IMPLEMENTING A LARGE-SCALE SPATIAL EPIDEMIC MODEL AS AGENT-BASED SYSTEM USING GPGPU AND DATABASES

Günter Schneckenreither[1], Günther Zauner[2], Niki Popper[2], Felix Breitenecker[1]

[1]Vienna University of Technology, Institute for Analysis and Scientific Computing
Karlsplatz 4, 1040 Vienna, Austria
[2]dwh simulation services
Neustiftgasse 57-59, 1070 Vienna, Austria

gschneck@osiris.tuwien.ac.at (Günter Schneckenreither)

Abstract

Epidemiological Models are basically composed of medical, demographic and spatial features of arbitrary complexity depending on available data and purpose of the simulation. With increasingly subtle structures the formulation and implementation becomes more and more challenging. Additionally the performance limit of simulators, applications and hardware interferes with the models fidelity. The purpose of this investigation is to find and test new approaches for implementing and simulating systems with large data volume and complexity. Some approved approaches/concepts involve matrix based programming languages, parallelisation, cellular automata and agent-based systems. Our goals are to swap costly calculations to equivalent methods with lower time penalty, make use of fundamental concepts such as databases during simulation or to facilitate access to advanced programming techniques like GPGPU (computing on graphics devices). This contribution presents and evaluates different approaches towards implementing large spatial systems using cellular automata and agent-based approaches. The discussion is based on a spatial model for simulating the spread of two competing diseases (serotypic shift).

Keywords: Cellular Automaton, Spatial Agent-Based System, Database, GPGPU, Disease Propagation

Presenting Author's Biography

Günter Schneckenreither studies mathematics at Vienna University of Technology. He joined the Modelling and Simulation Group in 2006 and works on Cellular Automata and Agent-Based Systems.

1 Introduction

Epidemiological models and simulations exist in varying forms. A very basic and imprecise yet often sufficient and used method for simulating epidemic scenarios or predicting patterns in the spread of diseases is by statistical analysis and evaluation of data.

The most common mathematical model for simulating spread of diseases is the classical SIR (susceptible-infected-recovered) ODE (ordinary differential equation) model or extensions (e.g. by spatial factors) derived from this dynamic approach. A population is stratified into susceptible, infected and recovered sub-populations and the accumulated flow between those groups is described by a system of ODEs with non-linear respectively linear reaction terms.

A more detailed demographic resolution by age or gender for example requires additional equations, which model the dynamics of the resulting sub-populations. Since such systems of ODEs become very complex, can no longer be treated with analytic methods, are unstable and difficult to parametrise, alternative approaches like agent-based models can be used to introduce finer grained systems.

When simulating spatial dispersion of pathogens it is necessary to use partial differential equations, which require numerical solution. Alternatively explicit top-down approaches like cellular automata can be used to achieve the same dynamic behaviour and simultaneously allow interweaving with agent-based components.

Furthermore important epidemiological factors like competing pathogens or different disease stages require the simulation of stochastic processes (e.g. Markov chains).

Often simpler approaches are chosen because some of those techniques are mathematically hard to handle or development is time-intensive. Furthermore if the model becomes more advanced, the implementation tends to become more complex so that it is necessary to improve the implementation in order to reduce the calculation time or memory consumption. But even a slightly more dedicated implementation can be very useful to reduce the time overhead. On the other side recent developments brought easier access to GPGPU and multi-core processors, which makes it possible to perform multiple or many operations in parallel and thus reduce the calculation time drastically.

2 Virtual Epidemiological Model

The virtual model simulates individual and unique persons of a population based on demographic data [1]. According to the data individuals are generated with specific features like age, gender, health, environment bonus, residence, family, profession, etc.

A second data set gives information about one or more disease pathogens. This data source must define the likelihood of infection and recovery with the same resolution as the demographic data. Furthermore the competition between multiple pathogens and its dynamics must be clearly defined.

The resulting system is being iterated over time and defines multiple phases of interaction within different groups of the population (p.e. within a family, at work or through social classification, etc.). Furthermore such groupings must not unexceptionally be static and predefined. Interaction can also happen according to geographic habits of individuals or the locations of their domiciles, working places or schools.

The previously demanded structure of the model is highly predestined for an agent-based approach. Every person of the population has individual features which are (randomly) generated from statistical data. Additionally complex and irregular interactions depend on the properties of the individuals and stochastic processes.

