GamesCrafters with GPU

Shangdian Han

04/20/2023

NVIDIA GeForce GTX 1080 Ti

- 28 streaming multiprocessors (SMs)
  - 128 CUDA cores per SM, total: 3584 CUDA cores
  - 2048 threads per SM, total: 57344 threads
- Many layers between SM and threads. Some can be 3D.

- Memory (dedicated): 11 GB
- Max Clock rate: 1582 MHz (1.58 GHz)

Intel® Core™ i7-8700K Processor

- Cores: 6
- Threads (Intel® Hyper-Threading): 12

- Memory: 16 GB (6~7 GB free)
- Processor Base Frequency: 3.70 GHz
- Max Turbo Frequency: 4.70 GHz

Can we solve Tic Tac Toe?

With a GPU?

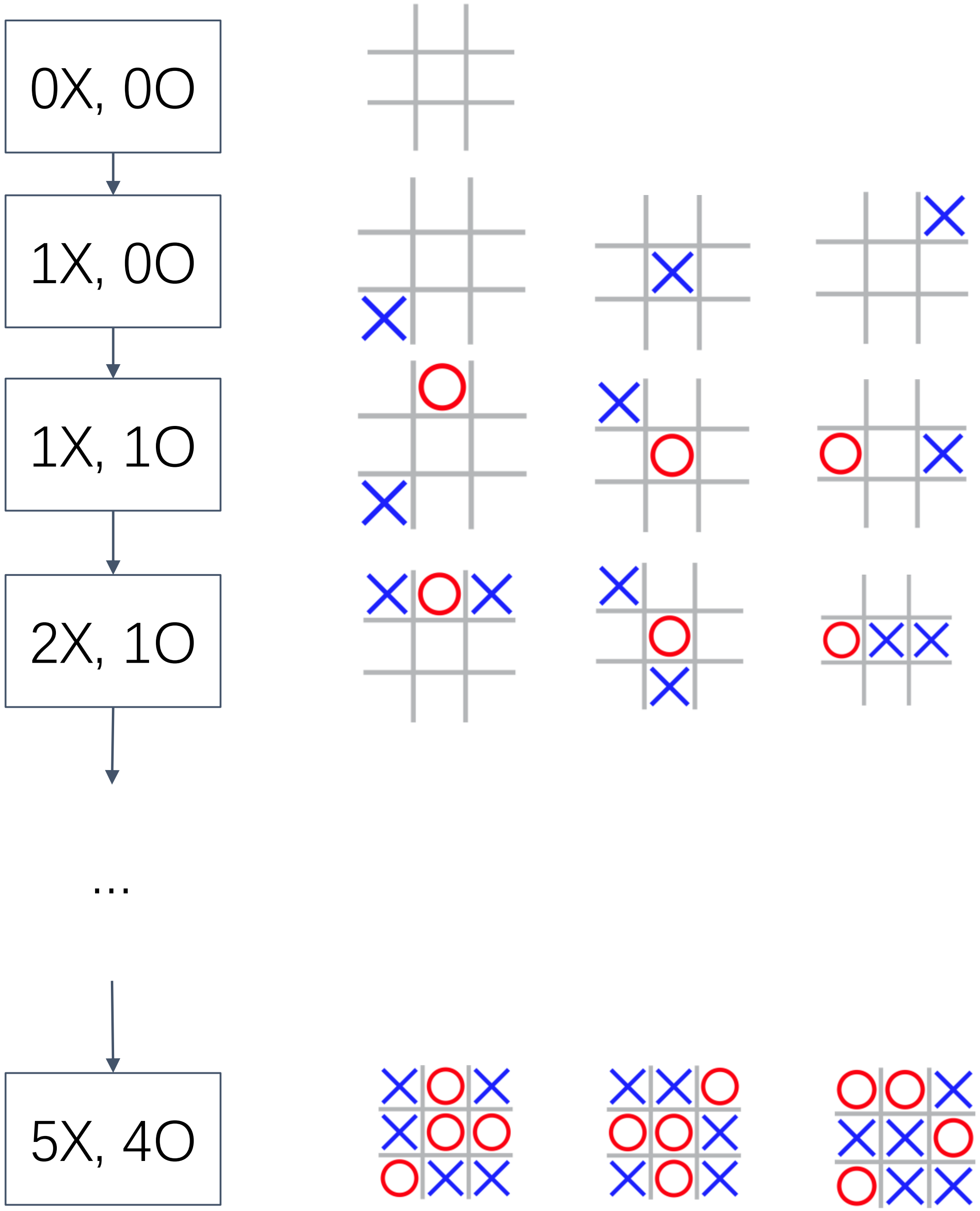
kingh0730/order_and_chaos (github.com)

Solver from homefun

- Recursion causes postorder tree traversal.
    - This order is not ideal for parallelism.

Tiers for parallelism

- Solve from bottom to top.
- Problem: symmetry removal
    - Symmetric positions are computed at the same time.
    - Even if we skip non-canonical positions, the next tier needs to wait until all computations finish in the current tier.
    - GPU threads are **not as independent** as CPU threads.



