Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

Analyse der Populationsdynamik in einer fiktiven Welt mit Vampiren

Arbeitsbereich Wissenschaftliches Rechnen

Fachbereich Informatik

Fakultät für Mathematik, Informatik und Naturwissenschaften

Betreuer:	Dr. Julian Kunkel
Semester:	Sommersemester 2015
Kurs:	Praktikum Parallele Programmierung
Vorgelegt von:	Jun-Patrick Raabe Matr.-Nr.: 6533187 Studiengang: B.Sc.WiInf 3raabe@informatik.uni-hamburg.de
	Kolya Feierabend Matr.-Nr.: 6527306 Studiengang: B.Sc.WiInf 3feierab@informatik.uni-hamburg.de

Hamburg, den 10.09.2015

Inhalt

1 Problemstellung

Dieses Projekt zielt darauf ab, die Entwicklung der Bevölkerungsgrößen in einer fiktiven Welt mithilfe einer Simulation darzustellen. Die Bevölkerungsgruppen Mensch, Vampirjäger und Blade kämpfen gegen die Vampire ums Überleben. Durch die Möglichkeit, verschiedene Parameter, wie zum Beispiel die einzelnen Bevölkerungsgrößen, zu Simulationsbeginn anzupassen, können viele vollkommen unabhängige Resultate erzielt werden. Das Programm bietet die Möglichkeit, die Bevölkerungsgröße in einer externen Datei und die Positionen aller Kreaturen visuell zu jedem Simulationsschritt darzustellen. Das Projekt wurde mit der Programmiersprache C realisiert. Die Parallelisierung erfolgte mittels MPI.

2 Lösungsansatz

2.1 Spielfeld

Bei dem Spielfeld handelt es sich um eine drei-dimensionale Welt, in der die ersten beiden Dimensionen für die Spielfeldkoordinaten stehen und die Dritte für den jeweiligen Platz der Kreaturen, die auf den Spielfeldkoordinaten gesetzt werden können. Die Anzahl Plätze auf einer Spielfeldkoordinate ist auf zwei Kreaturen festgelegt. Das Feld ist nicht rund, so dass die einzelnen Kanten als Spielfeldende angesehen werden und die Ecken keine Nachbarn sind. Die Größe kann dabei vom Anwender beliebig angepasst werden.

Das Spielfeld *gamefield* ist ein ein-dimensionales Array vom Typ *Creature*, (siehe Abb. 2) das in der Funktion *initGamefield* initialisiert wird. Abbildung 1 zeigt den Aufbau dieser Funktion. Die Funktion *generateAndSetDefault* ist dafür zuständig, jedes Feld mit „Default“ zu initialisieren. „Default“ hat die Bedeutung, dass keine Kreatur auf einem Feld im Spielfeld steht. Um die Spielfeldkoordinaten und die dazugehörigen Plätze in *gamefield* zu berechnen wird die Funktion *getPositionInArray* genutzt. *getPositionInArray* gibt einen Integerwert aus, der für genau einen Platz auf dem Spielfeld steht. Die einzelnen Positionen und Plätze werden in gamefield spaltenweise gespeichert. Dies erleichtert die Aufteilung des Spielfelds und somit die Parallelisierung. Weiteres dazu wird in Kapitel 3 Parallelisierungsschema erläutert.

```
Creature *gamefield;

void initGamefield(){
    X_SIZE = world_size_x + 1;
    Y_SIZE = world_size_y + 1;
    Z_SIZE = 2;

    gamefield = (Creature *)malloc(sizeof(Creature) * X_SIZE*Y_SIZE*Z_SIZE);

    generateAndSetDefault();
}
```

