Stable Marriage problem

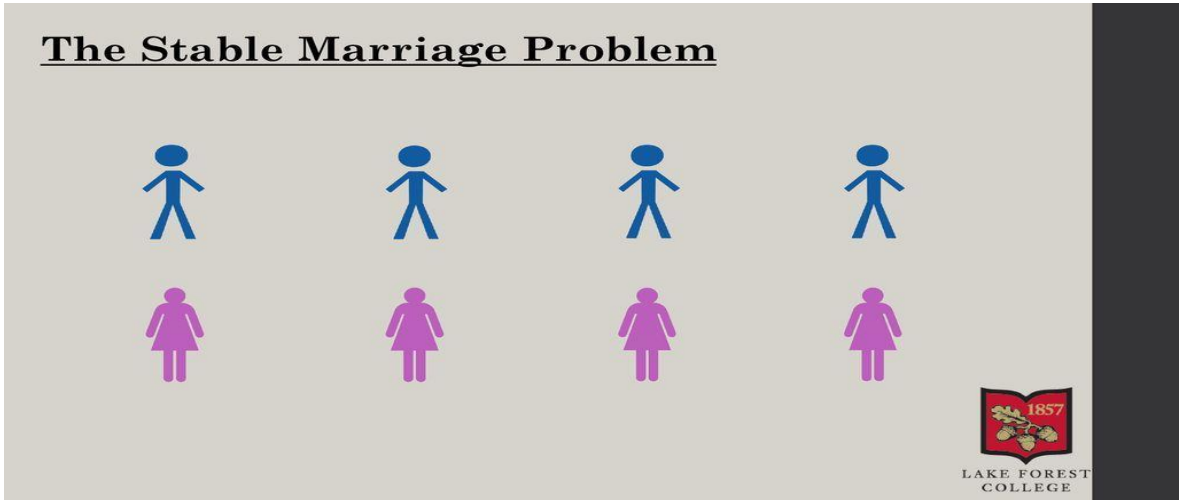
CS516 Parallelization of Programs

Hemant Gaur, Hrishik Kanade, Parth Khandenath

The Stable Marriage Problem

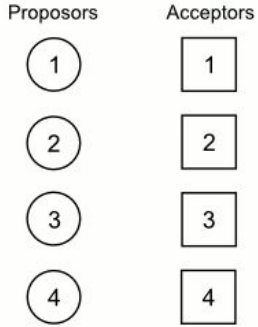
1. The stable marriage problem involves finding a stable matching between two sets of elements, A and B, ensuring that each element in set A is paired with an element from set B in a stable manner. This problem has significant applications in various fields, including education and resource allocation.

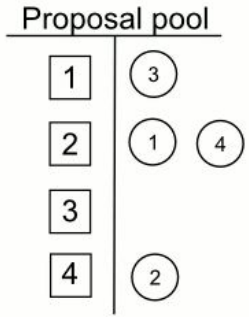
Gale Shapley Algorithm:



1. Everyone has a preference list for others.
2. Unmatched proposers take turns proposing to their most preferred **unmatched** acceptor.
3. Acceptors choose their best current option (unmatched or new proposal).
4. Matches are formed, rejected proposers keep searching their list.
5. Repeat until everyone is matched (stably).

Round : 1





- 1-4 propose, as none are currently tentatively attached

Preferences

□ → ○

Acceptor Table

1	1	3	2	4
2	3	4	1	2
3	4	2	3	1
4	3	2	1	4

○ → □

Proposor Table

1	2	1	3	4
2	4	1	2	3
3	1	3	2	4
4	2	3	1	4

Limitation of previous

1. research on stable marriage problems has mainly focused on theory and mathematical rigor previously
2. several recently published algorithms for computing greedy matchings were in fact special cases of classical algorithms for stable marriage problems
3. Earlier work proposed a new approach called Parallel Iterative Improvement (PII), where algorithm is implemented in OpenMP using n^2 processing elements (PEs), to obtain an O(n log n) running time compared to the sequential Gale and Shapley algorithm, which is $O(n^2)$.
4. However their implementation required $O(n^2)$ PEs, its not scalable in a GPU context since we have limited hardware resources.

