Problem selection

For this programming assignment choose one of the projects below or propose an alternate project of interest to you. If you choose a different project we need to agree in advance of the selection date on the suitability/feasibility of the project and the work to be done. As with the earlier programming assignments, you may work together with a partner and submit a single project. Please email your selection by Fri Nov 8.

Parallel Platforms

- Phaedra is available for shared memory (Cilk or OpenMP) and V100 accelerator projects.
- ITS Research Computing longleaf cluster. This requires a longleaf account. If you do not have one, you can request one online. The longleaf nodes are like Phaedra but only a few nodes have GPUs. There are some longleaf nodes with four V100 GPUs, use of these requires an extra step after obtaining a login. Longleaf requires you to use the slurm job submission system.

Project 1: Parallel quicksort using Cilk or OpenMP tasks

Implement a parallel quicksort on 64-bit double values. Parallel quicksort is easy to express in the W-T model (see the material in lecture 3 [PRAM (2)]). You will need to use tasking to implement the nested parallelism in the divide and conquer step. In addition you should investigate parallelizing the partitioning step of quicksort – otherwise you will have an (expected) $O(n)$ step complexity with only $O(lg n)$ average available parallelism.

Present your results as a description of the implementation and a set of performance graphs that display the sorting performance on random and specific input data sets (e.g. sorted, reverse sorted) as a function of problem size and number of processors. For expedience use {1, 4, 8, 12, 16, 20} processors. Show the performance (speedup) compared with an efficient sequential quicksort. Investigate and describe any performance bottlenecks.

Project 2: k-means using Nvidia V100 GPUs

Partition a sequence $L$ of $n$ points into $k$ clusters using the Lloyd $k$-means algorithm implemented on phaedra’s Nvidia V100 GPU. Each element of $L$ is a 2-tuple interpreted as a coordinate in 2-space. Use Euclidean distance as the distance metric between values in $L$.

The algorithm repeats an assignment phase and an update phase until it reaches a fixpoint in the assignment. In the assignment phase, assign each point in $L$ to the nearest of the $k$ clusters. In the update phase the cluster centers are repositioned to the coordinate-wise mean of points assigned to the cluster (a.k.a. the cluster centroid). Iterate until the cluster assignment stays the same. The output should be a
list of the $k$ centroids and a permutation of $L$ partitioned into $k$ segments corresponding to the points assigned to each cluster. Identify the number of points in each segment.

For performance analysis we are interested in the computation rate $C = \frac{k \times n \times h}{t}$, where $1 \leq k \leq 1000$ is the number of clusters, $1 \leq n \leq 10^8$ is the number of 2D points, $h$ is the number of iterations until convergence, and $t$ is the elapsed time. Feel free to stop the implementation after $h_{\text{max}} = 30$ iterations instead of waiting for convergence. However, if convergence is achieved in less than $h_{\text{max}}$ iterations, do not continue past convergence (because the computation is not representative). Show your performance results for when varying of $k$, $n$ and $h$ indicdually.

To obtain the elapsed time of the computation on the GPU, without the overhead of transferring data from the host to the device and back again, you can use CUDA GPU timers described in section 8.1.2 in the CUDA C Best Practices document on our class website to time just the $knh$ work at the heart of the algorithm.

**Project 3: Your choice.**

This project should be related to your research or other interests and must present a non-trivial parallelization problem. You should talk to me as early as possible to discuss the suitability of the project.