Abbildung 1: Aufbau der Funktion initGamefield

2.2 Kreaturen

Die Kreaturen stehen für die einzelnen Bevölkerungsgruppen, die auf dem Spielfeld zu finden sind. Jede Kreatur besitzt ein Alter, ein Geschlecht und eine Rasse. Die Startpopulation jeder einzelnen Bevölkerungsgruppe kann vom Anwender angepasst werden. Das Setzen aller Kreaturen auf das Spielfeld erfolgt einmalig zum Beginn der Simulation zufällig. Für die auf dem Spielfeld auftretenden Kreaturen wurde die Struktur *Creature* erstellt. Sie beinhaltet als Komponenten alle relevanten Eigenschaften zur Identifikation der einzelnen Kreaturarten. Abbildung 2 zeigt den Aufbau von *struct Creature*.

```
typedef struct{
    int age;
    bool woman;
    bool isHunter;
    bool isHuman;
    bool isBlade;
    bool isVampire;
    bool alreadyMoved;
    int hungerRate;
}Creature;
```

Abbildung 2: Aufbau von Creature

Alle Kreaturarten besitzen eine „Weitsicht“-Funktion, die es ihnen ermöglicht, Feinden aus dem Weg zu gehen oder Opfer aufzuspüren und zu verfolgen. Dazu werden alle Nachbarfelder einer einzelnen Kreatur betrachtet. Befinden sich auf den Nachbarpositionen eines Nachbarfelds Feinde, so zählt ein Zähler die potentiellen Feinde, die im nächsten Simulationsschritt eine Nachbarposition der Kreatur erreichen können. Eine Kreatur wählt demnach ein Nachbarfeld, dass von möglichst wenig oder gar keinen Feinden im nächsten Schritt erreicht werden kann. Abbildung 3 erläutert die Weitsicht-Funktion für den Kreaturtyp Mensch. In diesem Fall würde der Mensch, mit „M“ dargestellt, zufällig eine Position im unteren, rechten Teil wählen. Vampire werden mit „V“ symbolisiert. Für Jäger-Kreaturen funktioniert die Weitsicht-Funktion ähnlich, jedoch bewegen sich diese in Richtung eines Nachbarfeldes, auf das im nächsten Schritt möglichst viel Beute gelangen kann.

V V		V		
	4	1	1	
V	1	M 0	0	
	1	0	0	

Abbildung 3: „Weitsicht“-Funktion für Menschen

2.2.1 Menschen

Die Menschen sind die schwächste Bevölkerungsgruppe. Als Beute von Vampiren probieren sie den Vampiren aus dem Weg zu gehen, sobald sich welche in der Nähe befinden. Trifft ein Mensch trotz allem auf einen Vampir, so stirbt der Mensch und wird mit einer gewissen Wahrscheinlichkeit ebenfalls zu einem Vampir verwandelt. In einer vom Modellanwender

bestimmten Altersspanne verwandelt der Mensch sich zu einem Vampirjäger, so dass er gezielt auf Vampire zugeht, um diese zu töten. Die Wahrscheinlichkeit zu siegen ist ebenfalls vom Anwender einstellbar. Ebenso ist einstellbar, ab und bis zu welchem Alter Kinder gezeugt werden können. Damit ein Neugeborenes entsteht und die Bevölkerung der Menschen nicht ausstirbt, müssen 2 Menschen mit unterschiedlichem Geschlecht und notwendigem Alter auf dem gleichen Feld im Spielfeld stehen und im direkten Umkreis muss mindestens ein Feld frei sein. Das Neugeborene wird dann auf eines der freien Nachbarfelder gesetzt. Da Menschen nicht unsterblich sind, ist das maximale Alter anzugeben. Übersteigt der Mensch das Alter, so stirbt er an Altersschwäche.

