GPU UNTERSTÜTZTE MULTI-AGENTEN SIMULATION

Philipp Kayser

Gliederung

- Problemstellung
- Motivation
- Multi-Agenten Simulation
- GPU Programmierung
- Stand der Technik
- Abgrenzung

- Multi-Agenten Simulation (MAS) simuliert durch eine Vielzahl von Agenten
- Die Größe / Granularität der Simulation wird durch die Ressourcen eingeschränkt

- MAS können parallelisiert werden
- GPUs bieten massive Parallelperformance

Motivation

- Bachelorarbeit
- Parallelität
- Hardwarenähe

Multi-Agenten Simulation

- Simulation auf Basis vieler Individuen
  - Jedes Individuum hat ein autonomes Verhalten
  - Agieren abhängig von der Umwelt

- Datengewinnung durch Beobachtung der Agenten und Zustand des Systems

Multi-Agenten Simulation

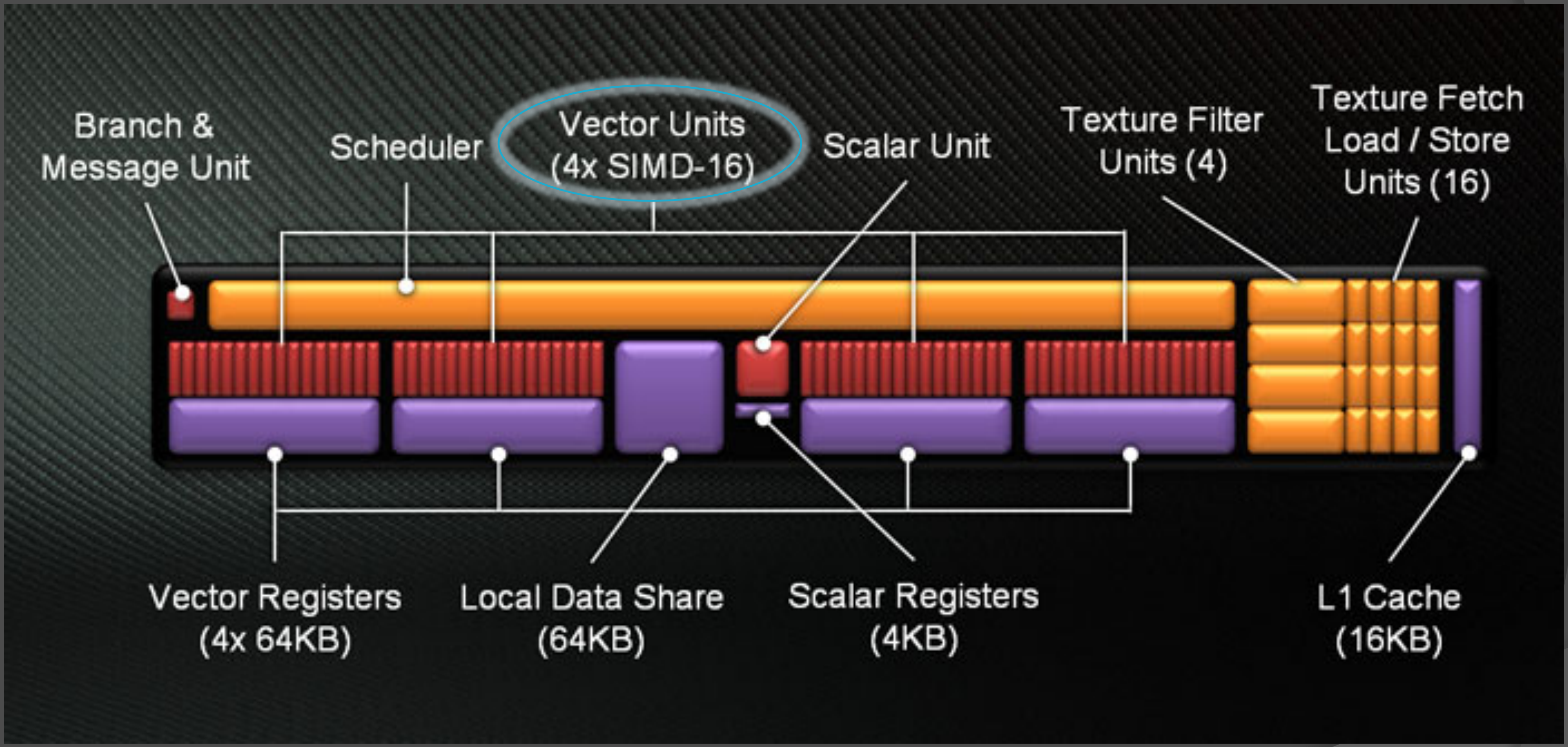
Multi-Agenten Simulation

- Zum Simulieren komplexer nicht linearer Systeme
  - Organismen
  - Ökologie
  - Massenverhalten
  - Verkehr