If interaction depends on spatial distances, each agent must maintain a geographic position and a system for integrating this information into the process of interaction (i.e. infection).

3 Implementing the Agent-Based System

A straight forward method for storing information per agent in a program is by using arrays or matrices. The data must be imported/generated at initialisation from CSV files, spreadsheets or databases, which is often a time consuming operation. Since all data within an array must usually be of the same data type (boolean, integer, double, string or even bitwise) there is a large memory overhead if one single matrix is used. For example in MATLAB a matrix that holds 1 million agents with 5 features each costs 40MB of RAM if stored in double; if 3 of the 5 columns can be stored as unsigned 8-bit integers (`uint8`), memory consumption is less than the half. Many programming languages provide additional data structures, which allow to combine fields of different data type. But such extended data structures often exhibit larger read and write times.

Another very intuitive approach is by object-oriented programming. Agents can easily be implemented as objects or as an array of objects. Each instance of the agent-class stores its own properties and provides the methods that are used for interaction. The object-oriented approach is practical to implement but does not perform very well in terms of calculation time.

What happens if systems become really large? Supposed that the number of agents and features as well as concurrently running tasks requires that the operating system swaps memory to the hard-disk the calculation time increases dramatically. In this case it might be of advantage to deliberately release some components to the hard-disk (compare section 5). It is very difficult to decide which data should be moved to to the hard-disk and when and in what manner this should be done. Furthermore if specific data is required for fast access, it is necessary to implement fast search algorithms and data structures.

Some of these requirements are achieved by typical database applications. They exhibit concepts like memory hierarchy, caching, simultaneous access to data, high capacity, indexing and much more. The process of initializing data can be avoided by restoring previously dumped snapshots of a database or by using temporary tables.

3.1 Technical Details of an External Database Approach

We connect to the PostgreSQL database management system from a simple C++ program using the pqxx library on a Linux desktop computer.

- **Database Connection:** It is important to connect to a local database via Unix domain sockets because TCP connections are much slower.

  Reads and writes from and to tables are performed as atomic operations. That means that if a so-called transmission fails or aborts no data will be lost or left behind incomplete.

- **Queries:** All queries and operations such as the change of a state of an agent must be implemented in SQL. In contrast to pure SQL PL/pgSQL is a procedural language, which allows to define functions and variables within the database.

- **Database Optimisation:** PostgreSQL provides indexing, which allows to make search operations faster, and an analyse functionality, which cleans up the database and updates indices.

- **Temporary Databases:** It is possible to use temporary databases during simulation. No changes will be made to the actual tables and all results are discarded when terminating the connection.

- **Storage:** If the model is small enough the database can be hosted in RAM. This can be done by using file-systems like ramfs or tmpfs. This allows to use the advantages of a database layout and experience lower access times (section 4.2.2) at the expense of a more complex installation. For larger systems and if continuity is important a RAID system might be the right choice.

- **Parallelisation:** A database allows multiple connections so that reading and writing from and to the database can be simultaneously depending on the exact operations and rules defined in the database.

- **Visualisation:** Visualisation must be done in the main program and requires copying data from the database into memory or hard-disk space.

3.2 Technical Details of an Embedded Database Approach

The Oracle Berkeley Database System exhibits the following features:

- **Programming Interface:** The Berkeley Database is written in C and provides interfaces (libraries) to nearly all common programming languages (C++, C#, Java, Python, Perl, etc.). This database system is used by many popular applications and operating systems.

- **Database Connection:** No inter-process communication is required since the database system is integrated into program as a library and uses the same address space. This makes the embedded database approach very fast compared to external databases.

  Like in other database systems, transactional data-management is provided.

- **Queries:** This database system does not provide a SQL-API. All data lookups must be known and programmed in advance. There is no search by value, all lookups must be managed through keys (key-data pairs).

- **Storage:** A database file must be specified at initialization.

- **Parallelisation:** The Berkeley Database supports many parallel threads and a “two-phase” locking system.

4 Modelling Spatial Dynamics

There are two points of attack for introducing spatial dynamics. On the one hand interactions occurring in working places, schools, universities or wherever many individuals come into contact can be modelled as cellular automata. This means a switch from the agent-based domain to cellular automata (hybrid modelling [2]). On the other hand the global system can be arranged with spatial information. This involves defining the location of a family’s home, working places etc. at initialization based on statistical data. The resulting information can then be used to generate/simulate a transport infrastructure, congested areas etc. Geographic breakup is an important factor since it represents a very natural condition.

