Computing Nash Equilibria in Bimatrix Games
GPU-based Parallel Support Enumeration

Safraz Rampersaud
Lena Mashayekhy
Advisor: Dr. Daniel Grosu
Department of Computer Science
Wayne State University, Detroit, MI
safraz@wayne.edu mlena@wayne.edu

Wayne State University

Department of Computer Science

Problem Statement

Computing Nash equilbria using GPUs

- Application Constraints: Real world strategic interactions usually require the modeling of large number of agents having a large number of choices or actions.
- Processing Constraints: Computing all Nash equilibria for large bimatrix games using single-processor computers is computationally expensive.

Nash Equilibria in Bimatrix Games

Bimatrix Game $\Gamma(A,B)$

- A set of two players: {Player 1, Player 2}
- A finite set of actions for each player
  - Player 1's actions: $M = (s_1, s_2, \ldots, s_m)$
  - Player 2's actions: $N = (t_1, t_2, \ldots, t_n)$
- Player payoff matrices $A, B \in \mathbb{R}^{m \times n}$

Strategies $\{(x,y)\}$

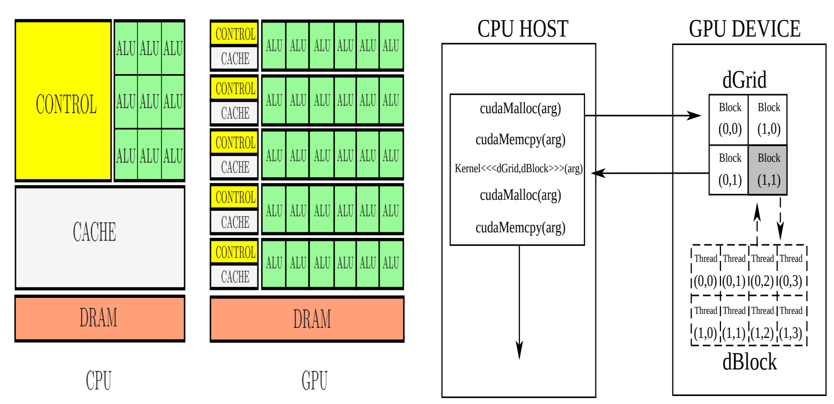
- Strategies are probabilities representing player's choice of actions.
- $x = \{(x_1, \ldots, x_m) \mid \Pr\{\text{(Player 1)} \leftarrow s_i\} = x_i\}$
- $y = \{(y_1, \ldots, y_n) \mid \Pr\{\text{(Player 2)} \leftarrow t_j\} = y_j\}$

Support Enumeration Method

1. Enumerate all pairs of supports (M_x, N_y)
   - $M_x = \{s_i | x_i > 0\}$ where $M_x \subset M$
   - $N_y = \{t_j | y_j > 0\}$ where $N_y \subset N$
2. Compute Nash equilibria (x, y) in $\Gamma(A,B)$
   - $x \mid \forall\, s_i \in M_x, (Ay)_i = u = \max_{q \in M} (Ay)_q$
   - $y \mid \forall\, t_j \in N_y, (x^T B)_j = v = \max_{r \in N} (x^T B)_r$

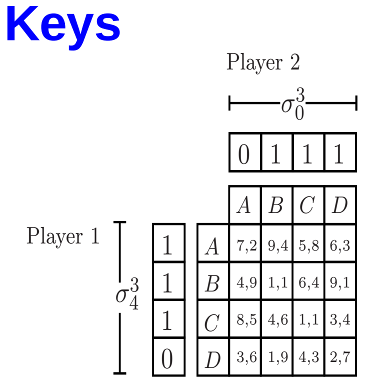
CPU-GPU

Graphics Processing Units (GPUs) are specialized hardware dedicated to building graphic images as well as supporting parallel processing.



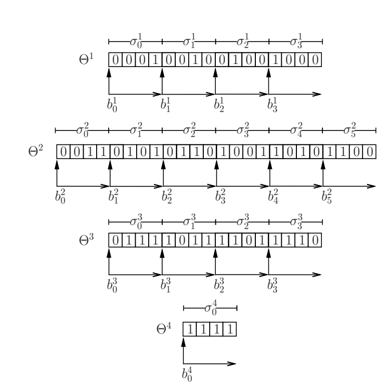
Support Keys

- Support key σ_j^i
- Indicates available actions
- Support size i
- Order of permutation j
- 69 strategy pair combinations



Support Key Set & Block Distribution

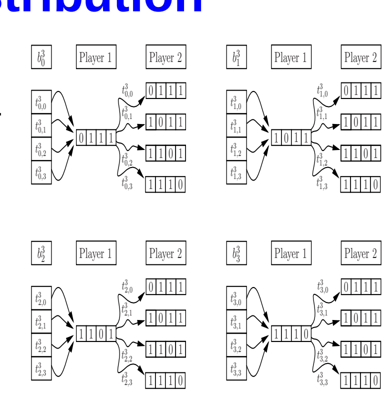
- Set of Support keys Θ^i
- Support size i
- Co-lexicographical order.