2.2.2 Vampirjäger

Als Spezialfall des Menschen sind die Vampirjäger den Vampiren nicht schutzlos ausgeliefert. Sie können sich zur Wehr setzen und kämpfen. Um die schutzlosen Menschen zu schützen, suchen und verfolgen sie in der Nähe befindliche Vampire. Stehen ein Jäger und ein Vampir auf demselben Feld, so ist die vom Modellanwender anpassbare Gewinnwahrscheinlichkeit des Jägers ausschlaggebend, wer als Sieger hervorgeht. Gewinnt der Jäger, so stirbt der Vampir. Verliert er, so kann dieser sterben oder selbst zum Vampir werden. Da es sich beim Vampirjäger nur um einen Spezialfall der Kreatur Mensch handelt, kann er ebenfalls Kinder kriegen, wenn sich ein Mensch oder Vampirjäger des anderen Geschlechts auf demselben Feld im Spielfeld befindet. Überschreitet der Jäger das maximale Alter zum Kämpfen, so verliert er die Fähigkeit gegen Vampire zu kämpfen und „verwandelt“ sich in einen Menschen zurück.

2.2.3 Vampire

Vampire sind die Jäger auf dem Spielfeld. Sie suchen und verfolgen gezielt Menschen, um ihren Durst zu stillen. Zusätzlich zu den Eigenschaften, die jede Kreatur besitzt, hat ein Vampir ein Hungerlevel, dass bei jeder Bewegung auf dem Spielfeld erhöht wird. Überschreitet das Hungerlevel einen festgelegten Wert, so verhungert der Vampir und stirbt. Trifft ein Vampir jedoch zuvor auf einen Menschen oder besiegt er einen Vampirjäger, so wird sein Hungerlevel zurück auf null gesetzt. Vampire vermehren sich nicht, sondern entstehen, wenn ein Vampir auf einen Menschen oder Jäger trifft und diese besiegt. Die Wahrscheinlichkeit, dass ein Opfer zum Vampir wird, ist vom Modellanwender anpassbar.

2.2.4 Blade

Blade ist die Geheimwaffe der Menschen und probiert diese zu beschützen. Befindet sich ein Vampir in unmittelbarer Nähe, so verfolgt er diesen. Blade ist unbesiegbar und tötet jeden

Vampir, wenn es zu einem Kampf kommt. Falls gewollt, kann Blade vom Modellanwender deaktiviert werden.

2.3 Simulation

2.3.1 Bewegungsphase

Nach der Initialisierung des Spielfelds und dem Setzen der Kreaturen werden zu Beginn eines jeden Simulationsschrittes alle Kreaturen bewegt. Die Bewegung ist dabei vom jeweiligen Typ verschieden. Befinden sich auf den Nachbarpositionen Opfer, so bewegen sich die Kreaturen Vampir, Vampirjäger und Blade darauf zu. Im Gegensatz dazu bewegt sich der Mensch in die für ihn am wenigsten gefährliche Position (siehe hierzu Abbildung 3). Sind alle Nachbarpositionen frei von Beute oder Opfern, so findet die Bewegung der Kreaturen zufällig statt.

2.3.2 Regelwerk

Nach Abschluss der Bewegungsphase werden alle Regeln angewendet. In dieser Phase werden sämtliche Kämpfe durchgeführt, falls zwei feindliche Kreaturen auf den zwei Feldern einer Spielfeldkoordinate stehen oder Neugeborene erstellt, falls Menschen oder Vampirjäger unterschiedlichen Geschlechts auf den zwei Feldern einer Koordinate stehen. Des Weiteren wird überprüft, ob das Hungerlevel eines Vampirs und das maximale Alter eines Menschen oder Vampirjägers überschritten ist. Ist dies der Fall, so stirbt die Kreatur.

