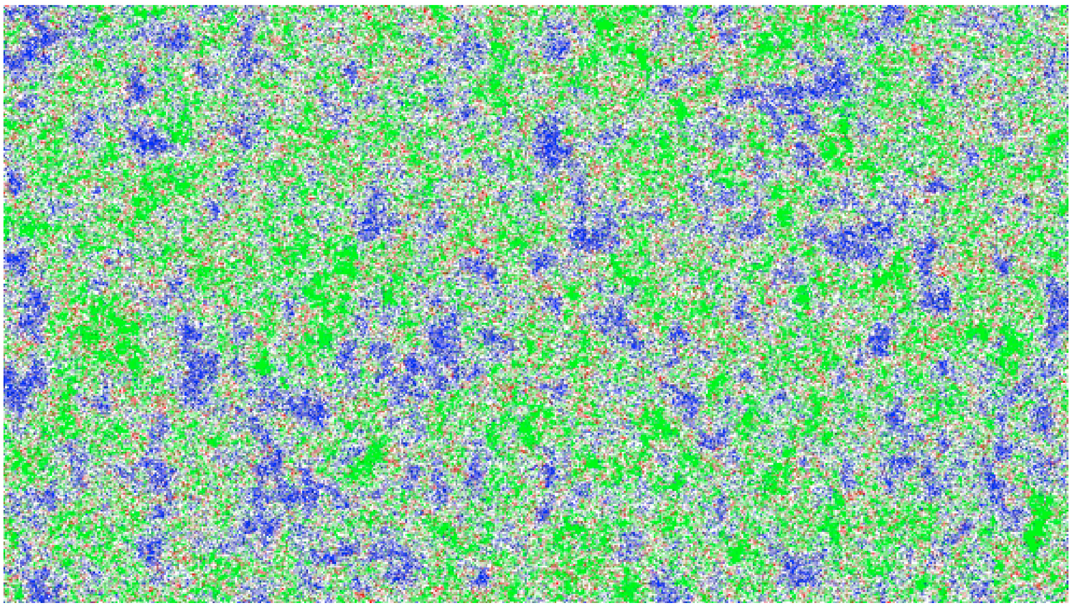
THE PREDATORS' - GUIDE

A parallelized prey-predator-simulation

Authors: Markus Fasselt, Julian Schmid, Kim Korte



Summer 2013

Contents

This project aims to show the consequences resulting from a simulation of prey and predator in a virtual world. By altering one or many of the key parameters, the behavior of the animals is controlled and different outcomes can be observed. Outcome does not necessarily mean a final state of the world in which the simulation eventually runs into but rather the state of the world after a pre-defined number of rounds. It is possible for the simulation to reach a dead-end before completing the set number of rounds. This can happen, for instance, if one population dies from starvation or one eats the other.
The final program is able to visualize the results and show appropriate data about the actions leading up to the results. Furthermore, it can be run simultaneously on multiple cores in order to improve performance.
The project is written mainly in the programming language C. The parallelization and communication between the processes is based on OpenMPI.

1 Introduction

The basic idea of the prey - predator - model is two populations interacting with each other, hence varying in size, location and spread. The particular model is often associated with diagrams showing the size of the two populations. These are often antiymmetric, as described in the Lotka - Volterra equation(Figure 1). The Lotka - Volterra equation describes the interaction between predators and prey, using the amount of specimen per species. This leads to an antisymmetric graph, as an excelling amount of predators reduces the amount of prey. This in turn reduces the amount of predators, since they can no longer find enough food.

