Optimal parallelization strategy on MiniMax search tree based on Kalah game
基於kalah之最大最小搜尋樹優化平行策略

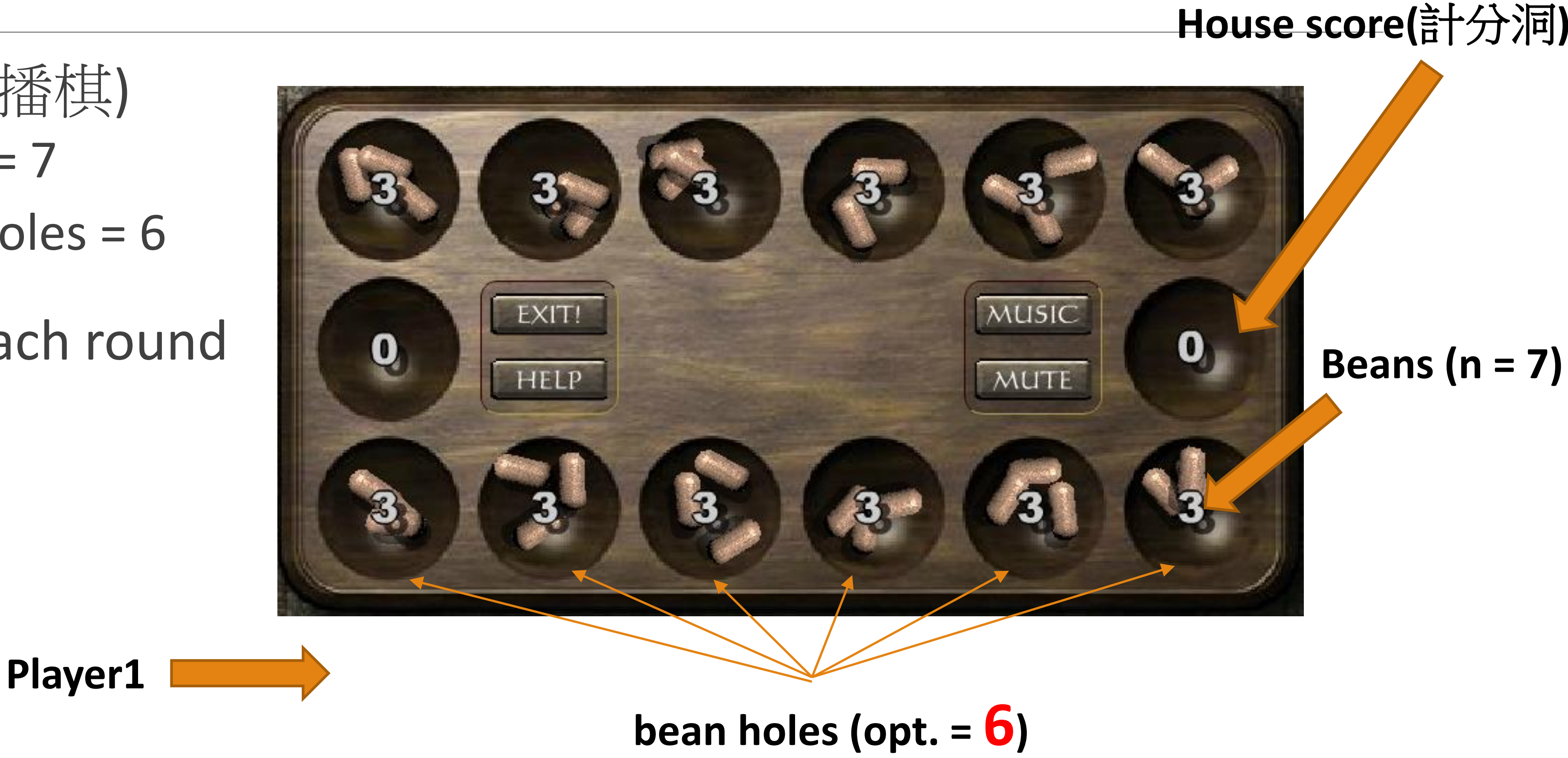
組員: 阮建元 0856566, 陳奕遠 0856148, 李家森 0856126

Outline

- Introduction- Kalah
- Minimax tree
- Pruned Minimax tree
- New Proposed Method: Advanced parallel method on game tree
- Result and Evaluation
- Conclusion

Introduction -Kalah

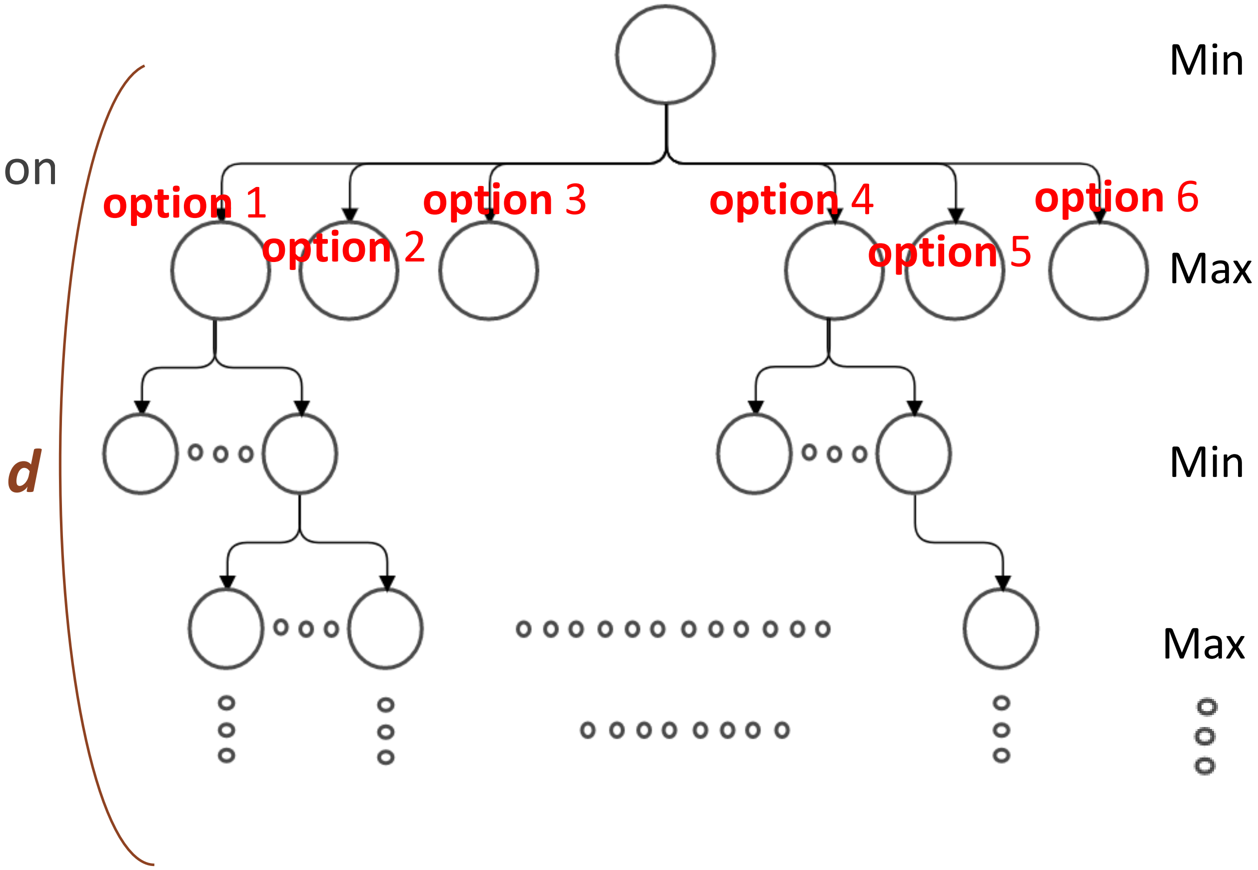
- Kalah、Kalaha(播棋)
  - Assume Beans = 7
  - Assume Bean holes = 6
- **6** options on each round