0X, 0O

1X, 0O

1X, 1O

2X, 1O

…

5X, 4O

Order & Chaos (4x4, 4 in a row)

```
#define FIR_ROW_OOOO
        (uint32_t)0b00000000000000000000000010101010
#define FIR_ROW_XXXX
        (uint32_t)0b00000000000000000000000011111111
#define FIR_COL_OOOO
        (uint32_t)0b00000010000000100000001000000010
#define POS_DIA_OOOO
        (uint32_t)0b10000000001000000000100000000010
```

CUDA Crash Course

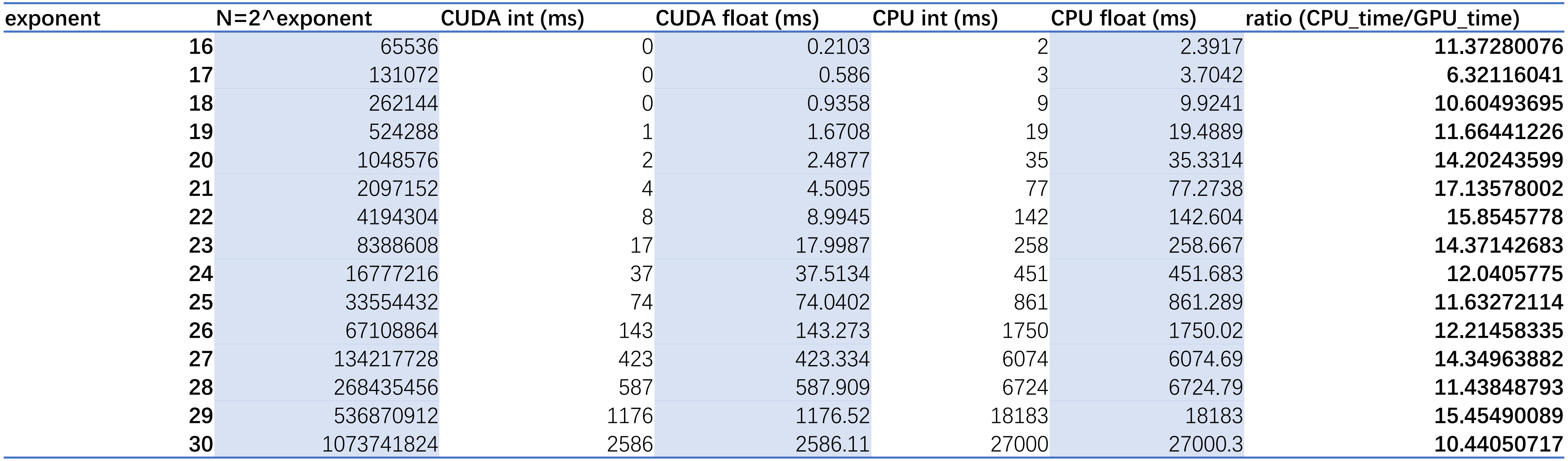
```
__global__ void call_cuda(uint32_t *a, bool *b, int N)
{
    int tid = (blockDim.x * blockIdx.x) + threadIdx.x; // thread ID

    if (tid < N)
    {
        // If within boundary, do stuff.
    }
}

call_cuda<<<GRID_SIZE(N, BLOCK_SIZE), BLOCK_SIZE>>>(a, b, N);
```

Speed comparison

- N is the number of positions calculated. Fix BLOCK_SIZE=1024.
- Order &Chaos (4x4, 4 in a row): Does Order win in this position?

