Wa-Tor Report

Author: Dawid Pionk

Student Number: C00273530

Date: 30/11/2024

Hardware Used

CPU: AMD Ryzen 5 3600X 6-Core Processor, 3800 Mhz, 6 Core(s), 12 Logical Processor(s)

GPU: NVIDIA GeForce RTX 3060 Ti

Memory: 16 GB

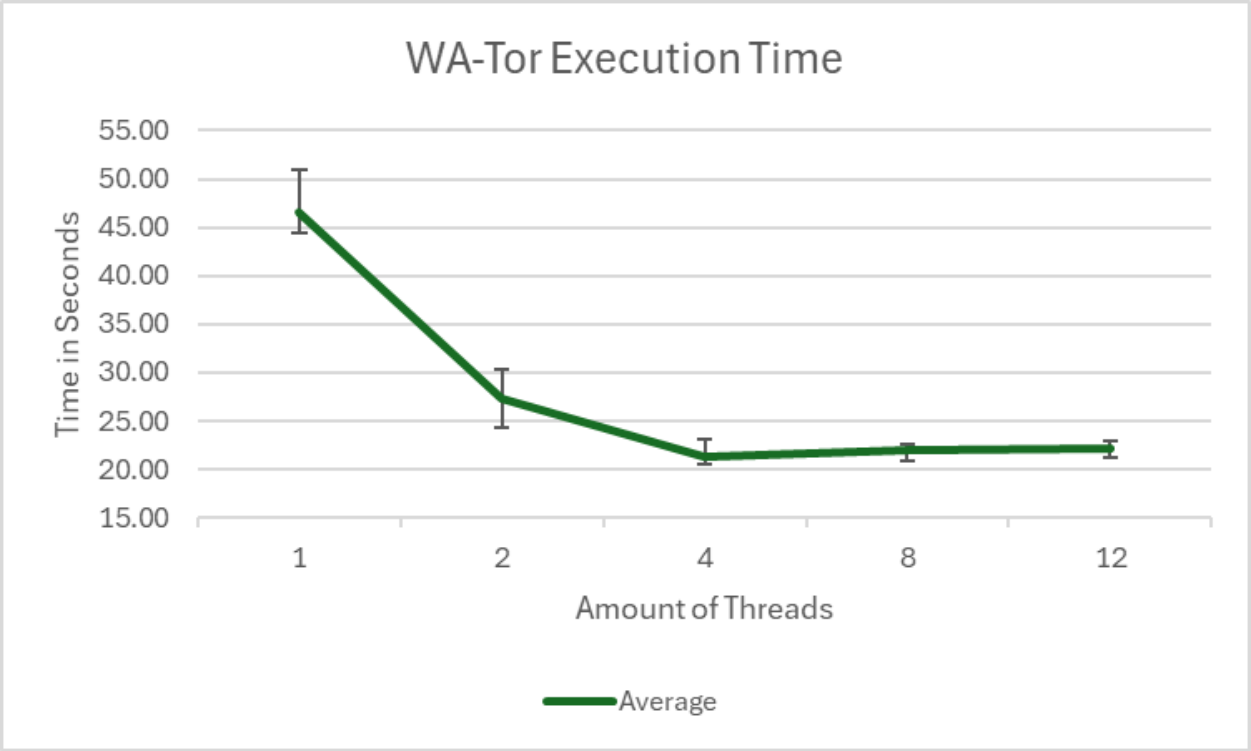
Operating System: Windows 10 Home

Table

Threads Used				
1	**2**	**4**	**8**	**12**
47.8424068	30.1917817	21.8534799	21.8096572	21.7050929
44.6670718	24.4328312	21.6627026	21.4514905	21.5283606
42.1184363	27.951975	22.1688529	22.1367344	22.3559146
46.1948084	27.8116292	21.1006886	21.8705083	22.2903672
46.1598785	27.3221434	21.1826294	22.896855	21.301662
46.1538008	25.6854399	22.0872409	22.0585692	21.9129858
48.6368084	28.9376126	20.206871	21.3197642	21.8079822
47.394985	24.2925332	22.1099113	21.9111846	22.446245
47.6097362	27.5530141	19.7205962	21.7380545	22.7294969
48.1895675	28.6998803	21.6664291	21.9585563	22.9069633

*Time is measured in seconds

Chart



Note: the bars on the average line show the range of values.

As you can see above there are diminishing returns (Amdahl's law) the more threads I used. This could be due to several reasons:

1. My CPU only has 6 cores so those threads will start competing to gain access to those cores.
2. There could be bottlenecks in the program that stop threads from being used properly.
3. There could also be overhead with context switching.

How was it measured

The program was run 10 each on 1, 2, 4, 8, 12 thread limit.

Parameters provided at started

Number of Sharks: 500	SharkBreed: 20
Number of Fish: 1000	Starve : 15
FishBreed: 10	GridSize: [2]int{250, 250}

Code

```
type Game struct {
    world     *World
    chronon   int
    startTime time.Time
}
```

```
func NewGame(numShark, numFish, fishBreed, sharkBreed, starve int, gridSize [2]int, threads int) *Game {
    w := NewWorld(numShark, numFish, fishBreed, sharkBreed, starve, gridSize, threads)
    return &Game{
        world:     w,
        chronon:   0,
        startTime: time.Now(),
    }
}
```

```
func (g *Game) Update() error {
    g.chronon++
    if g.chronon == 1000 {
        elapsed := time.Since(g.startTime)
        print(elapsed.String())
        os.Exit(0)
    }
    g.world.IterateProgram()
    return nil
}
```

- The time was taken when the program was initialized.
- The chronons are counted until they reach 1000
- Then the difference in time from 0 to 1000 chronons is measured and printed.
- The program is exited.