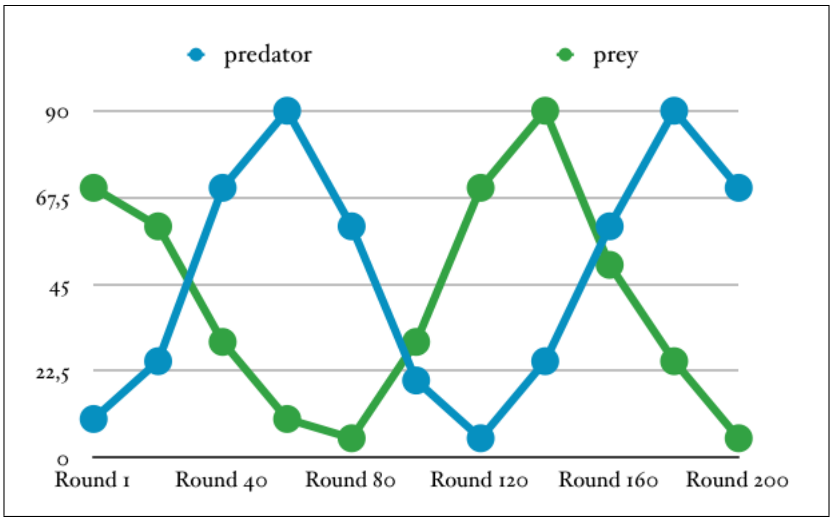


Figure 1: The lottka - volltera equation

2 The Model

In this section we will explain the main concepts of our simulation.

2.1 World

The world is a two dimensional map. Its size can be adjusted in the configuration. Depending on its size, it contains a specific amount of fields, Each of which can contain a creature and/or a plant. Each field has 8 direct neighboring fields (Figure 2). The world is considered to be round, therefore the fields on the edges of the map are neighboring the edge - fields on the other side of the map (Figure 2).

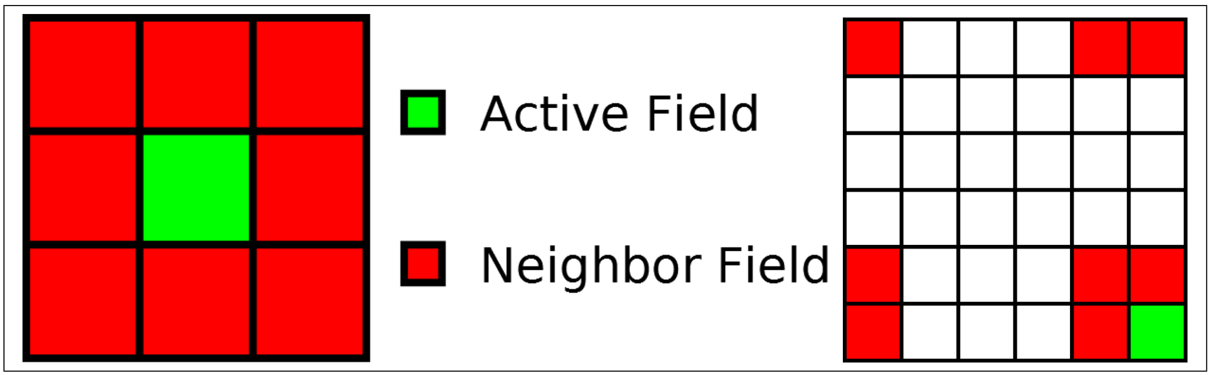


Figure 2: Fields and their neighbors

2.2 Creatures

This term includes both predators and prey. Predators are equal to carnivores, as prey is equal to herbivores. There can only be one (or no) creature on each field. Every creature has an energy level and an age, which determine whether the creature lives another round.

2.2.1 Energy

The energy level represents the hunger of a creature. It has a maximum value of ten, to which is is restored when a herbivore eats a plant, or respectively a carnivore eats a herbivore. Each round the creature does not eat, the energy level decreases by two until a minimum value of zero is reached. By then the creature will die of starvation. A carnivore will not eat plants, but only creatures from the herbivore population.
Additionally, the energy level suffers from fights between a predator and its prey, decreasing the energy level even further.