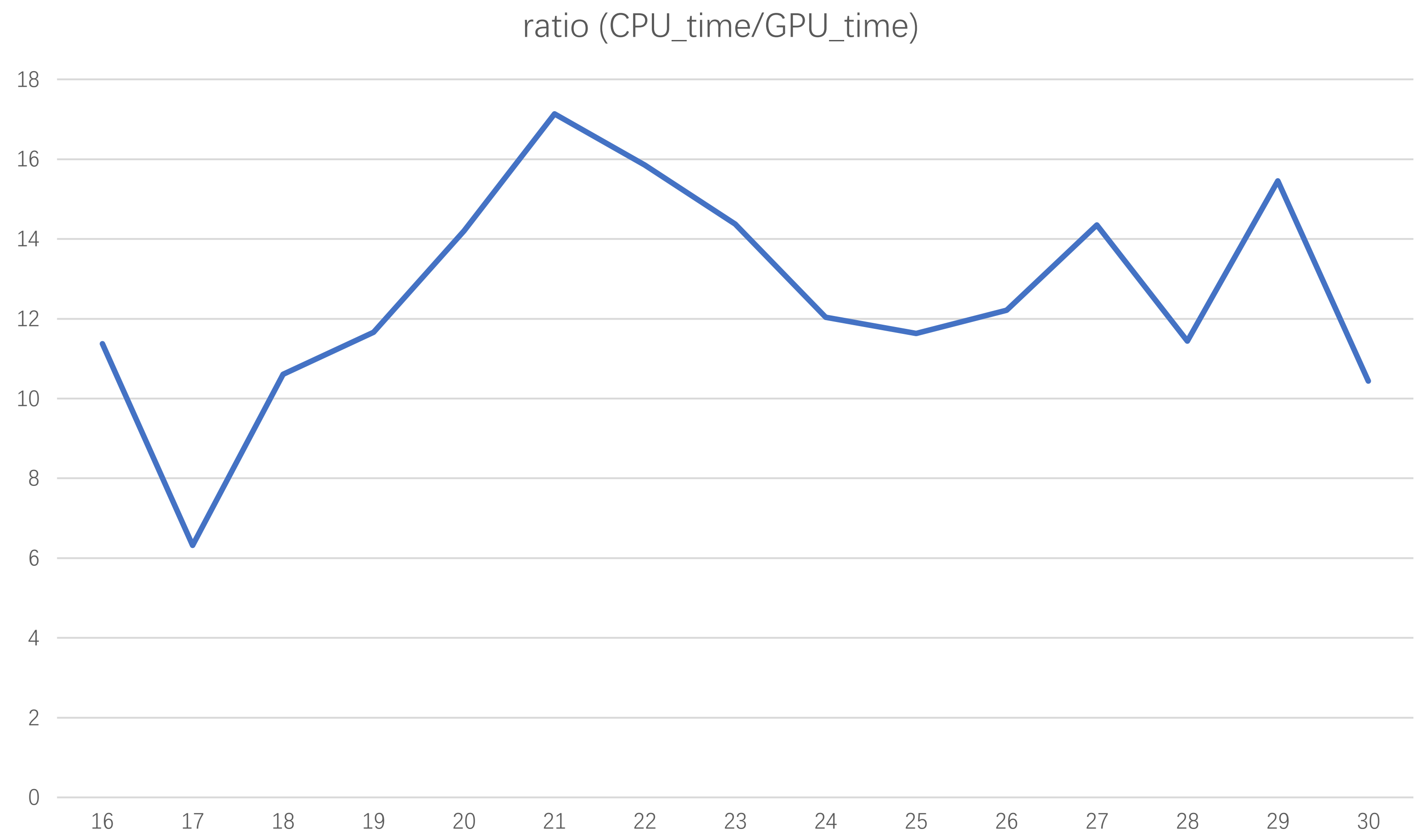


exponent	N=2^exponent	CUDA int (ms)	CUDA float (ms)	CPU int (ms)	CPU float (ms)	ratio (CPU_time/GPU_time)
16	65536	0	0.2103	2	2.3917	**11.37280076**
17	131072	0	0.586	3	3.7042	**6.32116041**
18	262144	0	0.9358	9	9.9241	**10.60493695**
19	524288	1	1.6708	19	19.4889	**11.66441226**
20	1048576	2	2.4877	35	35.3314	**14.20243599**
21	2097152	4	4.5095	77	77.2738	**17.13578002**
22	4194304	8	8.9945	142	142.604	**15.8545778**
23	8388608	17	17.9987	258	258.667	**14.37142683**
24	16777216	37	37.5134	451	451.683	**12.0405775**
25	33554432	74	74.0402	861	861.289	**11.63272114**
26	67108864	143	143.273	1750	1750.02	**12.21458335**
27	134217728	423	423.334	6074	6074.69	**14.34963882**
28	268435456	587	587.909	6724	6724.79	**11.43848793**
29	536870912	1176	1176.52	18183	18183	**15.45490089**
30	1073741824	2586	2586.11	27000	27000.3	**10.44050717**

Speed comparison

GPU provides a consistent 10x improvement.

