VPS 5	Parallel and Distributed Software Systems	ST 16, Exercise 1

Deadline: 1st of April 2016 17:00

Name_____

Points _____ **Effort in hours** _____

1. Wator – Eat or be eaten … (8 Points)

Wator is the Name of a small circular planet, far far away from our galaxy, were no man has ever gone before. On Wator there live two different kinds of species: *sharks* and *fish*. Both species live according to a very old set of rules which hasn't been changed for the last thousands of years.

For **fish** the rules are:

- at the beginning of all time there were *f* fish
- each fish has a constant energy E_f
- in each time step a fish moves randomly to one of its four adjacent cells (up, down, left or right), if and only if there is a free cell available
- if all adjacent cells are occupied, the fish doesn't move
- in each time step fish age by one time unit
- if a fish gets older than a specified limit B_f, the fish breeds (i.e., a new fish is born on a free adjacent cell, if such a cell is available)
- after the birth of a new fish the age of the parent fish is reduced by B_f

For **sharks** the rules are:

- at the beginning of all time there were *s* sharks, each with an initial energy of E_s
- in each time step a sharks consumes one energy unit
- in each time step a shark eats a fish, if a fish is on one of its adjacent cells
- if a shark eats a fish, the energy of the shark increases by the energy value of the eaten fish
- if there is no fish adjacent to the shark, the shark moves like a fish to one of its neighbor cells
- if the energy of a shark gets 0, the shark dies
- if the energy of a shark gets larger than a specified limit B_s, the shark breeds and the energy of the parent shark is equally distributed among the parent and the child shark (i.e., a new shark is born on a free adjacent cell, if such a cell is available)

In the Moodle course you find a ready to use implementation of Wator. Make a critical review of the code and analyze its design, performance, readability, etc. **Write a short report** which outlines the results of your review.

To get a fair comparison of the application's performance, **analyze** a Wator world of 500 x 500 cells. How long takes a run of 100 iterations on average with deactivated graphical output? Execute several independent test runs and **document the results in a table** (also calculate the mean value and the standard deviation). Then **answer the following questions**: Where and what for is most of the runtime consumed? What can be done to improve performance? What are the most performance-critical aspects?

2. Wator – Optimization (16 Points)

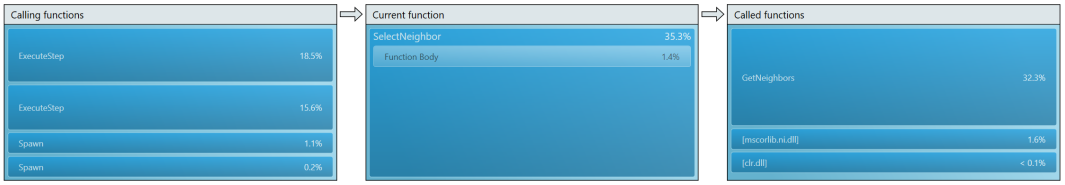
Based on your analysis, change the application step by step to improve performance. Think of at least **three concrete improvements** and implement them. Document for each improvement how the runtime changes (in comparison to the prior and to the initial version) and calculate the speedup. Each single optimization should yield a speedup of at least 1.05 compared to the prior version. Test your improvements with the settings given in the previous task.

1 1. Wator – Eat or be eaten

1.1 Analyse

Der Sourcecode ist grundsätzlich gut dokumentiert und auch lesbar. Die Architektur der Anwendung loose gekoppelt und nachvollziehbar. Zur Implementierung gibt es zu erwähnen, dass in der selben Methode oft auf die gleichen Feldelemente zugegriffen wird. Hier wäre es vielleicht besser, wenn der Wert einmal zwischengespeichert werden würde, da sonst bei jedem Zugriff Laufzeitüberprüfungen durchgeführt werden. Weiters fällt auf, dass sehr viele Objekte der Klasse `Point` angelegt werden. Zur Methode `GetNeighbors` gibt es zu erwähnen, dass hier für alle vier Richtungen eigentlich der selbe Code verwendet wird und dieser vielleicht besser in eine Methode ausgelagert werden könnte (bringt warscheinlich keine Performancesteigerung aber der Code wäre leichter lesbar).

1.2 Where and what for is most of the runtime consumed?



Grundsätzlich wird die meiste CPU-Zeit in der Methode `ExecuteStep` und in weiterer Folge dann in `GetNeighbors` verbraucht. Diese Methode ist für das Suchen der Nachbarn eines Feldes zuständig und wird daher sehr oft aufgerufen. Die Analsyse der Methode zeigt, dass sehr viele `Point` Objekte angelegt und dann sogar nochmals kopiert werden. Dadurch wird sehr viel CPU-Zeit verbraucht.

Die Methode `RandomizeMatrix` ist eine weitere Methode die sehr viel CPU-Zeit benötigt. Diese ist für das Mischen der Durchlauf-Matrix zuständig.

1.3 What can be done to improve performance?

- Umgang mit den Koordinaten in der Methode `GetNeighbors` optimieren, damit nicht mehr so viele Objekte angelegt werden müssen.

  Verwenden eines statischen Arrays.

  Kopieren des Arrays am Ende der Methode vermeiden.

- Umstellen der zweidimensionalen Felder auf eindimensionales Felder.
- Ändern `Moved` Eigenschaft um die zusätzliche Schleife über alle Elemente zu sparen.

  Eine Möglichkeit wäre hier die Iterationsnummer zu verwenden.

2 2. Wator – Optimization