Notflowing add bitonic sort and radix select topk a1555f1 · 3 minutes ago History

Code Blame 227 lines (198 loc) · 7.82 KB Raw

```cpp
/**
 * @brief Flips bits of single-precision floating-point number
 *
 *  flip a float for sorting
 *  finds SIGN of fp number.
 *  if it's 1 (negative float), it flips all bits
 *  if it's 0 (positive float), it flips the sign only
 * @param[in] f floating-point input (passed as unsigned int)
 * @returns uint that stores the flipped version of the input
 * @see floatUnflip
**/
__device__ uint float_flip(uint f)
{
    uint mask = -int(f >> 31) | 0x80000000;  // -int(0): 0x00000000; -int(1): 0xffffffff
    return f ^ mask;
}

/**
 * @brief Reverses bit-flip of single-precision floating-point number
 *
 * flip a float back (invert FloatFlip)
 *  signed was flipped from above, so:
 *  if sign is 1 (negative), it flips the sign bit back
 *  if sign is 0 (positive), it flips all bits back
 * @param[in] f floating-point input (passed as unsigned int)
 * @returns uint that stores the unflipped version of the input
 * @see floatFlip
**/
__device__ uint float_unflip(uint f)
{
    uint mask = ((f >> 31) - 1) | (0x80000000);
    return f ^ mask;
}

template <int BitsPerPass>
static __global__ void histogram_kernel(const uint* in_buf, int* histogram,
                                        const Counter* counter, const int pass, bool select_min)
{
    const int num_elements = counter->len;
    const int tid = blockIdx.x * blockDim.x + threadIdx.x;

    for (int idx = tid; idx < num_elements; idx += gridDim.x * blockDim.x)
    {
        uint mask = float_flip(in_buf[idx]);
        mask = mask >> (32 - BitsPerPass * (pass + 1));
        uint bucket = mask & ((0x01 << BitsPerPass) - 1);
        if (!select_min)
        {
            // bucket = ~bucket;  // 不能按位取反，正确做法是对最低8位取反
            bucket = bucket ^ ((0x01 << BitsPerPass) - 1);
        }

        atomicAdd(&histogram[bucket], 1);
        // atomicAggInc_cg(histogram + bucket);  // 结果异常，因为一个warp中的所有线程都是活跃的，只是执行原子操作的bucket不同
    }

}

template <int BitsPerPass>
static __global__ void scan_select_kernel(int* histogram, Counter* counter)
{
    constexpr int num_buckets = cal_num_buckets<BitsPerPass>();
```

```
        constexpr int num_buckets = cal_num_buckets<BitsPerPass>();
        constexpr int num_warps   = num_buckets / kWarpSize;

        __shared__ int sums[num_warps];
        __shared__ int histogram_smem[num_buckets];
        const int tid = blockIdx.x * blockDim.x + threadIdx.x;

        for (int i = tid; i < num_buckets; i += blockDim.x)
        {
            histogram_smem[i] = histogram[i];
            histogram[i] = 0;  // 设置histogram的所有值为0，用于初始化下一次pass
        }
        __syncthreads();

        int lane_id = tid % kWarpSize;
        int warp_id = threadIdx.x / kWarpSize;

        int value = histogram_smem[tid];
        // using a shfl instruction for a scan. (Hillis Steele scan)
        for (int i = 1; i < kWarpSize; i *= 2)
        {
            uint mask = 0xffffffff;
            int n = __shfl_up_sync(mask, value, i, kWarpSize);
            if (lane_id >= i)
            {
                value += n;
            }
        }

        // write the sum of the warp to smem
        if (threadIdx.x % kWarpSize == kWarpSize - 1)
        {
            sums[warp_id] = value;
        }
        __syncthreads();

        // scan sum the warp sums
        if (warp_id == 0 && lane_id < num_warps)
        {
            int warp_sum = sums[lane_id];
            uint mask = (0x01 << num_warps) - 1;
            for (int i = 1; i < num_warps; i *= 2)
            {
                int n = __shfl_up_sync(mask, warp_sum, i, num_warps);
                if (lane_id >= i)
                {
                    warp_sum += n;
                }
            }

            sums[lane_id] = warp_sum;
        }
        __syncthreads();

        // perform a uniform add across warps in the block
        // read neighbouring warp's sum and add it to threads value
        int block_sum = 0;
        if (warp_id > 0)
        {
            block_sum = sums[warp_id - 1];
        }
        histogram_smem[tid] = value + block_sum;

        // choose bucket
        int k = counter->k;
        int len = counter->len;
        __syncthreads();