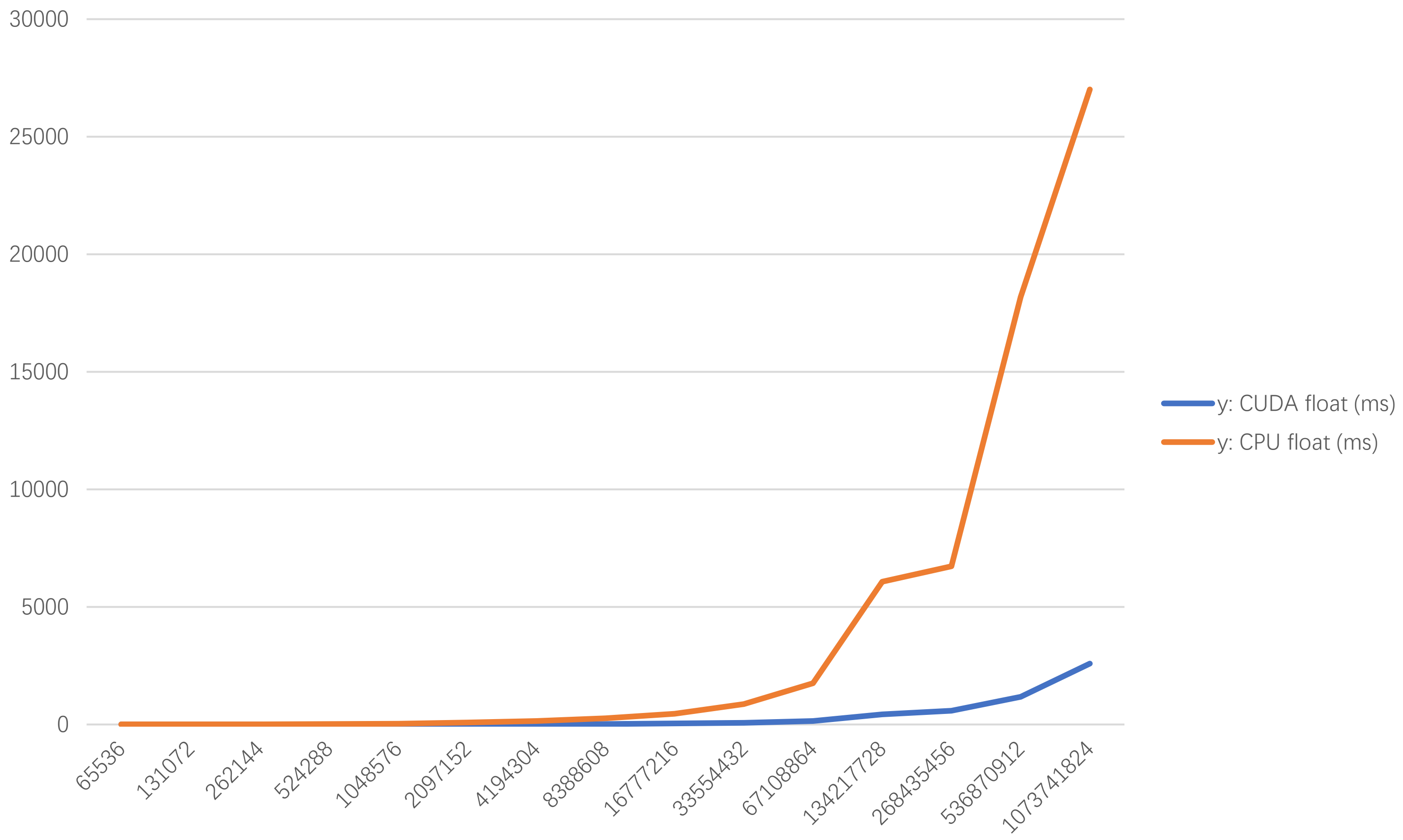
x-axis is log(N).



Speed comparison

Computation time seems to scale exponentially with the number of positions.

x-axis is N.