4.1 Implementing Spatial Subsystems

Aggregated interaction within temporary groupings can be modelled using the classical ODE-system or by stochastic means. A more subtle approach is by cellular automata or weighted interactions combined with stochastic features.

A non-aggregated method requires more calculation time and thus should be optimised. Sub-simulations in cellular automata are initialized by placing interacting agents on a two dimensional domain, where they move and interact during simulation [3]. Accelerating cellular automaton simulations can happen through parallelisation on different levels. On the one hand the domain can be sliced into multiple equal sized sub-domains. Agents or interaction processes which leave a sub-domain are passed over to the adjacent sub-domain

using traditional message passing systems. The other approach is easier to implement and features a parallel execution of code on a lower level by running multiple independent cellular automata in parallel.

An implementation in MATLAB of the first approach can use PVM (parallel virtual machine) and a free parallelisation toolbox [4, 5] as interface to the parallelisation technique. With Java or C++ threads or processes can scatter the calculation effort for different sub-domains or automata to different processors or cores. The typical matrix structure of cellular automata also favours an implementation of the first approach on graphics devices using OpenCL, CUDA, etc.

The development effort for accelerations using threads or graphics devices is of course greater. Early tests with the MATLAB-PVM approach delivered nearly linear speed-up, which means that four processor cores deliver the simulation in one fourth of the time for one cellular automaton. We currently work on a cellular automata parallelisation with MATLAB and PVM, which is compatible to an agent-based model [6]. The following (incomplete) listing shows the simple modifications that are necessary to scatter calculations of sub-domains to multiple virtual machines [7]. The installation on the other side is slightly more complex since it involves compilation and linking of the interface written in C.

```
pid=dpparent;
if isempty(pid)          % MAIN PROGRAM
 tid=dpspawn(hosts,'thisfile');
 dpscatter(geometry,tid);
 dpsend(tid,tid);
 u=dpgather(tid);
else               % PARALLEL EXECUTION
 grid=dprecv(dpparent);
 tid=dprecv(dpparent);
 for all iterations
  if POS<SIZE
   dpsend(grid(end,:,:),tid(POS+1));
  end
  if POS>1
   northborder=dprecv(tid(POS-1));
   dpsend(grid(1,:,:),tid(POS-1));
  end
  if POS<SIZE
   southborder=dprecv(tid(POS+1));
  end
 end
 dpsend(u,dpparent);
end
```

A C/C++ implementation typically uses arrays of pointers to access different layers of the grid. Each layer represents one of the possible moving directions (4 or 6 for rectangular respectively hexagonal grids) respectively a position within a cell. In order to improve the execution time of the streaming operator, it is possible to shift the layers sidewise by swapping pointers to the rows or columns of these layers. This allows further to process different layers in parallel, which means that this is a third possible approach for parallelisation within a cellular automaton.

```
// allocation:
int** gridN = new int*[gridsize];
for (int y=0;y<gridsize;y++)
   gridN[y] = new int[gridsize];
...
// parallel streaming:
int* Nbuffer = gridN[0];
for (int y=0;y<gridsize-1; y++)
   gridN[y] = gridN[y+1];            // up
gridN[gridsize-1] = Nbuffer;

int* Wbuffer = gridW[gridsize-1];
for (int x=gridsize-1;x>0; x--)
   gridW[x] = gridW[x-1];            // left
gridW[0] = Wbuffer;
```

4.2 Introducing Geographic Information in the Global System

If the global agent-system is to be equipped with spatial dynamics each agent must store its current position (coordinates), which leads to two more (double) variables for each agent. The model must provide the functionalities to perform motion and interaction depending on the distance between two agents.

For example if an infection can take place if the distance is smaller then r, it is necessary to find all neighbouring agents within radius r of an agent. Therefore the distance of each agent to every other agent must be taken into account for each time step. It is clear that this operation is very time consuming (compare [8]). A further very complex problem are path-finding algorithms, if agents should move to specific locations [9] - for example from home to the working place.

We discuss two completely different approaches for implementing such dynamics efficiently.

4.2.1 Spatial Dynamics Using GPUs

A hypothetical scenario was tested on GPUs using CUDA and compared with a native C++ implementation [10]. The iterative test model involved moving agents and static "signposts" on a two dimensional domain. According to the signposts direction-vector, the agents change their moving direction.