圖片來源:http://two.game.tw/html/paper1364.html

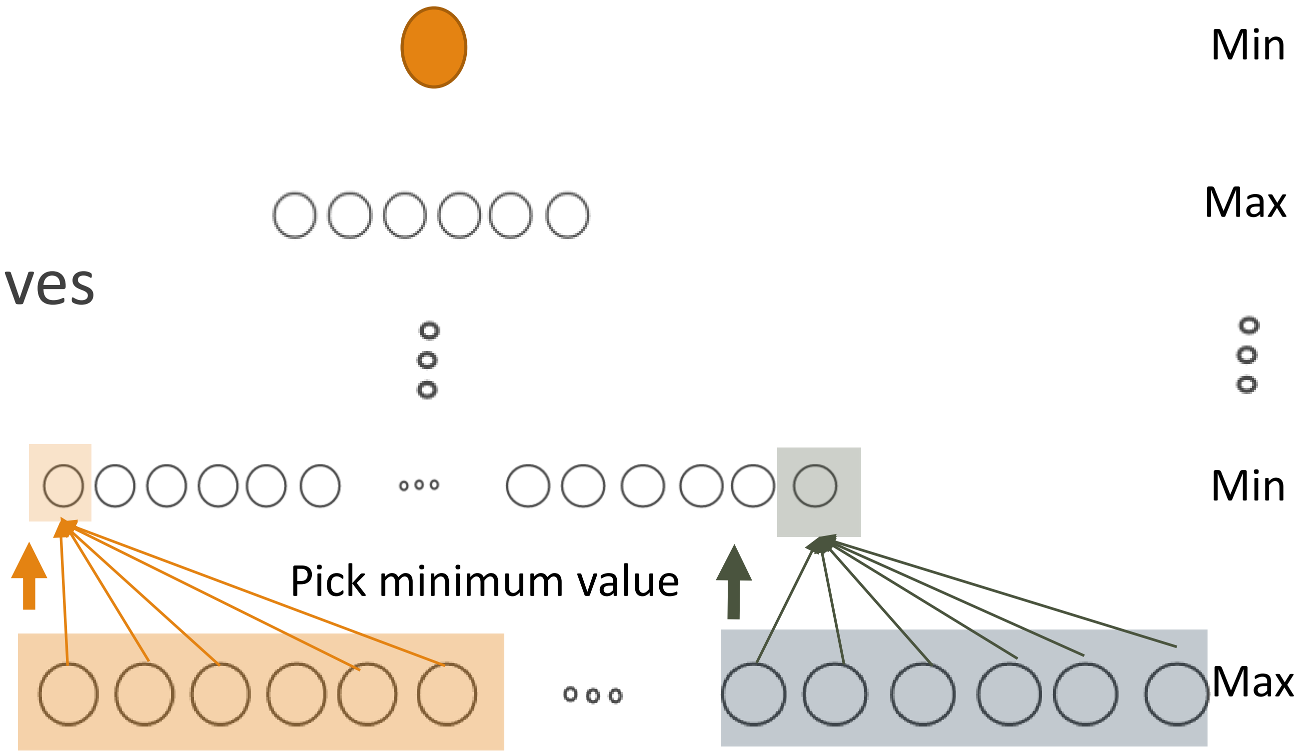
Minimax tree

- Concept:
  - Best **option** to win opponent's best step
  - Expand further ***d*** steps, and predict best option
- complexity: $O(6^{d})$
  - Maxmum game length (*d*)
  - As far as we know, length can be more than ***50***
- Minimax prediction generally use recursion



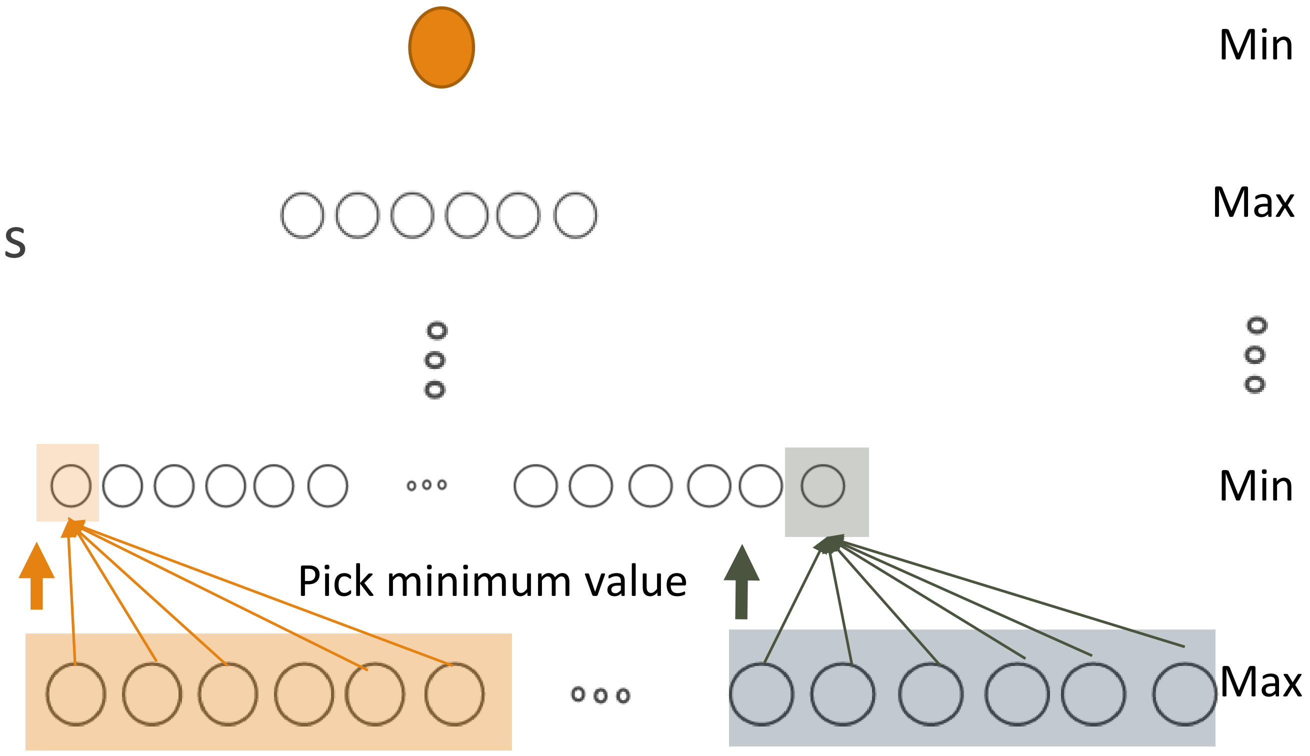
Experiment- (1)

- Time overhead:
  - Reduction
  - Traverse
- assume d =10, yield 362,797,056 leaves
  - Need $6^{10} + \cdots + 6^{1}$ comparison
  - How much parallelization can improve?



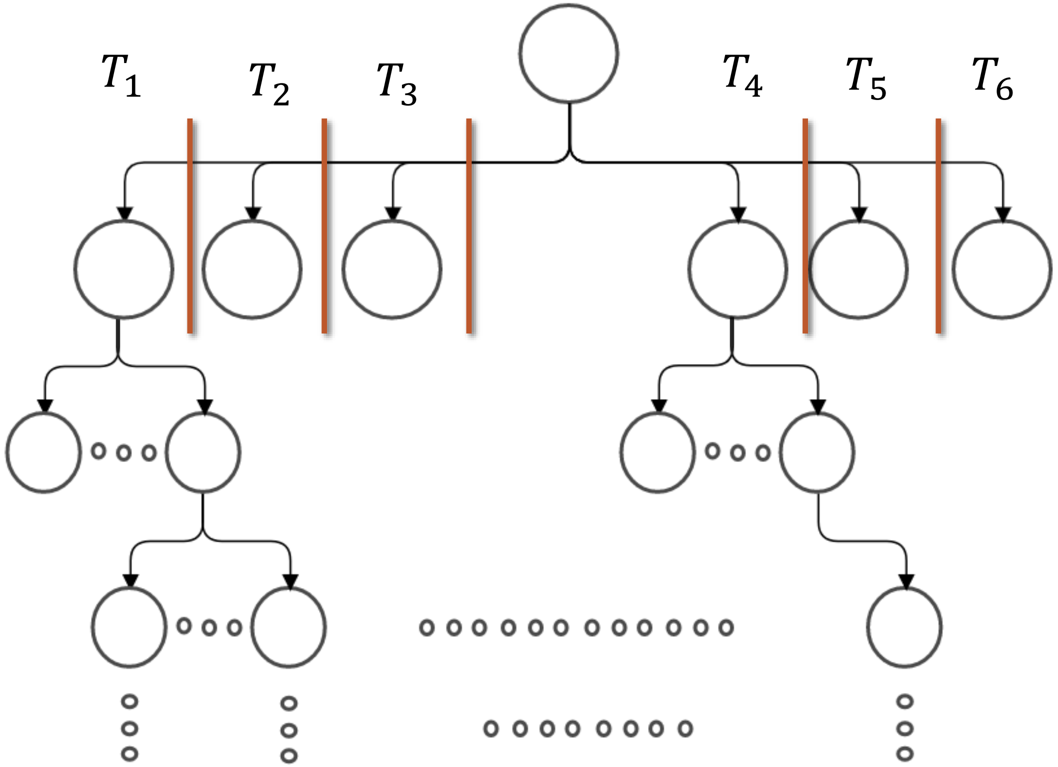
Experiment- (1)

- Time overhead:
  - Reduction
  - Traverse
- assume d =10, yield 362,797,056 leaves
  - Need $6^{10} + \cdots + 6^{1}$ comparison
- Serial Reduction
  - $Time_{reduction}$ < 1% of Total time
  - Most of time spend on traverse