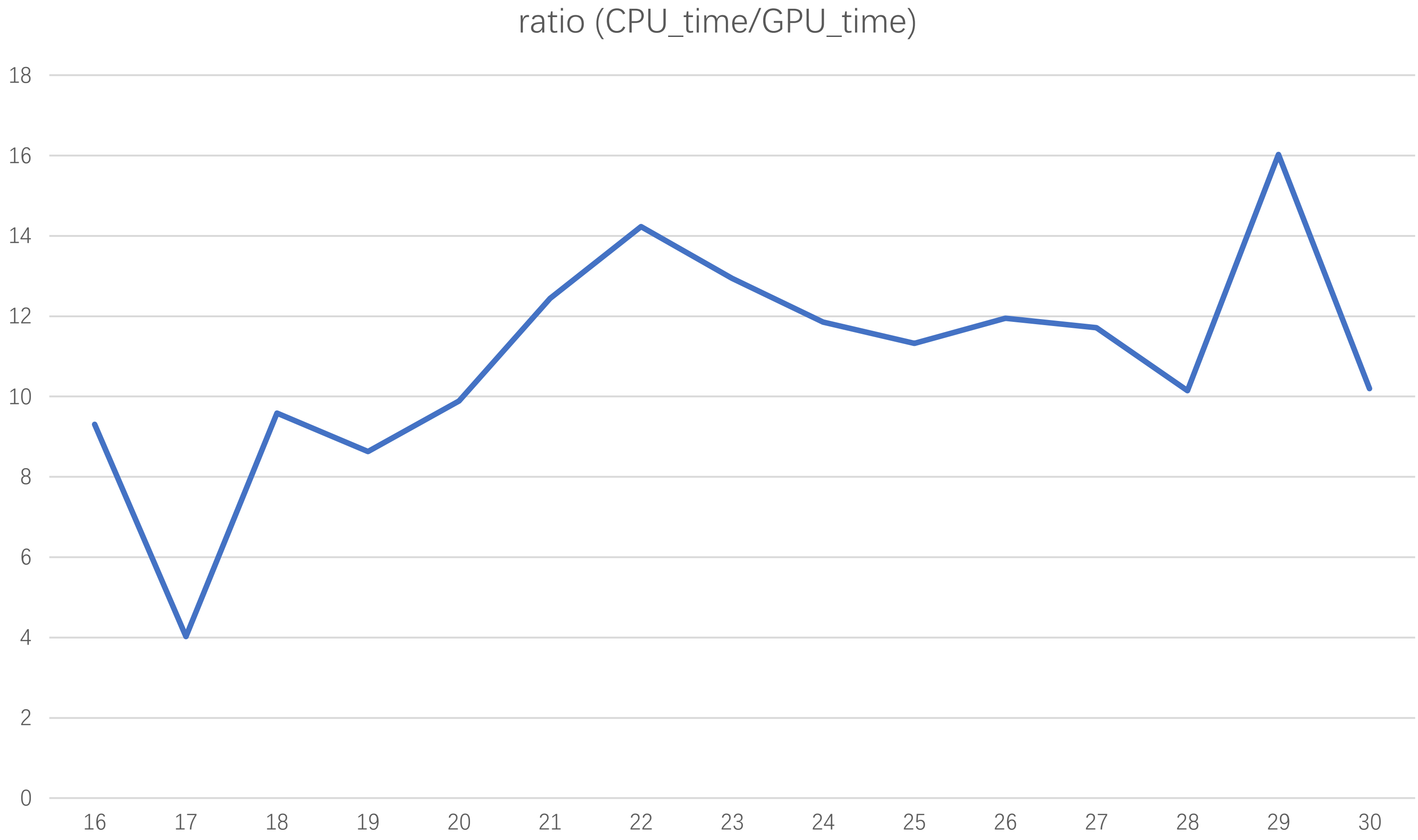
Speed comparison

If we include the time taken to allocate memory (different memories).

GPU still provides a consistent 10x improvement.

x-axis is log(N).

Why the same shape?



Can we solve 6x6 Order & Chaos?

- Assume we use the same GPU.
  - No symmetry removal.
  - Handing-waving a little bit about the complexity of the program.
- Only include checking win condition:
- $\left(\frac{3^{6*6}}{1073741824}\right) * 2586\, ms \approx 11.5\; years$

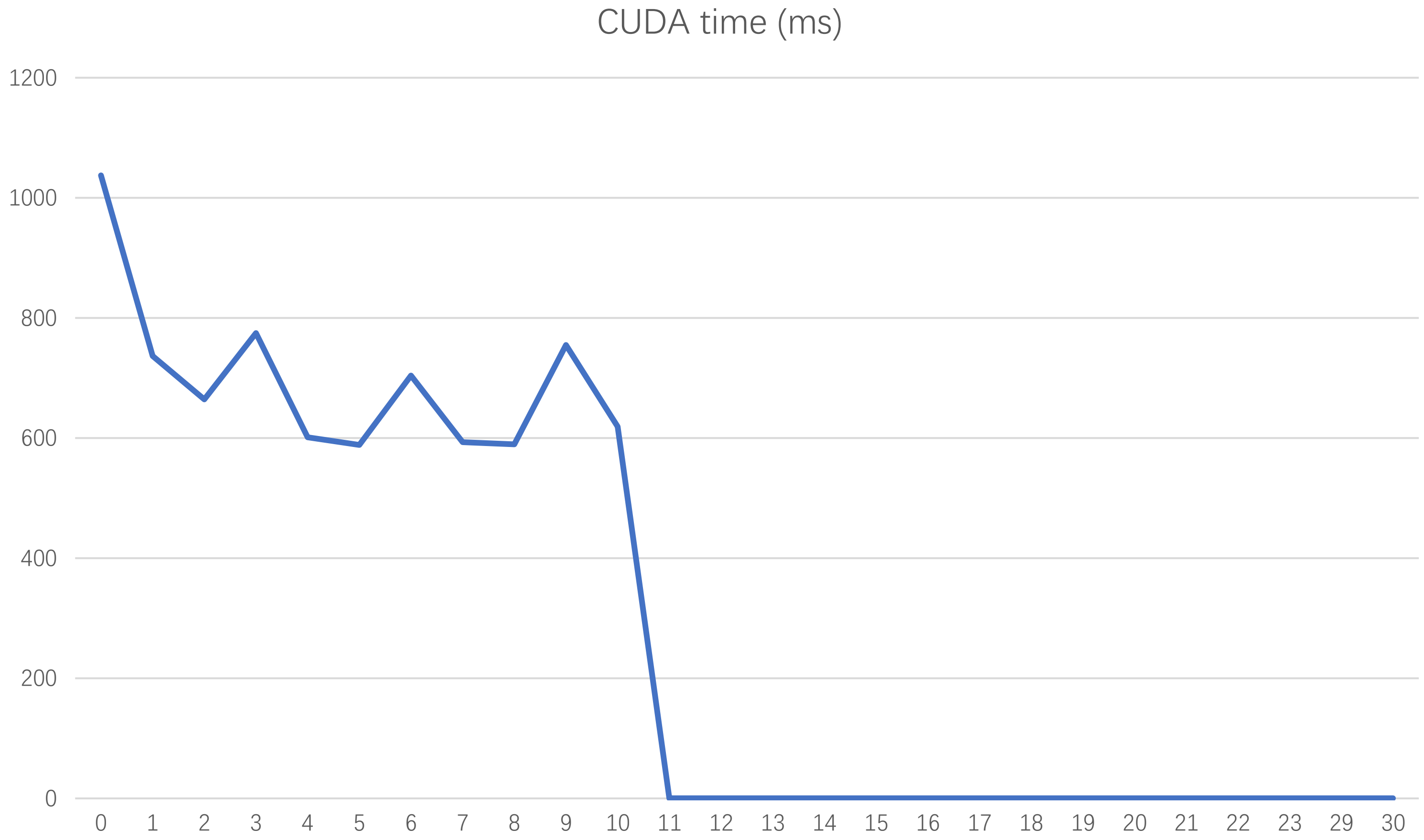
Speed comparison

Fix N=2^28.

