Homework #1

(Due: March 19)

Task 1. [250 Points] Multithreading Floyd-Warshall's All-Pairs Shortest Path (APSP) Algorithm.

Consider a directed graph $\mathcal{G} = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$, and each edge (v_i, v_j) is labeled by a real-valued length $l(v_i, v_j)$. If $(v_i, v_j) \notin E$, $l(v_i, v_j)$ is assumed to have a value $+\infty$. The length of a path is defined as the sum of the lengths of the edges in the path, taken in order. For each pair $v_i, v_j \in V$, $\delta[i, j]$ is defined to be the smallest of the lengths of all paths going from v_i to v_j. We assume that G does not have any negative length cycle.

Floyd-Warshall's algoithm for computing the length of the shortest path between every pair of vertices of G has a standard iterative implementation as shown in Figure 1. The input is an $n \times n$ matrix $D[1 \ldots n, 1 \ldots n]$ in which for every pair $i, j \in [1, n]$, $D[i, j]$ is initialized with $l(v_i, v_j)$. After the algorithm terminates $D[i, j] = \delta[i, j]$ for all $i, j \in [1, n]$.

In this task we will consider a number of recursive divide-and-conquer implementations of Floyd-Warshall's algoithm which are shown in Figures 2, 4 and 5. For simplicity we will assume n to be a power of 2. Each of these implementations is launched by calling $\mathcal{A}_{FW}(X, X, X)$, where X points to $D[1 \ldots n, 1 \ldots n]$ initialized with edge lengths of G as described in the previous paragraph. In general, for each $\mathcal{F} \in \{\mathcal{A}, \mathcal{B}, \mathcal{C}\}$, function $\mathcal{A}_{FW}$ / $\mathcal{A}_{loop\text{-}FW}$ accepts $X \equiv D[i_1 \ldots i_2, j_1 \ldots j_2]$, $U \equiv D[i_1 \ldots i_2, k_1 \ldots k_2]$ and $V \equiv D[k_1 \ldots k_2, j_1 \ldots j_2]$ as inputs, where $i_1, i_2, j_1, j_2, k_1, k_2 \in [1, n]$ with $i_2 - i_1 = j_2 - j_1 = k_2 - k_1 \geq 0$. Each update $D[i, j] \leftarrow \min(D[i, j], D[i, k] + D[k, j])$ applied by $\mathcal{F}_{FW}(X, U, V)$ / $\mathcal{F}_{loop\text{-}FW}(X, U, V)$ updates $D[i, j] \in X$ using $D[i, k] \in U$ and $D[k, j] \in V$, where $i_1 \leq i \leq i_2$, $j_1 \leq j \leq j_2$ and $k_1 \leq k \leq k_2$. In order to reduce the overhead of recursion each recursive function $\mathcal{F}_{FW}(X, U, V)$ switches to an iterative function $\mathcal{F}_{loop\text{-}FW}(X, U, V)$ as shown in Figure 3 when the problem size becomes small but still requires enough work to solve it so that the overhead of recursively reaching that problem size does not dominate the cost of solving the problem. We want to avoid all determinacy races in this task.

LOOP-FLOYD-WARSHALL-APSP(D, n)

```
1. for k ← 1 to n do
2.     for i ← 1 to n do
3.         for j ← 1 to n do
4.             D[i, j] ← min(D[i, j], D[i, k] + D[k, j])
```

Figure 1: [ITERATIVE IMPLEMENTATION] Looping code implementing Floyd-Warshall's APSP algorithm.

(*a*) **[15 Points]** Figure 1 shows the standard serial iterative implementation of Floyd-Warshall's APSP algorithm. Explain how you would parallelize this implementation by only replacing the serial ***for*** loops with parallel ***for*** loops. Compute the work, span and parallelism of your parallel implementation.

$\mathcal{A}_{FW}(X, U, V)$ $\{possibly\ X \equiv U \equiv V\}$

1. ***if*** X is an $m \times m$ matrix ***then*** $\mathcal{A}_{loop\text{-}FW}(X, U, V)$

   else

2. $\quad \mathcal{A}_{FW}(X_{11}, U_{11}, V_{11})$
3. $\quad$ **parallel:** $\mathcal{A}_{FW}(X_{12}, U_{11}, V_{12})$, $\mathcal{A}_{FW}(X_{21}, U_{21}, V_{11})$
4. $\quad \mathcal{A}_{FW}(X_{22}, U_{21}, V_{12})$
5. $\quad \mathcal{A}_{FW}(X_{22}, U_{22}, V_{22})$
6. $\quad$ **parallel:** $\mathcal{A}_{FW}(X_{21}, U_{22}, V_{21})$, $\mathcal{A}_{FW}(X_{12}, U_{12}, V_{22})$
7. $\quad \mathcal{A}_{FW}(X_{11}, U_{12}, V_{21})$