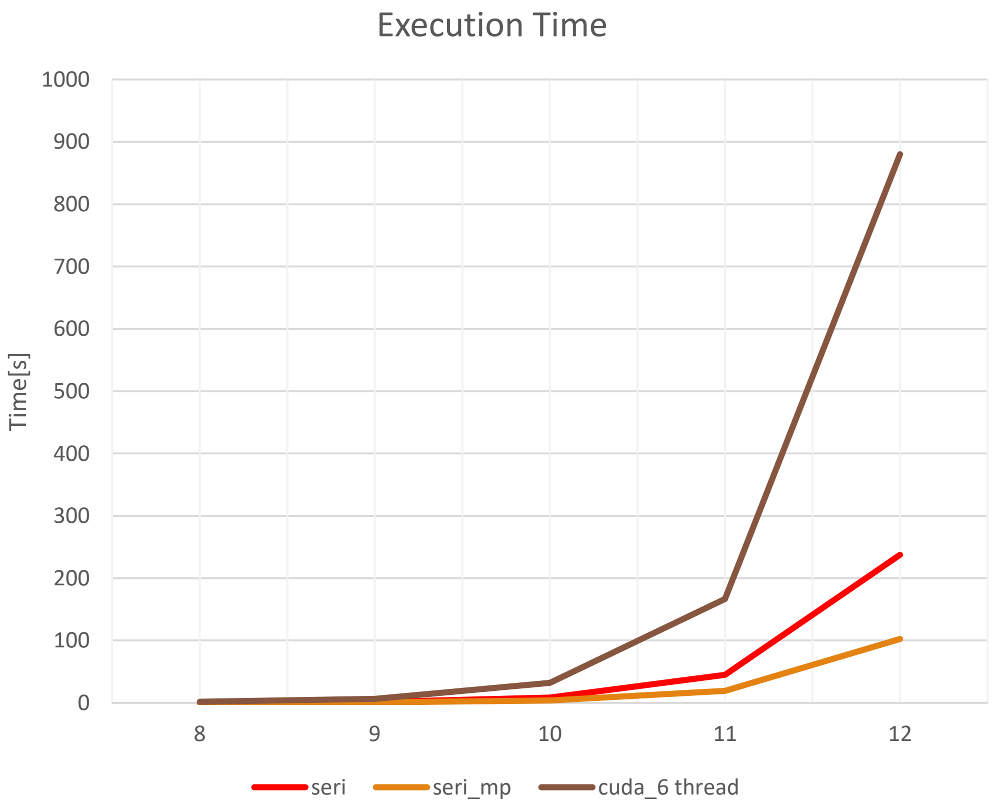


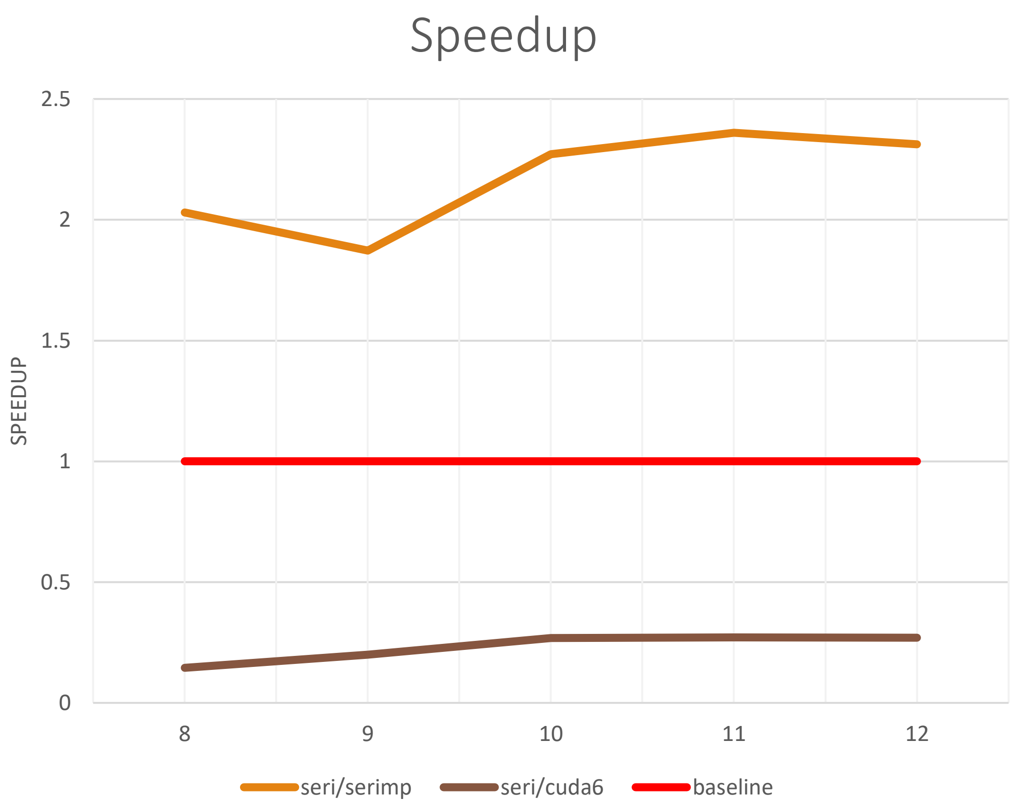
Minimax tree - traverse parallelization

- 6 threads parallel
  - CUDA, openMP respectively



Seri & Seri_Mp

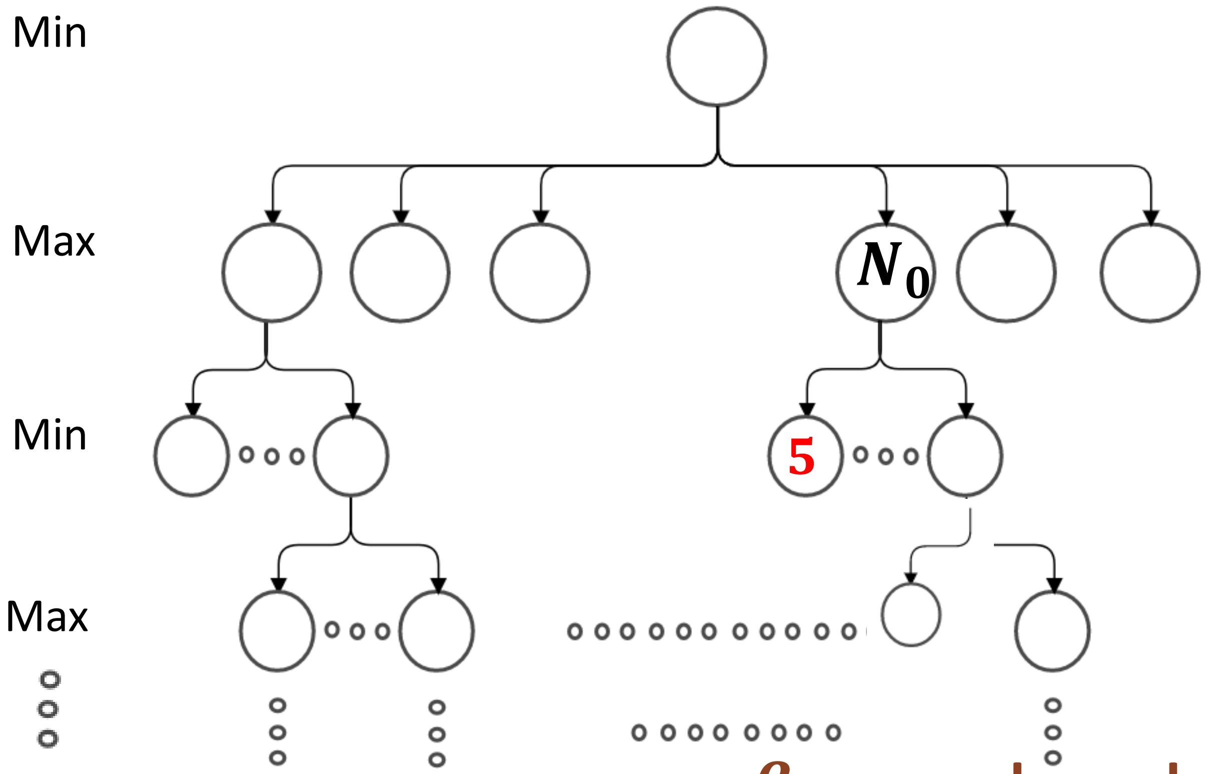

Execution Time
1000
900
800
700
600
500
400
300
200
100
0
Time[s]
8
9
10
11
12
seri
seri_mp
cuda_6 thread


