size * N + warp_group_id * N
    }
}

__global__ void softmax_one_warp_two_row_float4(float *input, float *output, int M, int
{
    constexpr int size = 2;
    constexpr int group_size = BLOCK_DIM_WARP / size;
    int row_group_id = blockIdx.x;
    int warp_group_id = threadIdx.x / group_size;
    int id_in_warp = threadIdx.x % group_size;
    int gap = group_size;

    __shared__ float globalMax[size];
    __shared__ float globalSum[size];

    float4 val4 = make_float4(-__FLT_MAX__, -__FLT_MAX__, -__FLT_MAX__, -__FLT_MAX__);
    for (int i = id_in_warp * 4; i < N; i += gap * 4)
    {
        float4 data = *reinterpret_cast<float4 *>(&input[row_group_id * size * N + warp
        val4.x = max(val4.x, data.x);
        val4.y = max(val4.y, data.y);
        val4.z = max(val4.z, data.z);
        val4.w = max(val4.w, data.w);
    }
    float val = max(max(val4.x, val4.y), max(val4.z, val4.w));
    for (int offset = gap / 2; offset > 0; offset /= 2)
    {
        val = max(val, __shfl_down_sync(0xffffffff, val, offset, group_size));
    }
    if (id_in_warp == 0)
    {
        globalMax[warp_group_id] = val;
    }
    __syncthreads();

    float4 sum4 = make_float4(0.0f, 0.0f, 0.0f, 0.0f);
    for (int i = id_in_warp * 4; i < N; i += gap * 4)
    {
        float4 data = *reinterpret_cast<float4 *>(&input[row_group_id * size * N + warp
        sum4.x += __expf(data.x - globalMax[warp_group_id]);
        sum4.y += __expf(data.y - globalMax[warp_group_id]);
        sum4.z += __expf(data.z - globalMax[warp_group_id]);
        sum4.w += __expf(data.w - globalMax[warp_group_id]);
    }
    float sum = sum4.x + sum4.y + sum4.z + sum4.w;
    for (int offset = gap / 2; offset > 0; offset /= 2)
    {
        sum += __shfl_down_sync(0xffffffff, sum, offset, group_size);
    }
    if (id_in_warp == 0)
    {
        globalSum[warp_group_id] = sum;
    }
    __syncthreads();

    for (int i = id_in_warp * 4; i < N; i += gap * 4)
    {
        float4 data = *reinterpret_cast<float4 *>(&input[row_group_id * size * N + warp
        data.x = __expf(data.x - globalMax[warp_group_id]) / globalSum[warp_group_id];
        data.y = __expf(data.y - globalMax[warp_group_id]) / globalSum[warp_group_id];
        data.z = __expf(data.z - globalMax[warp_group_id]) / globalSum[warp_group_id];
        data.w = __expf(data.w - globalMax[warp_group_id]) / globalSum[warp_group_id];
        *reinterpret_cast<float4 *>(&output[row_group_id * size * N + warp_group_id * N
    }
}
```

3 结果分析

3.1 运行环境

Geforce RTX 4060 8G