Figure 2: [Recursive Implementation 1] The initial call is $\mathcal{A}_{FW}(X, X, X)$, where X points to $D[1 \ldots n, 1 \ldots n]$ and n is assumed to be a power of 2. By X_{11}, X_{12}, X_{21} and X_{22} we denote the top-left, top-right, bottom-left and bottom-right quadrant of X, respectively.

$\mathcal{F}_{loop\text{-}FW}(X, U, V)$ $\{\mathcal{F} \in \{\mathcal{A}, \mathcal{B}, \mathcal{C}\}\}$

1. let $i_1, i_2, j_1, j_2, k_1, k_2 \in [1, n]$ with $i_2 - i_1 = j_2 - j_1 = k_2 - k_1 \geq 0$ be indices such that

   $X \equiv D[i_1 \ldots i_2, j_1 \ldots j_2]$, $U \equiv D[i_1 \ldots i_2, k_1 \ldots k_2]$ and $V \equiv D[k_1 \ldots k_2, j_1 \ldots j_2]$.

2. ***for*** $k \leftarrow k_1$ ***to*** k_2 ***do***
3. $\quad$ ***for*** $i \leftarrow i_1$ ***to*** i_2 ***do***
4. $\quad\quad$ ***for*** $j \leftarrow j_1$ ***to*** j_2 ***do***
5. $\quad\quad\quad D[i, j] \leftarrow \min\left(D[i, j], D[i, k] + D[k, j]\right)$

Figure 3: Looping base case for the recursive implementation of Floyd-Warshall's APSP algorithm.

(*b*) [**60 Points**] Assuming m to be a (small) constant[1] independent of n, compute the work, span and parallelism of each of the three recursive divide-and-conquer implementations of Floyd-Warshall's APSP given in Figures 2, 4 and 5.

(*c*) [**15 Points**] Observe that $\mathcal{A}_{loop\text{-}FW}$, $\mathcal{B}_{loop\text{-}FW}$, $\mathcal{C}_{loop\text{-}FW}$, and $\mathcal{D}_{loop\text{-}FW}$ must implement the same triply nested ***for*** loop as shown in Figure 3. But explain why $\mathcal{B}_{loop\text{-}FW}$ and $\mathcal{C}_{loop\text{-}FW}$ are more optimizable than $\mathcal{A}_{loop\text{-}FW}$, and $\mathcal{D}_{loop\text{-}FW}$ is the most optimizable among the four even when none of them are parallelized.

(*d*) [**30 Points**] Parallelize $\mathcal{A}_{loop\text{-}FW}$, $\mathcal{B}_{loop\text{-}FW}$, $\mathcal{C}_{loop\text{-}FW}$, and $\mathcal{D}_{loop\text{-}FW}$ using only parallel ***for*** loops. Compute the work, span and parallelism of each of those four functions.

(*e*) [**40 Points**] Implement the standard iterative Floyd-Warshall's APSP (from Figure 1) and the three recursive versions (from Figures 2, 4 and 5) using `Cilk`. Optimize the codes as much as possible. For each of three recursive implementations find the value of m (the switching

[1] for this subtask you may even assume $m = 1$ if you like

$\mathcal{A}_{FW}(X,U,V)$ $\{$*possibly* $X \equiv U \equiv V\}$

1. ***if*** X is an $m \times m$ matrix ***then*** $\mathcal{A}_{loop\text{-}FW}(X,U,V)$

   else

2. $\quad \mathcal{A}_{FW}(X_{11},U_{11},V_{11})$
3. $\quad$ **parallel:** $\mathcal{B}_{FW}(X_{12},U_{11},V_{12})$, $\mathcal{C}_{FW}(X_{21},U_{21},V_{11})$
4. $\quad \mathcal{A}_{FW}(X_{22},U_{21},V_{12})$
5. $\quad \mathcal{A}_{FW}(X_{22},U_{22},V_{22})$
6. $\quad$ **parallel:** $\mathcal{B}_{FW}(X_{21},U_{22},V_{21})$, $\mathcal{C}_{FW}(X_{12},U_{12},V_{22})$
7. $\quad \mathcal{A}_{FW}(X_{11},U_{12},V_{21})$