Speedup
2.5
2
1.5
1
0.5
0
SPEEDUP
8
9
10
11
12
seri/serimp
seri/cuda6
baseline

Pruned Minimax tree

- Complexity improvement:
  - $O(6^d) \rightarrow O(6^{d/2})$

Step1: got min value 5



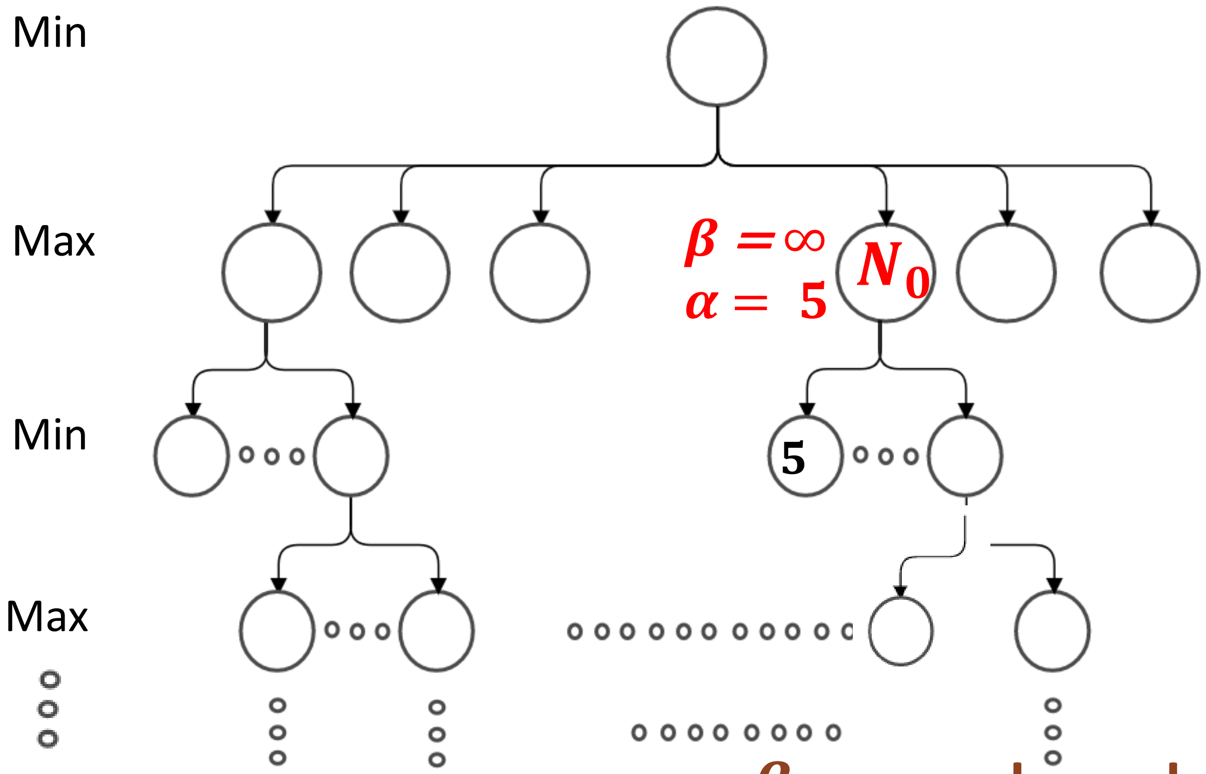
β : upper bound
α : lower bound

Pruned Minimax tree

- Complexity improvement:
  - $O(6^d) \to O(6^{d/2})$

Step1: got min value 5

Step2: N_0 value at least bigger than 5



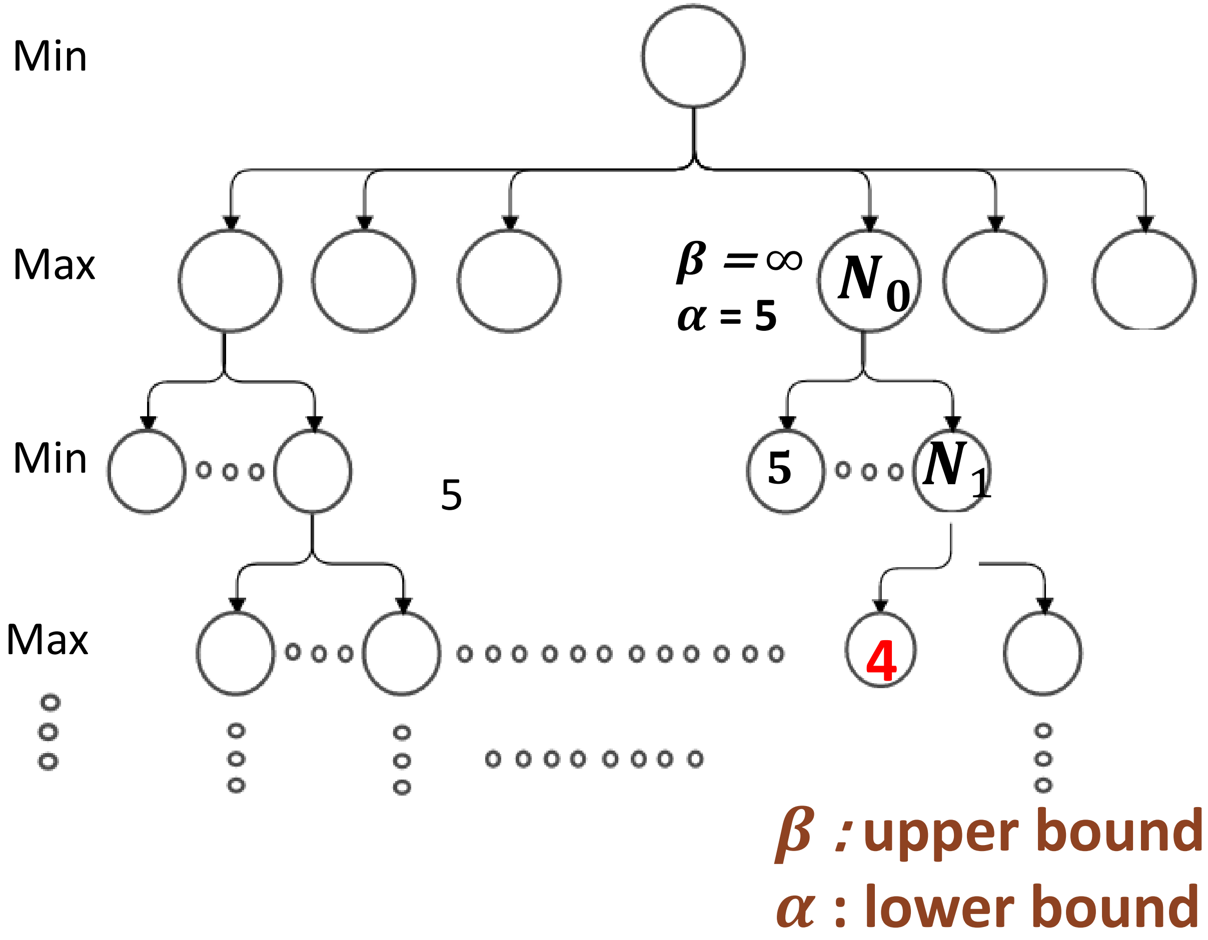
β : upper bound

α : lower bound

Pruned Minimax tree

- Complexity improvement:
  - $O(6^d) \rightarrow O(6^{d/2})$

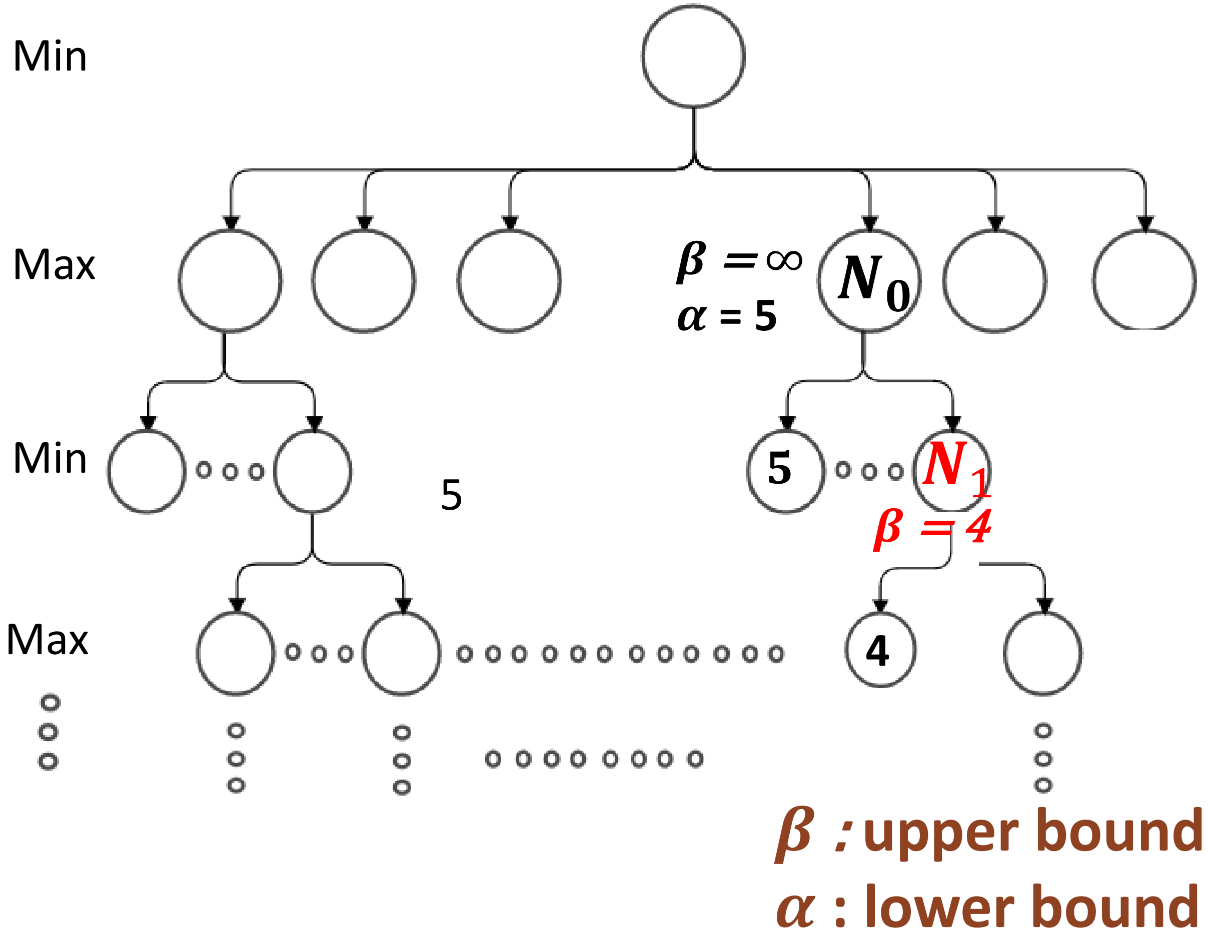
Step1: got min value 5
Step2: N_0 value at least bigger than 5
Step3: Max value 4 of N_1 child