2.2.2 Death

Another dying reason is the age of the creature. With each round it increases by one until a certain value is reached. This value can be set as a parameter and stands for the lifespan of the creature. All creatures die eventually, as is only natural, in order to make room for their children.
There is an addional death rate for both carnivore and herbivore. It is independent from age or energy level of the creature and represents all other possible ways of death, for example a natural accident, sickness or death while birthing a child. The natural dying rate is 0,5% for the prey and 2% for the predators.
These values are also parameters, which can be adjusted.

2.2.3 Birth

Except for the initial round of a simulation no creatures will be spawned randomly. There is a birth rate in order to ensure sufficient successors. Each creature has a chance of spawning an offspring every round. A creature of the prey population will bear a child roughly every 2 rounds (50% birth rate), whereas a predator gives birth less often (at a birth rate of 20 %). There does not have to be another creature from the same population nearby for the creature to bear a child.

2.2.4 Fighting

When a carnivore attacks a herbivore it comes to a fight until one of them dies. Fighting costs energy, so the more energy a creature has when engaging in a fight, the higher the chance that it will survive that fight. Only one creature can survive a fight. There are several fighting rounds, which are all completed within one simulation round.

2.3 Plants

Plants will randomly spawn on fields of the map. The occurence of a plant is independent from any creatures residing on that field. They serve as the food for all herbivore creatures. When a herbivore stands on a field with a plant on it, the creature will eat the food and therefore resets its energy level. This process removes the plant from the field.
Plants cannot become extinct, since they spawn randomly and independently from any other parameters. This assumes that the weather is providing a constant stream of growing conditions and sets aside the real-word neccesarity of seeds.

2.4 behavior

Herbivores and Carnivores will both search for food in their surrounding area. Should they find any food, they will engage it. For herbivores this means that they will move towards the plant and eat it. Carnivores, however, will find herbivores not as easy to consume. When they engage the herbivores, a fight will take place. The result of the fight is dependent on the energy level of both participants. Although carnivores will have a natural benefit, a strong and well fed herbivore will be able to defeat a weak carnivore.

3 Simulation

This section explains the order in which the steps of the simulation are executed, as well as the generated statistics.

3.1 Step execution

The first step of the simulation is the creatures looking for food (prey or plants) in the adjacent fields. If they find something, they move to the field and eat the food residing on it. It is not until all creatures have searched for food that the next step is executed. Neighboring fields are the ones left, right, above, below, upper left, upper right, lower left and lower right, so eight in total. Every creature who did not move in the previous step, moves randomly to an empty, adjacent field. In the next step creatures bear children, depending on a combination of birth rate and chance. In the last step the program checks for dying reasons, which can be either old age, starvation or the natural death rate. It will then remove all creatures who happen to have at least one dying reason. (figure 3)

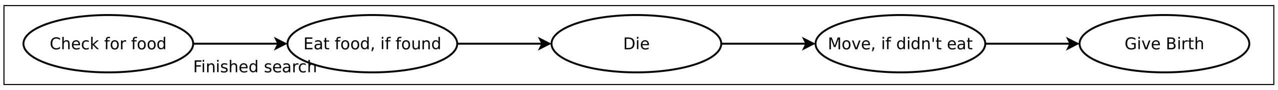


Figure 3: General steps of the execution

3.2 Statistics

Since the program is parallelized, each process generates its own statistics. After every simulation step the process counts the number of herbivores and carnivores. The data is then sent to the master process, which cumulates it and stores it in a file for later evaluation. Using gnuplot, the program is able to turn this data into a useful graph (figure 4).