2.4 Modellparameter

Die Modellparameter können vom Anwender in verschiedenster Weise verändert werden. Abbildung 4 zeigt die Anpassungen, die in *config.c* gemacht werden können. Neben der Möglichkeit Spielfeld-Bilder zu jedem Simulationsschritt der anpassbaren Simulationsdauer im .bmp Format auszugeben, kann die Größe des Spielfelds sowohl in X, als auch in Y Richtung beliebig angepasst werden. Hierbei ist jedoch zu beachten, dass es mit mehreren Prozessen zu Problemen kommen kann, wenn die X-Dimension zu klein gewählt wurde. Zusätzlich kann das Alter der Menschen verändert werden, in welchem diese zu Vampirjäger werden oder aber Kinder erzeugt werden können. Das eingestellte Todesalter variiert für jeden Menschen um den Wert 10 herum. So kann es sein, dass ein Mensch nach 85 oder gar erst nach 105 Simulationsschritten aufgrund von Altersschwäche stirbt, wenn es auf 95 voreingestellt ist. Die oben beschriebene Hungerrate für Vampire, die Wahrscheinlichkeit, dass ein Mensch oder Vampirjäger zu einem Vampir wird, und die Gewinnwahrscheinlichkeit für Jäger in einem Kampf gegen Vampire ist ebenfalls vom Modellanwender einstellbar. Die wohl wichtigste Einstellungsmöglichkeit ist die Startpopulation jeder einzelnen Kreaturart.

```
const bool picturePrinting = true;
const int world_size_x = 500;
const int world_size_y = 500;
const int sim_duration = 3000;
const int min_fighting_age = 20;
const int max_fighting_age = 67;
const int min_childbirth_age = 15;
const int max_childbirth_age = 67;
const int human_death_age = 95;
const int vampire_max_hunger_rate = 50;
const int start_popu_vampire = 5000;
const int start_popu_human = 100000;
const int start_popu_hunter = 10000;
const bool blade_available = true;
const int human_to_vampire_percentage = 30;
const int hunter_win_percentage = 50;
```

Abbildung 4: Anpassung der Modellparameter in config.c

3 Parallelisierungsschema

Abhängig von der Anzahl zur Verfügung stehender Prozesse wird das Spielfeld spaltenweise auf je einen Prozesse aufgeteilt. Angenommen es stehen 4 Prozesse zur Verfügung und man hätte eine Spielfeldbreite von 500, so gibt es vier Segmente mit einer Breite von je 125 Spalten. Bei Restbeträgen werden diese gleichmäßig aufgeteilt und die Breite der Teilspielfelder dementsprechend angepasst. Abbildung 5 verdeutlicht die Aufteilung des Spielfelds.

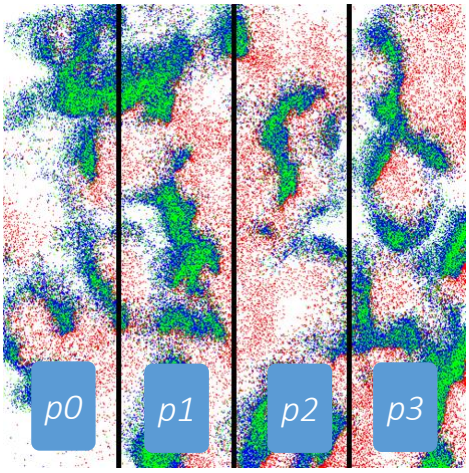

Abbildung 5: Aufteilung des Spielfelds bei 4 Prozessen

3.1 Setzen der Kreaturen auf das Spielfeld

Da beim Setzen der Kreaturen keine Abhängigkeiten zwischen den einzelnen Teilspielfeldern bestehen, können alle Threads die Kreaturen gleichzeitig auf das Spielfeld setzen. Dazu werden die Kreaturen gleichmäßig und zufällig auf die einzelnen Teilspielfelder aufgeteilt. Eine Kommunikation zwischen den einzelnen Prozessen ist hier aus diesem Grund nicht notwendig.

3.2 Ermitteln der Bevölkerungsgröße

Zur Ermittlung der Bevölkerungsgrößen aller Kreaturarten hat jeder Prozess einen Counter zu den Bevölkerungsarten, welche während der gesamten Zeit jedoch unabhängig voneinander gezählt werden. Der Master wiederum addiert die gesammelten Zähler am Ender der kompletten Simulation und gibt die Summe der jeweiligen Bevölkerung aus. Der Versand der Zähler erfolgt über einen extra MPI-Datentyp („mpi_counterSumAtSimStep“).