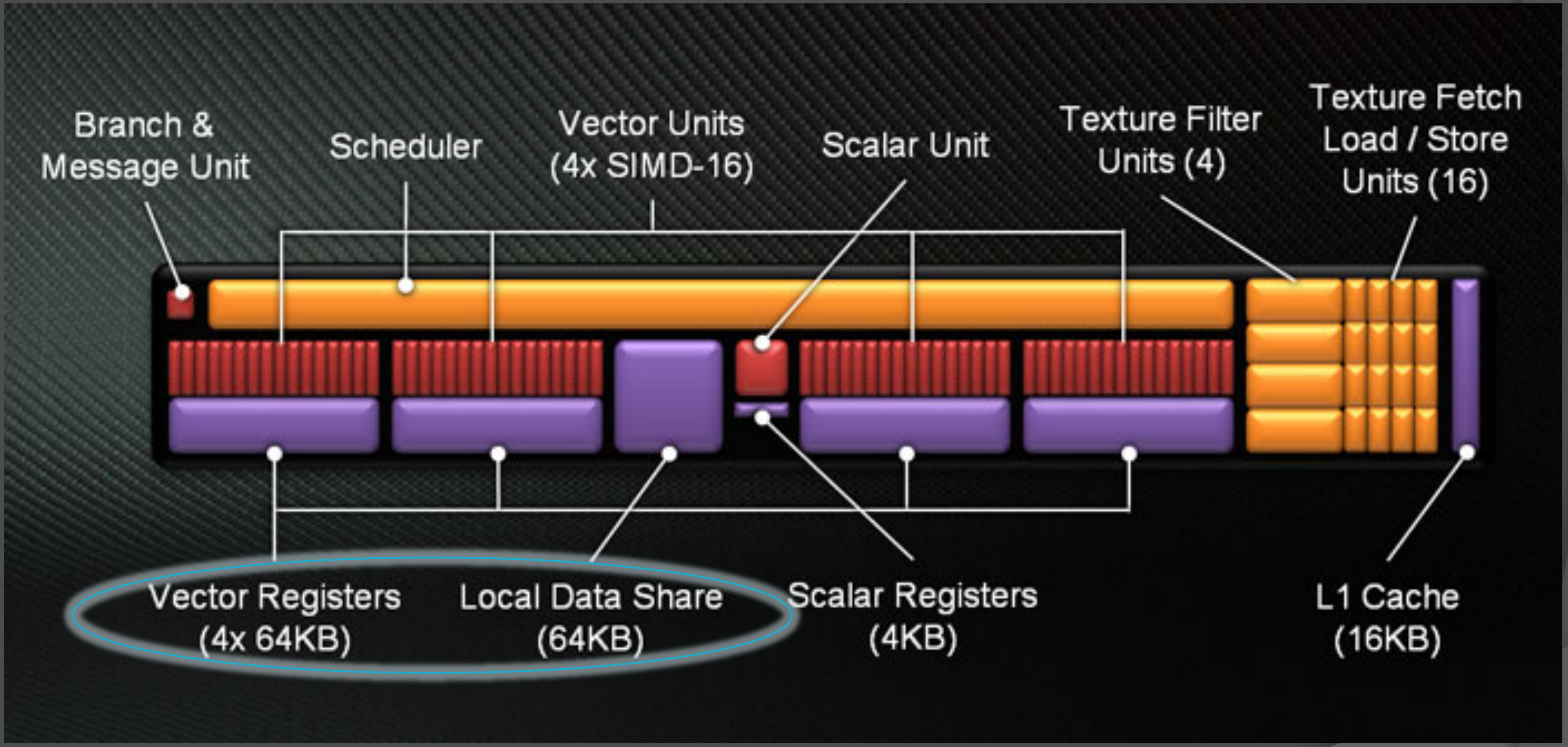
GPU Aufbau

Compute Unit

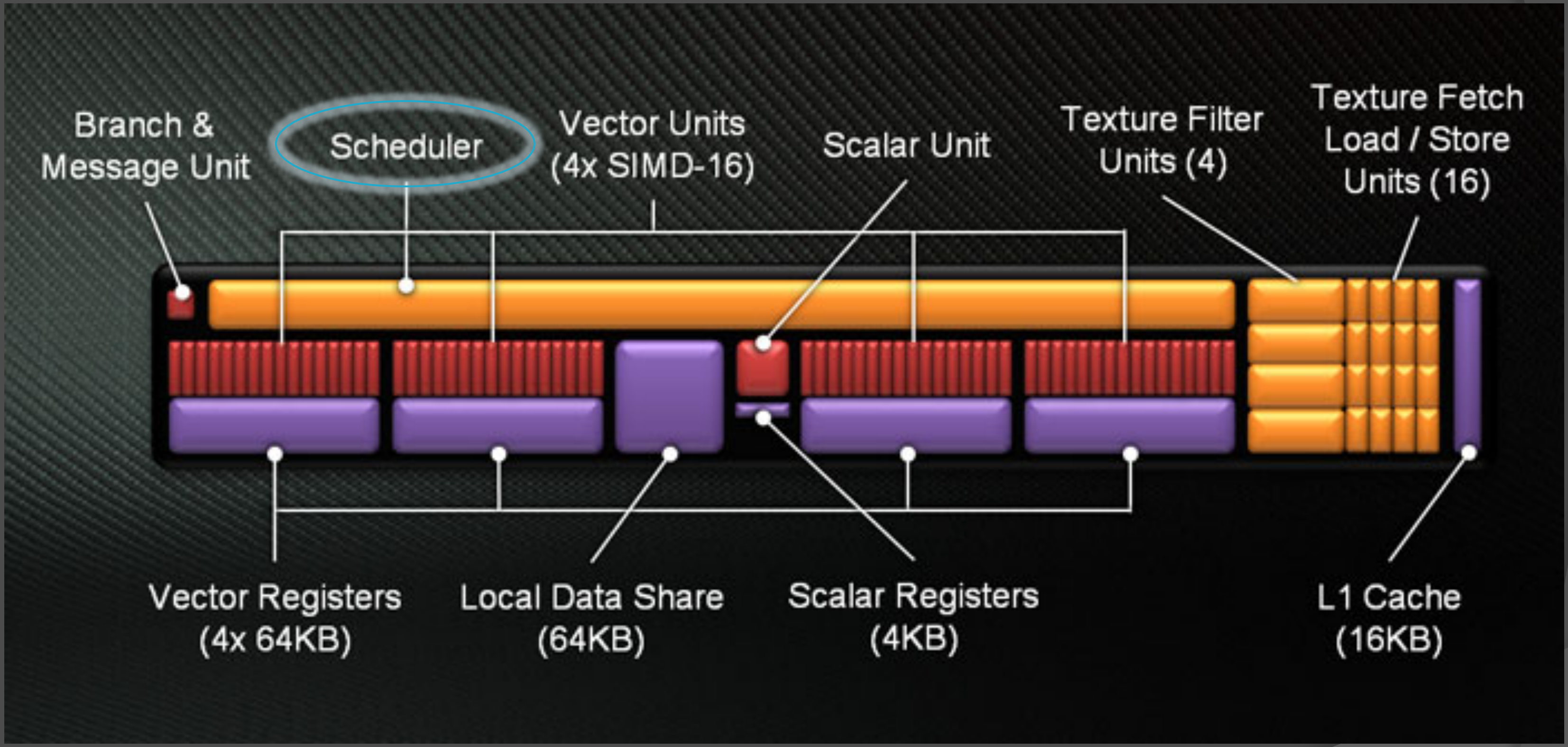
Compute Unit



Compute Unit



Compute Unit



GPGPU Programming

- **Compute Unified Device Architecture (CUDA)** von Nvidia
- **Open Computing Language** (**OpenCL**) von Apple Inc. / Khronos Group