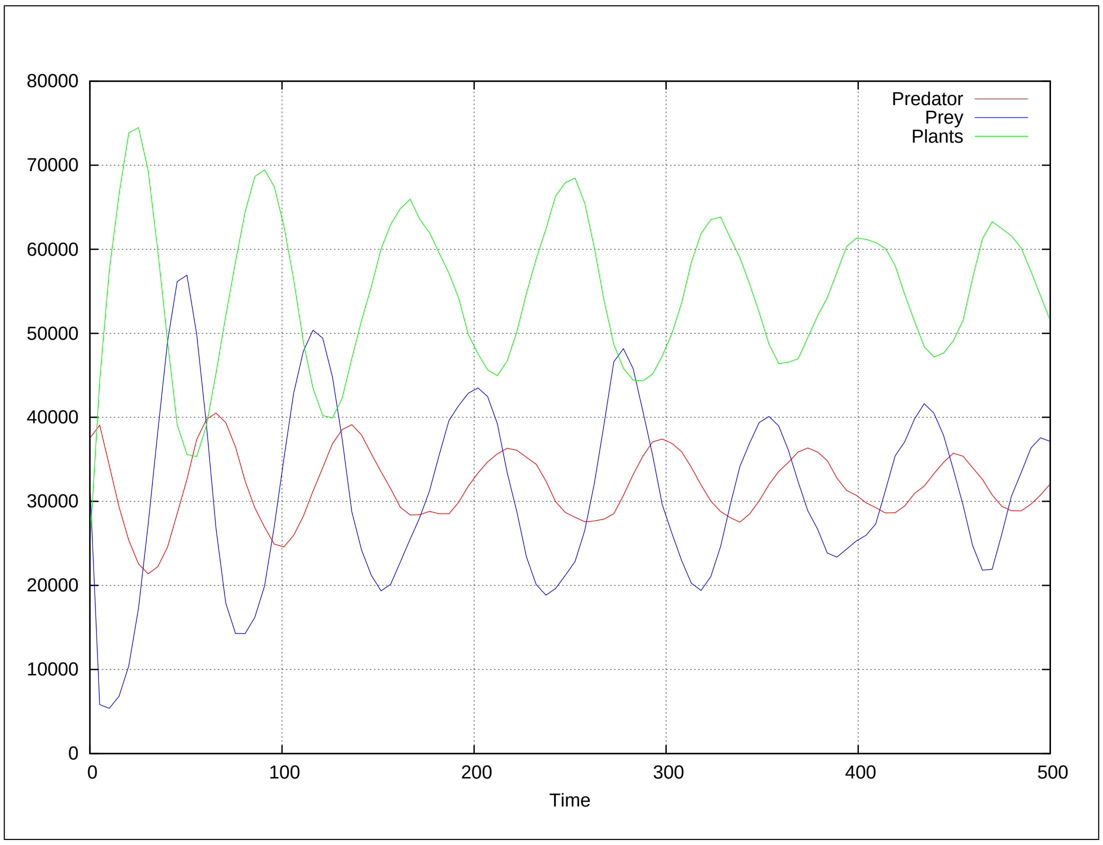


Figure 4: Simulated population over time

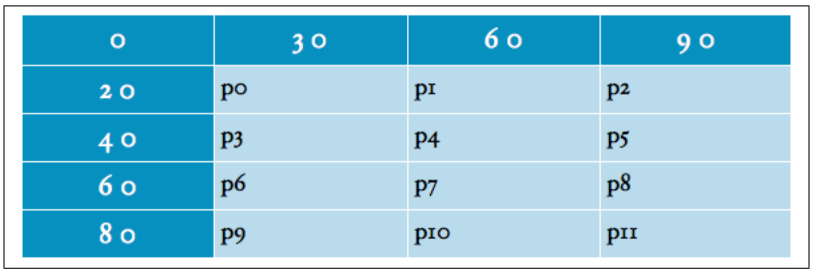
4 Implementation

4.1 Simulation

The implementation of the simulation itself.

4.1.1 World

The world is divided into several segments depending on the number of processes available for computing. For example: a simulation using a $90 \cdot 80$ map with 12 processes will be divided up into 12 $30 \cdot 20$ segments. The numbers in the upper row are the left border/starting points of the segments (x1), whereas the leftmost column contains the upper border of the segment (y1). As seen here, each segment is 30 fields wide and 20 fields high. If we take, for example, the segment p4, its delimiting corner fields would be $x1 = 30, x2 = 59, y1 = 20, y = 39$.