General Limitations

1. Lack of scalability: high time complexity or memory requirements, infeasible for large problem sizes, limited applicability in real-world

2. Lack of efficiency: high running times, a large number of operations, impractical for time-sensitive applications or quick decision-making

3. Lack of parallelization: not designed to use parallel computing architectures, can’t exploit the computational power of modern hardware, suboptimal performance

4. Lack of adaptability: not easily adaptable to different problem variations or extensions, incomplete preference lists or multiple matches per participant, limits their usefulness

5. Lack of optimality guarantees: may give suboptimal, approximate solutions to the stable marriage problem, can be a limitation in situations

Techniques Used to Parallelize Stable Marriage Problem

CUDA for Massively Parallel Systems:

CUDA allows offloading computations to a GPU with thousands of cores for significant speedups. Here's how it could be implemented:

- **Data Transfer:** Preference lists and matching information need to be transferred from CPU memory to GPU memory. This transfer can introduce overhead and needs optimization.
- **Kernels:** Design kernels (functions executed on multiple GPU threads) to perform the core logic of the Gale-Shapley algorithm. Each thread could handle a specific proposer and iterate through their preference list, making proposals and updating matches stored on GPU memory.
- **Synchronization:** Implement synchronization mechanisms like thread barriers to ensure proper execution flow and avoid race conditions when accessing shared data structures.

Challenges:

- GPU memory limitations: Large preference lists might not fit entirely on GPU memory, requiring careful data management and potentially reducing the benefits. It resulted in Out of Memory (OOM) for size > 10^6.

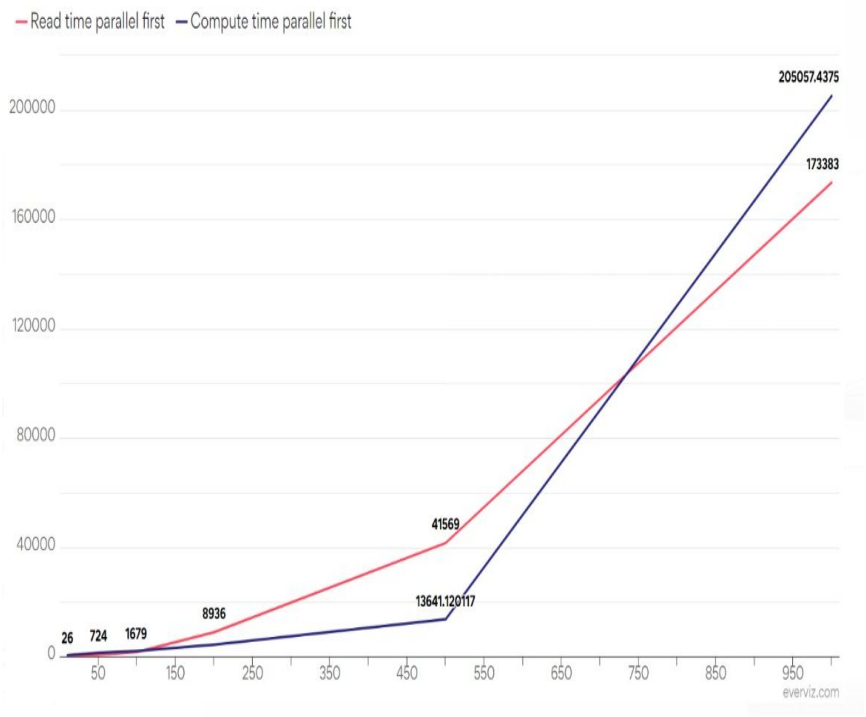
```
__global__ void stable_matching(int n, int *d_men, int *d_women,
        int *d_menacc, int *d_womenacc, int *d_menpre, int *d_matched, int *d_womenlock) {
    int j = threadIdx.x + 1, idx;
    idx = d_men[j*(n+1) + d_menpre[j]];
    if(j <= n && d_menacc[j] == -1) {
        bool isSet = false;
        do {                                    int idx
            if(isSet = atomicCAS(&d_womenlock[idx], 0, 1) == 0) {
                if(d_womenacc[idx] == -1) {
                    d_womenacc[idx] = j;
                    d_menacc[j] = idx;
                    atomicAdd(d_matched, 1);
                }
                else if(d_women[idx*(n+1) + d_womenacc[idx]] > d_women[idx*(n+1) + j]) {
                    d_menacc[d_womenacc[idx]] = -1;
                    d_menacc[j] = idx;
                    d_womenacc[idx] = j;
                }
            }
            if(isSet) {
                atomicCAS(&d_womenlock[idx], 1, 0);
            }
        } while(!isSet);
        d_menpre[j]++;
    }
}
```

