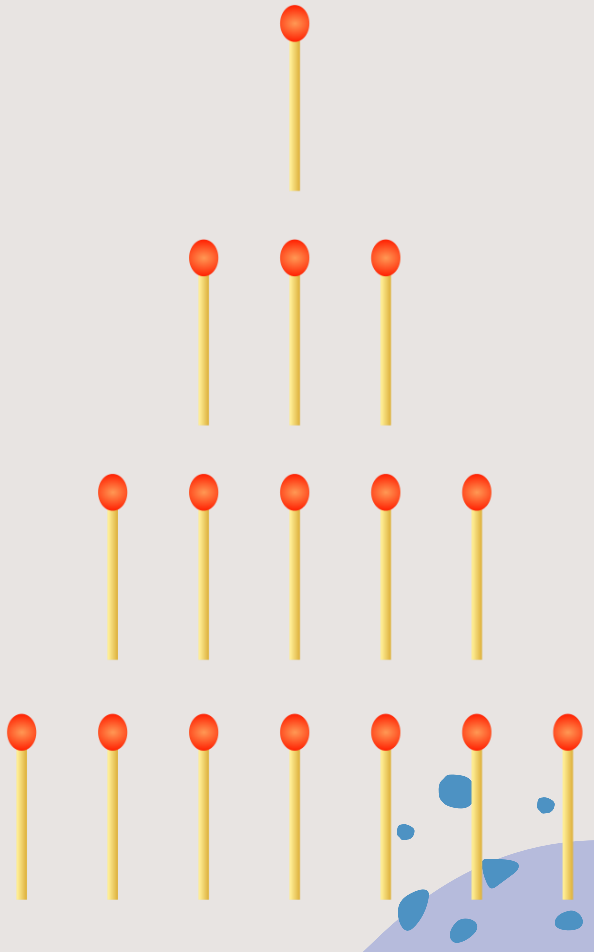
Parallel MinMax for Nim

Faculty of Computer Engineering, Politecnico di Torino, Italy

The Nim game

Nim is a two-player mathematical game of strategy in which players take turns removing objects from distinct heaps or piles of the board:

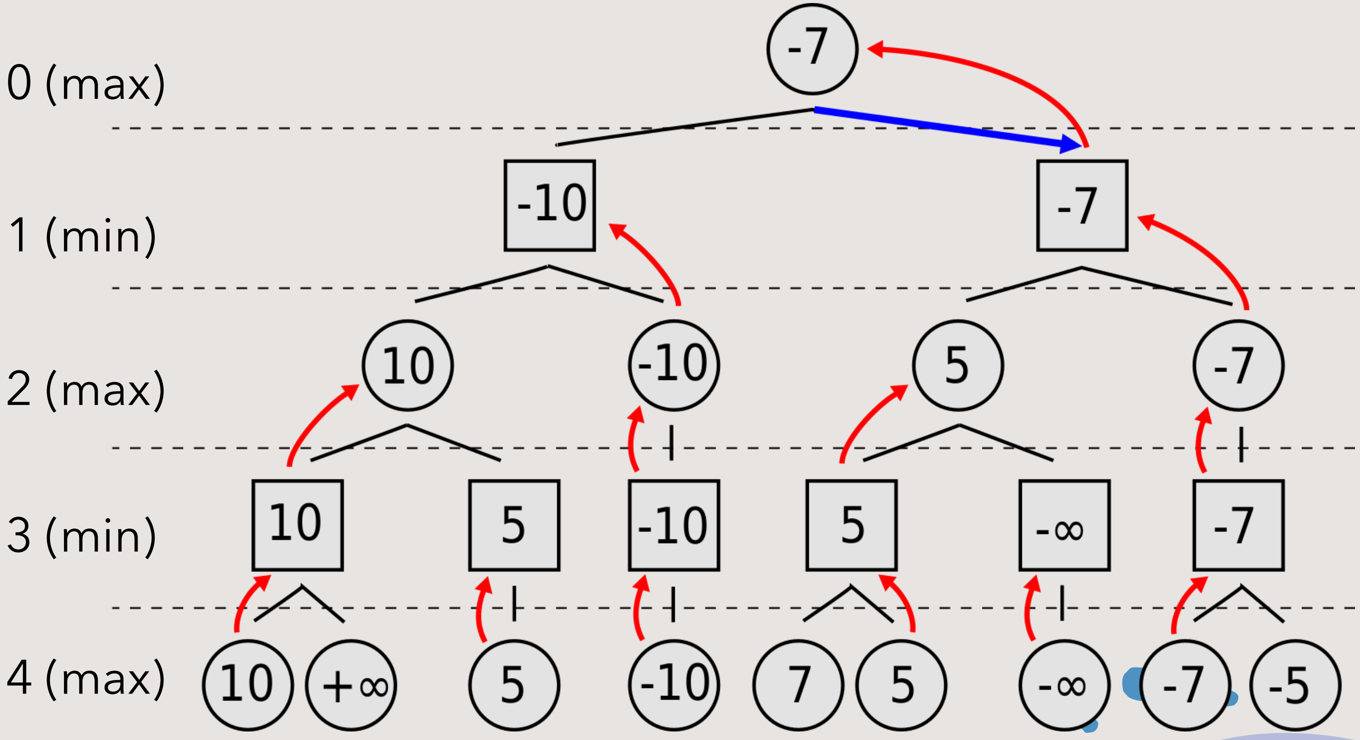
- The board typically consists of several piles of objects each pile contains an increasing odd number of sticks.
- On each turn, a player must remove one or more objects from any pile.
- The goal of the game is to be the player to take the last object.