0	30	60	90
20	p0	p1	p2
40	p3	p4	p5
60	p6	p7	p8
80	p9	p10	p11

Figure 5: segmentation of a map

4.1.2 Fields

The fields are the basic components of the map on which the simulation runs. The fields are storing a lot of the information needed to run the simulation. Each Field contains the following information:
The coordinates of the field, the population type currently residing on the field, the age and energy level of that creature and whether or not the field has a plant growing on it. Should the animal move to another field, these values will be copied to the other field.

4.1.3 Animal behavior

The behavior of the animals is different for prey and predators.
Any animal which is considered prey will check its eight neighboring fields for plants. If any of these fields should contain a plant it will move onto that field, as long as there is no other creature blocking it. If none of the surrounding fields have a plant the animal will wander around and move to a random direction. Predators will search its eight neighboring fields for prey. If the neighboring fields have prey on them, both predator and prey will start fighting. Only one of them will survive. Should the predator survive, it will eat the prey. Otherwise the predator will die and the prey will survive.

4.1.4 Fighting

When a predator engages its prey, a fight will break out between them. The fight will last until one of the participants dies. A creature dies in combat by losing all of its energy. The fight lasts multiple fighting rounds, not to be confused with simulation rounds. The entire fight will be resolved within one simulation round.
For each fighting round, every participant generates a random number between 0 and its fighting energy. The fighting energy equals the normal energy. Predators get a bonus of five added to their fighting energy.
When the energy level of one of the participants reaches zero, the participant will die and the fight is over.

4.1.5 The configuration

In order to allow easy and quick modification of the parameters for the simulation, the program offers an easy to use configuration. It offers the ability to change almost every parameter that contributes to the simulation. The offered parameters are: The size of the world described with a width and height parameter.
The number of simulation steps before termination. If offers the option to go for an infinite number of steps, although this is not recommended.
Spawn-rates, determining the percentage of the map which gets filled with creatures in the beginning of the simulation. Aditionally, there is an option for how many of the creatures are spawned as prey and how many as predators. An additional spawn rate exists for plants, unrelated to the creature spawn rate.
Additional creature parameters, such as the age at which creatures die of old age, the maximum energy level, the birth - rate and the death - rate, which represents natural causes of death, except old age and starvation.

4.2 Parallelization

The implementation of running the simulation on multiple processes.

4.2.1 Segmentation and animal movements

As mentioned earlier, the world is divided into several segments on startup. The number and size of them is determined by the number of processes available.
The program determines the distribution of segments using prime factorization. If the prime factorization results in more than two prime numbers, the program will multiply

the results with each other until only 2 results remain. To make this process as balanced as possible the program see-saws between multiplying the results at the front of the list and the results at the end of the list.
For example: There are have 64 processes. The prime factorization yields $2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2$. The programm will then multiply the first two results, resulting in $4 \cdot 2 \cdot 2 \cdot 2 \cdot 2$. Then the last two results. $4 \cdot 2 \cdot 2 \cdot 4$. In the end we will get the results $8 * 8$.
Based on this process the program will then divide the map into 64 segments, dividing it eight times on the x - axis and eight times on the y - axis.
Each segment is assigned to exactly one process which will then calculate the movements and events on the fields of the segment during the simulation.
Since creatures are able to cross segments, a process does not only have to store the fields it is directly responsible for, but also the fields that surround his segment, called outer - fields (figure 6).
The fields that are on the edges of a processors segment are called border - fields and are important for communication between processors.