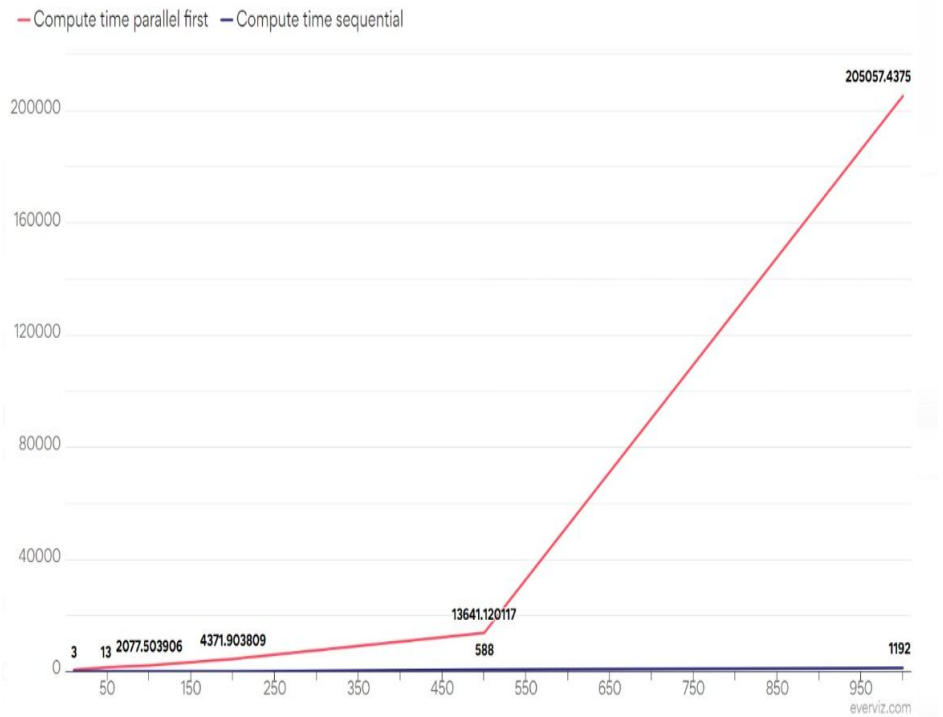
```
while(matched != n) {
    ct++;
    stable_matching <<< 1, n >>>(n, d_men, d_women, d_menacc, d_womenacc, d_menpre, d_matched, d_womenlock);
    cudaMemcpy(&matched, d_matched, sizeof(int), cudaMemcpyDeviceToHost);
}
```