OpenCL

- Host
  - Organisiert den Speicher
  - Kompiliert und erstellt Kernel Programme
  - Erstellt Command Queues

- Kernel
  - Parallele Ausführung auf dem Device
  - Wird in einer Abwandlung von C programmiert (OpenCL C)

```
clGetPlatformIDs(1, &platform, NULL);
clGetDeviceIDs(platform, CL_DEVICE_TYPE_GPU, 1,
   &device, NULL);
context = clCreateContext(NULL, 1, &device, NULL,
   NULL, &err);
```

Set platform/ device/context

```
program_handle = fopen(PROGRAM_FILE, "r");
fseek(program_handle, 0, SEEK_END);
program_size = ftell(program_handle);
rewind(program_handle);
program_buffer = (char*)malloc(program_size + 1);
program_buffer[program_size] = '\0';
fread(program_buffer, sizeof(char), program_size,
   program_handle);
fclose(program_handle);
```

Read program file

```
program = clCreateProgramWithSource(context, 1,
   (const char**)&program_buffer, &program_size, &err);
free(program_buffer);
clBuildProgram(program, 0, NULL, NULL, NULL, NULL);
```

Compile program

```
kernel = clCreateKernel(program, KERNEL_FUNC, &err);
queue = clCreateCommandQueue(context, device, 0, &err);
```

Create kernel/queue

```
mat_buff = clCreateBuffer(context, CL_MEM_READ_ONLY |
   CL_MEM_COPY_HOST_PTR, sizeof(float)*16, mat, &err);
vec_buff = clCreateBuffer(context, CL_MEM_READ_ONLY |
   CL_MEM_COPY_HOST_PTR, sizeof(float)*4, vec, &err);
res_buff = clCreateBuffer(context, CL_MEM_WRITE_ONLY,
   sizeof(float)*4, NULL, &err);
clSetKernelArg(kernel, 0, sizeof(cl_mem), &mat_buff);
clSetKernelArg(kernel, 1, sizeof(cl_mem), &vec_buff);
clSetKernelArg(kernel, 2, sizeof(cl_mem), &res_buff);
```

Set kernel arguments

```
work_units_per_kernel = 4;
clEnqueueNDRangeKernel(queue, kernel, 1, NULL,
   &work_units_per_kernel, NULL, 0, NULL, NULL);
```