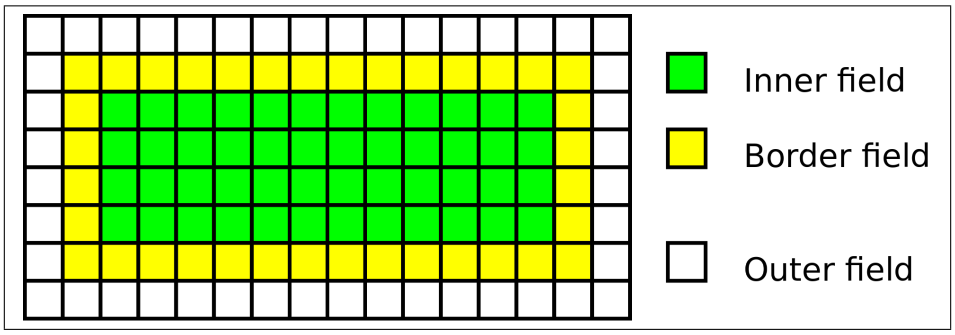


Figure 6: Fields used by a process

4.2.2 Communication

This section explains the communication between the different processes. A process will send out information when it updates a field that is either an outer field or a border field.
When a border or outer field is updated, all processes that can see this field will get notified. This can be up to three processes depending on the location of the field. For example: If the field is on the corner, rather than the edge, of a segment more processes need to be notified.

In our simulation a field is implemented as a struct. Structs cannot be transfered with standard MPI data types, so we created our own MPI data type. This brings the advantage of being able to directly transfer the field, without any transformations.

The communication itself is implemented using the blocking MPI_send() and the non - blocking MPI_Irecv().

4.3 Statistics

After each round, every process calculates its statistics for the remaining, living creatures. These will be sent to the master process, using another self - created MPI data type. After these statistics have been received, they get merged using MPI_reduce() and a self - created MPI operation.
At last, should any creature become extinct, every process is notified that the simulation has to be terminated.
Otherwise the simulation continues with the next round.