Experimental Setup

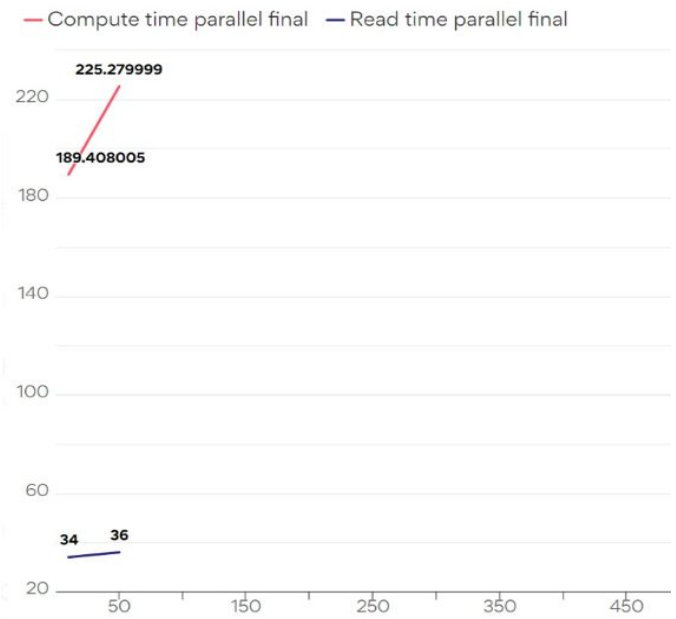
1. Here our aim was to compare the performance of serialised gale shapely with that of our parallel implementation.
2. So to do this, we generated data points for parameters like number of persons involved, number of proposal made while execution, execution time of code on gpu(device), and overall execution time.
3. These collected data points are then used for graph plotting of parallel vs serial implementation on various parameters vs n(number of participants).
4. We used google colab with CPU: Intel Xeon @ 2.20 GHz, RAM: approximately 13 GB and GPU: A Tesla K80 accelerator with 12 GB GDDR5 VRAM .

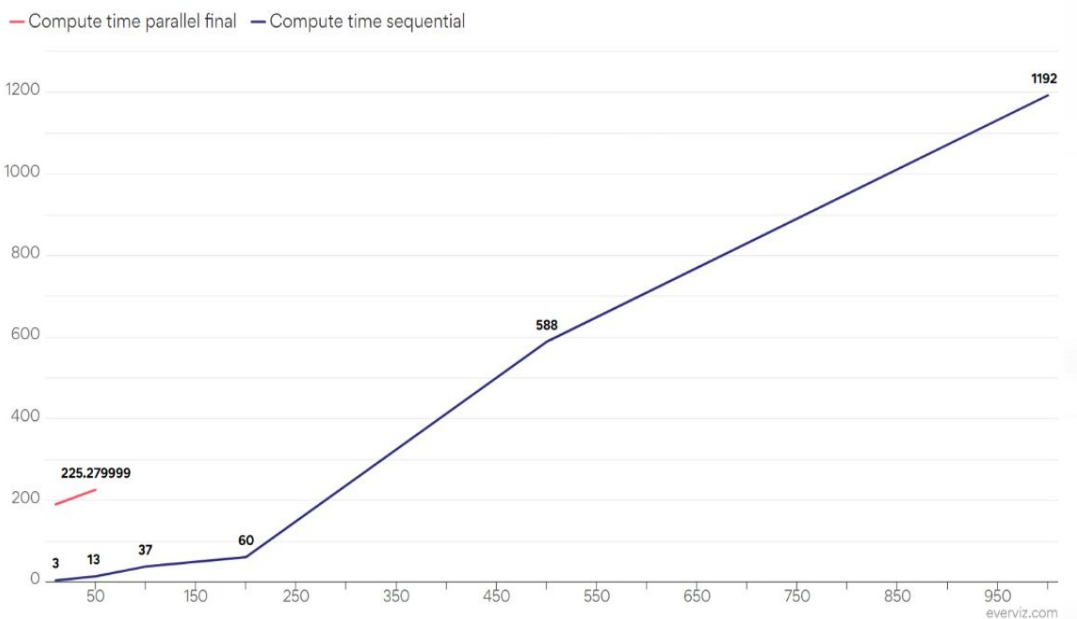
Results





Multiple kernel launches, atomic operations





With Single kernel launch and synchronisation barriers

Tradeoffs:

1. Multiple kernel launches - waste time & save resource
2. Single kernel - save time & exhaust resources

In line with the results of many previous research attempts:

1. A parallel approach to stable marriage problem (Jesper Larsen DIKU { Department of Computer Science University of Copenhagen , 1997) [1]

