CSCE 435 Group project

1. Group members:

1. Trey Wells
2. Aaron Weast
3. Jacob Miller
4. David Vilenchouk

---

2. *due 10/25* Project topic

2. *due 10/25* Brief project description (what algorithms will you be comparing and on what architectures)

For the duration of this project, our team plans on communicating via Slack.

For our algorithms, we plan on implementing various sorting algorithms. The three sorting algorithms we are planning on implementing are Bubble sort, Merge Sort, and Quick sort.

For each of the algorithms, we are planning on implementating in both OpenMP and CUDA so that we can compare the differences in CPU vs. GPU parallelization. Not only will we be comparing the differences in CPU and GPU speed but we will also be testing the differences in the algorithms on various types of inputs. For example, we might run each algorithm on a completely random input, then on a partially sorted one, then on a completely sorted one.

Psuedocode for Algorithms

For example:

- Bubble Sort (MPI)
- Bubble Sort (CUDA)
- Quick Sort (MPI)
- Quick Sort (CUDA)
- Merge Sort (MPI)

  function merge(X, n, tmp): i = 0 j = n / 2 ti = 0 while i < n / 2 and j < n: if X[i] < X[j]: tmp[ti] = X[i] ti = ti + 1 i = i + 1 else: tmp[ti] = X[j] ti = ti + 1 j = j + 1 while i < n / 2: tmp[ti] = X[i] ti = ti + 1 i = i + 1 while j < n: tmp[ti] = X[j] ti = ti + 1 j = j + 1 copy(tmp, X, n) end merge

  function mergesort(X, n, tmp): if n < 2: return //Sort the first half #pragma omp task (X, n, tmp) mergesort(X, n / 2, tmp) //Sort the second half #pragma omp task (X, n, tmp) mergesort(X + (n / 2), n - (n / 2), tmp) //wait for both tasks to complete #pragma omp taskwait //Merge the sorted halves merge(X, n, tmp) end mergesort

  Source: https://avcourt.github.io/tiny-cluster/2019/03/08/merge_sort.html

- Merge Sort (CUDA) function Device_Merge(d_list, length, elementsPerThread): index = blockIdx.x * blockDim.x + threadIdx.x for i in 0 to elementsPerThread - 1: if (index + i < length): tempList[current_list][elementsPerThread * threadIdx.x + i] = d_list[index + i] synchronize_threads() for walkLen = 1 to length - 1 by walkLen *= 2: my_start = elementsPerThread * threadIdx.x my_end = my_start + elementsPerThread left_start = my_start while left_start < my_end: old_left_start = left_start if left_start > my_end: left_start = len break left_end = left_start + walkLen if left_end > my_end: left_end = len right_start = left_end if right_start > my_end: right_end = len right_end = right_start + walkLen if right_end > my_end: right_end = len solve(tempList, left_start, right_start, old_left_start, my_start, my_end, left_end, right_end, headLoc) left_start = old_left_start + 2 * walkLen current_list = not current_list synchronize_threads() index = blockIdx.x * blockDim.x + threadIdx.x for i in 0 to elementsPerThread - 1: if (index + i < length): d_list[index + i] = tempList[current_list] [elementsPerThread * threadIdx.x + i] synchronize_threads() return end Device_Merge

  function MergeSort(h_list, len, threadsPerBlock, blocks): d_list allocate_device_memory(d_list, len * sizeof(float)) copy_input_to_device(d_list, h_list, len * sizeof(float)) elementsPerThread = ceil(len / float(threadsPerBlock * blocks)) Device_Merge<<<blocks, threadsPerBlock>>>(d_list, len, elementsPerThread) copy_output_to_host(h_list, d_list, len * sizeof(float)) free_device_memory(d_list) end MergeSort

  function solve(tempList, left_start, right_start, old_left_start, my_start, my_end, left_end, right_end, headLoc): for i = 0 to walkLen - 1: if tempList[current_list][left_start] < tempList[current_list][right_start]: tempList[not current_list][headLoc] = tempList[current_list][left_start] left_start = left_start + 1 headLoc = headLoc + 1 if left_start == left_end: for j = right_start to right_end - 1: tempList[not current_list] [headLoc] = tempList[current_list][right_start] right_start = right_start + 1 headLoc = headLoc + 1 else: tempList[not current_list][headLoc] = tempList[current_list][right_start] right_start = right_start + 1 if right_start == right_end: for j = left_start to left_end - 1: tempList[not current_list][headLoc] = tempList[current_list][right_start] right_start = right_start + 1 headLoc = headLoc + 1 end solve

Source: https://pushkar2196.wordpress.com/2017/04/19/mergesort-cuda-implementation/