BLOCK_SIZE >= 2048 seems optimal.

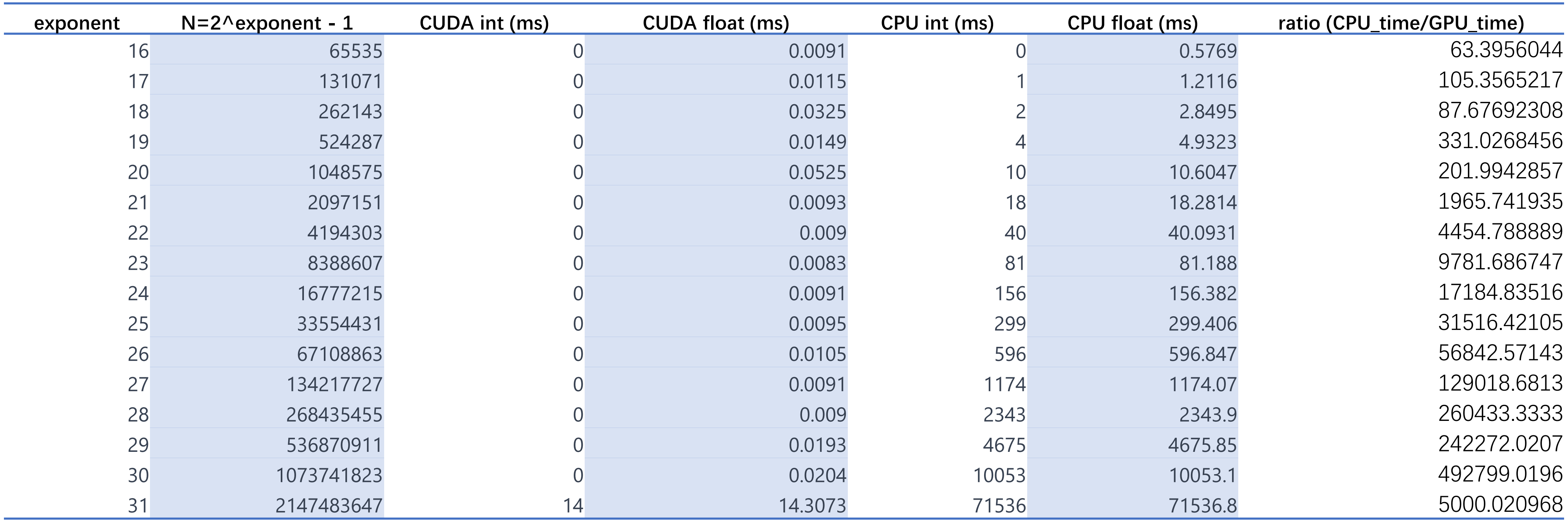
But I used 1024!

x-axis is log(BLOCK_SIZE).



