Outbreak – Simulation ansteckender Krankheiten
Praktikum "Parallele Programmierung"

Hajo Möller

29. November 2012
Sommersemester 2012

Inhaltsverzeichnis

Zusammenfassung

Outbreak ist ein Kommandozeilen-basiertes Werkzeug für unixartige Betriebssystem zur Simulation einer ansteckenden Krankheit auf einer torusformigen Welt, wobei Eigenschaften wie Weltgröße, das Verhältnis von Landfläche zu Wasser, Population je Simulationspunkt sowie Heilungsrate, Mortalität und Ansteckungswahrscheinlichkeit als Kommandozeilenparameter übergeben werden. Zur schnellen Visualisierung wird je Simulationsschritt der aktuelle Ausbreitungsstatus als Portable Pixmap ausgegeben.

1 Einführung

1.1 Ziele

Die rechnerische Simulation von Krankheiten ist seit einigen Jahren eine erfolgreiche Unterstützung bei der Prognose und darauf folgender Eindämmung schwerer Epidemien[1]. Durch bessere Simulationen können gute Vorhersagen über die zu erwartende Ausbreitung einer Krankheit getroffen werden, was eine leichtere Koordination von Gegenmaßnahmen wie gezielte Schutzimpfungen, Importstopps und Grenzschließungen ermöglicht.
Für die Pandemie-Simulation eignen sich rein statistische Modelle, die auf Bevölkerungsparametern basieren, oder Agentensysteme, bei denen die Personen der Bevölkerung jeweils einzeln betrachtet werden und so eine wesentlich höhere Komplexität erreichen. Ich versuchte *Outbreak* als Agentensystem wie in „Predicting the spread of pandemics in urban environments“[2] aufzubauen, aufgrund mangelnder *C*-Kenntnisse und so entstehender Speicherlecks und anderer Probleme musste ich diesen Ansatz verwerfen und auf ein simpleres, rein mathematisches Modell zurückgreifen.
So kann mit *Outbreak* zur Zeit auch jeweils nur eine statt, wie ursprünglich geplant, bis zu $2^{64} - 1$ Krankheiten simuliert werden. Ferner musste auch auf die Implementierung von Immunitäten und Mutationsfähigkeit der Erreger verzichtet werden.

1.2 Anwendung von *Outbreak*

Ein Aufruf von *Outbreak* ohne Parameter führt zu Ausgabe der Revisionsnummer, des Kompilierungszeitpunktes und der erwarteten Parameter:

```
% ./outbreak
Outbreak (4fd8216489, compiled Nov 29 2012 15:49:29)

Usage: outbreak [width] [height] [countries] [watertoland] [population]
    [travelchance] [contagiousness] [deadliness] [curability]

  [width]           (uint32_t) width of simulation
  [height]          (uint32_t) height of simulation
  [countries]       (uint8_t)  countries to spawn
  [watertoland]     (double_t) water / land
  [population]      (uint32_t) count of people per area
  [travelchance]    (double_t) chance any given person will travel this
      round
  [contagiousness]  (double_t) people infected by 1 sick person
  [deadliness]      (double_t) chance of a sick person dying
  [curability]      (double_t) chance of spontaneous healing
could not parse args
```

Listing 1: Beispielausgabe bei Aufruf ohne Parameter

1. `width` steuert die Breite in Simulationspunkten der simulierten Oberfläche
2. `height` steuert die Höhe
3. `countries` bestimmt, wie viele Zufallspunkte der Oberfläche als Startpunkte für Landmasse markiert werden, siehe *Welterstellung*
4. `watertoland` ist ein Koeffizient, mit dem das Verhältnis von Wasser- zu Landstartpunkten bestimmt wird
5. `population` legt die Anfangspopulation jedes Simulationspunktes fest
6. `travelchance` ist die Wahrscheinlichkeit, mit der eine Person seine Standpunkt auf einen anderen Simulationspunkt verlegt
7. `contagiousness` ist die Basisreproduktionsrate der betrachteten Krankheit, sie gibt den Durchschnitt von je Simulationsschritt durch eine infektiöse Person infizierte Personen an
8. `deadliness` ist die durch die simulierte Krankheit verursachte Mortalität
9. `curability` ist die Wahrscheinlichkeit, zu der eine infizierte Person innerhalb eines Simulationsschritts geheilt wird