```
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 536.40                 Driver Version: 536.40       CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                     TCC/WDDM  | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  NVIDIA GeForce RTX 4060      WDDM  | 00000000:01:00.0  On |                  N/A |
|  0%   33C    P5              53W / 115W |    493MiB /  8188MiB |      6%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|    0   N/A  N/A       180    C+G   ...__8wekyb3d8bbwe\WindowsTerminal.exe    N/A      |
|    0   N/A  N/A      3992    C+G   ...t.LockApp_cw5n1h2txyewy\LockApp.exe    N/A      |
|    0   N/A  N/A      6472    C+G   ...on\127.0.2651.98\msedgewebview2.exe    N/A      |
|    0   N/A  N/A      8708    C+G   C:\Windows\explorer.exe                   N/A      |
|    0   N/A  N/A      9032    C+G   D:\Microsoft VS Code\Code.exe             N/A      |
|    0   N/A  N/A      9252    C+G   ...2txyewy\StartMenuExperienceHost.exe    N/A      |
|    0   N/A  N/A      9276    C+G   ...nt.CBS_cw5n1h2txyewy\SearchHost.exe    N/A      |
|    0   N/A  N/A     10128    C+G   ...ekyb3d8bbwe\PhoneExperienceHost.exe    N/A      |
|    0   N/A  N/A     10708    C+G   D:\MyToDesk\ToDesk\ToDesk.exe             N/A      |
|    0   N/A  N/A     10928    C+G   ...CBS_cw5n1h2txyewy\TextInputHost.exe    N/A      |
|    0   N/A  N/A     11116    C+G   ...42.0_x64__8wekyb3d8bbwe\GameBar.exe    N/A      |
+---------------------------------------------------------------------------------------+
PS C:\Users\Administrator\Desktop>
```

图 1: 显卡配置

3.2 分析

综合来看，1 个 warp 处理一行的表现最好，但是在行数比列数大得多的情况下（如 2024，32 1024，32）的情况下,1 个 warp 处理多行的效果更好，并且在这个项目的数据范围内，一个 warp 处理 4 行优于一个 warp 处

理 2 行。此外，使用 float4 优化后的效果几乎总是优于不使用的情况。根据所有数据规模，最好的优化效果相比于 baseline 每次都有 2-8 倍的性能提升 (详细数据见下一部分)

3.3 数据展示

3.3.1 M=32, N=32

baseline:

M = 32, N = 32

average kernel time: 0.0095 ms, average total time: 0.0247 ms

2.6002e-05 7.0681e-05 1.9213e-04 5.2226e-04 1.4197e-03 3.8590e-03 1.0490e-02 2.8515e-02 7.7511e-02 2.1070e-01

one warp one row:

M = 32, N = 32

average kernel time: 0.0044 ms, average total time: 0.0221 ms

Difference from baseline: 0.0000e+00

one warp one row float4 version:

M = 32, N = 32

average kernel time: 0.0053 ms, average total time: 0.0222 ms

Difference from baseline: 0.0000e+00

one warp two rows:

M = 32, N = 32

average kernel time: 0.0058 ms, average total time: 0.0234 s

Difference from baseline: 0.0000e+00

one warp two rows float4 version:

M = 32, N = 32

average kernel time: 0.0049 ms, average total time: 0.0202 s

Difference from baseline: 0.0000e+00

one warp four row:

M = 32, N = 32

average kernel time: 0.0050 ms, average total time: 0.0219 ms

Difference from baseline: 0.0000e+00

one warp four row float4 version:

M = 32, N = 32

average kernel time: 0.0047 ms, average total time: 0.0221 ms
Difference from baseline: 0.0000e+00

3.3.2 M=32, N=1024

baseline:
M = 32, N = 1024
average kernel time: 0.0093 ms, average total time: 0.0250 ms
7.6482e-07 2.0790e-06 5.6513e-06 1.5362e-05 4.1758e-05 1.1351e-04 3.0855e-04 8.3873e-04 2.2799e-03 6.1974e-03