3.3 Bewegen der Kreaturen auf dem Spielfeld

Beim Bewegen der einzelnen Kreaturen kann es passieren, dass diese sich entweder in ein anderes Teilspielfeld hinein- oder hinausbewegen. Damit dies reibungslos vonstattengeht, müssen die jeweils benachbarten Teilspielfelder miteinander kommunizieren. Zu Beginn werden die Kreaturen auf den Teilspielfeldern der Prozesse bewegt, die eine gerade Prozesszahl

besitzen. Da es hier nun dazu kommen kann, dass Kreaturen das Teilspielfeld verlassen und in das benachbarte Feld wechseln, müssen dem benachbarten Spielfeld mit ungerader Prozesszahl die Änderungen mitgeteilt werden. Da das gesamte Nachbarspielfeld für die einzelnen Prozesse nicht von Nöten ist, wird nur der aktualisierte Randbereich gesendet. Es werden also die äußeren zwei Spalten des eigenen Bereichs zzgl. der zwei Spalten vom Nachbarn an eben diesen geschickt. Die zwei Spalten kommen aus dem Grund zu Stande, da die Weitsichtfunktion einer Kreatur die Nachbarkoordinaten des eigentlichen Nachbars benötigt. Anschließend führen die ungeraden Prozesse die Bewegung ihrer Teilspielfelder aus und senden die Spaltenränder ihren Nachbarn zu.

Keines der verfügbaren Prozesse erhält das gesamte Spielfeld, sondern nur die Ränder des jeweiligen Nachbars zur Aktualisierung seines eigenen Spielfelds.

3.4 Anwenden der Regeln

Die in 2.2 beschriebenen Regeln werden innerhalb der Teilspielfelder von den einzelnen Prozessen parallel angewendet. Eine Kommunikation zwischen den einzelnen Threads während der Regelanwendung ist nicht notwendig, da die Spielfeldgrenzen nicht überschritten werden. Einzige Ausnahme ist das Setzen von Neugeborenen auf dem Spielfeld. Aufgrund dessen, dass Neugeborene auf eine Nachbarposition von zwei Menschen oder Vampirjägern mit entsprechendem Alter gesetzt werden können, besteht die Möglichkeit, dass diese auf ein Nachbarspielfeld gesetzt werden. Aus diesem Grund wird für das Setzen der Neugeborenen äquivalent vorgegangen wie beim Bewegen aller Kreaturen und die Spaltenränder den Prozessen zugesendet, die für die Berechnung der Nachbarspielfelder zuständig sind. Dieser Vorgang wurde bereits in 3.3 beschrieben.

3.5 Visualisierung

Falls der Modellanwender in der Konfiguration das Erstellen von Bildern im jeweiligen Simulationsschritt eingeschaltet hat, so werden dem Master-Prozess alle Teilspielfelder der anderen Prozesse zugeschickt. Nach dem Erhalt fügt der Master das Spielfeld zusammen und erstellt die Bilder. Hierzu gibt es den Datentyp „mpi_creature“.

4 Laufzeitmessungen

Um Aussagen über die Laufzeit zu treffen, wurde eine Simulationsdauer von 3000 Simulationsschritten bei einer Spielfeldgröße von 501 in X und Y Richtung festgelegt. Alle sonstigen Parameter wurden ebenfalls so gewählt, dass alle Arten von Kreaturen bis zum Ende der Simulation überleben.