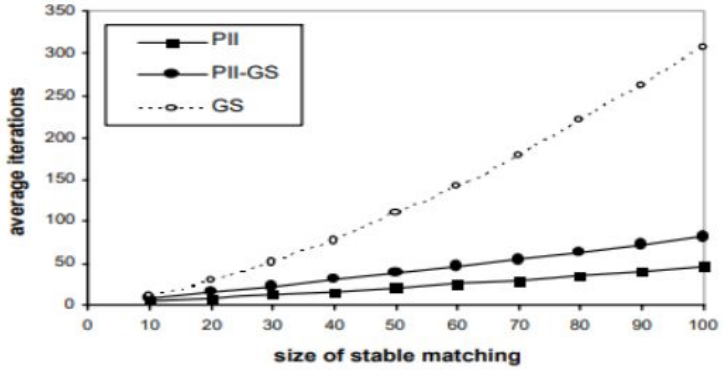
Algorithm	p	Random data		Worst-case	
		S_p	E_p	S_p	E_p
Parallel	3	$2.94 \cdot 10^{-2}$	$9.80 \cdot 10^{-3}$	$4.19 \cdot 10^{-3}$	$2.62 \cdot 10^{-4}$
Gale-Shapley	7	$5.37 \cdot 10^{-3}$	$7.67 \cdot 10^{-4}$	$4.14 \cdot 10^{-3}$	$2.59 \cdot 10^{-4}$
	15	$3.65 \cdot 10^{-3}$	$2.43 \cdot 10^{-4}$	$3.81 \cdot 10^{-3}$	$2.38 \cdot 10^{-4}$
"normal"	3	$9.12 \cdot 10^{-3}$	$3.04 \cdot 10^{-3}$	0.356	$2.22 \cdot 10^{-2}$
Parallel	7	$6.49 \cdot 10^{-3}$	$9.28 \cdot 10^{-4}$	0.302	$1.88 \cdot 10^{-2}$
Tseng-Lee	15	$5.29 \cdot 10^{-3}$	$3.53 \cdot 10^{-4}$	0.243	$1.52 \cdot 10^{-2}$
"optimal"	3	$1.02 \cdot 10^{-2}$	$6.38 \cdot 10^{-4}$	0.310	$1.94 \cdot 10^{-2}$
Parallel	7	$7.10 \cdot 10^{-3}$	$4.44 \cdot 10^{-4}$	0.278	$1.73 \cdot 10^{-2}$
Tseng-Lee	15	$6.41 \cdot 10^{-3}$	$4.01 \cdot 10^{-4}$	0.258	$1.61 \cdot 10^{-2}$

Table 6: The speedup and efficiency for the parallel Gale-Shapley and Tseng-Lee algorithms

6 Conclusion

Our experiments clearly show that the parallel algorithms are slower than the sequential ones, and they get even slower the more processors we use. As stated earlier, the communication is too expensive compared to the amount of work to be done. Clearly the stable marriage problem is to simple to be solved with commercial parallel computers.

2. A parallel iterative improvement of Stable Matching Algortihm (Enyue Lu and S. Q. Zheng Department of Computer Science, University of Texas at Dallas, 2003) [2]



In some applications, such as real-time packet/cell scheduling for a switch, stable matching is desirable, but may not be found quickly within tight time constraint. Thus, finding a "near-stable" matching by relaxing solution quality to satisfy time constraint is more important for such applications. Most of existing parallel stable matching algorithms cannot guarantee a matching with

a small number of unstable pairs within a given time interval. Interrupting the computation of such an algorithm does not result in any matching. However, the PII algorithm can be stopped at any time. By maintaining the matching with the minimum number of unstable pairs found so far, a matching that is close to a stable matching can be computed quickly.

Even the parallel versions cannot run in a time complexity better than O(no. of proposals).

Overheads in parallel versions like copying massive datasets between device and host machines and initializing kernel can take a lot of time and increase the overall runtime of the parallel program.

Demo