$\mathcal{B}_{FW}(X,U,V)$ $\{X$ *and* U *are dsjoint, but possibly* $X \equiv V\}$

1. ***if*** X is an $m \times m$ matrix ***then*** $\mathcal{B}_{loop\text{-}FW}(X,U,V)$

   else

2. $\quad$ **parallel:** $\mathcal{B}_{FW}(X_{11},U_{11},V_{11})$, $\mathcal{B}_{FW}(X_{12},U_{11},V_{12})$
3. $\quad$ **parallel:** $\mathcal{B}_{FW}(X_{21},U_{21},V_{11})$, $\mathcal{B}_{FW}(X_{22},U_{21},V_{12})$
4. $\quad$ **parallel:** $\mathcal{B}_{FW}(X_{21},U_{22},V_{21})$, $\mathcal{B}_{FW}(X_{22},U_{22},V_{22})$
5. $\quad$ **parallel:** $\mathcal{B}_{FW}(X_{11},U_{12},V_{21})$, $\mathcal{B}_{FW}(X_{12},U_{12},V_{22})$

$\mathcal{C}_{FW}(X,U,V)$ $\{X$ *and* V *are dsjoint, but possibly* $X \equiv U\}$

1. ***if*** X is an $m \times m$ matrix ***then*** $\mathcal{C}_{loop\text{-}FW}(X,U,V)$

   else

2. $\quad$ **parallel:** $\mathcal{C}_{FW}(X_{11},U_{11},V_{11})$, $\mathcal{C}_{FW}(X_{21},U_{21},V_{11})$
3. $\quad$ **parallel:** $\mathcal{C}_{FW}(X_{12},U_{11},V_{12})$, $\mathcal{C}_{FW}(X_{22},U_{21},V_{12})$
4. $\quad$ **parallel:** $\mathcal{C}_{FW}(X_{12},U_{12},V_{22})$, $\mathcal{C}_{FW}(X_{22},U_{22},V_{22})$
5. $\quad$ **parallel:** $\mathcal{C}_{FW}(X_{11},U_{12},V_{21})$, $\mathcal{C}_{FW}(X_{21},U_{22},V_{12})$

Figure 4: [Recursive Implementation 2] The initial call is $\mathcal{A}_{FW}(X,X,X)$, where X points to $D[1\ldots n, 1\ldots n]$ and n is assumed to be a power of 2. By X_{11}, X_{12}, X_{21} and X_{22} we denote the top-left, top-right, bottom-left and bottom-right quadrant of X, respectively.

point[2]) that gives you the best running time. Use $n = 2^{13}$, and try each power of 2 from 2^0 to 2^{13} as value of m. Produce a table or a graph for each recursive implementation showing how the running time varies as you change m.

(f) [**20 Points**] Run all four of your implementations from part (e) on all cores. Plot the running times of each implementation as you vary n from 2^4 to 2^{13} (i.e., consider only powers of 2).

[2]recall that for $\mathcal{F} \in \{\mathcal{A},\mathcal{B},\mathcal{C}\}$, $\mathcal{F}_{FW}$ switches to $\mathcal{F}_{loop\text{-}FW}$ when the size of the input matrices reduce to $m \times m$

$\mathcal{A}_{FW}(X,U,V)$ $\{X \equiv U \equiv V\}$

1. ***if*** X is an $m \times m$ matrix ***then*** $\mathcal{A}_{loop\text{-}FW}(X,U,V)$

   else

2. $\quad \mathcal{A}_{FW}(X_{11},U_{11},V_{11})$
3. $\quad$ **parallel:** $\mathcal{B}_{FW}(X_{12},U_{11},V_{12})$, $\mathcal{C}_{FW}(X_{21},U_{21},V_{11})$
4. $\quad \mathcal{D}_{FW}(X_{22},U_{21},V_{12})$
5. $\quad \mathcal{A}_{FW}(X_{22},U_{22},V_{22})$
6. $\quad$ **parallel:** $\mathcal{B}_{FW}(X_{21},U_{22},V_{21})$, $\mathcal{C}_{FW}(X_{12},U_{12},V_{22})$
7. $\quad \mathcal{D}_{FW}(X_{11},U_{12},V_{21})$