Bei Aufruf mit korrekter Parameterzahl erfolgt keine Ausgabe, stattdessen beginnt die Simulation und legt nach jedem Schritt den aktuellen Zustand als Portable Pixmap (`world00000.ppm`, `world00001.ppm`, ...) im Arbeitsverzeichnis ab.
Die Simulation beendet sich selbstständig sobald alle infizierten Personen geheilt wurden oder verstorben sind, sie kann bei Bedarf jederzeit mit `Strg+c` abgebrochen werden.
In den ausgegebenen Bildern entspricht ein Pixel einem Simulationspunkt, dabei haben die auftretenden Farben folgende Bedeutungen:

Blau Wasser

Grün Punkt mit durchgehend gesunder Bevölkerung

Gelb Mindestens eine infizierte Person in diesem Punkt

Rot Die Bevölkerung dieses Punkts ist vollständig erkrankt

Schwarz Kaum oder keine lebenden Personen in diesem Punkt

Je dunkler ein Punkt erscheint, desto niedriger ist das Verhältnis von aktuell lebenden zu bereits verstorbenen Personen.

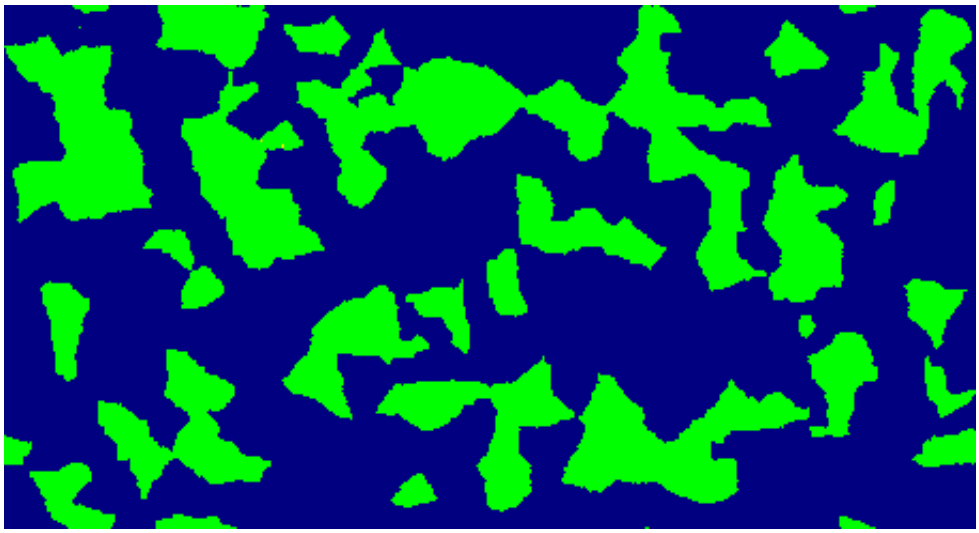

Abbildung 1: Schritt 0, `./outbreak 455 245 100 2 1000 0.2 1.2 0.02 0.02`

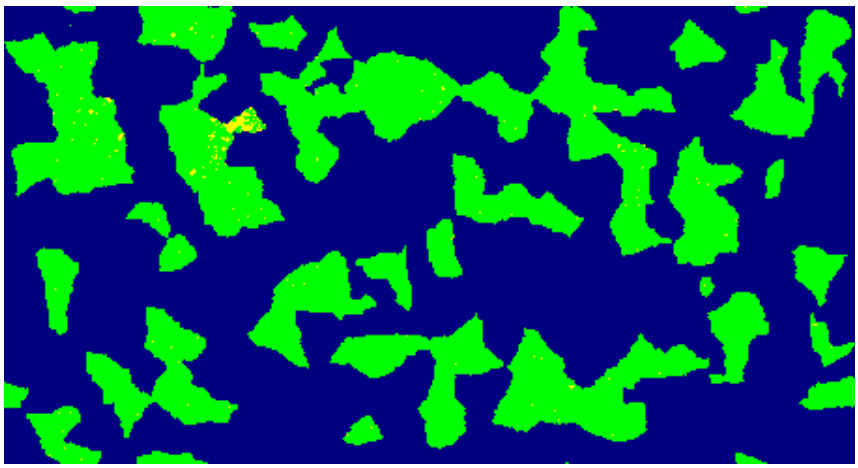

Abbildung 2: Simulation in Schritt 25

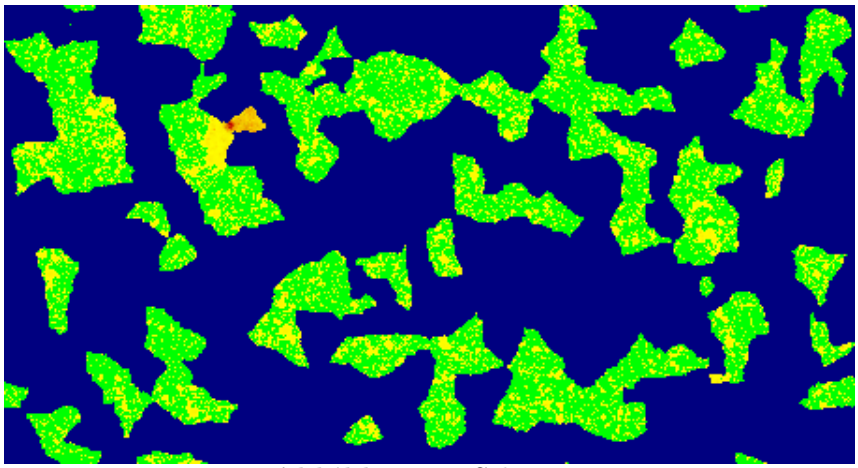

Abbildung 3: Schritt 70

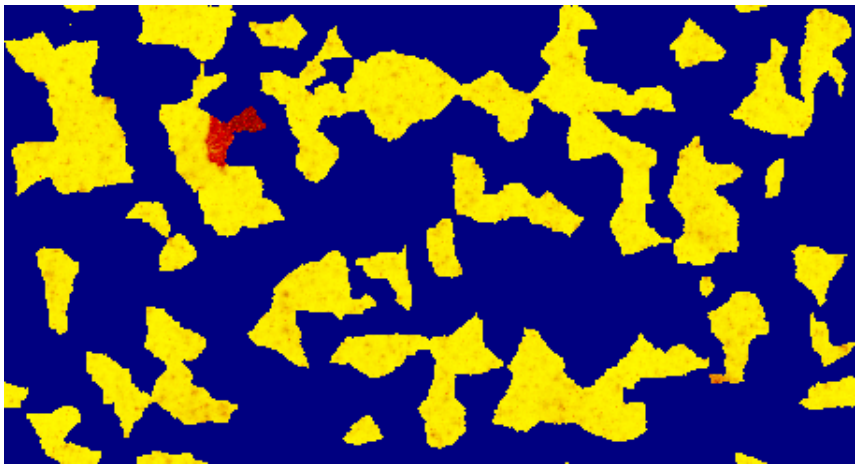

Abbildung 4: Schritt 90

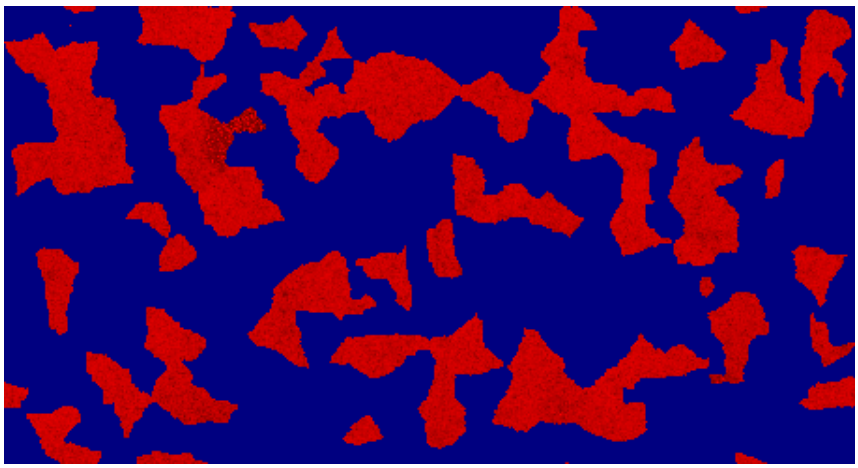

Abbildung 5: Schritt 110

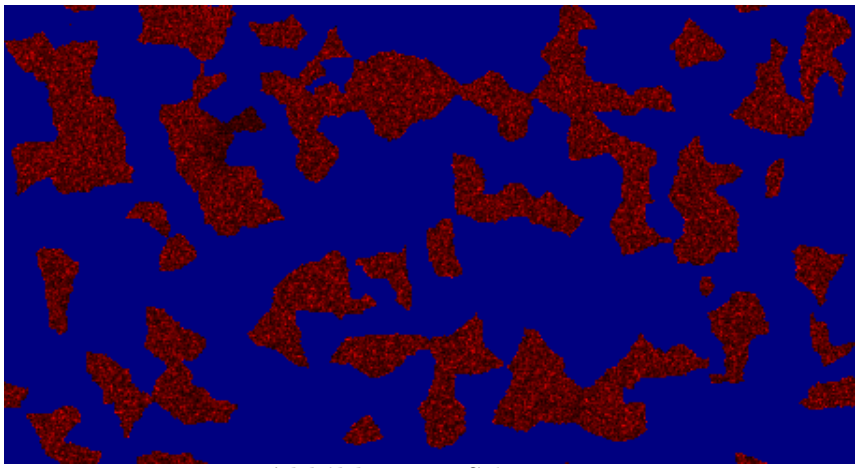

Abbildung 6: Schritt 160

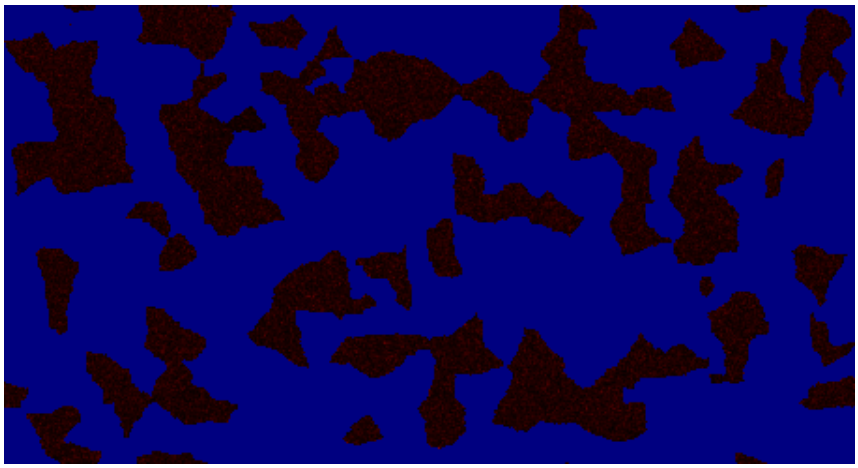

Abbildung 7: Schritt 200

2 Methoden

Der vollständige Quelltext von *Outbreak* steht unter der GPL2-Lizenz[3] und kann über `http://wr.informatik.uni-hamburg.de/_media/teaching/sommersemester_2012/papo-12-outbreak-sourcecode.tgz` bezogen werden.

2.1 Datenstrukturen und Algorithmen

2.1.1 Structs

Jeder Simulationspunkt entspricht einem Struct `AREA` mit folgenden Elementen:

```
typedef struct s_area {
    uint8_t type; /* 0: empty, 1: water, else: land */
    float coeff_age;
    float coeff_reproduction;
    uint64_t population;
    uint64_t infected;
    uint64_t alive;
    uint64_t dead;
    //~ PERSON *people;
} AREA;
```

Listing 2: Struct `AREA`

Die Elemente `coeff_age`, `coeff_reproduction` und `*PEOPLE` kommen in der aktuellen Fassung nicht zum Einsatz, wie unter *Ziele* erläutert. Der (somit unbenutzte) Datentyp `PERSON` ist so konstruiert:

```
typedef struct s_person {
    uint8_t health;
    uint8_t age;
    uint8_t infected; /* disease_1 & disease_2 & ... */
    uint8_t immune; /* immunity_1 & immunity_2 & ... */
    uint8_t *incubation;
    uint8_t *latency;
} PERSON;
```

Listing 3: Struct `PERSON`

Es war geplant, jede lebendige Person einzeln zu simulieren und am Ende jedes Berechnungsschrittes die reisenden Personen zusammenzufassen und per MPI zu den Zielpunkten zu schicken.