Pruned Minimax tree

- Complexity improvement:
  - $O(6^d) \rightarrow O(6^{d/2})$

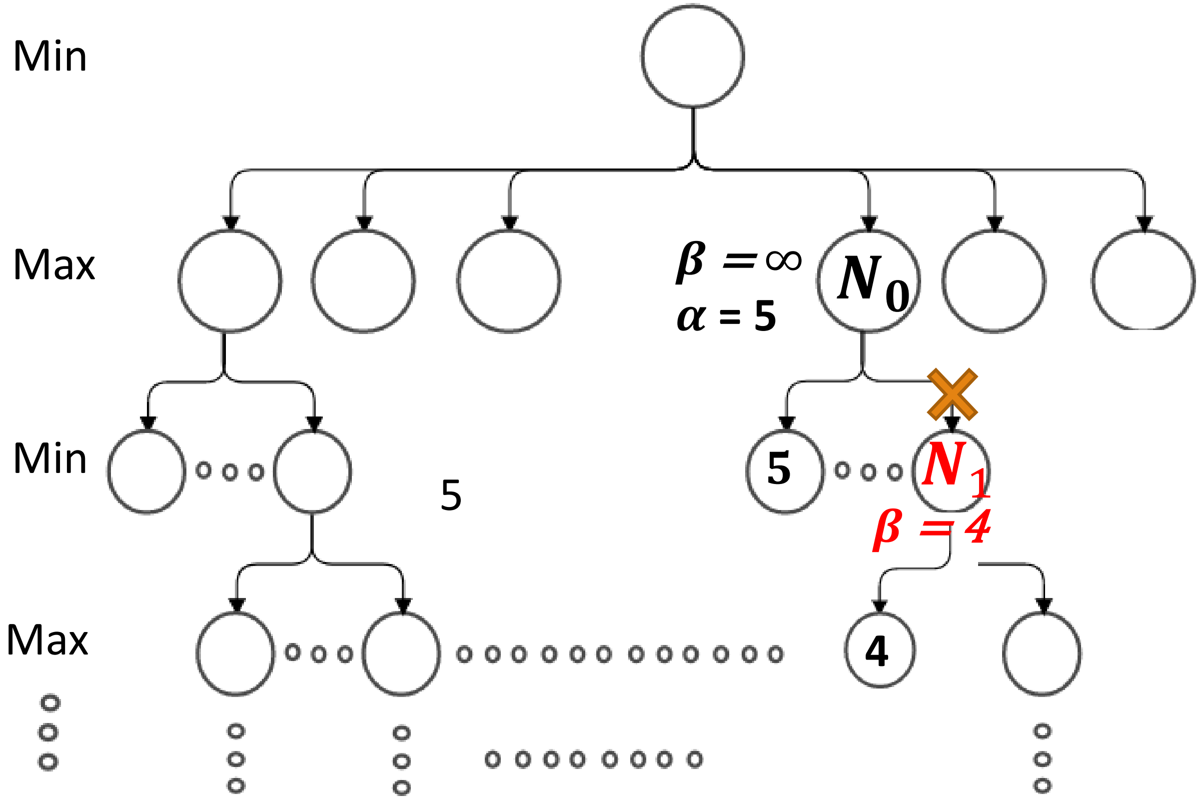
Step1: got min value 5
Step2: N_0 value at least bigger than 5
Step3: Max value 4 of N_1 child
Step4: N_1 value at most 5



Pruned Minimax tree

- Complexity improvement:
  - $O(6^d) \rightarrow O(6^{d/2})$

Step1: got min value 5
Step2: N_0 value at least bigger than 5
Step3: Max value 4 of N_1 child
Step4: N_1 value at most 5
Step5: N_1 would never find number better than 5
-> prune N_1