Do everything again…

- With BLOCK_SIZE=2048

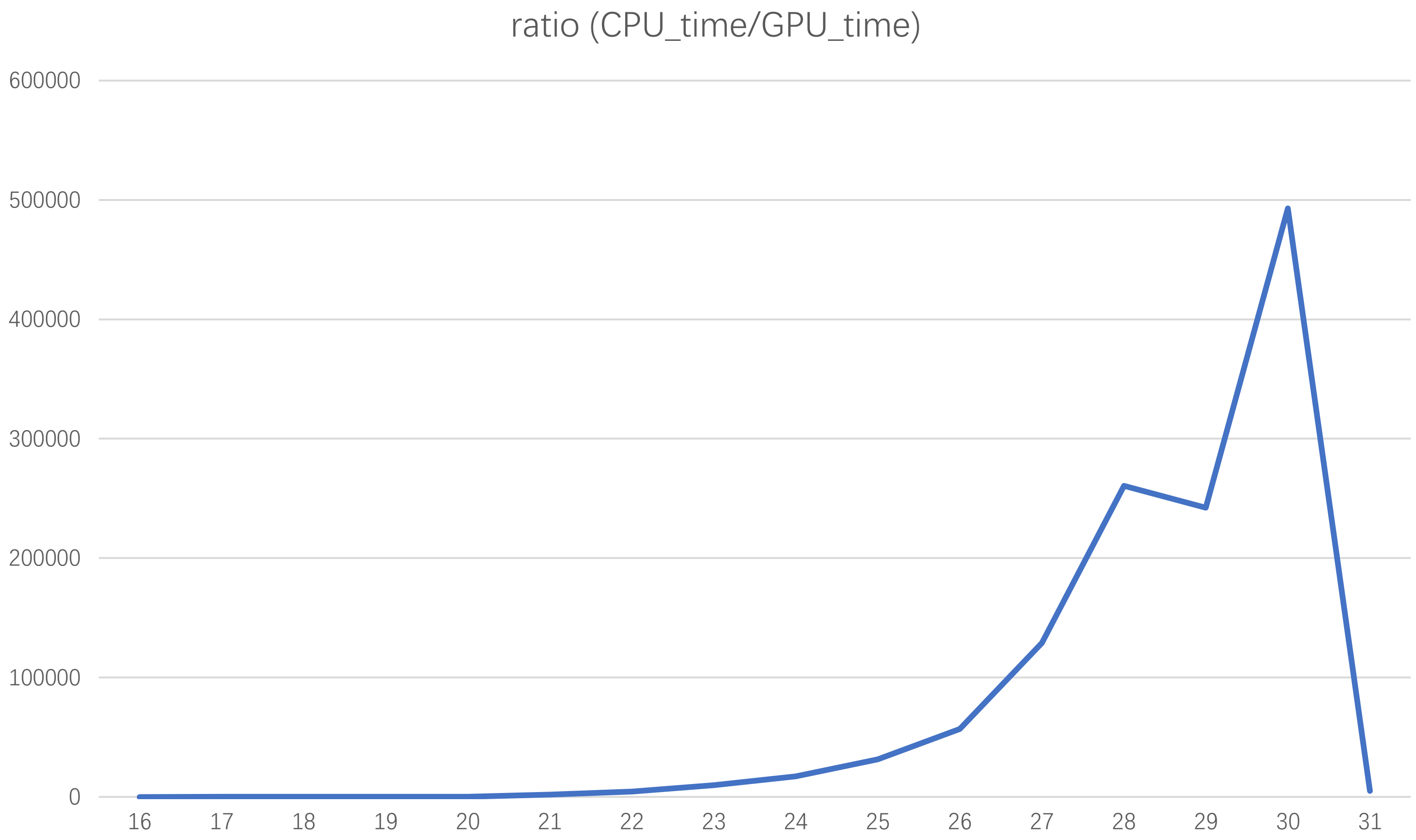


exponent	N=2^exponent - 1	CUDA int (ms)	CUDA float (ms)	CPU int (ms)	CPU float (ms)	ratio (CPU_time/GPU_time)
16	65535	0	0.0091	0	0.5769	63.3956044
17	131071	0	0.0115	1	1.2116	105.3565217
18	262143	0	0.0325	2	2.8495	87.67692308
19	524287	0	0.0149	4	4.9323	331.0268456
20	1048575	0	0.0525	10	10.6047	201.9942857
21	2097151	0	0.0093	18	18.2814	1965.741935
22	4194303	0	0.009	40	40.0931	4454.788889
23	8388607	0	0.0083	81	81.188	9781.686747
24	16777215	0	0.0091	156	156.382	17184.83516
25	33554431	0	0.0095	299	299.406	31516.42105
26	67108863	0	0.0105	596	596.847	56842.57143
27	134217727	0	0.0091	1174	1174.07	129018.6813
28	268435455	0	0.009	2343	2343.9	260433.3333
29	536870911	0	0.0193	4675	4675.85	242272.0207
30	1073741823	0	0.0204	10053	10053.1	492799.0196
31	2147483647	14	14.3073	71536	71536.8	5000.020968

Speed comparison

Now, the ratio is NOT a constant.

x-axis is exponent.

