Parallel Nash Equilibrium Solver

Ally Du, Michael Cui

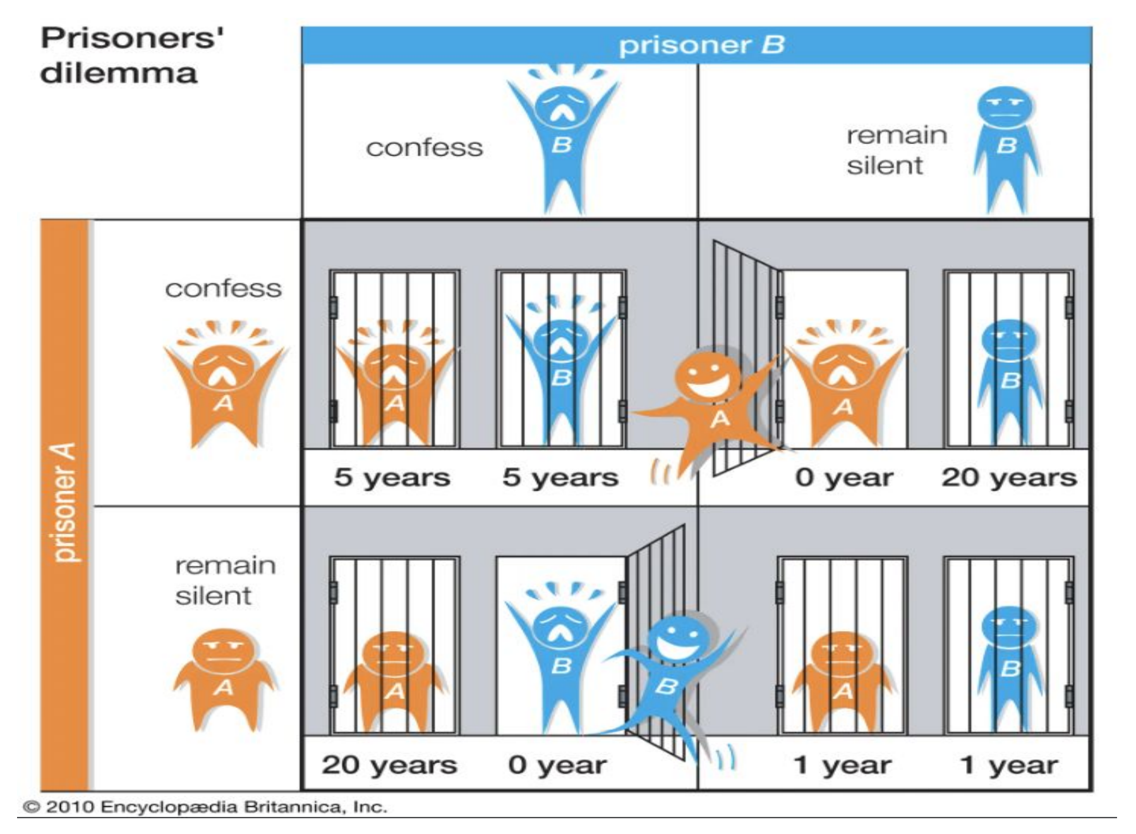
15-418 Parallel Computing Architecture and Programming, Fall 2024

Introduction

Game theory plays a pivotal role in understanding strategic interactions in various fields, including economics, biology, and artificial intelligence.

A **Nash Equilibrium** is a fundamental concept in game theory that describes a stable state in a game. It occurs when all players in the game choose strategies such that:

1. Each player's strategy is the best response to the strategies chosen by others.
2. No player has an incentive to deviate from their chosen strategy, given the strategies of the other players.



Finding NE in general 2-player games is PPAD-Complete (Papadimitrios 1994)

	Prisoner B Confess	Prisoner B Remain Silent
Prisoner A Confess	(-5, -5)	(0, -20)
Prisoner A Remain Silent	(-20, 0)	(-1, -1)

Sequential Solver

The algorithm we used for our sequential version is the following. This is inspired by (Grosu 2008) :

1. **Initialize and Input Data.**
2. **Generate and Pair Strategy Supports.**
3. **Evaluate Each Support Pair.**
4. **Validate and Construct Full Strategies.**
5. **Verify Equilibrium Conditions.**
6. **Store and Output Equilibria.**

OpenMP Results

Parallel Strategy (v1):
We insert #pragma omp parallel between step 2 and step 3, to parallelize the evaluation.