4.1 Simulationsschritte pro Sekunde

Bei der Messung der Simulationsschritte, die pro Sekunde berechnet werden können, konnte ein erheblicher Anstieg bei der Erhöhung der Prozesse festgestellt werden. So können bei einem Knoten und 12 Prozessen ca. 4 Schritte pro Sekunde berechnet werden. Das sequentielle Programm mit nur einem Prozess schafft weniger als einen Simulationsschritt in der Sekunde zu berechnen. Bei mehr als 12 Prozessen ist kein Anstieg mehr zu beobachten. Bei zwei Knoten und 24 Prozessen werden 6 Simulationsschritte in der Sekunde berechnet. Eine Prozesszahl höher als 24 führte dazu, dass weniger Schritte berechnet werden können. Die Nutzung von 4 Knoten erbrachte bei 46 Prozessen den höchsten Wert. Hier wurden über 7 Simulationsschritte in der Sekunde berechnet. Abbildung 6 zeigt die Anzahl Simulationsschritte, die mit einem, zwei und vier Knoten berechnet werden konnten. Dass die Simulation mit einem Knoten und 12 Prozessen bzw. zwei Knoten und 24Prozessen bereits an einem Maximum angelangt ist, ist auf die Architektur des Clusters zurückzuführen.

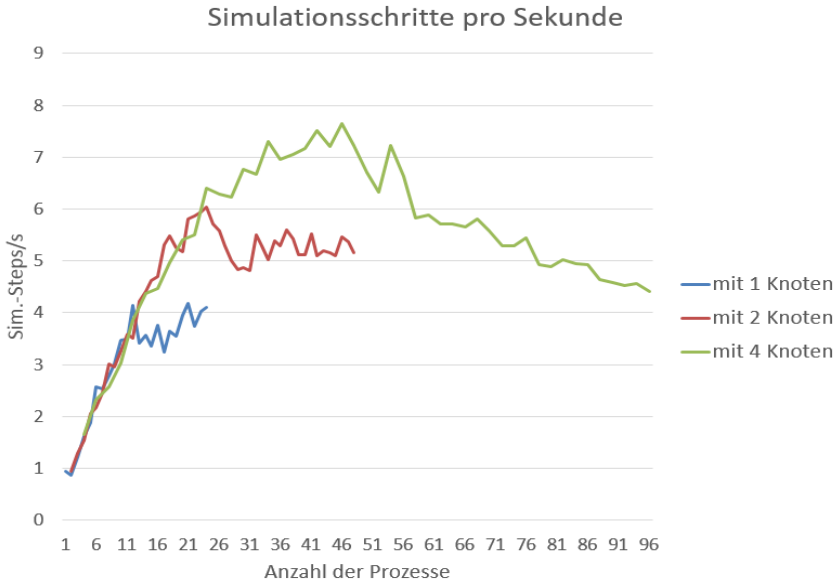

Abbildung 6: Simulationsschritte pro Sekunde

4.2 Effizienzdiagramm

Abbildung 7 zeigt die Effizienz bei der Berechnung der Simulationsschritte pro Sekunde. Hier ist zu erkennen, dass bei einer Hinzunahme von mehreren Prozessen die Effizienz deutlich

abnimmt. Bei der Nutzung von einem Knoten ist die Abnahme wesentlich schneller als beispielsweise bei vier Knoten. So ist bei einem Knoten und 24 Prozessen nur noch eine Effizienz von ca. 18% vorhanden. Die Effizienz liegt bei der Nutzung von zwei Knoten und