It is very important to understand that just implementing an algorithm in a straight-forward fashion on **high-performance** hardware does not necessarily improve the runtime of the program. This is especially true for the test model. The crucial point of the system is that all agents must find their neighbouring (radius r) signposts (**search algorithm**). This search can be done by going through all signposts for every agent (linear search). A much faster approach is by using an uniform grid search method. This means that the domain is divided into a grid of $(r \times r)$-sized cells thus limiting the lookup-area of every agent to 4 ("neighbouring") cells. Furthermore an index-array stores the index or identity of the first and last signpost within each cell and thereby additionally speeds-up the search process. Both search approaches were compared in a GPU and CPU implementation (Fig. 1), showing that there is a time difference of multiple orders of magnitude.

In contrast to an implementation in MATLAB or C/C++

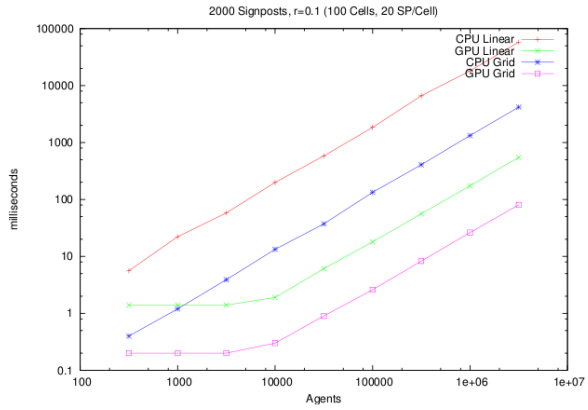

Fig. 1 log-log scaled plot of time needed for simulating a system with a given number of agents [10]. The figure shows that the CPU implementation and the linear search approach are orders of magnitude slower than the GPU respectively grid search approach.

the architecture of the graphics device must be taken into account when building an algorithm. CUDA for example provides access to different memory spaces, which exhibit certain advantages and drawbacks concerning life time, access times and interchangeability. Furthermore the significant time penalty for exchanging data with the CPU or main memory should be avoided, thus making it necessary to avoid connecting to different sub-models concurrently running on the CPU. This is of course a great drawback and makes GPU programming a challenging task (**advanced programming technique**). The following listing shows parts of the CUDA implementation for finding an agents neighbouring signposts within a certain neighbouring cell with cell-index `(idxX,idxY)` (linear search within a grid cell).

```
__device__ int searchCell(float px,float py,
     int idxX,int idxY,float &vx,float &vy) {
 int vacc = 0; // number of acc.-vectors
 // load signpost indices from index array
 // into (fast and temporary) texture memory:
 int2 idx = tex2D(texSPIndices, idxX, idxY);
 // find signposts within radius:
 for (int i=idx.x; i < idx.y; i++) {
  // load required signposts:
  float4 sp = tex2D(texSignposts,
       i % (1<<16), i / (1<<16));
  // calculate distance to signpost:
  float dx = sp.x - px;
  float dy = sp.y - py;
  float dist = dx*dx + dy*dy;
  // add signposts direction vector
  // to agents velocity vector
  if ( dist < cdRadiusSqr ) {
   vx += sp.z;
   vy += sp.w;
   vacc++;
  }
 }
 return vacc;
}
```

In contrast to CPU implementations it is possible to implement **real-time interactive visualization** using OpenGL.

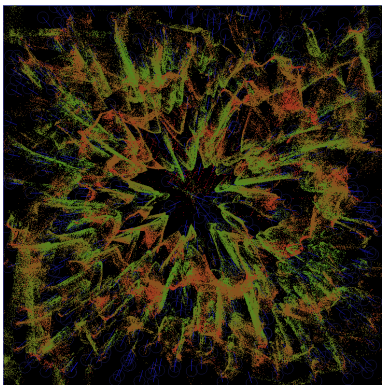

Fig. 2 Interactive real-time visualization of a GPU simulation. Red agents move fast; green agents move slowly. Signposts and their directions are shown in blue.

4.2.2 Spatial Dynamics Using the External Database Approach

The PostgreSQL database management system is provided with a highly optimised extension (PostGIS) for storing and manipulating geographic and geometric information. PostGIS adds support for geometry columns, which store **geometric information** in a binary format including cartographic projections. Geometric objects include points, lines, circles, polygons, etc. Additionally a **wide range of functions** abstract queries for the distance between two objects, whether one object lies within another, to calculate the area of an object etc.