Execute kernel

Beispiel Kernel

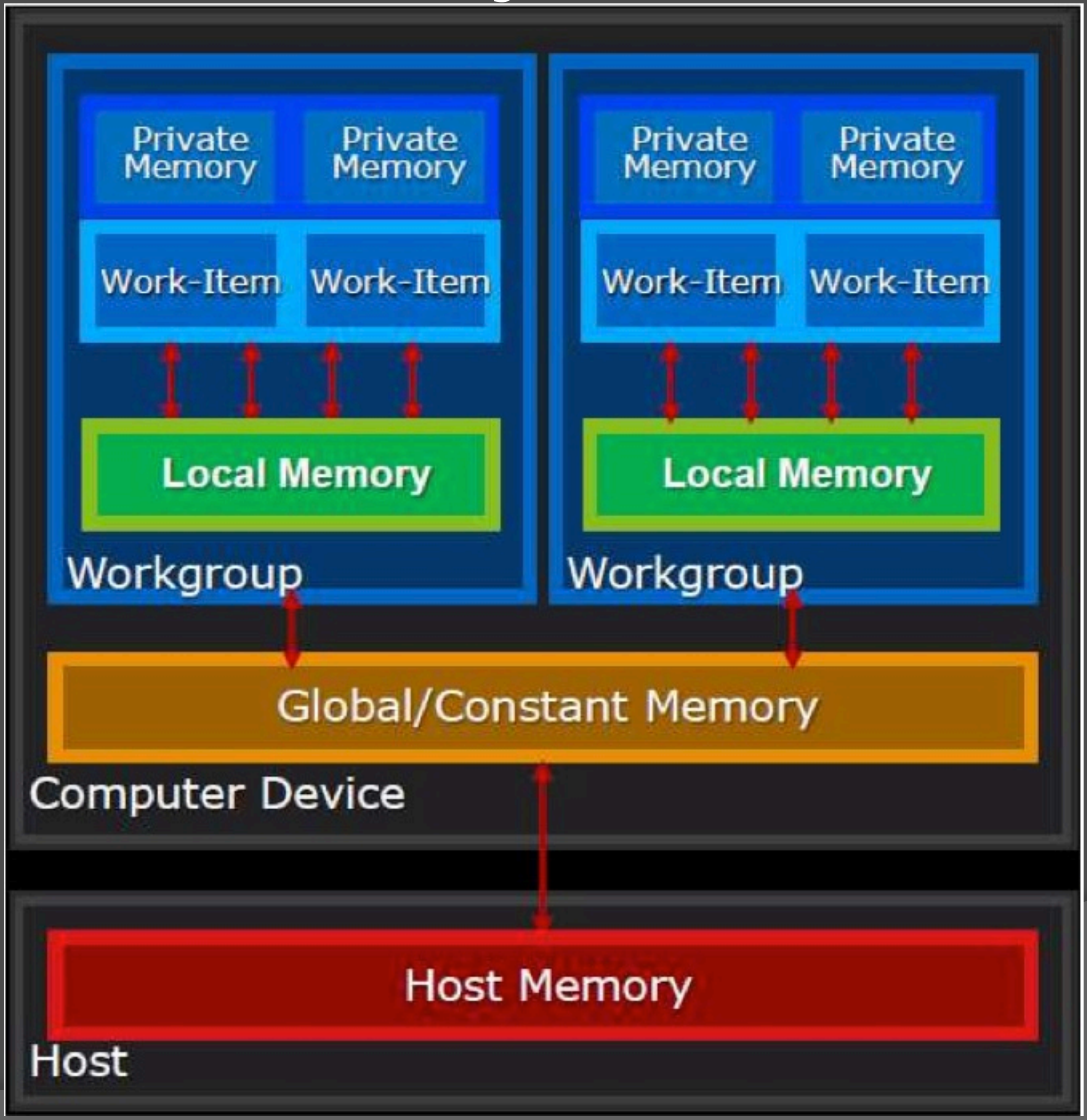
Standard C

```c
for(int i = 0; i < size, i++){
    result[i] = input[i]*input[i];
}
```

Kernel

```c
__kernel void square(__global float* input,
                     __global float* output,
                     const unsigned int count)
{
    int i = get_global_id(0);
    if(i < count)
        output[i] = input[i] * input[i];
}
```