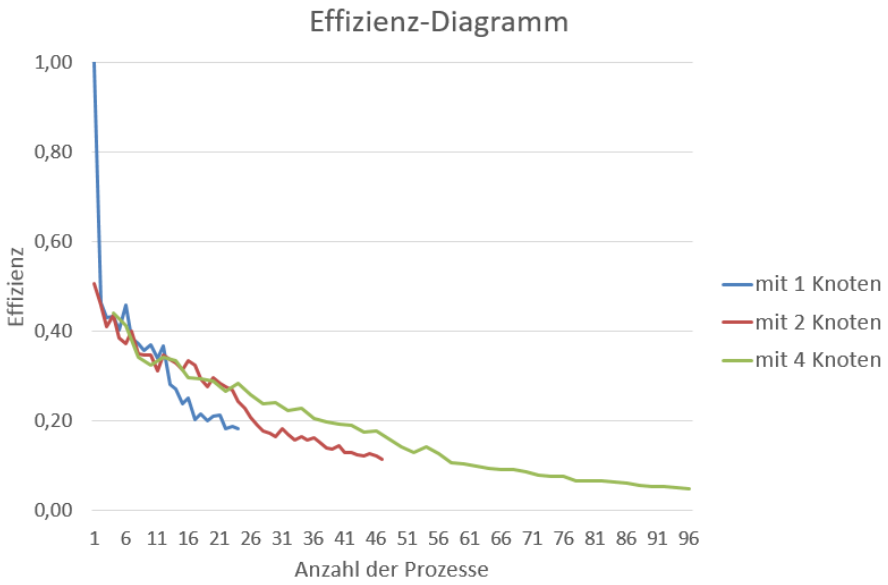

Abbildung 7: Effizienz Diagramm

derselben Prozessanzahl bei etwa 27% und bei vier Knoten bei 28%. Die Werte des Effizienzdiagramms zeigen, dass bei der Nutzung mehrerer Knoten und Prozesse die Anzahl zu berechnender Simulationsschritte pro Sekunde ansteigen, jedoch nur mit einer niedrigen Effizienz. Der größte Effizienzwert mit 45,9% lag bei der Nutzung von einem Knoten und 6 Prozessen. Eine Effizient von mehr als 50% war nicht zu erwarten, da in der Bewegungsphase der größte Teil der Zeit verbraucht wird und diese nur je die Hälfte der Prozesse nutzt.

5 Leistungsanalyse

5.1 Analyse mittels Vampir

Bei der Leistungsanalyse mittels Vampir kam das zu erwartende „Schachbrettmuster“ zur Erscheinung. Aufgrund der versetzt statt findenden Kreaturbewegung auf den einzelnen Teilspielfeldern warten die Prozesse mit ungerader Prozesszahl auf die Übertragung der Spaltenränder, die erst von den Prozessen mit gerader Prozesszahl berechnet werden müssen. Erst dann können die Prozesse mit ungerader Prozesszahl mit der Berechnung des Teilspielfelds beginnen. Die MPI Funktionen sind in Rot und die Berechnung der eigentlichen Anwendung in Grün dargestellt. Abbildung 8 zeigt dieses „Wechselspiel“ bei der Allokation von 12 Prozessen.