- Blocks b_j^i select strategies for Player 1
- Support size i
- Order of permutation j

Thread Distribution

- Threads $t_{j,k}^i$ selects strategies for Player 2
- Processes strategy pairs
- Computes (x,y) probabilities
- Support size i
- Thread index k in block j



GPU Parallel Support Enumeration

Input: Player 1 payoff, Player 2 payoff (A, B)
Output: Set of equilibria (Φ)

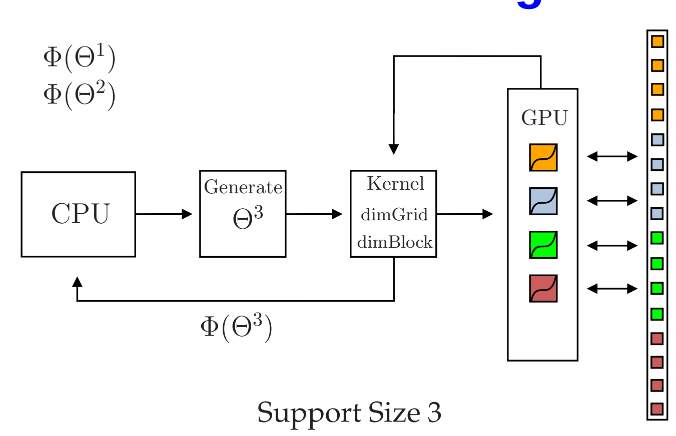
1: $\Phi = \emptyset$
2: $q = \min(m, n)$
3: $\Theta =$ Generate(1, q)
4: $\Phi =$ Pure(A, B, q, Θ)
5: **for** $k = 2, \ldots, q$
6: $\quad\Theta =$ Generate(k, q)
7: $\quad\Phi = \Phi \cup$ Mixed(A, B, k, q, Θ)
8: output Φ

Computing Nash Equilibria

Pure(A, B, q, Θ):
Computes pure Nash Equilibria in $\Gamma(A,B)$.
A *pure* strategy x is Nash equilibrium strategy where players choose a single action with probability 1.

Mixed(A, B, k, q,Θ):
Computes mixed Nash Equilibria in $\Gamma(A,B)$.
A *mixed* strategy x is Nash equilibrium where players choose actions according to a probability distribution over their pure actions.

GPU Processing



Experimental Setup

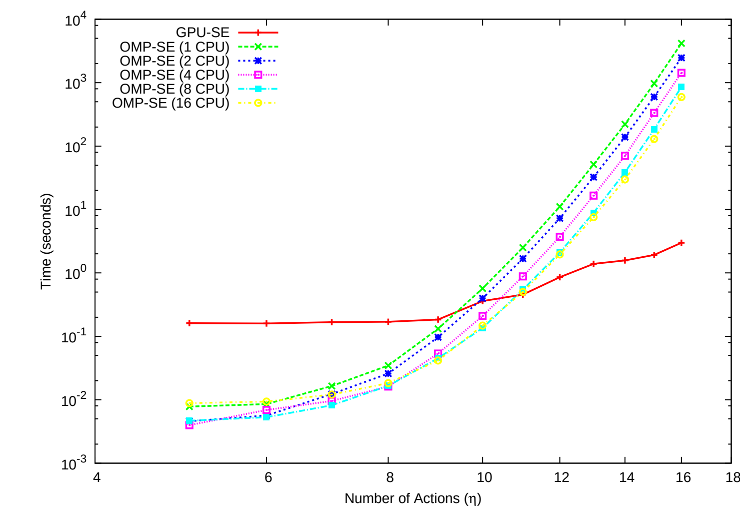
GPU

- Nvidia™ GT 440
- 96 CUDA Cores
- 1.6 (GHz) Processor Speed
- 13 (billion/sec.) Texture Fill Rate

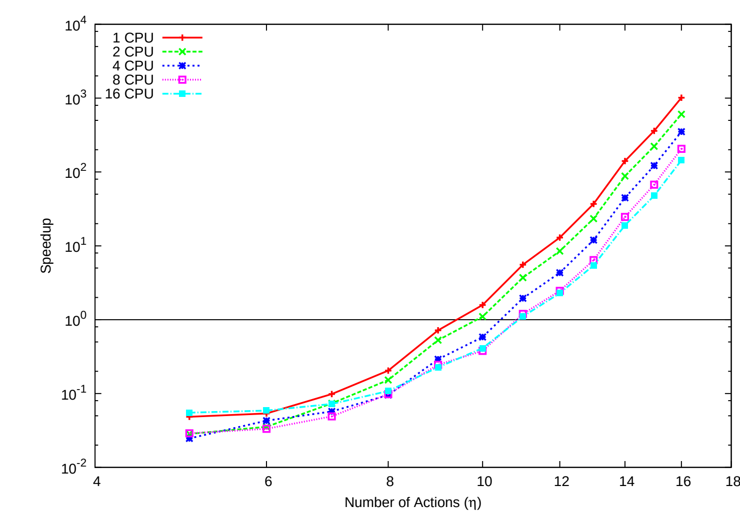
OpenMP

- Wayne State University Grid
- 40 Node 16-Core Quad AMD Processors
- 2.6 (GHz) Processor Speed
- 128 GB RAM

Average Execution vs. Number of Actions



Average Speedup vs. Number of Actions



Conclusion

- GPU processing outperforms OpenMP implementations for computing equilibria in larger games.
- GPU speedups range from 144.07 to 1013.53 against OpenMP configurations from 1 to 16 CPUs.

Acknowledgment

This research was partially supported by NSF grants DGE-0654014 and CNS-1116787.