The MinMax algorithm

Minmax is an algorithm used in **game theory** to determine the best move in a two-player, zero-sum game:

- The opponent will make the move that is most detrimental to the current player, so the he will always try to *minimize the reward*.
- The current player choose the move that minimizes the worst-case outcome, so he will always try to *maximize his reward*.



Project aim

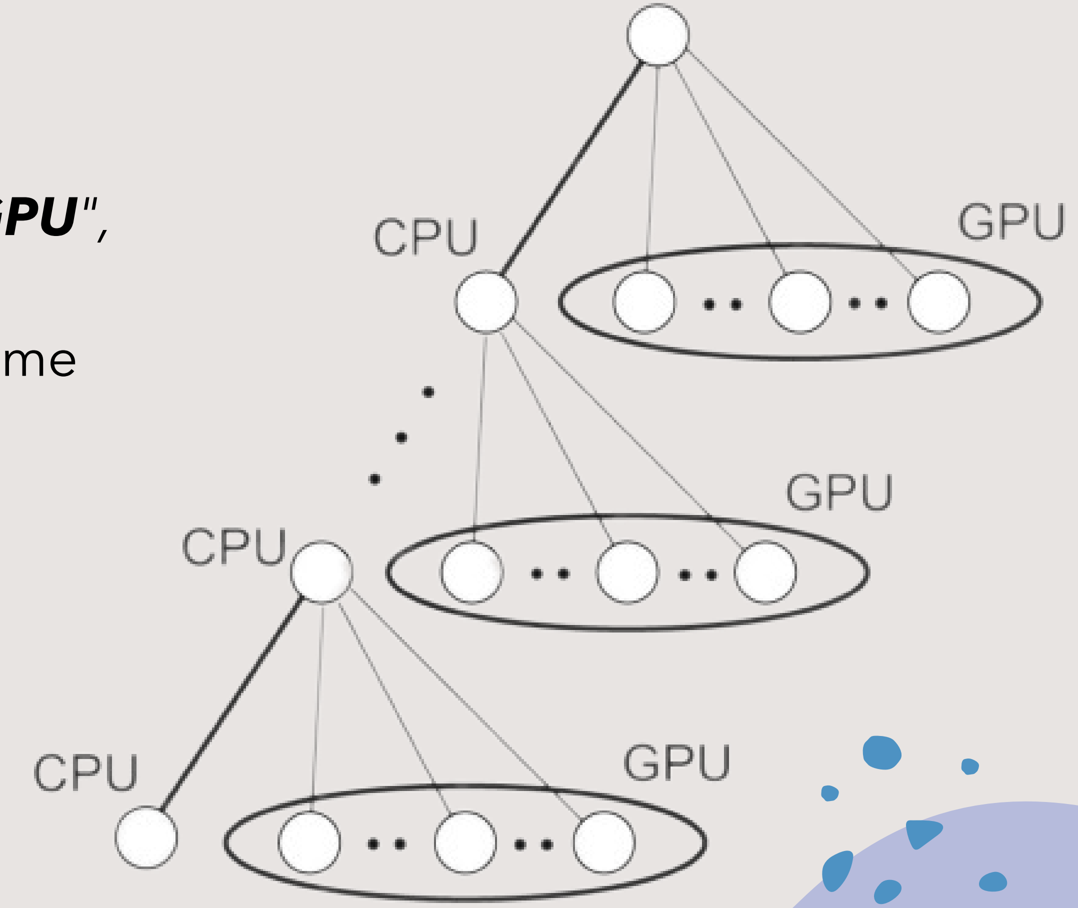
The aim of the project is exploiting the power of GPUs to compute the optimal move for Nim by creating a **parallel** version of the minmax algorithm, by achieving:

- Performance
- Scalability
- Portability
- Power efficiency

Related Work

In the paper "***Parallel Alpha-Beta Algorithm on the GPU***", *Damjan Strnad* and *Nikola Guid* describe the parallel implementation of the alpha-beta algorithm for the game of Reversi by using the **PV-Split algorithm:**

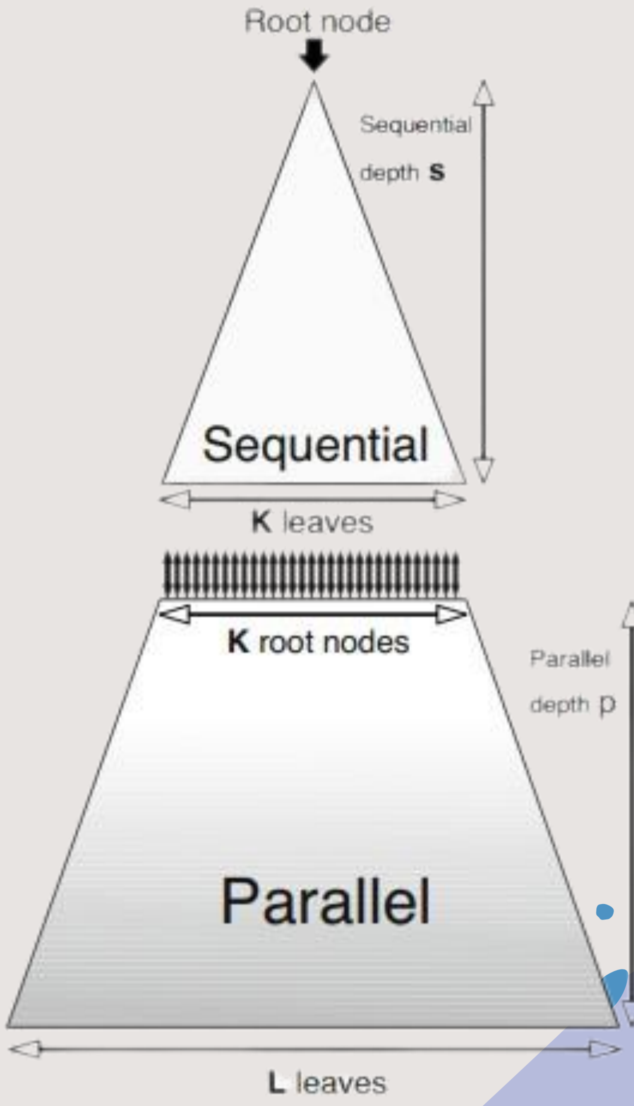
- The leftmost child of each PVnode is searched on the CPU
- The rest of PVnode's descendants are searched in parallel on the GPU.



Related Work

Kamil Rocki and *Reiji Suda*, in their "***Parallel Minimax Tree Searching on GPU***" adapted the Minmax algorithm to the Reversi game by splitting the tree into two parts:

- The upper tree of depth is processed in a *sequential* manner.
- The lower part of depth is processed *parallelly* and sliced into subtrees, so that each of them can be searched separately.



Proposed Method

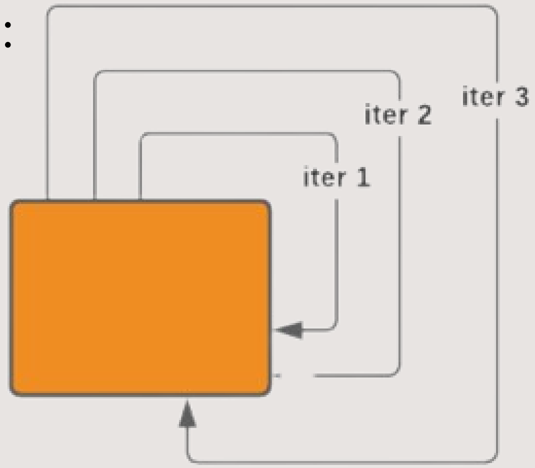
During the development, four main version of the parallel algorithm were implemented, plus one version of the sequencial algorithm:

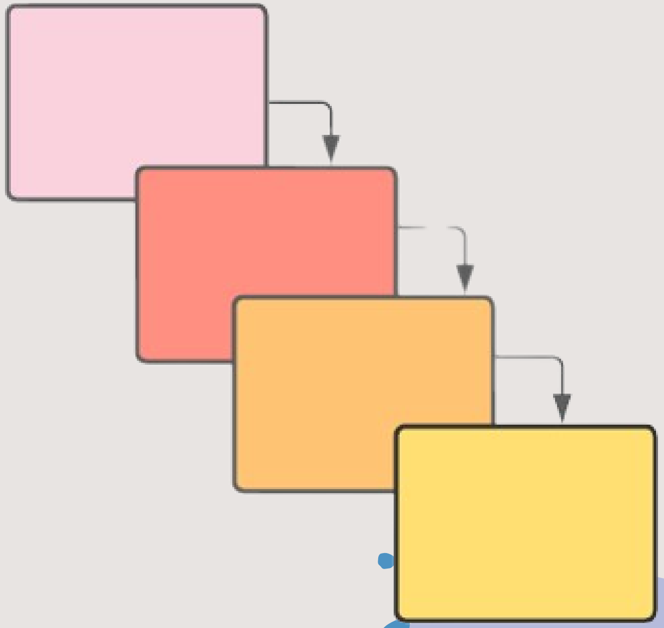
- ***V0*** – Iterative C Implementation
- ***V1*** – First CUDA version
- ***V2*** – Data structures optimization
- ***V3*** – More levels in shared memory
- ***V4*** - Memory transactions reduced

V0 – Iterative C Implementation

The **Nim library** and the **minmax algorithm** were adapted from Python to C, before parallelizing the algorithm itself in a CUDA kernel:

- From recursive to iterative form
- Usage of a Stack
- Dynamic memory
- High complexity
- Faster than Python

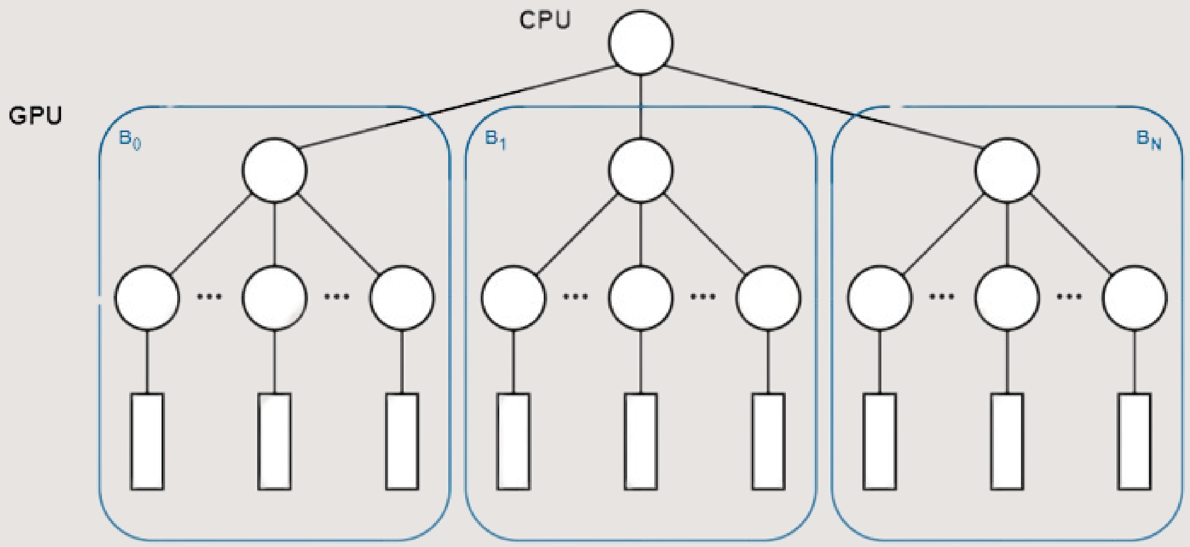


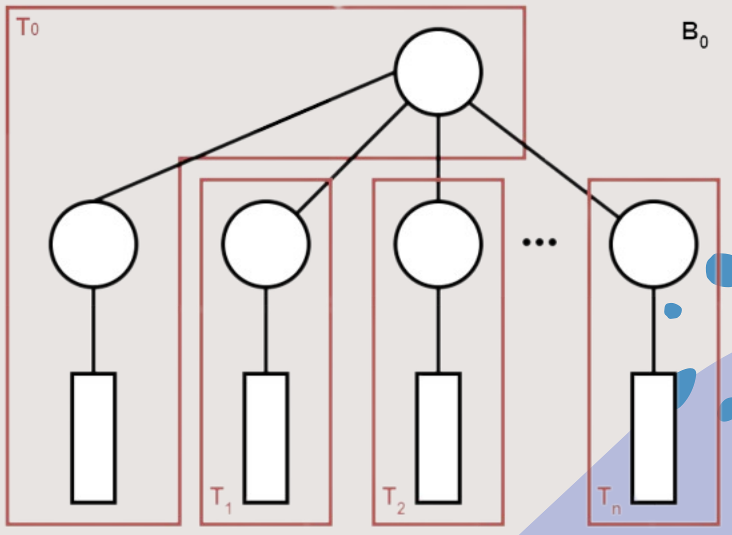


V1 – First CUDA version

The GPU is used to **parallelize** the search and node-creation process:

- The game tree is divided into smaller subtrees
- Each subtree is processed by one block
- Each block evaluates one child nodes for each thread
- The results are evaluated in the host

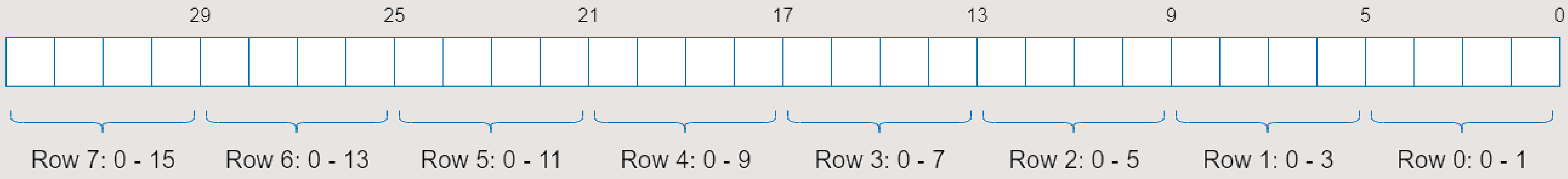




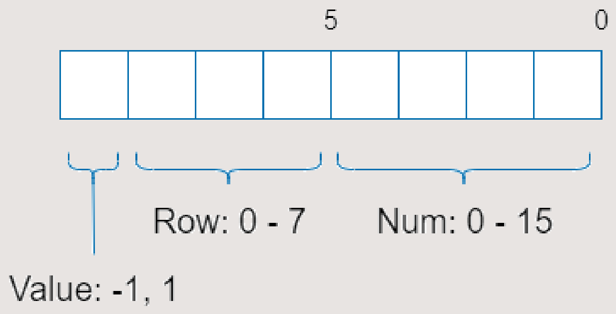
V2 – Data structures optimization

The **memory read/write transactions** were reduced, and the constant memory was used.

Nim representation



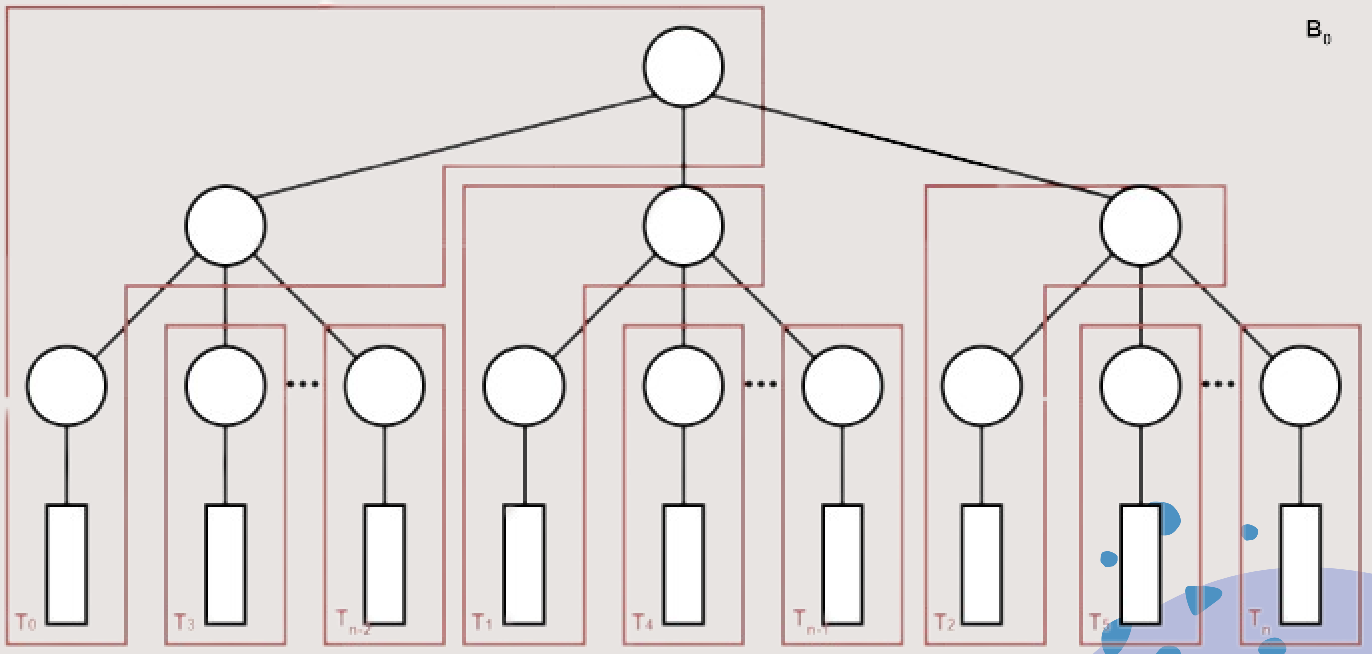
Nimply representation



V3 – More levels in shared memory

The third version tries to achieve a **higher level of parallelism**:

- One level of tree search removed from stack.
- Square the threads.
- Several results arrays in the *shared memory*.
- Each block evaluates one child nodes for each thread.



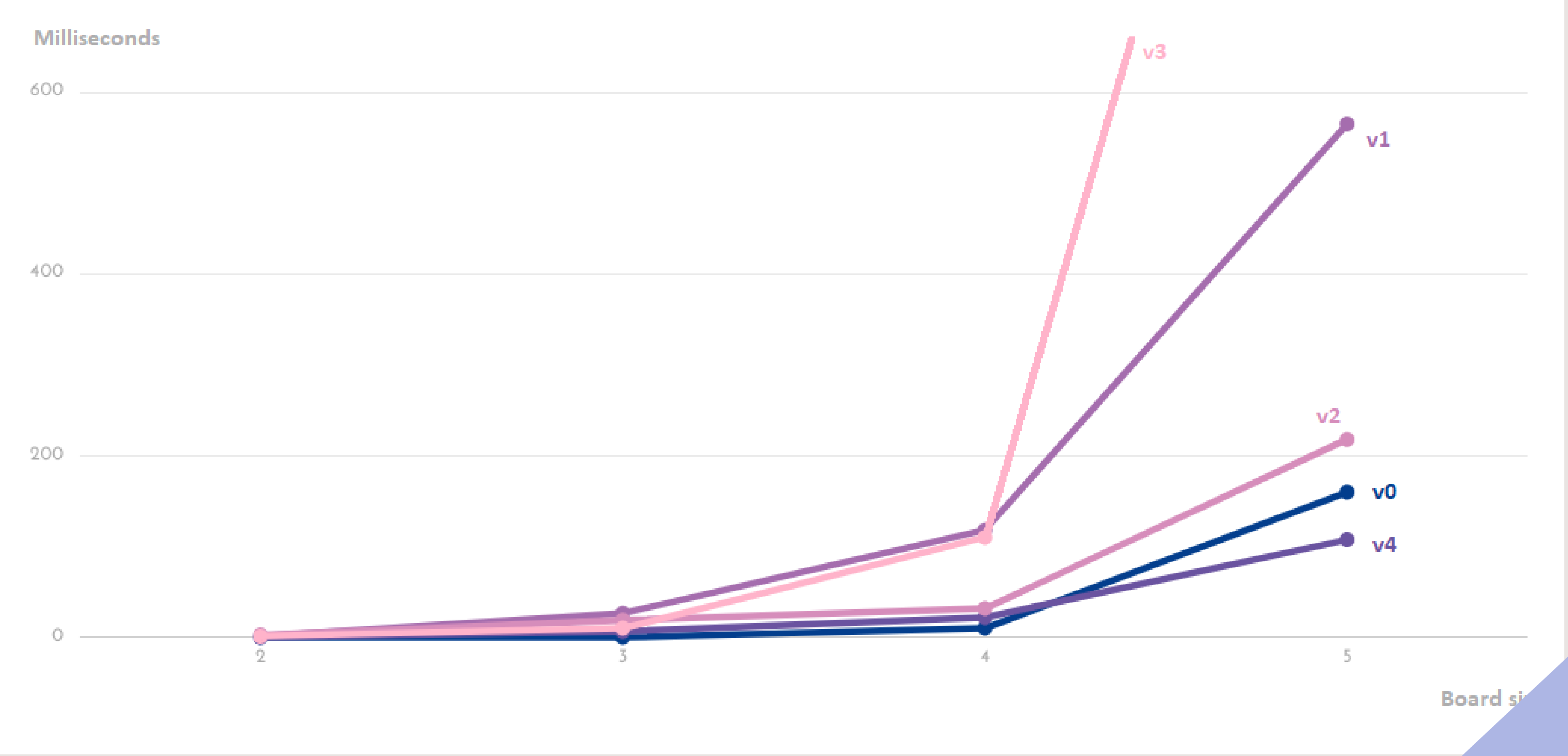
V4 - Memory transactions reduced

The fourth version tries to remove the overhead added by the **memory transactions between the host and the device**:

- Based on v2
- The kernel starts from the very first game state
- The optimal move is returned

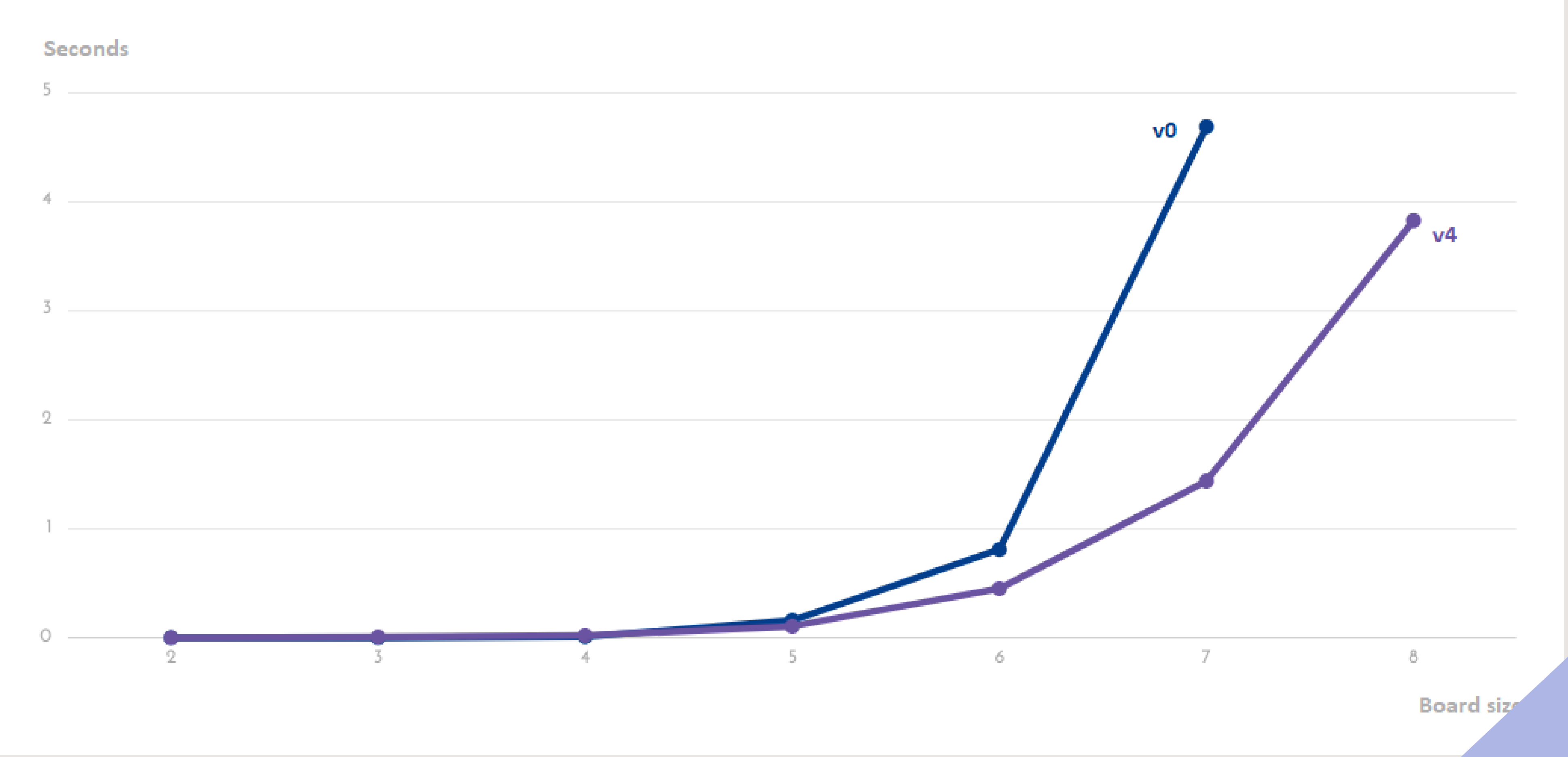
Results and Analysis

Duration in *milliseconds* of the minmax algorithm for different board sizes. The maximum depth was set to 7.