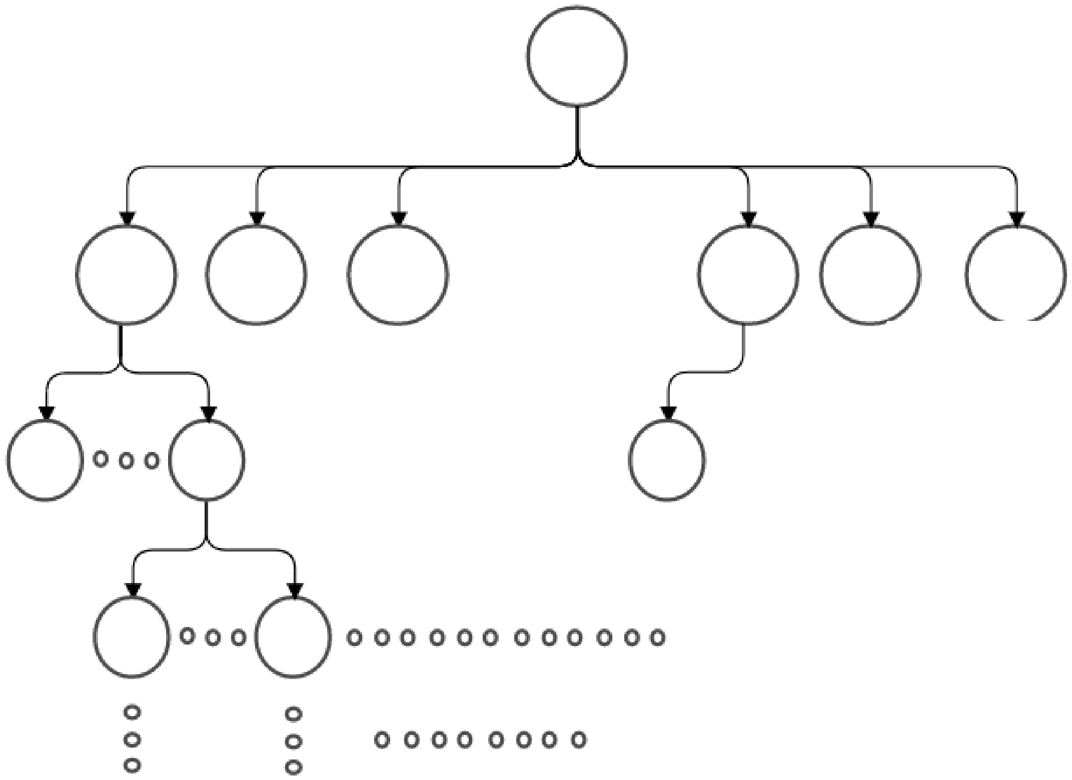


β : upper bound
α : lower bound

Pruned Minimax tree

- Complexity improvement:
  - $O(6^d) \rightarrow O(6^{d/2})$

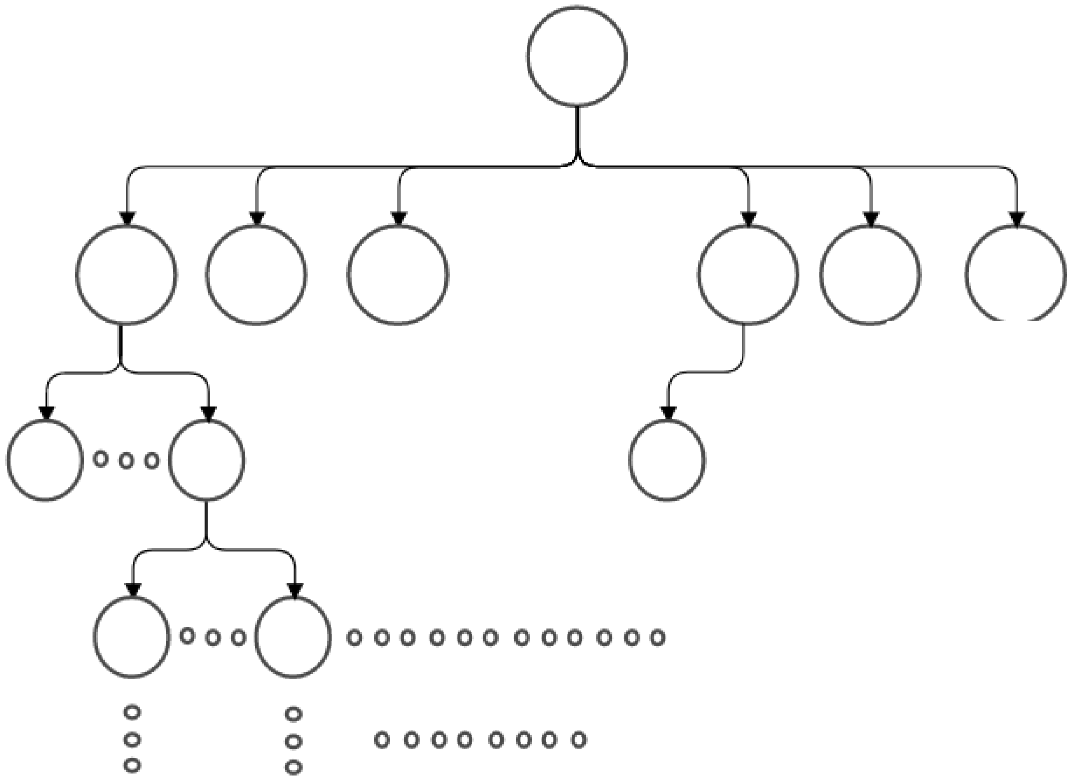
Get a ***pruned tree***



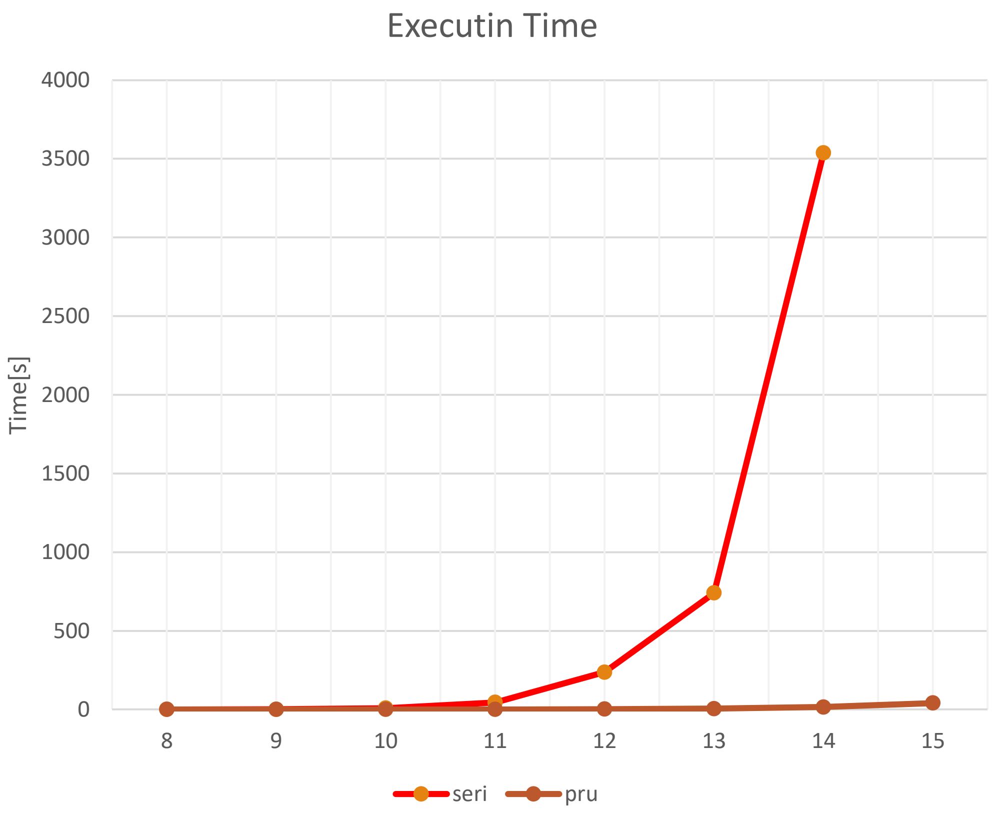
Pruned Minimax tree

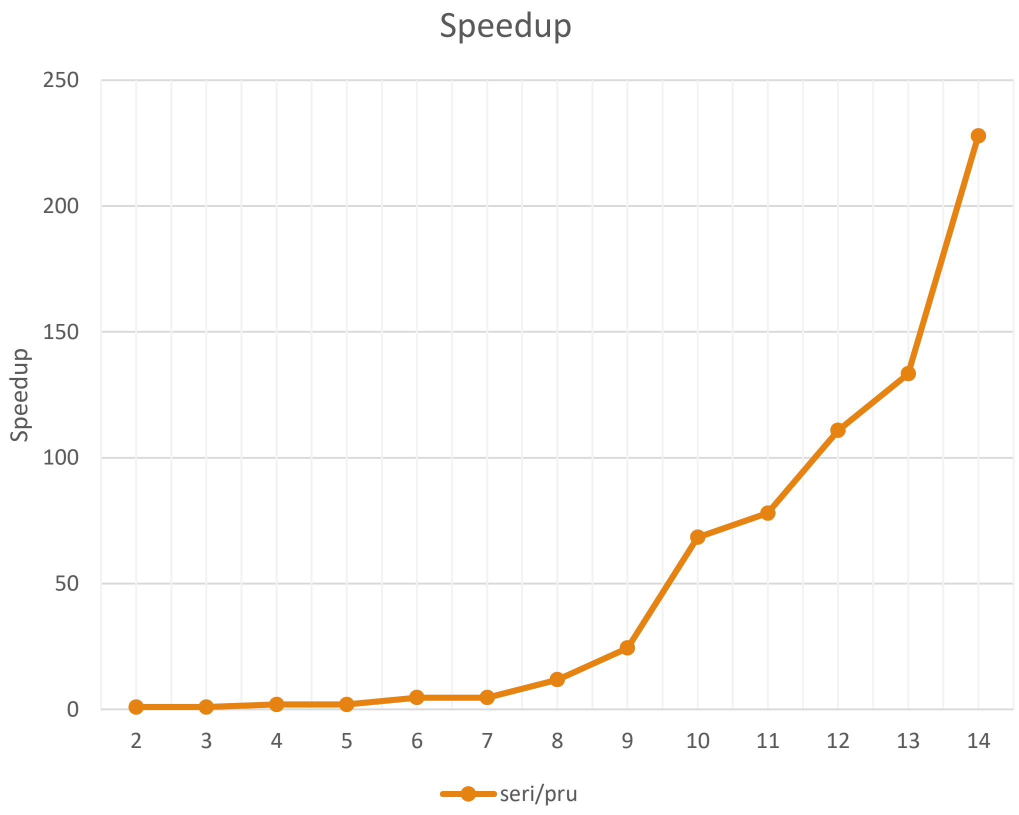
- Complexity improvement:
  - $O(6^d) \rightarrow O(6^{d/2})$

Predict layer	original	pruned	save
1	6	6	0%
2	36	11	69%
3	216	42	81%
4	1,296	123	91%
5	7,776	507	93%
6	46,656	1,622	96%
7	279,936	6,348	98%
8	1,679,616	15,400	99%
9	10,077,696	79,274	99%