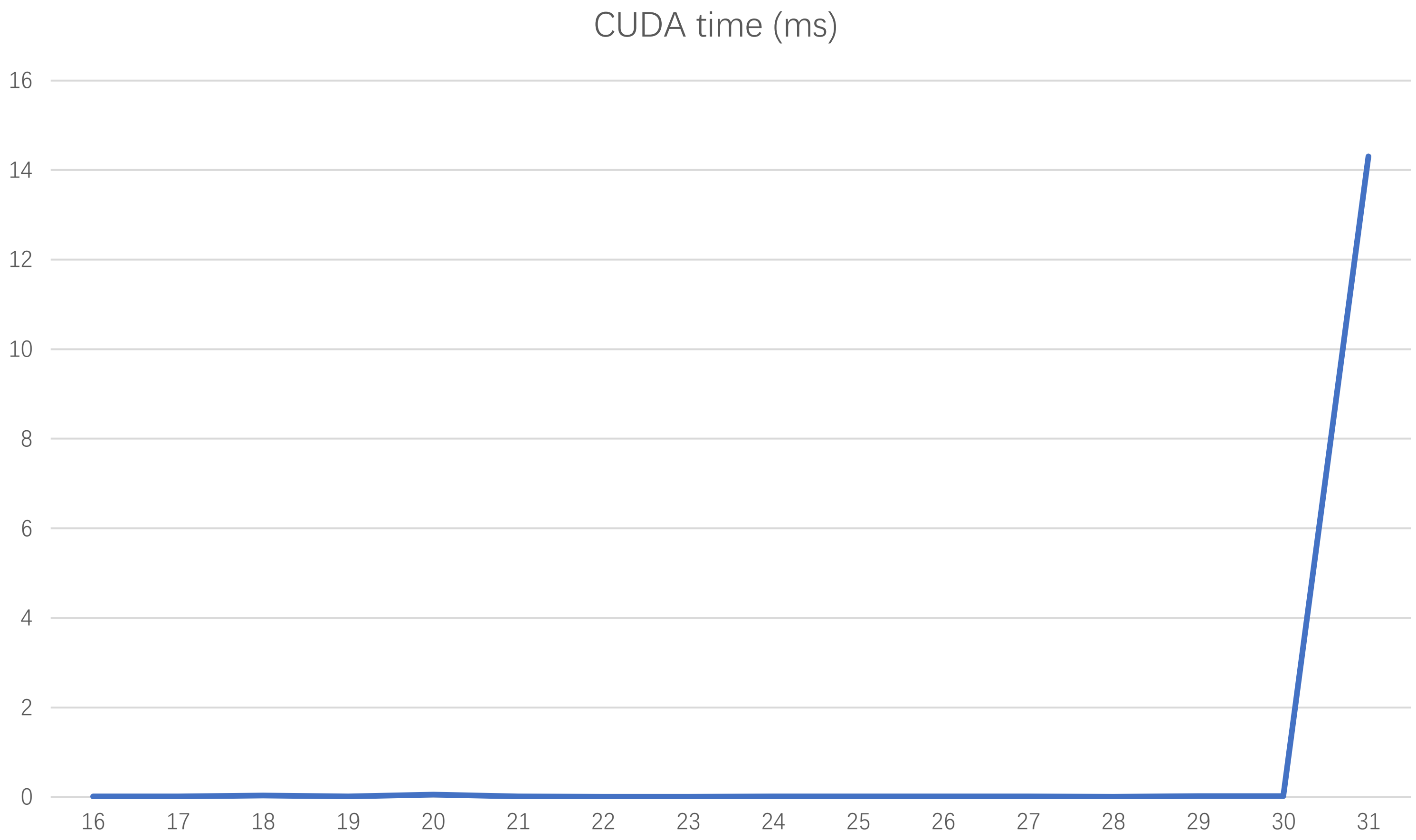


GPU Time

GPU computation time barely changes.

Until the problem space becomes too big.

x-axis is exponent.



Can we solve 6x6 Order & Chaos?

- Assume we use the same GPU.
  - No symmetry removal.
  - Handing-waving a little bit about the complexity of the program.
- Only include checking win condition:
- $\left(\frac{3^{6*6}}{1073741823}\right) * 0.0204\, ms \approx 47.5\, minutes$

Limitations of experiments

- Speed comparison depends highly on the devices.
  - CPU/GPU specs, memory available, clock speed (overclocked?), etc.
- My computer runs many processes, so the speed test environment is not stable.
  - Whether I’m running Chrome, VS Code, Slack, PPT, Excel, etc. makes a huge difference for the CPU.
- CPU computation is unoptimized.

Order & Chaos (4x4, 4 in a row)

- Problems
  - Symmetry removal during parallelism (from previous slides)
  - Memoization using a C++ map might block parallel threads.
  - Memoization using a hash table might consume too much memory.
  - **Cannot** use CPU functions (e.g. std lib) in GPU. Must rewrite.
  - The majority of the runtime is spent constructing the game tree.

- 😢Did not get speed comparison for solving the game
  - Because it's too big to solve using a CPU within the time available.
  - And I have not solved it using a GPU.

Suggestions for future research

- How to parallelize other tasks, like **game tree construction**?
- Does the optimal **tier division** depend on the specs of the GPU?
- What exactly are the layers between SM and threads on a GPU?
  - Warps, thread blocks (3D), grids (3D), etc.
  - So that we can optimize the **parameters fed into the GPU**.
- Why is BLOCK_SIZE >= 2048 optimal? Due to GPU structure?
- Why did the same shape show up (from previous slides)?

- With these, hopeful for solving Order & Chaos!

GamesCrafters with Rust

Shangdian Han & Max Fierro

04/20/2023

Rust solver

- kingh0730/gamescrafters-10-to-0-by-1-or-2 (github.com)

- Advantages
  - Minimal rookie mistakes (strongly typed + rich compiler feedback)
  - Performance
  - Fearless concurrency
- Disadvantages
  - Needs good background in memory management.
  - Initial learning curve steeper than Python, Java, etc.

Examples

```
#[derive(Debug)]
pub struct Solver<P, M, PV, RV>
where
    P: Position<M, PV> + PositionKey,
    M: PlayerMove,
    PV: PrimitiveValue + ToRecursiveValue<RV>,
    RV: RecursiveValue,
```

Examples

```
let mut solver = Solver::<_, _, _,
                 GameResultWithRmt>::new(...);

let result = solver.solve(TicTacToePosition {
    board: [[None, None, None],
            [None, None, None],
            [None, None, None]],
    player: TicTacToePlayer::X,
});
```