In order to improve queries on geometric objects, geometry columns can be indexed using **geometric indexing**, which involves bounding boxes or so-called R-trees [11]. Accordingly it is not necessary to implement search algorithms or data structures. Nevertheless it is **important to be familiar with SQL** since choosing the right way to implement a query can be rather tricky but it is necessary to avoid time overhead coming from sequential scans or badly designed queries. Of course the overall performance is lower than for a very well-designed C++ or GPU approach. Depending on the programmers taste SQL can make things either easier or worse. Analysis tools like `EXPLAIN` and `ANALYZE` can be used to calculate the exact time consumption and operations for each query, which helps to improve the code. The following (badly designed) SQL query was used in the test model to perform interaction in working-places.

```
CREATE OR REPLACE FUNCTION interact_work()
RETURNS
 integer AS $$
BEGIN
 UPDATE agents
 SET state = (CASE
    WHEN rand < prob.P_02A THEN 1
```

```
    WHEN rand < prob.P_02A+prob.P_02B THEN 2
    ELSE 0 END)
 FROM
  ( SELECT numwork.workingplace AS work,
   (1-(1-P02A)^numwork.NumA)*
       (1-P02B)^numwork.NumB AS P_02A,
   (1-(1-P02B)^numwork.NumB)*
       (1-P02A)^numwork.NumA AS P_02B
  FROM
   ( SELECT
    count((SELECT ag.id
       WHERE ag.state = 1)) AS numA,
    count((SELECT ag.id
       WHERE ag.state = 2)) AS numB,
    ag.workingplace AS workingplace
   FROM agents AS ag
   GROUP BY ag.workingplace
   ) AS numwork
  ) AS prob,
  random() AS rand
 WHERE prob.work = agents.workingplace AND
  agents.state = 0;
RETURN 0;
END;
$$ LANGUAGE plpgsql;
```

The main program is written in C++ and provides access to the database through the pqxx library. This allows to do automated testing, Monte Carlo analysis or comparison with other approaches. Graphical output can be generated in parallel to the execution of SQL queries. We store all geometric information in the SVG file format and then use a script to convert the vector graphics to a bit-mapped image format (PNG) using ImageMagick and finally create a video using FFmpeg.

If the database stays small enough, it can be completely hosted in memory. This is done by mounting a file-system in RAM using ramfs at boot time. A PostgreSQL table-space must be defined so that all databases within this table-space are stored on the mounted virtual device. After the file-system is mounted and before the simulation starts a database layout must be restored from dump. Therefore at initialization of the simulation no new set of agents must be generated. Since memory consumption can be rather high it is necessary to use computers with large memory and fast disk access (RAID), which makes this approach rather complicated or expensive.

4.2.3 Spatial Dynamics Using the Embedded Database Approach

In contrast to the PostgreSQL system, the **Berkeley DB system does not provide predefined functions or geometric data types**. All lookups must be built upon hashed searches or B-trees. This fact places the embedded database approach somewhere between a native implementation and a full-featured database approach.

The remaining benefits compared to a native implementation are the **caching** infrastructure, **storing to the hard-disk**, **controlled (locking) simultaneous access to the data** and two methods for finding data by keys (namely **hash-search and B-tree search**).

Datasets can either be loaded at start-up or generated before the actual simulation. The library features routines for retrieving, setting and updating datasets. All those operations take a pointer to a variable and copy or change the contents of this variable. All updates to a database can be flushed to disk by committing the changes.

Therefore, in contrast to the external database approach, **no SQL statements** must be composed, transmitted and parsed and the retrieved data must not be copied from the database engine to the program but can be **directly accessed through pointers**. Accordingly it is much easier to handle large database entries.

At the time of writing the performance comparison was not yet fully implemented but if this approach is equipped with spatial indexing (p.e. index tables, uniform grid approach), a great performance boost in comparison to the external database implementation can be expected.

5 Results and Conclusions

The main result from our comparison is that there are several techniques that provide high performance. The difficulty lies in accessing those techniques. That is for example the CUDA language, which requires to understand the physical condition of a graphics device, or SQL, which requires analysing and optimising the resulting queries. Furthermore setting up a working CUDA environment or a high-performance PostgreSQL environment is not straight forward.