OpenCL Memory Architecture



Analogie

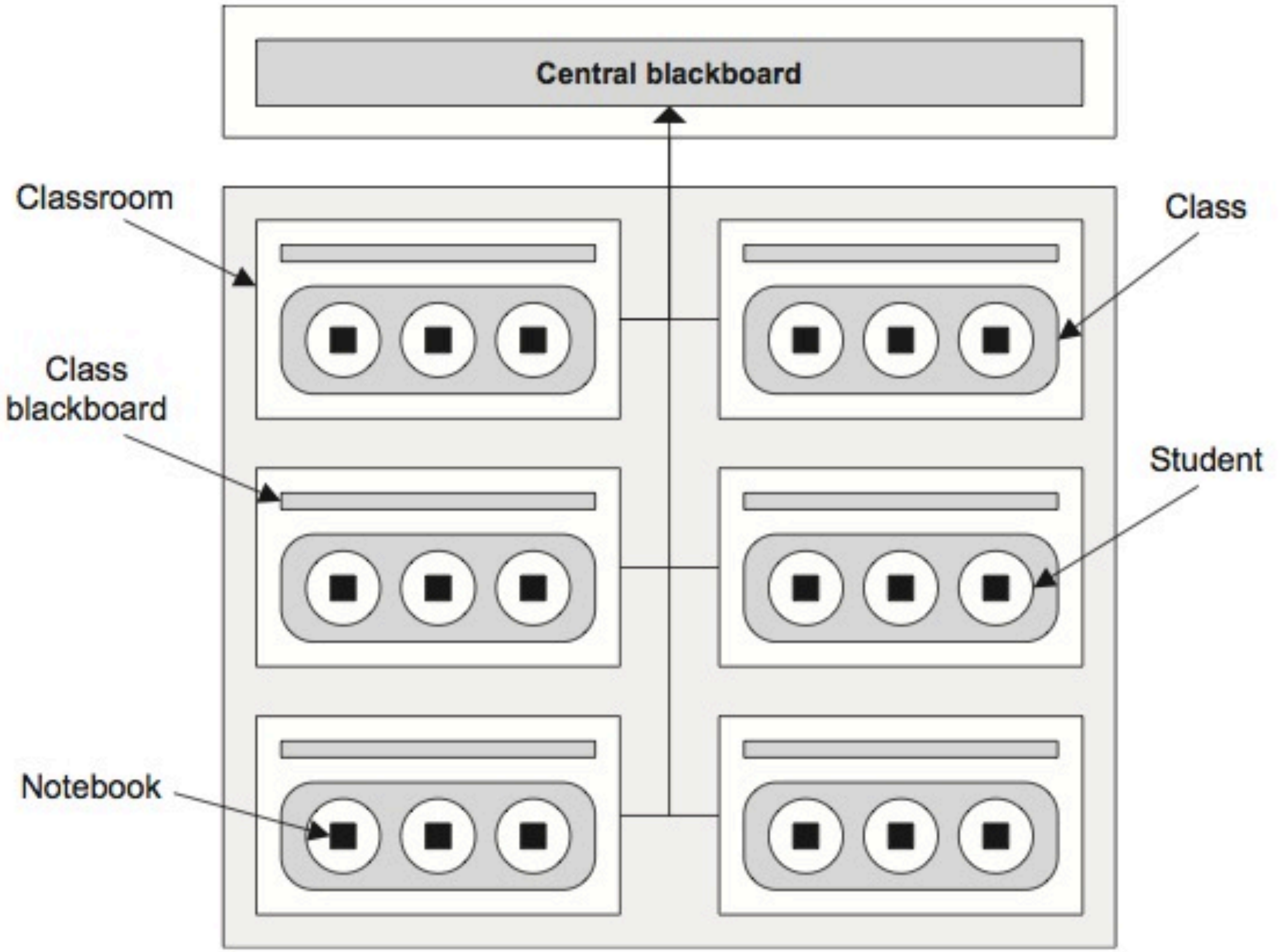


Figure 4.6 School of math students in OpenCL device analogy

Analogie

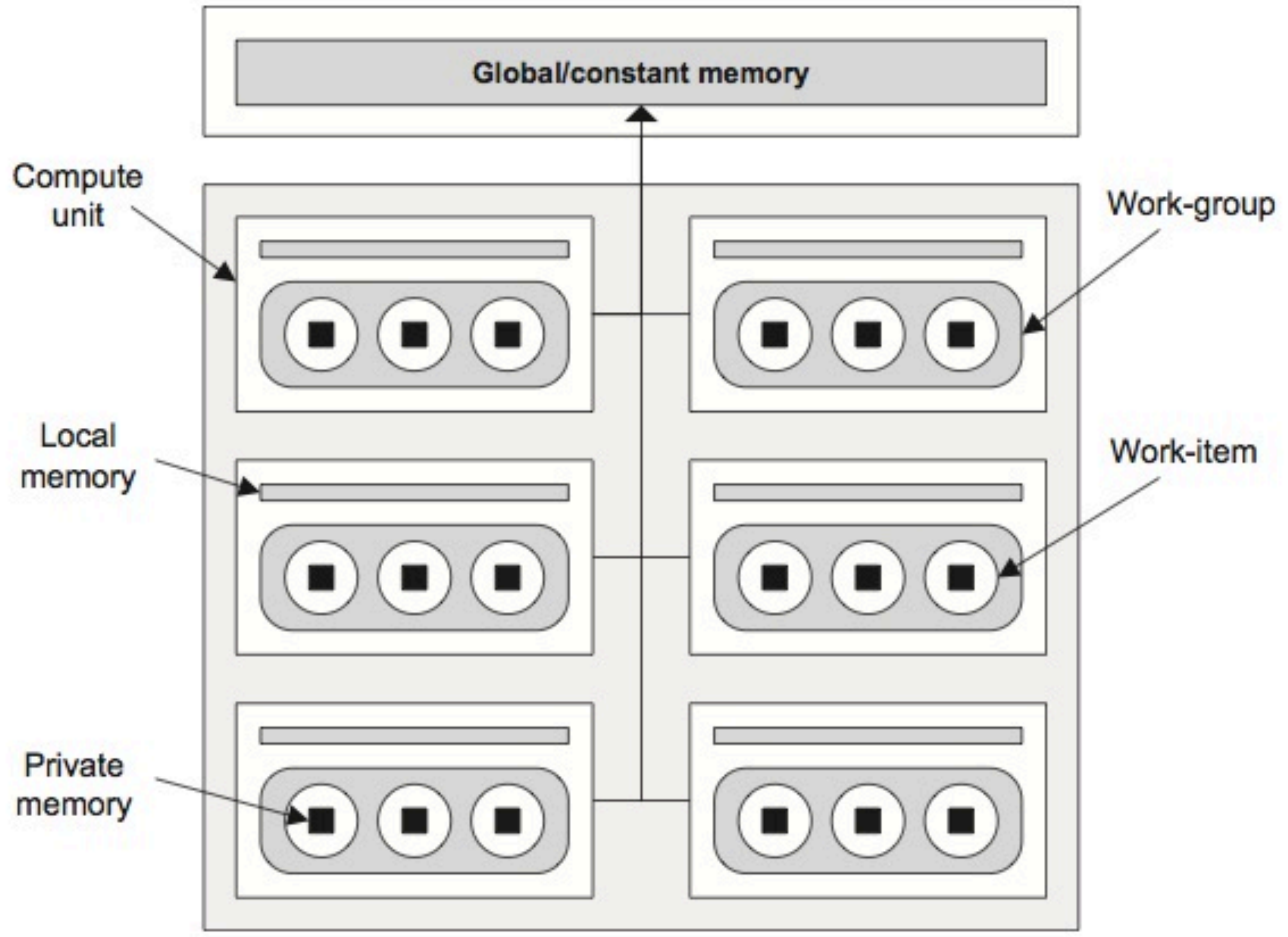


Figure 4.7 **OpenCL device model**

SAMPO (Kofler at al. (2014))

- Simuliert Population des *Anopheles gambiae* Mosquitos (Malaria)
- Basiert auf AGiLESim
- Vollständige Umsetzung der Simulation in OpenCL
- Starke Anpassung der Struktur

SAMPO (Kofler at al. (2014))