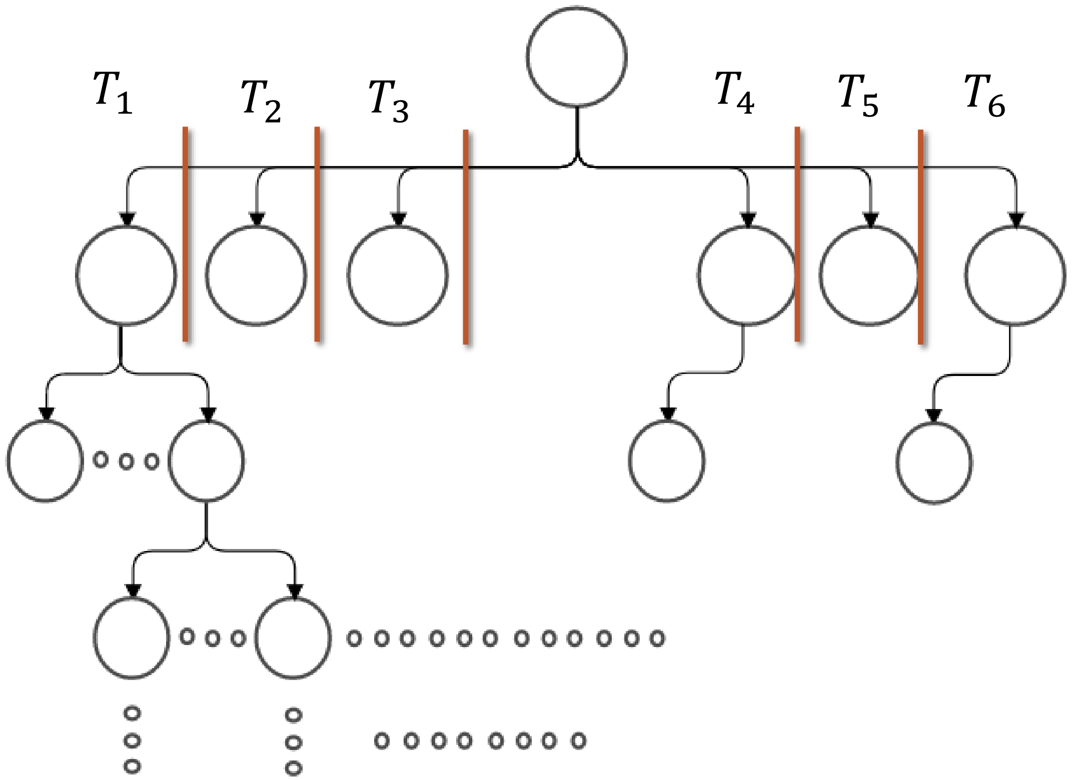
Min-max &min-max prune



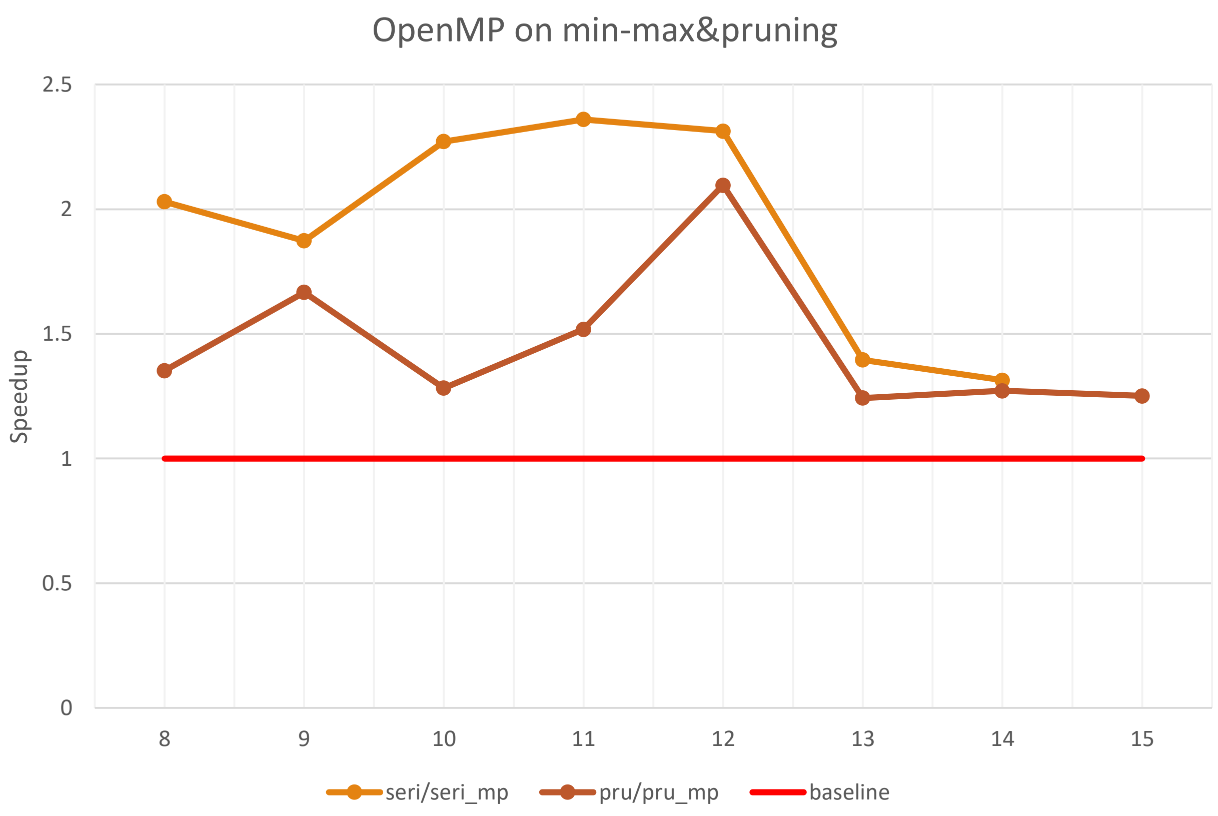


Pruned Minimax tree – Parallelization

- Hard to parallelize recursive algorithm
  - Thread number: 6

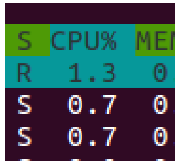


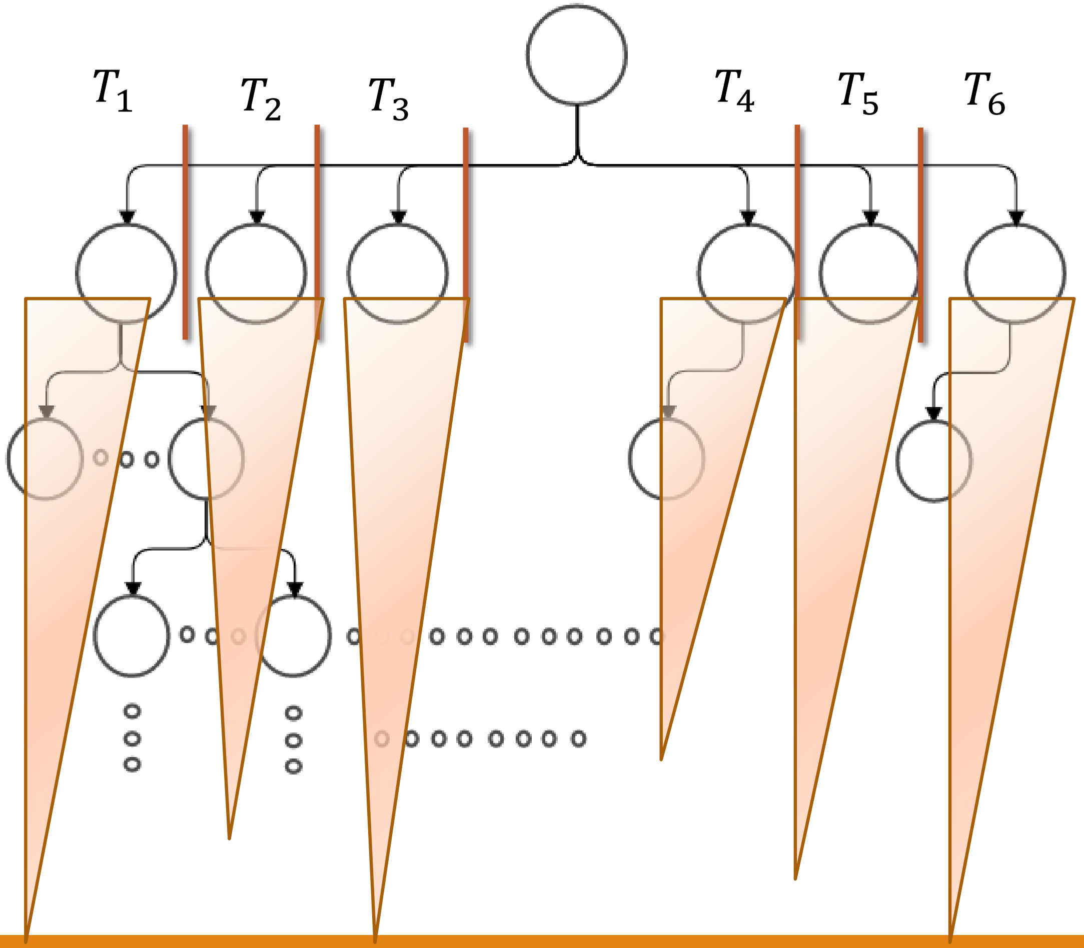
Min-max_mp &prune_mp speedup



Pruned Minimax tree – Parallelization

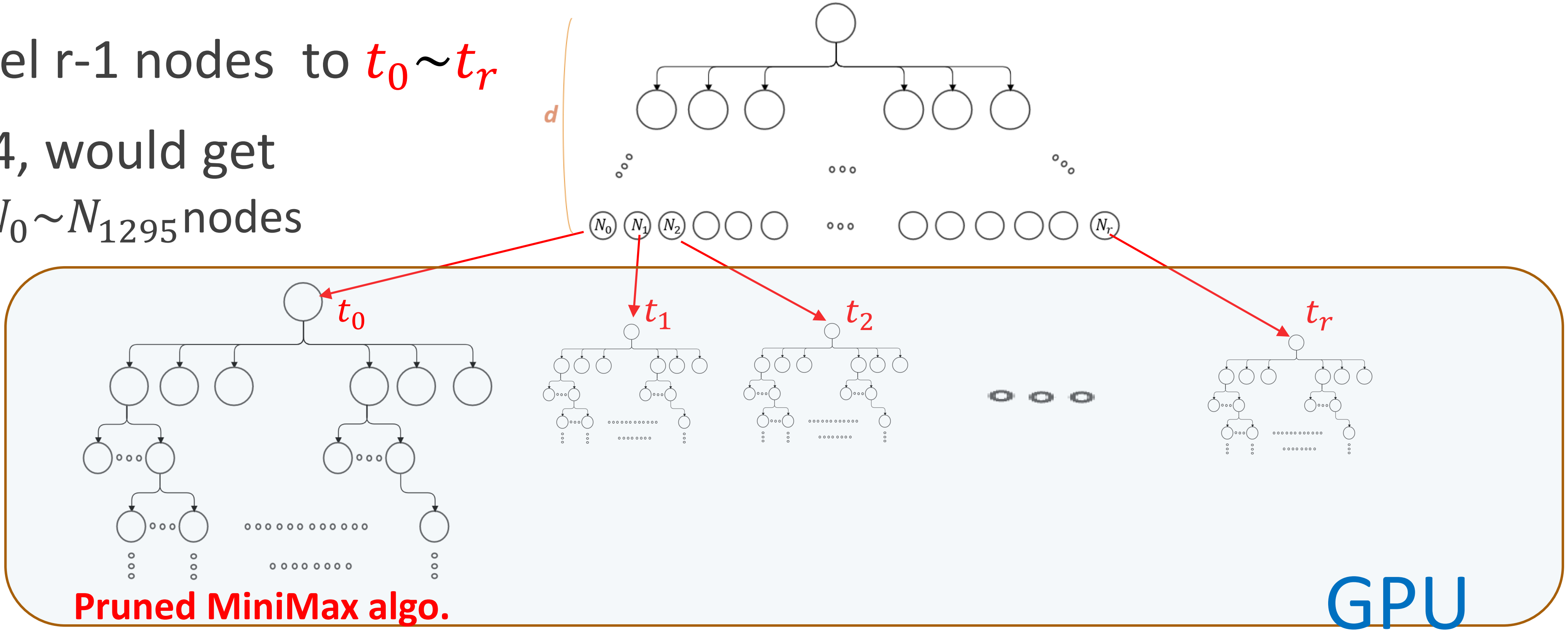
- Hard to parallelize recursive algorithm
  - Thread number: 6
- Still face with load imbalance.
  - Note: htop