$\mathcal{B}_{FW}(X,U,V)$ $\{X$ *and* U *are dsjoint, but* $X \equiv V\}$

1. ***if*** X is an $m \times m$ matrix ***then*** $\mathcal{B}_{loop\text{-}FW}(X,U,V)$

   else

2. $\quad$ **parallel:** $\mathcal{B}_{FW}(X_{11},U_{11},V_{11})$, $\mathcal{B}_{FW}(X_{12},U_{11},V_{12})$
3. $\quad$ **parallel:** $\mathcal{D}_{FW}(X_{21},U_{21},V_{11})$, $\mathcal{D}_{FW}(X_{22},U_{21},V_{12})$
4. $\quad$ **parallel:** $\mathcal{B}_{FW}(X_{21},U_{22},V_{21})$, $\mathcal{B}_{FW}(X_{22},U_{22},V_{22})$
5. $\quad$ **parallel:** $\mathcal{D}_{FW}(X_{11},U_{12},V_{21})$, $\mathcal{D}_{FW}(X_{12},U_{12},V_{22})$

$\mathcal{C}_{FW}(X,U,V)$ $\{X$ *and* V *are dsjoint, but* $X \equiv U\}$

1. ***if*** X is an $m \times m$ matrix ***then*** $\mathcal{C}_{loop\text{-}FW}(X,U,V)$

   else

2. $\quad$ **parallel:** $\mathcal{C}_{FW}(X_{11},U_{11},V_{11})$, $\mathcal{C}_{FW}(X_{21},U_{21},V_{11})$
3. $\quad$ **parallel:** $\mathcal{D}_{FW}(X_{12},U_{11},V_{12})$, $\mathcal{D}_{FW}(X_{22},U_{21},V_{12})$
4. $\quad$ **parallel:** $\mathcal{C}_{FW}(X_{12},U_{12},V_{22})$, $\mathcal{C}_{FW}(X_{22},U_{22},V_{22})$
5. $\quad$ **parallel:** $\mathcal{D}_{FW}(X_{11},U_{12},V_{21})$, $\mathcal{D}_{FW}(X_{21},U_{22},V_{12})$

$\mathcal{D}_{FW}(X,U,V)$ $\{X,\ U$ *and* V *are dsjoint}*

1. ***if*** X is an $m \times m$ matrix ***then*** $\mathcal{D}_{loop\text{-}FW}(X,U,V)$

   else

2. $\quad$ **parallel:** $\mathcal{D}_{FW}(X_{11},U_{11},V_{11})$, $\mathcal{D}_{FW}(X_{12},U_{11},V_{12})$, $\mathcal{D}_{FW}(X_{21},U_{21},V_{11})$, $\mathcal{D}_{FW}(X_{22},U_{21},V_{12})$
3. $\quad$ **parallel:** $\mathcal{D}_{FW}(X_{11},U_{12},V_{21})$, $\mathcal{D}_{FW}(X_{12},U_{12},V_{22})$, $\mathcal{D}_{FW}(X_{21},U_{22},V_{21})$, $\mathcal{D}_{FW}(X_{22},U_{22},V_{22})$

Figure 5: [Recursive Implementation 3] The initial call is $\mathcal{A}_{FW}(X,X,X)$, where X points to $D[1\ldots n, 1\ldots n]$ and n is assumed to be a power of 2. By X_{11}, X_{12}, X_{21} and X_{22} we denote the top-left, top-right, bottom-left and bottom-right quadrant of X, respectively.

(g) [**20 Points**] For each of your four implementations from part (e) generate a `Cilkview` scalability plot by fixing n to 2^{13}.

(*h*) **[40 Points]** Repeat parts (*e*)–(*g*) using OpenMP, but instead of a `Cilkview` scalability plot generate a standard strong scalability plot for $n = 2^{13}$.

(*i*) **[10 Points]** Consider the recursive implementation from Figure 5. Can you improve its parallelism even further by using extra space for storing intermediate values the way we did for recursive matrix multiplication? Show your analysis.

APPENDIX 1: What to Turn in

One compressed archive file (e.g., zip, tar.gz) containing the following items.

- Source code, makefiles and job scripts.
- A PDF document containing all answers and plots.

APPENDIX 2: Things to Remember

- **Please never run anything that takes more than a minute or uses multiple cores on TACC login nodes.** TACC policy strictly prohibits such usage. They reserve the right to suspend your account if you do so. All runs must be submitted as jobs to compute nodes (even when you use Cilkview or PAPI).
- Please store all data in your work folder ($WORK), and not in your home folder ($HOME).
- When measuring running times please exclude the time needed for reading the input and writing the output. Measure only the time needed by the algorithm.