Result (observation): when player1 and player2 has 18 and 2 strategies, speedup is 4.6490x, but when the number of strategies flipped, the speedup is only 2.1391x.

Only works well when player 1's number of strategies (m) is larger than player 2's number of strategy (n).

Reason: We generated strategy subsets separately for player 1 and player 2, and run a 2-layer for-loop to iterate through all the strategies. Using the parallel operation for the inner and outer loop has very different performance when m << n (or n << m).

Parallel Strategy (v2):
Generate pairs of strategies in parallel, so we could have a good throughput for all combinations of m and n.

Result:

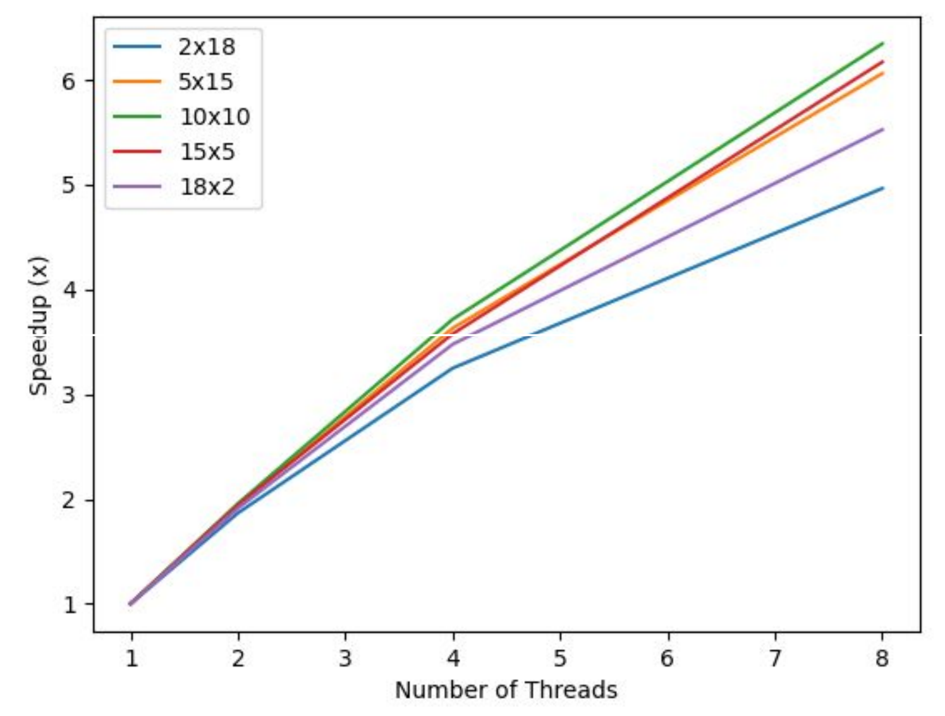


Fig1. Speedup vs. number of threads when parallelized using OpenMP.

Scheduling policy:

- We experimented with different scheduling policy and realized that fine-grained dynamic scheduling using a dynamic work queue works best on average. This is because different strategy subsets may take very different amount of time to evaluate.

Submatrix construction: building submatrices incrementally, reducing computation time for each thread.

MPI Results

Parallel Strategy (v1): Message passing between step 2 and step 3, broadcast matrix pairs.

Result:

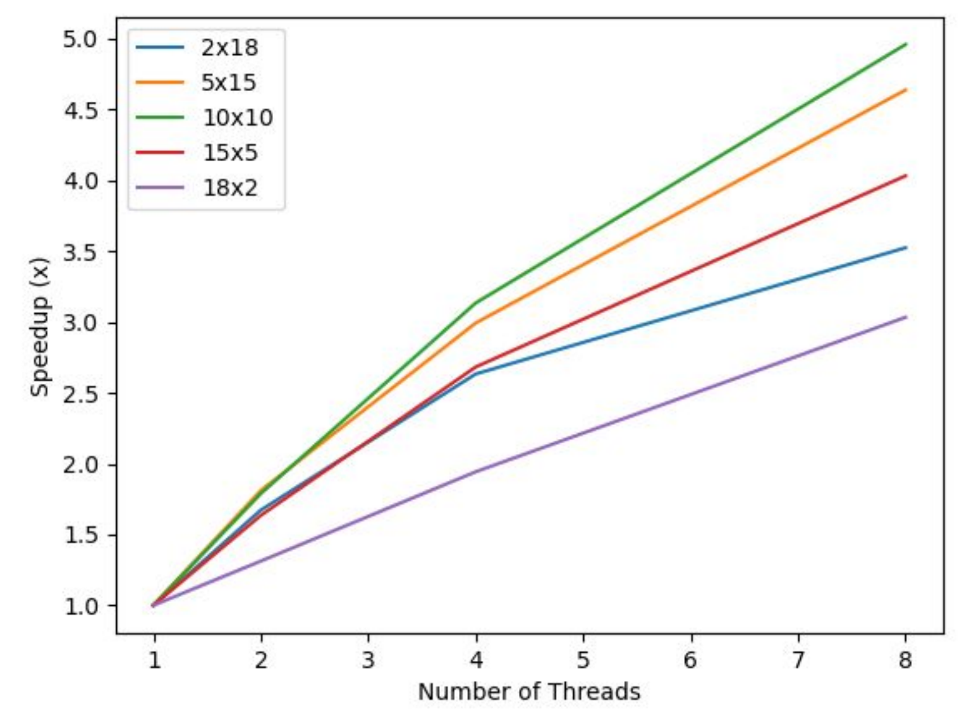


Fig2. Speedup vs. number of threads when parallelized using MPI.

Reason: broadcasting the support pair costs too much.

Parallel Strategy (v2): Ask each thread to generate the support pairs locally, reducing message passing time.

Result:

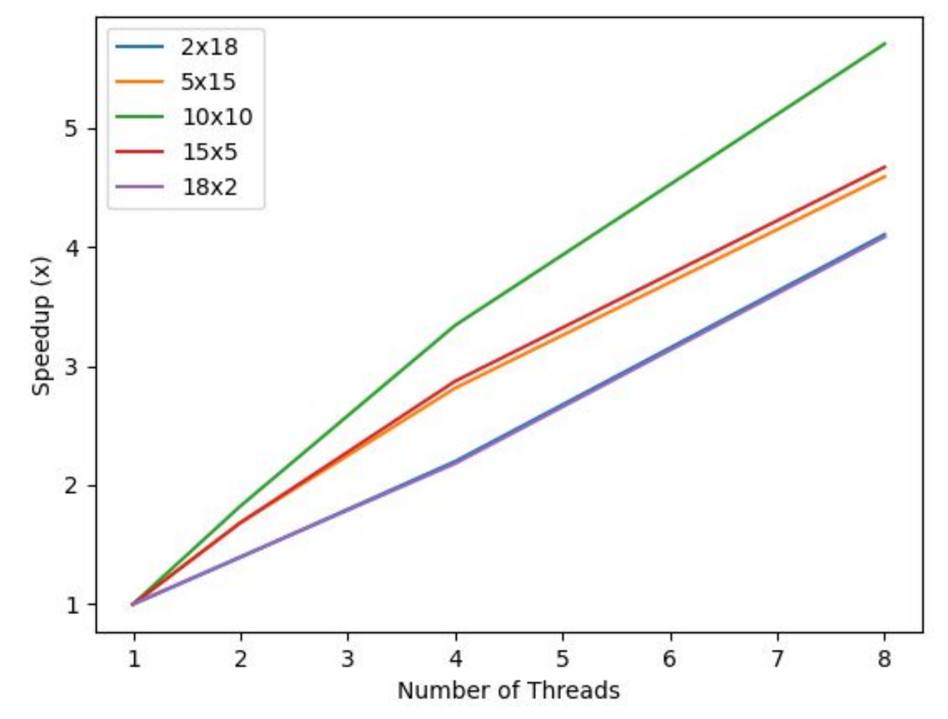


Fig3. Speedup vs. number of threads when parallelized using improved version of MPI.

Improvement: This indicate room for improvement, we plan to implement load balancing using master-worker model.
We can also consider a hierarchical synchronization.

Limitations/Potential Improvements

- The algorithm assumes well-structured input and doesn't seem to address degenerate cases, such as games with multiple or no Nash equilibria, or scenarios where the payoff matrices are not invertible.
- We can also explore some $\epsilon-$approximation algos.
- We can extend this to >2 players games!

Carnegie Mellon University

Reference

Papadimitriou, C. H. (1994). Complexity of finding Nash equilibria. *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS)*, 195-204. IEEE.

Grosu, D. (2008). Parallel computation of Nash equilibria in N-player games. *Parallel Computing*, 34(10), 625-633. https://doi.org/10.1016/j.parco.2008.08.003

Encyclopaedia Britannica. (n.d.). *The prisoner's dilemma*. In *Game theory*. Retrieved December 13, 2024, from https://www.britannica.com/science/game-theory/The-prisoners-dilemma