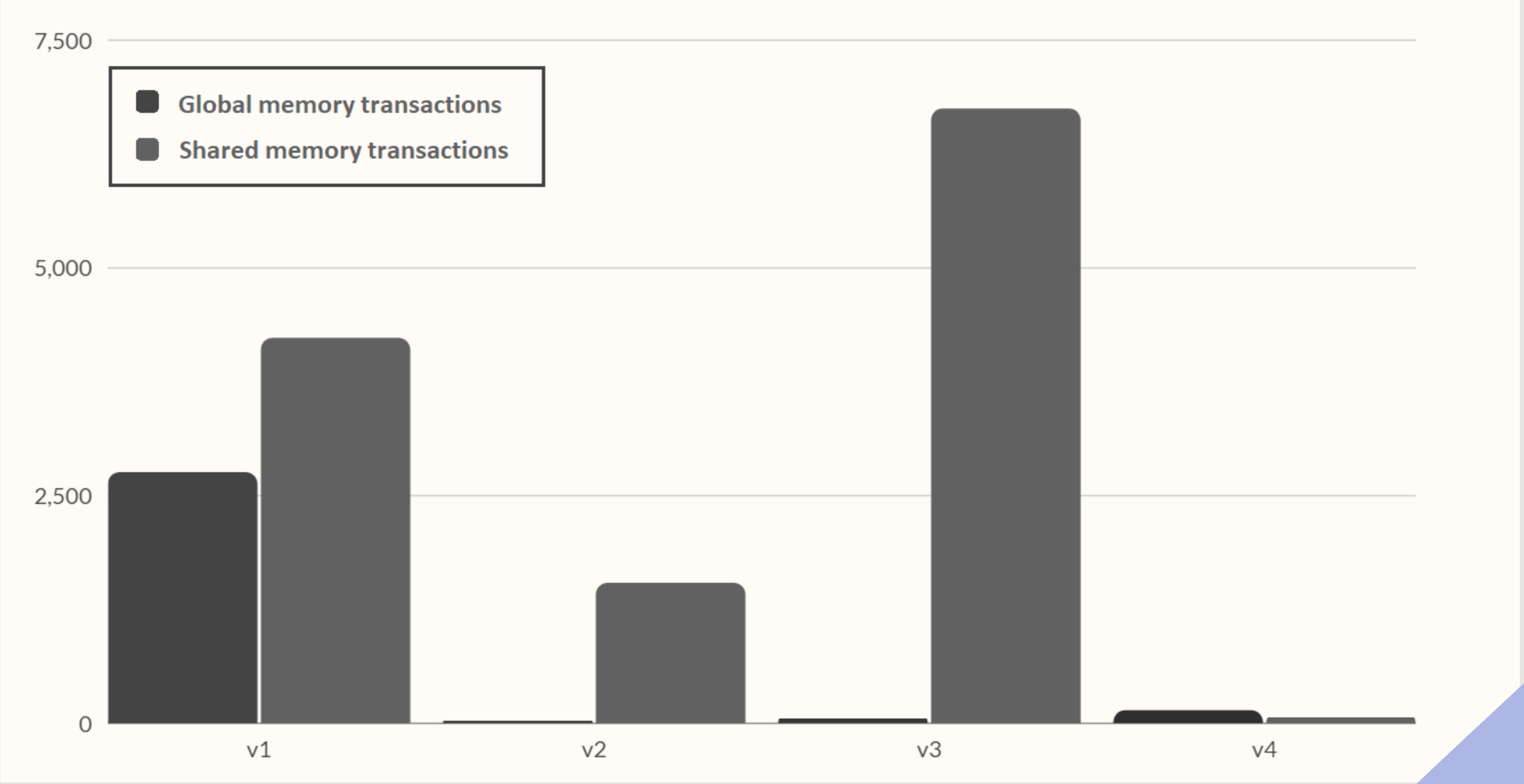
Results and Analysis

Duration in *milliseconds* for different board sizes. The best version (v4) is compared with the base one.



Results and Analysis

Comparison between the number of **global and shared transactions** (read and write) with a board of 5 rows.



Results and Analysis

Some observations:

- Between v1 and v2, the **global transactions** decreased by 98.98%, while the **shared** ones decreased by 63.65%.
- Despite the **higher level of parallelization**, v3 performs too many transactions in the shared memory.
- The **global transactions** between v4 and v2 increased, but the **shared** and the **host-device** ones decreased, leading to better performances.

Results and Analysis

Some observations:

- The algorithm resulted extremely efficient for bigger boards, while the **GPU resources** are underused on smaller boards.
- One of the main problems of these implementation is the **warp divergence**.
- The used **GPU technology** is not able to perform a full create-and-search tree algorithm, without setting up a maximum explorable depth.

Conclusion

The GPU is able to calculate the optimal move **faster** than the CPU for almost all the board sizes, despite the relatively **low complexity** of the Nim game.

Some possible new implementations:

- Parallelize the **evaluation function**
- **Remove** the standard minmax from the threads
- Extend the number of **threads**
- **Split the tree** search between the host and the device

The end

Thank you for your attention!