The benefit of existing search algorithms in the PostGIS approach could not be used efficiently since the design of fast SQL queries can be rather complicated and the time penalty for transmitting queries and results outweighs the great performance of these algorithms.

On the other hand the GPGPU approach proved to be rather static, since it is a closed system that does not allow (efficient) communication with the CPU.

A threaded C++ implementation proved to be the most flexible approach but does of course not deliver the same degree and level of parallelisation as the GPU approach. The main advantage is the possibility to connect to databases, graphics devices and image processing tools from one basic program.

Furthermore an embedded database for example with the Berkeley Database system could facilitate the data management in a native implementation.

In a next step it would be interesting to analyse a combination of those approaches. A potential implementation of the test model could for example host the large datasets in an embedded database and perform appropriate calculations on the GPU. In order to optimise the performance of such a system further benchmarks and tests with all approaches and techniques would be necessary. Additionally an event-driven scheduling system could help to avoid unnecessary calculations where possible. For example if only a random subset of the agents is involved in certain instances of a periodic geometric operation.

6 References

[1] F. Miksch, N. Popper, G. Zauner, I Schiller-Frühwirth, and G. Endel. Modelling spread of pneumococcal diseases in austria: Long term behavior and impact of vaccination. In G. Elst, editor, *ASIM-Workshop 2009 in Dresden mit integrierter DASS'2009 - Workshop Beiträge*, volume Tagungsband (der Fullpaper), pages 147–152, Dresden, Germany, 2009. Fraunhofer IRB Verlag.

[2] N. Popper and F. Breitenecker. Combining different modelling approaches - parallel implementation and hybrid implementation. In G. Elst, editor, *ASIM-Workshop 2009 in Dresden mit integrierter DASS'2009 - Workshop Beiträge*, volume Tagungsband (der Fullpaper), pages 221 – 230, Dresden, Germany, 2009. Fraunhofer IRB Verlag.

[3] S. Emrich. *Comparison of Mathematical Models and Development of a Hybrid Approach for the Simulation and Forecast of Influenza Epidemics within Heterogeneous Populations*. Diploma thesis, Inst. f. Analysis und Scientific Computing, Vienna University of Technology, 2007.

[4] R. Fink. *Untersuchungen zur Parallelverarbeitung mit wissenschaftlich-technischen Berechnungsumgebungen*. Number 12 in Fortschrittsberichte Simulation. ARGESIM/ASIM-Verlag, Wien, 2008.

[5] S. Pawletta. *Erweiterung eines wissenschaftlich-technischen Berechnungs- und Visualisierungssystems zu einer Entwicklungsumgebung für parallele Applikationen*. Number 7 in Fortschrittsberichte Simulation. ARGESIM/ASIM-Verlag, Wien, 2000.

[6] G. Endel, I Schiller-Frühwirth, N. Popper, G. Zauner, F. Miksch, and F. Breitenecker. Multi agent simulation techniques for dynamic simulation of social interaction and spread of diseases with different serotypes. In J. Mauskopf, editor, *ISPOR TWELFTH ANNUAL EUROPEAN CONGRESS, Research Poster Abstracts*, volume 12. Nummer 7 of *Value in Health*. Wiley Periodicals Inc., October 2009.

[7] L. Mayer. Cellular automata model using MATLAB-PVM, bachelor work, 2009.

[8] Andrew Crooks, Christian Castle, and Michael Batty. Key challenges in agent-based modelling for Geo-Spatial simulation. *UCL, Working Papers Series*, Paper 121, September 2007.

[9] S. Tauböck, M. Bruckner, N. Popper, D. Wiegand, S. Emrich, and S. Mesic. Ein hybrides modell zur simulation von raummanagement und räumungszeiten. In A. Gnauck and B. Luther, editors, *ASIM 2009 - 20. Symposium Simulationstechnik/20th Symposium Simulation Techniques, Cottbus, 23.-25. September 2009, Tagungsband auf CD-ROM*, volume 124 of *ASIM-Mitteilungen*, pages 423–430, BTU Cottbus, 2009.

[10] S. Hepp, D. Prokesch, and G. Schneckenreither. Construction and implementation of a simple Agent-Based system on GPU-Architectures. *Simulation News Europe SNE (draft submitted for contribution)*, 2010.

[11] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proceedings of the 1995 ACM-SIGMOD International Conference on Management of Data*, page 9, San Jose, USA, 1995.