Pruned Minimax tree parallel method with GPU

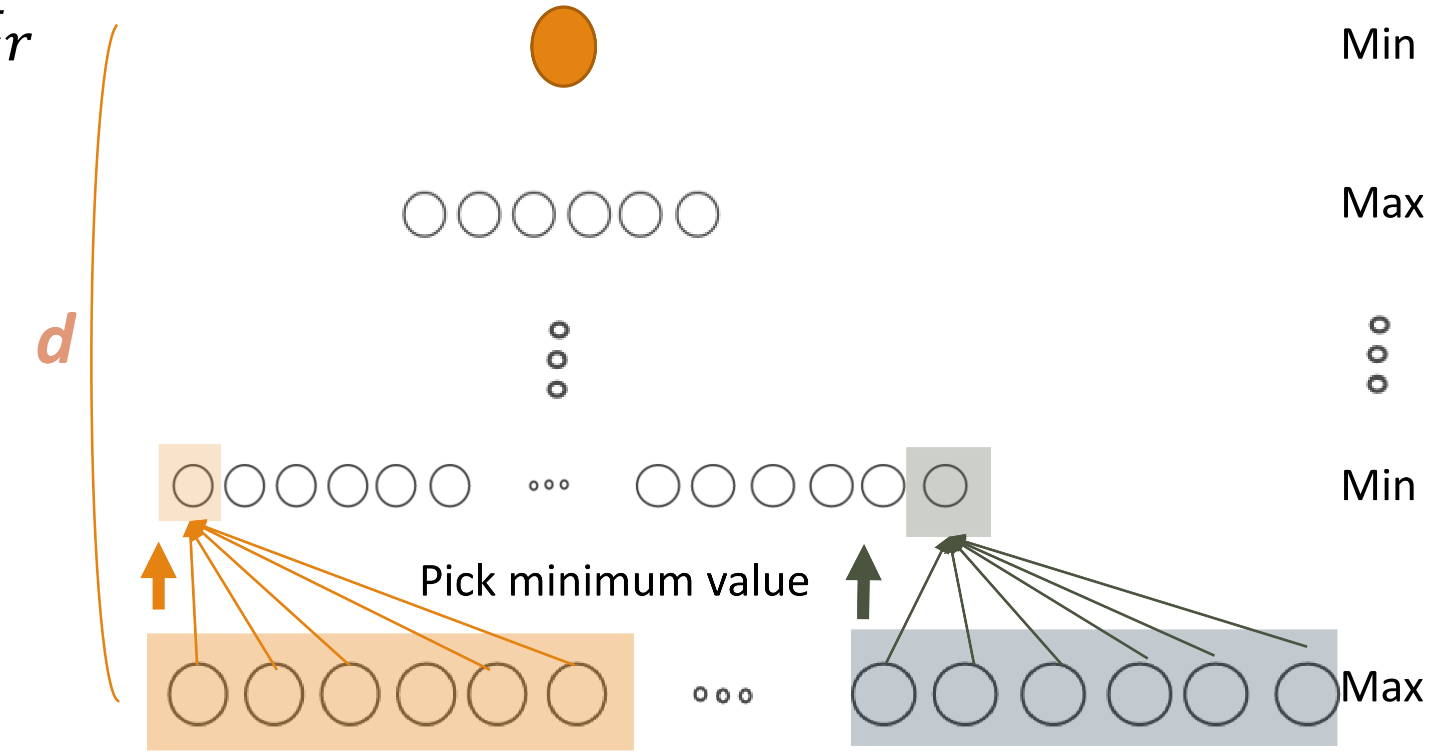
- Brutely expand ***d*** layer tree, yield nodes $N_0 \sim N_r$, (r = 6^d)
- Parallel r-1 nodes to $t_0 \sim t_r$
- If d =4, would get $N_0 \sim N_{1295}$ nodes



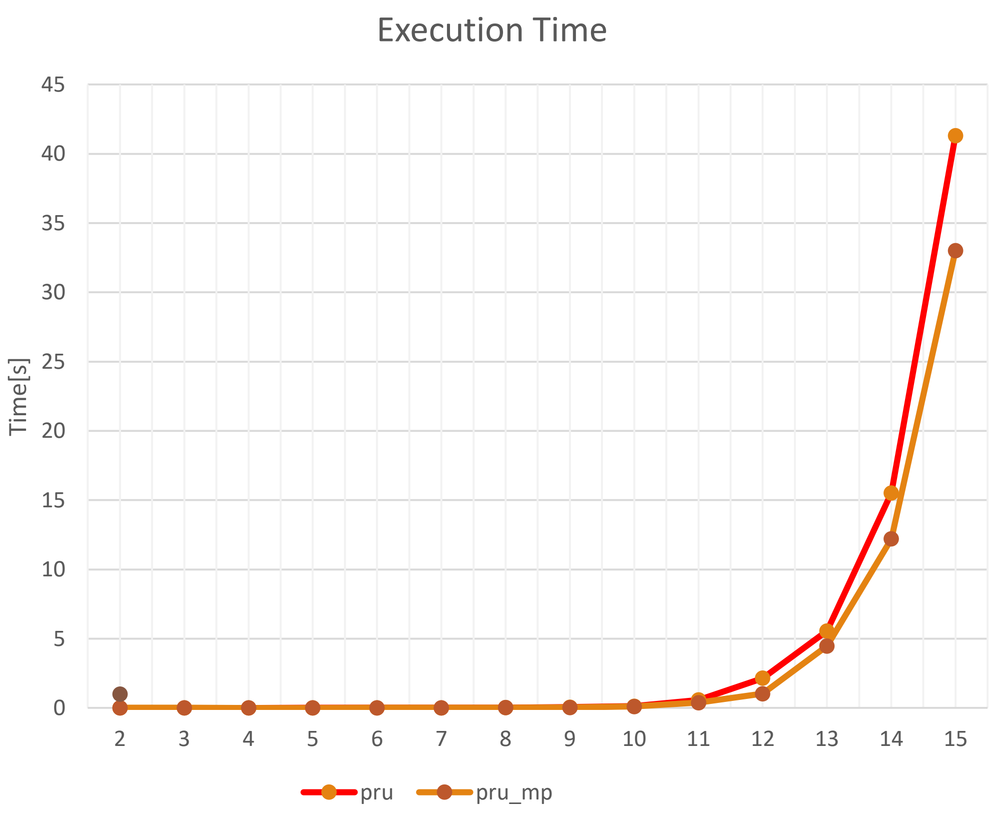
Proposed Method: Advanced Pruned Minimax tree parallel method with GPU

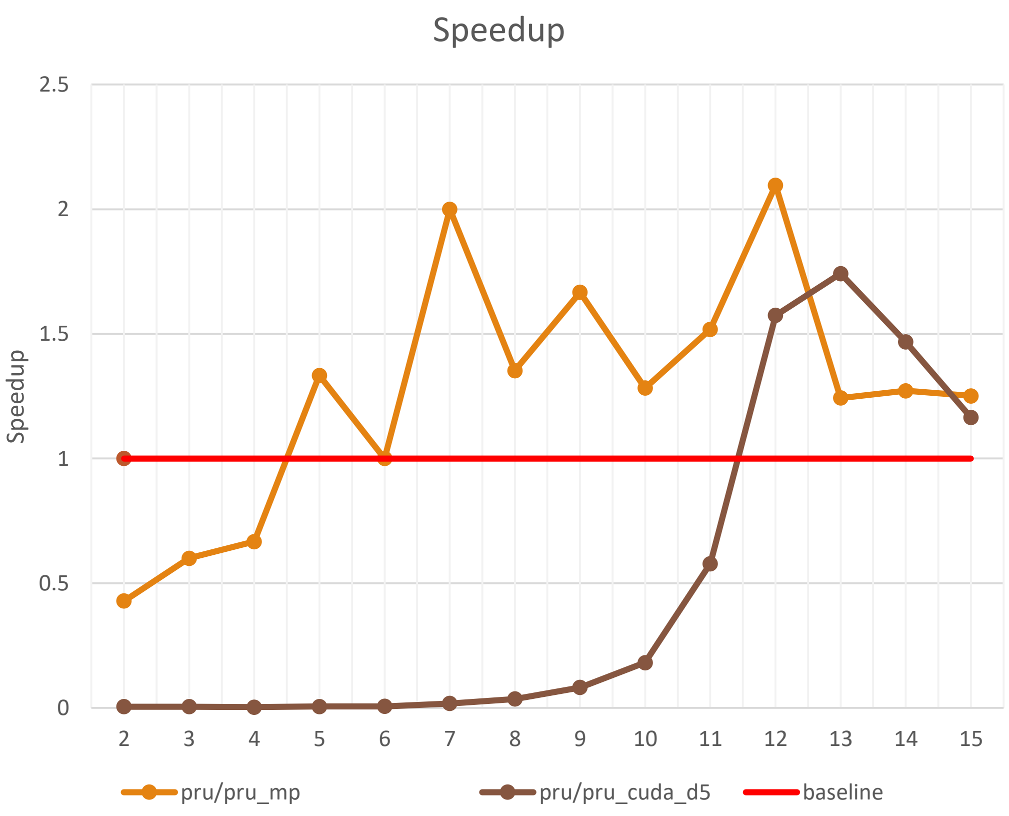
- Brutely expand ***d*** layer tree, yield nodes $N_0 \sim N_r$, (r = 6^d)
- Parallel r-1 nodes to $t_0 \sim t_r$
- Collect predictions and

  get best option.



Pruned Minimax tree parallel method with GPU





Conclusion

- When we are design a parallel algorithm. The issues firstly come to our mind are synchronization problem and dependency problem. However, picking a good algorithm to parallelize is also a big deal.

e.q. recursion.