        for (int i = tid; i < num_buckets; i += blockDim.x)
        {
            int pre = (i == 0) ? 0 : histogram_smem[i - 1];
            int cur = histogram_smem[i];

            // one and only one thread will satisfy this condition, so counter is written by only one thread
            if (pre < k && cur >= k)
            {
```

```
                counter->k = k - pre;      // how many values still are there to find
                counter->len = cur - pre;  // number of values in next pass
                counter->prev_len = len;
                counter->bucket_bits = i;  // bucket
            }
        }

        if (tid == 0)
        {
            counter->filter_cnt = 0;  // 重新设置filter_cnt的值为0
        }
    }

    template <int BitsPerPass>
    static __global__ void filter_kernel(const uint* in_buf, const int* in_idx,
                                         uint* out_buf, int* out_idx,
                                         int* out_index, Counter* counter,
                                         const int pass, const bool select_min)
    {
        constexpr int num_buckets = cal_num_buckets<BitsPerPass>();
        uint select_bucket = counter->bucket_bits;
        uint bucket_bits = (select_min ? select_bucket : (select_bucket ^ ((0x01 << BitsPerPass) - 1)));

        const int tid = blockIdx.x * blockDim.x + threadIdx.x;
        for (int idx = tid; idx < counter->prev_len; idx += gridDim.x * blockDim.x)
        {
            uint mask = float_flip(in_buf[idx]);
            mask = mask >> (32 - BitsPerPass * (pass + 1));
            uint bits = mask & ((0x01 << BitsPerPass) - 1);

            if (bits == bucket_bits)
            {
                // int pos = atomicAdd(&counter->filter_cnt, 1);
                int pos = atomicAggInc_cg(&counter->filter_cnt);
                out_buf[pos] = in_buf[idx];
                out_idx[pos] = in_idx[idx];
            }
            else if ((bits < bucket_bits) == select_min)
            {
                // int pos = atomicAdd(&counter->output_cnt, 1);
                int pos = atomicAggInc_cg(&counter->output_cnt);
                out_index[pos] = in_idx[idx];
                // 如果实际的topk的值，在数据集中有多个相等的值，(这里收集到的值肯定小于topk)
                // 则在最后一个pass后，将out_idx的指标都附加到out_index后面，直到数据个数为topk个
            }
        }

    }

    static __global__ void gather_kernel(int* out_index, int* in_index,
                                         Counter* counter, const int* num_elements,
                                         const int topk)
    {
        const int output_len = MIN(*num_elements, topk);
        const int tid = blockIdx.x * blockDim.x + threadIdx.x;
        for (int idx = tid; idx < (output_len - counter->output_cnt); idx += gridDim.x * blockDim.x)
        {
            out_index[counter->output_cnt + idx] = in_index[idx];
        }

    }

    // topk operator
    {
        for (int pass = 0; pass < num_passes; ++pass)
        {
            // 1. Calculate histogram
            constexpr int TS_topk = 16;
            const int GS_topk = (total_num_score_ + (BS * TS_topk) - 1) / (BS * TS_topk);
            histogram_kernel<BitsPerPass><<<GS_topk, BS, 0, cu_stream>>>
                            (in_buf, histogram, counter, pass, select_min);

            // 2. Scan the histogram (Inclusive prefix sum)
            // 3. Choose the bucket (Find the bucket j of the histogram that the k-th value falls into)
            scan_select_kernel<BitsPerPass><<<1, num_buckets, 0, cu_stream>>>
                                (histogram, counter);
```

```
214
215             // 4. Filtering (Input elements whose digit value <j are the top-k elements)
216             filter_kernel<BitsPerPass><<<GS_topk, BS, 0, cu_stream>>>
217                             (in_buf, in_idx, out_buf, out_idx,
218                             output_index, counter, pass, select_min);
219
220             in_buf  = (pass % 2 == 0) ? buf_value_1 : buf_value_2;
221             in_idx  = (pass % 2 == 0) ? buf_index_1 : buf_index_2;
222             out_buf = (pass % 2 == 0) ? buf_value_2 : buf_value_1;
223             out_idx = (pass % 2 == 0) ? buf_index_2 : buf_index_1;
224         }
225
226         gather_kernel<<<1, BS, 0, cu_stream>>>(output_index, in_idx, counter, ws1_data, total_num_out_);
227     }
```