5 Performance analysis

5.1 Speedup with one node

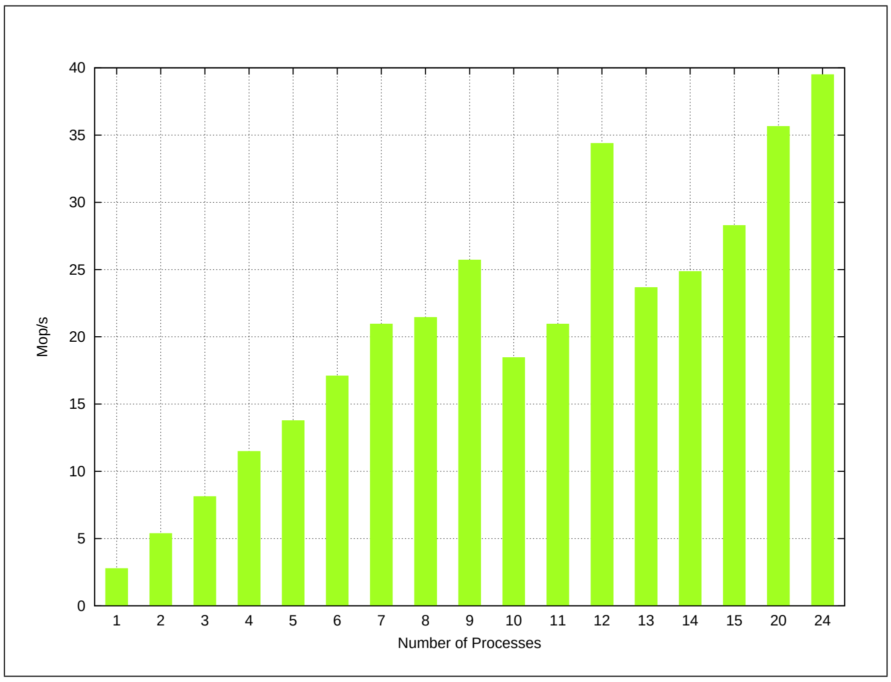


Figure 7: Speedup in Million Operations per second

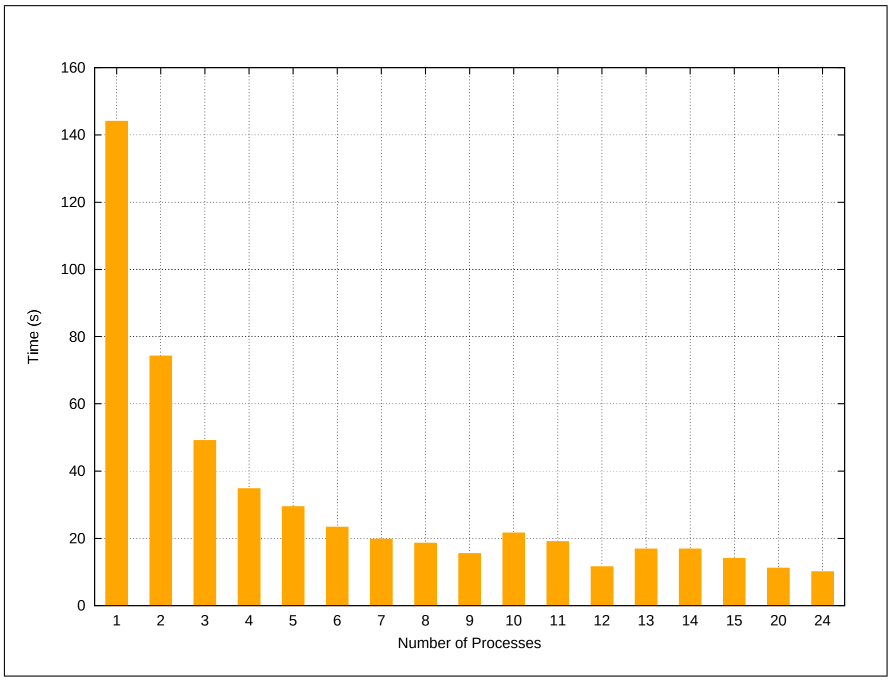


Figure 8: Speedup in time

The speedup is shown in figure 7 and figure 8.
While the speed does increase over an increasing number of processes, there is some data standing out due to a sudden increase of computing time.
We assume this is caused by a uneven distribution of segments. The problem seems to occur every time the number of processes does not allow for a balanced segmentation of the map. For example: Ten processes result in a segmentation of $2 \cdot 5$. Twelve processes result in a segmentation of $4 \cdot 3$, which is much more balanced.
However, the fact that seven processes seem to be almost as fast as eight processes, while not allowing for a balanced segmentation at all, seems to contradict this theory.

5.2 Speedup with multiple nodes

With multiple nodes the speedup is significantly greater (figure 9), than with only one node. Using four nodes we can observe that the speedup graph follows Amdahl's law. A higher number of nodes seems to pull the point, were increasing the number of processes, decreases the speed of the program, farther back.

