LSU EE 7722 **Homework 3** **Due: 14 April 2025**

Formatted 17:43, 7 April 2025

Basic Setup

Follow the instructions for class account setup on `https://www.ece.lsu.edu/gp/proc.html`.
This assignment does not require writing or running code, but doing so can be helpful. Code for this assignment is in directories `../hw/gpm/2024/hw01` and `../hw/gpm/2024/hw02`. Note that those are 2024 directories and that they have the solution to the 2024 assignments (not to the questions asked here). For further instructions on how to run the 2024 Homework 1 and Homework 2 code see the Homework 1 assignment and Homework 2 assignment.

Background

This assignment is based on the solutions to 2024 Homework 1 and 2024 Homework 2. The subject of those assignments was a kernel that normalized vectors so that the average element value in a normalized vector is zero. (That's not the same a normalizing a vector so that its length is one.) The solution to those assignments is available (as of this writing in the repo only). To complete this 2025 assignment one needs to understand those solutions. That said, it might be helpful to attempt to solve those past assignments yourself.

In 2024 Homework 1 a modified kernel was to be written that lessens the impact of four issues that hurt performance: premature writebacks, bank conflicts, workload imbalance, and cache pressure. These are eliminated or reduced writing a kernel in which several threads normalize a vector, rather than just one. The number of threads normalizing a vector is called the *group size* and template parameter `grp_size` is set to its value. A disadvantage of increasing the group size is a drop in instruction efficiency (a fifth issue that affects performance). See 2024 Homework 1 for a detailed discussion of these five issues.

In 2024 Homework 2 the problem of reduced instruction efficiency and access latency is to be addressed by applying loop unrolling. That's not as simple as telling the compiler to unroll the loop by placing an `unroll` pragma before the loop. As explained in the problem, the compiler won't do a good job. The solution is to unroll a loop in such a way that instruction efficiency is improved and latency can be hidden.

The solutions to these two problems were the subject of 2024 Final Exam Problem 2. This 2025 assignment is a recycled version of that problem.

Problem 1: The following question originally appeared as 2024 Final Exam Problem 2. In the original exam the multiple choice parts of (b) through (e) were part of the question. Here the correct choice is given, and all one needs to do is explain the answer. Appearing below is the solution to 2024 Homework 2.

```
template<int D_L = 0, int grp_size = 1, int unroll_degree = 1>
__global__ void norm_group_u(elt_t* __restrict__ l_out, const elt_t* __restrict__ l_in) {
  const int tid = threadIdx.x + blockIdx.x * blockDim.x;
  const int n_threads = blockDim.x * gridDim.x;
  const int d_l = D_L ?: c_app.d_l, n_l = c_app.n_l;
  const int sub_lane = threadIdx.x % grp_size;
  constexpr int wp_lg = 5, wp_sz = 1 << wp_lg;
  const int lane  = threadIdx.x % wp_sz, wp_id = tid >> wp_lg;

  constexpr int vec_p_wp = wp_sz / grp_size;
  const int vec_p_wp_iter = vec_p_wp * unroll_degree;
  const int h_wp_start = wp_id * vec_p_wp_iter,   inc = unroll_degree * n_threads / grp_size;

  for ( int h_wp = h_wp_start;  h_wp < n_l;  h_wp += inc ) {
      const int hi = h_wp + lane / grp_size;
      elt_t thd_sum[unroll_degree]{};

      for ( int j = 0;  j < unroll_degree;  j++ )
        {
          const int h = hi + j * vec_p_wp;
          const size_t idx_vec_start = h * d_l;
          const size_t idx_vec_thd_start = idx_vec_start + sub_lane;

          for (int i=0;  i<d_l;  i+=grp_size ) thd_sum[j] += l_in[ idx_vec_thd_start + i ];
        }

      elt_t sum[unroll_degree]{};
      for ( int j = 0;  j < unroll_degree;  j++ ) sum[j] = group_sum(thd_sum[j],grp_size);

      for ( int j = 0;  j < unroll_degree;  j++ )
        {
          const int h = hi + j * vec_p_wp;
          const size_t idx_vec_start = h * d_l;
          const size_t idx_vec_thd_start = idx_vec_start + sub_lane;
          const elt_t avg = sum[j] / d_l;

          for ( int i = 0;  i < d_l;  i += grp_size )
            l_out[ idx_vec_thd_start+i ] = l_in[ idx_vec_thd_start + i ] - avg;
        }
    }}
```

(*a*) What is the minimum L1 cache size for which the `l_in` access in the Normalization Loop hits the L1 cache? Answer in terms of δ (unroll degree), d_l (vector length), g (group size), and B (number of threads per block). Assume that the number of blocks equals the number of SMs.

☐ In terms of δ, d_l, g, and B, the minimum cache size is:

To help answering the parts below it might be helpful to look at the sample output on the last pages of 2024 Homework 2.

(*b*) Consider two configurations of `norm_group_u`. Configuration X is launched with $B \geq 256$ threads per block, and an unroll degree of δ. Configuration Y is launched with $B/2$ threads per block and an unroll degree of 2δ. For both configurations n_l **is large**, $d_l = g = 32$, and the number blocks is equal to the number of SMs.

☐ Might Y run ◯ *at least* 1.5× *faster than X*, ◯ *at least* 1.5× *slower than X*, ⊗ *or at about the same speed as X.* ☐ Explain your answer based on your understanding of how GPUs work. ☐ Indicate whether the data included in the 2024 Homework 2 handout is consistent with the answer or does something different.

(*c*) Consider a system with 114 SMs. In configuration X $B = 256$, $d_l = g = 32$, $\delta = 1$, and $n_l = 912$. Configuration Y is the same except $B = 512$. In both cases 114 blocks are launched. (The unroll degree is **not** changed.).

☐ Will Y run ◯ *faster than X*, ◯ *slower than X*, ⊗ *or at about the same speed as X.* ☐ Explain your answer based on your understanding of how GPUs work. ☐ Don't forget to account for the value of n_l, which is not large in this part.

(*d*) Consider a system with 114 SMs. In configuration X $B = 256$, $d_l = g = 32$, $\delta = 1$, and $n_l = 912$. Configuration Y is the same except $\delta = 2$. (The block size is **not** changed.).

☐ Given the way the homework code performed, will Y run ◯ *faster than X*, ◯ *slower than X*, ⊗ *or at about the same speed as X.* ☐ Explain your answer based on your understanding of how GPUs work. ☐ Don't forget to account for the value of n_l, which is not large in this part.

(*e*) Consider a system with 114 SMs. In configuration X $B = 256$, $d_l = 1024$, $g = 32$, $\delta = 1$, and $n_l = 912$. Configuration Y is the same except $\delta = 2$. (The block size is **not** changed.).

☐ Given the way the homework code performed, will Y run ◯ *faster than X*, ⊗ *slower than X*, ◯ *or at about the same speed as X.* ☐ Explain your answer based on your understanding of how GPUs work. ☐ Don't forget to account for the value of n_l, which is not large in this part.