one warp one row:
M = 32, N = 1024
average kernel time: 0.0073 ms, average total time: 0.0264 ms
Difference from baseline: 0.0000e+00

one warp one row float4 version:
M = 32, N = 1024
average kernel time: 0.0074 ms, average total time: 0.0247 ms
Difference from baseline: 0.0000e+00

one warp two rows:
M = 32, N = 1024
average kernel time: 0.0092 ms, average total time: 0.0251 s
Difference from baseline: 0.0000e+00

one warp two rows float4 version:
M = 32, N = 1024
average kernel time: 0.0075 ms, average total time: 0.0228 s
Difference from baseline: 0.0000e+00

one warp four row:
M = 32, N = 1024
average kernel time: 0.0157 ms, average total time: 0.0310 ms
Difference from baseline: 0.0000e+00

one warp four row float4 version:
M = 32, N = 1024
average kernel time: 0.0105 ms, average total time: 0.0275 ms
Difference from baseline: 0.0000e+00

3.3.3 M=32, N=2048

baseline:

M = 32, N = 2048

average kernel time: 0.0102 ms, average total time: 0.0269 ms

3.8217e-07 1.0388e-06 2.8238e-06 7.6760e-06 2.0866e-05 5.6718e-05 1.5418e-04 4.1910e-04 1.1392e-03 3.0967e-03

one warp one row:

M = 32, N = 2048

average kernel time: 0.0090 ms, average total time: 0.0265 ms

Difference from baseline: 0.0000e+00

one warp one row float4 version:

M = 32, N = 2048

average kernel time: 0.0084 ms, average total time: 0.0259 ms

Difference from baseline: 0.0000e+00

one warp two rows:

M = 32, N = 2048

average kernel time: 0.0125 ms, average total time: 0.0288 s

Difference from baseline: 0.0000e+00

one warp two rows float4 version:

M = 32, N = 2048

average kernel time: 0.0108 ms, average total time: 0.0276 s

Difference from baseline: 0.0000e+00

one warp four row:

M = 32, N = 2048

average kernel time: 0.0230 ms, average total time: 0.0396 ms

Difference from baseline: 0.0000e+00

one warp four row float4 version:

M = 32, N = 2048

average kernel time: 0.0167 ms, average total time: 0.0328 ms

Difference from baseline: 0.0000e+00

3.3.4 M=1024, N=32

baseline:

M = 1024, N = 32

average kernel time: 0.1045 ms, average total time: 0.1248 ms

2.6002e-05 7.0681e-05 1.9213e-04 5.2226e-04 1.4197e-03 3.8590e-03 1.0490e-02 2.8515e-02 7.7511e-02 2.1070e-01

one warp one row:

M = 1024, N = 32

average kernel time: 0.0060 ms, average total time: 0.0211 ms

Difference from baseline: 0.0000e+00

one warp one row float4 version:

M = 1024, N = 32

average kernel time: 0.0072 ms, average total time: 0.0267 ms

Difference from baseline: 0.0000e+00

one warp two rows:

M = 1024, N = 32

average kernel time: 0.0062 ms, average total time: 0.0255 s

Difference from baseline: 0.0000e+00

one warp two rows float4 version:

M = 1024, N = 32

average kernel time: 0.0057 ms, average total time: 0.0248 s

Difference from baseline: 0.0000e+00

one warp four row:

M = 1024, N = 32

average kernel time: 0.0052 ms, average total time: 0.0269 ms

Difference from baseline: 0.0000e+00

one warp four row float4 version:

M = 1024, N = 32

average kernel time: 0.0045 ms, average total time: 0.0186 ms

Difference from baseline: 0.0000e+00

3.3.5 M=1024, N=1024

baseline:

```
M = 1024, N = 1024
average kernel time: 0.1155 ms, average total time: 0.1362 ms
7.6482e-07 2.0790e-06 5.6513e-06 1.5362e-05 4.1758e-05 1.1351e-04 3.0855e-
04 8.3873e-04 2.2799e-03 6.1974e-03
```

one warp one row:

```
M = 1024, N = 1024
average kernel time: 0.0167 ms, average total time: 0.0317 ms
Difference from baseline: 0.0000e+00
```

one warp one row float4 version:

```
M = 1024, N = 1024
average kernel time: 0.0173 ms, average total time: 0.0358 ms
Difference from baseline: 0.0000e+00
```

one warp two rows:

```
M = 1024, N = 1024
average kernel time: 0.0209 ms, average total time: 0.0424 s
Difference from baseline: 0.0000e+00
```

one warp two rows float4 version:

```
M = 1024, N = 1024
average kernel time: 0.0206 ms, average total time: 0.0410 s
Difference from baseline: 0.0000e+00
```

one warp four row:

```
M = 1024, N = 1024
average kernel time: 0.0305 ms, average total time: 0.0530 ms
Difference from baseline: 0.0000e+00
```

one warp four row float4 version:

```
M = 1024, N = 1024
average kernel time: 0.0216 ms, average total time: 0.0406 ms
Difference from baseline: 0.0000e+00
```

3.3.6 M=1024, N=2048

baseline:

M = 1024, N = 2048

average kernel time: 0.1254 ms, average total time: 0.1486 ms

3.8217e-07 1.0388e-06 2.8238e-06 7.6760e-06 2.0866e-05 5.6718e-05 1.5418e-04 4.1910e-04 1.1392e-03 3.0967e-03

one warp one row:

M = 1024, N = 2048

average kernel time: 0.0368 ms, average total time: 0.0542 ms

Difference from baseline: 0.0000e+00

one warp one row float4 version:

M = 1024, N = 2048

average kernel time: 0.0301 ms, average total time: 0.0520 ms

Difference from baseline: 0.0000e+00

one warp two rows:

M = 1024, N = 2048

average kernel time: 0.0322 ms, average total time: 0.0495 s

Difference from baseline: 0.0000e+00

one warp two rows float4 version:

M = 1024, N = 2048

average kernel time: 0.0302 ms, average total time: 0.0467 s

Difference from baseline: 0.0000e+00

one warp four row:

M = 1024, N = 2048

average kernel time: 0.0564 ms, average total time: 0.0739 ms

Difference from baseline: 0.0000e+00

one warp four row float4 version:

M = 1024, N = 2048

average kernel time: 0.0360 ms, average total time: 0.0566 ms

Difference from baseline: 0.0000e+00

3.3.7 M=2048, N=32

baseline:

M = 2048, N = 32

average kernel time: 0.2050 ms, average total time: 0.2271 ms

2.6002e-05 7.0681e-05 1.9213e-04 5.2226e-04 1.4197e-03 3.8590e-03 1.0490e-02 2.8515e-02 7.7511e-02 2.1070e-01

one warp one row:

M = 2048, N = 32

average kernel time: 0.0093 ms, average total time: 0.0295 ms

Difference from baseline: 0.0000e+00

one warp one row float4 version:

M = 2048, N = 32

average kernel time: 0.0084 ms, average total time: 0.0256 ms

Difference from baseline: 0.0000e+00

one warp two rows:

M = 2048, N = 32

average kernel time: 0.0064 ms, average total time: 0.0231 s

Difference from baseline: 0.0000e+00

one warp two rows float4 version:

M = 2048, N = 32

average kernel time: 0.0066 ms, average total time: 0.0217 s

Difference from baseline: 0.0000e+00

one warp four row:

M = 2048, N = 32

average kernel time: 0.0065 ms, average total time: 0.0225 ms

Difference from baseline: 0.0000e+00

one warp four row float4 version:

M = 2048, N = 32

average kernel time: 0.0053 ms, average total time: 0.0272 ms

Difference from baseline: 0.0000e+00

3.3.8 M=2048, N=1024

baseline:

M = 2048, N = 1024

average kernel time: 0.2258 ms, average total time: 0.2490 ms

7.6482e-07 2.0790e-06 5.6513e-06 1.5362e-05 4.1758e-05 1.1351e-04 3.0855e-04 8.3873e-04 2.2799e-03 6.1974e-03

one warp one row:

M = 2048, N = 1024

average kernel time: 0.0280 ms, average total time: 0.0487 ms

Difference from baseline: 0.0000e+00

one warp one row float4 version:

M = 2048, N = 1024

average kernel time: 0.0255 ms, average total time: 0.0425 ms

Difference from baseline: 0.0000e+00

one warp two rows:

M = 2048, N = 1024

average kernel time: 0.0320 ms, average total time: 0.0487 s

Difference from baseline: 0.0000e+00

one warp two rows float4 version:

M = 2048, N = 1024

average kernel time: 0.0300 ms, average total time: 0.0467 s

Difference from baseline: 0.0000e+00

one warp four row:

M = 2048, N = 1024

average kernel time: 0.0393 ms, average total time: 0.0563 ms

Difference from baseline: 0.0000e+00

one warp four row float4 version:

M = 2048, N = 1024

average kernel time: 0.0330 ms, average total time: 0.0495 ms

Difference from baseline: 0.0000e+00

3.3.9 M=2048, N=2048

baseline:

M = 2048, N = 2048

average kernel time: 0.2870 ms, average total time: 0.3100 ms

3.8217e-07 1.0388e-06 2.8238e-06 7.6760e-06 2.0866e-05 5.6718e-05 1.5418e-04 4.1910e-04 1.1392e-03 3.0967e-03

one warp one row:

M = 2048, N = 2048

average kernel time: 0.1397 ms, average total time: 0.1642 ms

Difference from baseline: 0.0000e+00

one warp one row float4 version:

M = 2048, N = 2048

average kernel time: 0.1348 ms, average total time: 0.1618 ms

Difference from baseline: 0.0000e+00

one warp two rows:

M = 2048, N = 2048

average kernel time: 0.1446 ms, average total time: 0.1748 s

Difference from baseline: 0.0000e+00

one warp two rows float4 version:

M = 2048, N = 2048

average kernel time: 0.1419 ms, average total time: 0.1665 s

Difference from baseline: 0.0000e+00

one warp four row:

M = 2048, N = 2048

average kernel time: 0.2431 ms, average total time: 0.2693 ms

Difference from baseline: 0.0000e+00

one warp four row float4 version:

M = 2048, N = 2048

average kernel time: 0.1973 ms, average total time: 0.2253 ms

Difference from baseline: 0.0000e+00