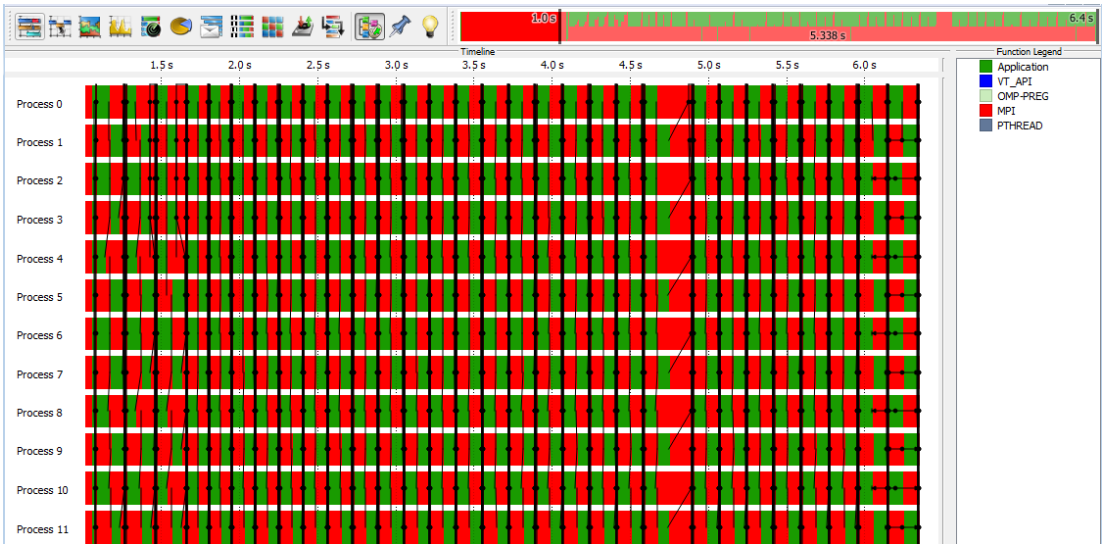

Abbildung 8: Leistungsanalyse mithilfe von Vampir bei 12 Prozessen

5.2 Analyse mittels Oprofile

Ca. 27% der CPU-Zeit sind durch die Funktion *getPositionInArray* ausgelastet. Wie der Name der Funktion vermuten lässt, ist *getPositionInArray* dafür zuständig die 3-dimensionale Spielfeldposition zu einem Integer Wert zu verändern, so dass die Position im Spielfeld-Array ermittelt werden kann. Bei jeder Kreaturbewegung und jedem Kreaturaufruf wird diese Funktion aufgerufen, so dass eine hohe Auslastung zu erwarten war. Eine ebenfalls hohe Auslastung der CPU-Zeit ist durch die für die Weitsicht der Kreaturen benötigten Funktionen *vampireDanger* und *getTargetCounter* zu beobachten. Diese beiden Funktionen werden auch mehrmals bei der Bewegung einer Kreatur aufgerufen, um Feinden aus dem Weg zu gehen und Opfer zu verfolgen. Die Funktionsweise wurde in 2.2 bereits erläutert. Abbildung 9 zeigt die Ergebnisse bei der Ausführung von Oprofile.

```
CPU: Intel Westmere microarchitecture, speed 2667 MHz (estimated)
Counted CPU_CLK_UNHALTED events (Clock cycles when not halted) with a unit mask of 0x00 (No unit m
samples  %        image name               app name                 symbol name
1824548  27.1089  vampire                  vampire                  getPositionInArray
1535904  22.8202  vampire                  vampire                  vampireDanger
974070   14.4726  vampire                  vampire                  GetTargetCounter
666474    9.9024  vampire                  vampire                  CreatureNotDefault
480131    7.1337  vampire                  vampire                  KeinPlatzInUmgebung
134612    2.0000  vampire                  vampire                  huntVictim
95818     1.4236  vampire                  vampire                  GetNextYPos
90560     1.3455  vampire                  vampire                  GetNextXPos
87367     1.2981  libc-2.15.so             vampire                  random
86409     1.2839  vampire                  vampire                  newbornPossible
81144     1.2056  vampire                  vampire                  moveCreaturesAndSetNewborns
76153     1.1315  libc-2.15.so             vampire                  random_r
75176     1.1170  vampire                  vampire                  useAllRules
60008     0.8916  vampire                  vampire                  moveHuman
```

Abbildung 9: Oprofile Ausgabe mittels opreport -l

6 Erweiterungspotential

Für ein solches Modell gibt es verschiedene Erweiterungspotentiale, die zeitbedingt nicht weiter verfolgt worden sind. Die Anzahl Kreaturen, die auf einer Spielfeldkoordinate gesetzt werden können, hätte auf eine vom Modellanwender beliebig gewählte Größe erweitert werden können. So wären Kämpfe möglich, an denen mehr als zwei Kreaturen gegeneinander antreten können. Käme es zu einem Kampf, so hätten die Kreaturen mit höherem Aufkommen einen Vorteil. Die Kämpfe könnten darüber hinaus mit Lebens- und Erfahrungspunkten für alle Kreaturen realitätsnäher stattfinden.

Eine weitere Anpassung käme den Neugeborenen zu Gute. In diesem Modell kann jeder Mensch beliebig viele Kinder zeugen. Eine Geburtenrate scheint angemessen.

Die Analyse mittels Vampir zeigt, dass Verbesserungspotentiale in der Parallelisierung möglich sind. So könnte beispielsweise das Spielfeld nicht nur in Spalten, sondern auch Zeilenweise unter den Prozessen geteilt werden. Dies könnte zu einer Erhöhung der Simulationsschritte, die pro Sekunden berechnet werden können, führen.