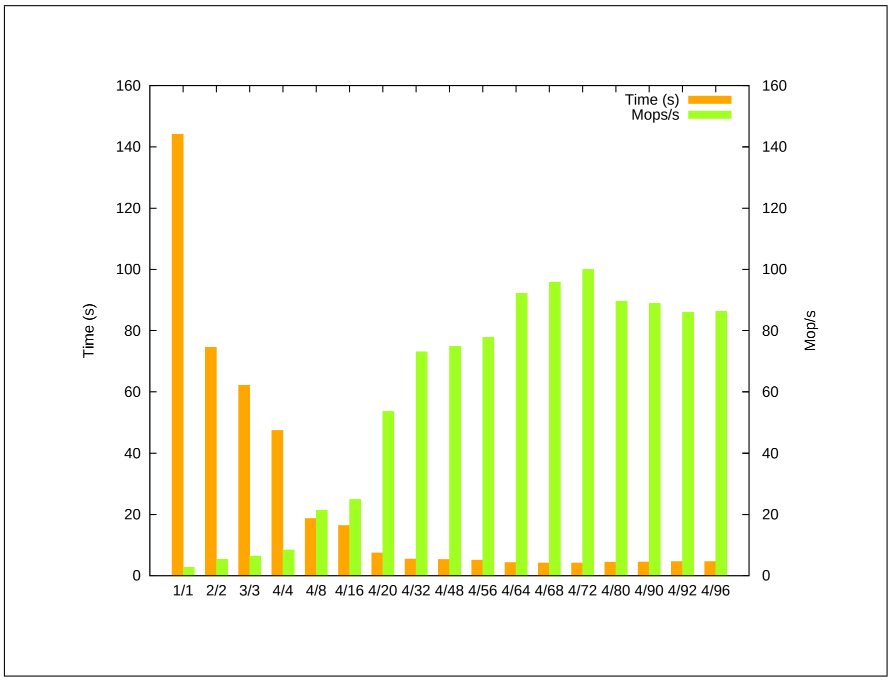


Figure 9: Speedup with 4 Nodes

5.3 Weak Scaling

The speedup using weak scaling. Weak scaling describes the method of having a fixed size for the segments and making the size of the map dependent on the number of processes(figure 10). We ran twelve processes using one node and up to 96 processes using multiple nodes to avoid hyper - threading. We can see that the graph for weak scaling is very simillar to strong scaling, although slower than strong scaling. The speed increases in a simillar manner and the sudden speeddrops for specific numbers of processes are still happening.

6 Addendum

The development of the program was aided by the IDE Eclipse and the compilers GC-C/MPICC.

For code collaboration, we used the version control system git.

For debugging purposes we used gdb, valgrind and DDT.

Additional software used for creation of the paper and presentation includes Apple Pages,

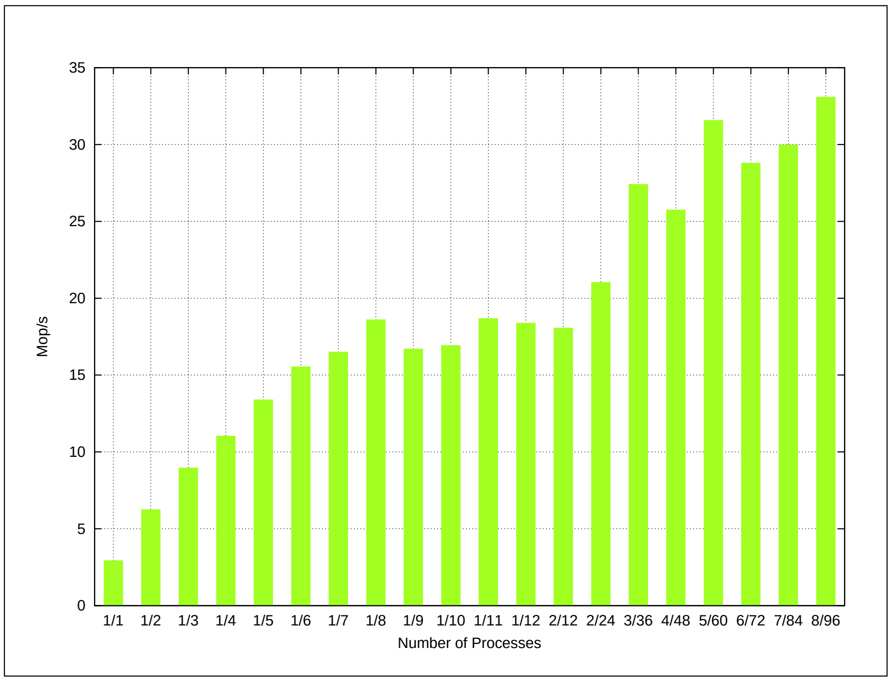


Figure 10: Speedup with weak scaling

LaTeX, dia and gnuplot.