2.1.2 Welterstellung

Die Welt ist als Torus implementiert, wenn man sie also z.B. über den linken Rand verlässt betritt man sie an der rechten Seite in selber Höhe.
Bei der Erstellung der Welt durch `int make_world(void);` geschieht folgendes:

1. Im Speicher wird der für die Oberfläche benötigte Bereich reserviert und auf 0 gesetzt
2. Es werden `countries` Punkte zufällig ausgewählt und als Landfläche (intern: $2 \ldots$ `countries` $+2$) markiert, Punkte mit gleicher Markierung werden später dem selben Land zugeordnet.
3. Nun werden `countries * watertoland` Punkte ausgewählt und als Wasserflächen (intern: 2) markiert
4. Bis jeder Punkt der Oberfläche mit einem anderen Wert als 0 belegt ist wachsen die Land- und Wasserflächen je Iteration ungleichmäßig in alle Richtungen ohne bereits belegte Punkte zu überschreiben

So ist zugesichert, dass die Welt vollständig initialisiert wird und unregelmäßige Formen entstehen.

2.1.3 Simulationsschleife

Die Welt wird durch `int run_simulation(AREA *myworld);` so lange simuliert bis es keine infizierten Personen mehr gibt, sie also entweder ausgestorben oder vollständig geheilt wurden:

1. Feststellung, dass der aktuell betrachtete Punkt nicht leer ist (d.h. lebende Personen, kein Wasser), sonst Sprung zum nächsten Punkt
2. Für jeden Infizierten überprüfen, ob er innerhalb dieses Schrittes geheilt wird und bei Bedarf Heilung durchführen
3. Krankheit innerhalb des Punktes ausbreiten lassen, neue Anzahl Infizierter = alte Anzahl $*$ `contagiousness`, höchstens jedoch so viele Infizierte wie überhaupt lebende Personen schaffen
4. Für jeden nun Infizierten wird überprüft, ob er in diesem Schritt stirbt. Bei Bedarf wird die Anzahl lebender, infizierter und toter Personen angepasst
5. Exakt ein Ziel für alle Reisende aussuchen, dabei zuerst die direkten Nachbarpunkte prüfen, dann einen zufälligen Punkt des selben Landes und schließlich eine zufälligen Punkt der Welt.
6. Die reisenden Personen werden „umgezogen“, im Ursprungs- und Zielpunkt werden die Zähler der lebenden und infizierten Personen angepasst

2.2 Parallelisierungsidee

Meine Idee war, die Welt in Zeilenblöcke mit jeweils etwa der selben Anzahl an Simulationspunkten auf die bearbeitenden Knoten aufzuteilen. Die Kommunikation, also das Reisen, würde nicht blockierend ablaufen. Jeder Knoten schickt am Ende eines Simulationsschrittes einen Puffer mit von ihm Reisenden (also `struct PEOPLE`) per `MPI_Isend(...)` an den entsprechenden Zielknoten, welcher vor seinem nächsten Schritt die Datenmenge empfängt und die einzelnen Personen den entsprechenden Punkten zuordnet.
Zusätzlich müsste der Master-Knoten kontinuierlich auf Statusupdates der Worker-Knoten horchen, um den aktuellen Status als Pixmap ausgeben zu können.

3 Ergebnis

Leider habe ich es nicht geschafft, eine funktionierende parallele Variante von *Outbreak* zu implementieren, eine Performance-Auswertung ist somit nicht möglich. Zusätzlich macht der hohe Anteil an zufallsgesteuerten Ereignissen die Rechenlast sehr unvorhersagbar und ungleichmäßig, bei einigen dem Programm übergebenen Parameterwerten beendet es sich mal nach nur sehr wenigen Runden, mal nach erst 500+ Runden.
Momentan bietet *Outbreak* also nur einen kleinen Einblick in die Welt der Pandemiesimulation und kann bereits existierenden Programmen wie InfluSim[4] keine Konkurrenz machen.

Literatur

[1] *Models of Infectious Disease Agent Study*; http://www.nigms.nih.gov/Research/FeaturedPrograms/MIDAS/

[2] *Predicting the spread of pandemics in urban environments*; Aleman, Dionne M.; http://www.informs.org/ORMS-Today/Public-Articles/February-Volume-39-Number-1/Predicting-the-spread-of-pandemics-in-urban-environments

[3] *GNU General Public License, Version 2* http://www.gnu.org/licenses/old-licenses/gpl-2.0.de.html

[4] *Influenza-Pandemie-Planungs-Simulator InfluSim* http://www.uni-tuebingen.de/modeling/Mod_Pub_Software_InfluSim_de.html