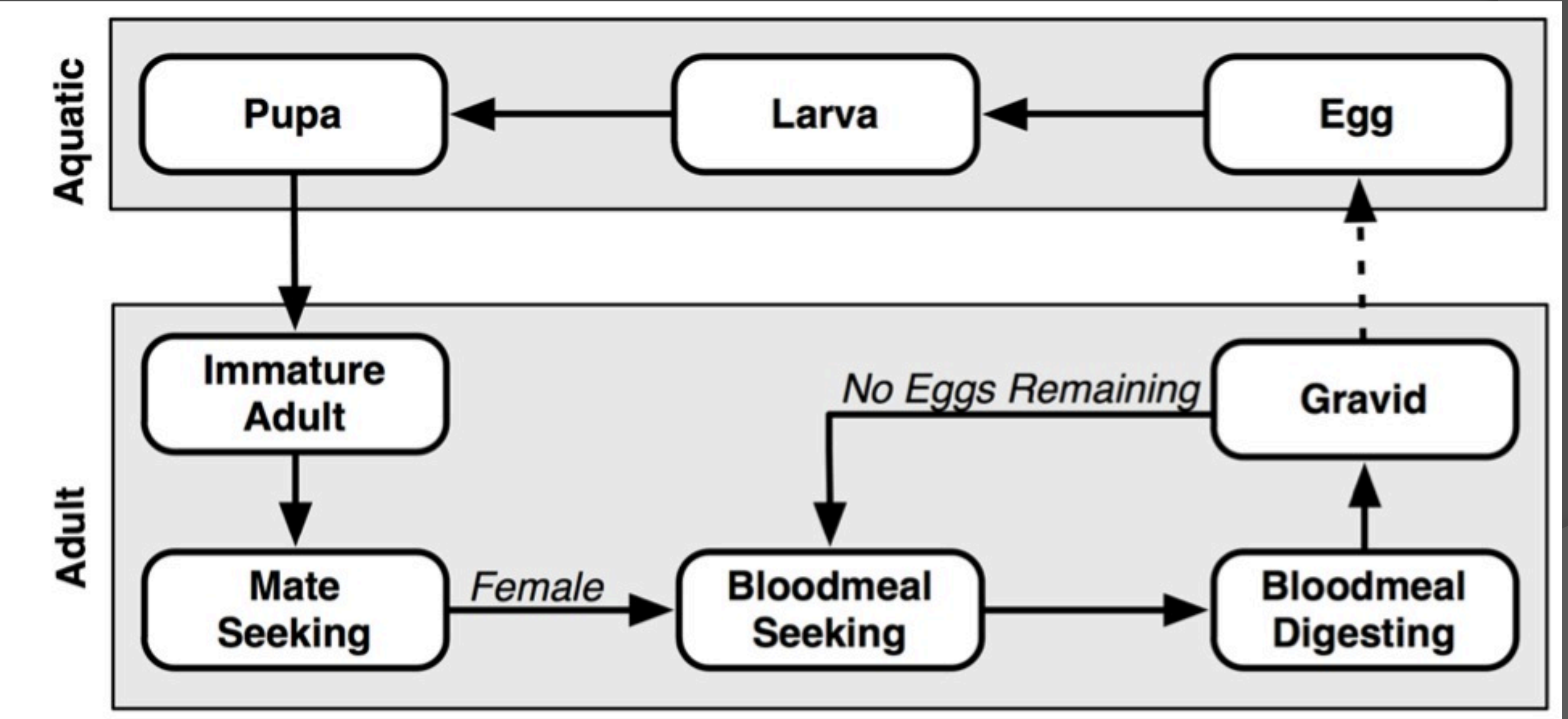


Figure 1. Mosquito life cycle.

SAMPO (Kofler at al. (2014))

Table 2. Memory used and transferred during the simulation

	Size in Bytes
host[a]	$240 + 24 * \#iterations$[b]
device[a]	$240 + 116 * \#agents$[c]
host to device[d]	64
device to host[d]	100

[a] total amount of allocated memory, constant during the entire simulation
[b] number of time-steps in the simulation
[c] maximum number of agents specified as described in Section 2.1.2.
[d] data transferred in each iteration

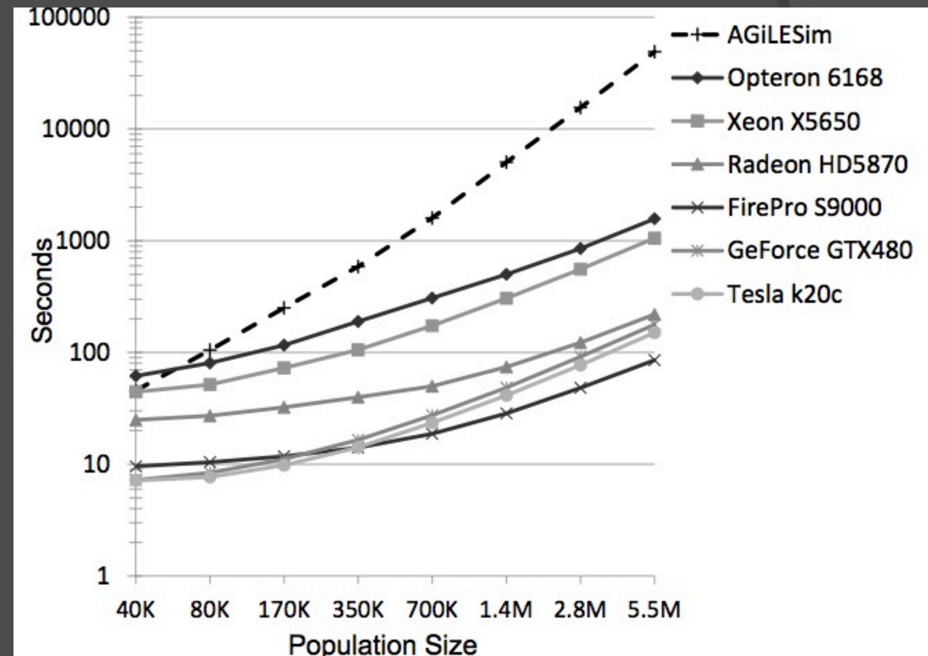


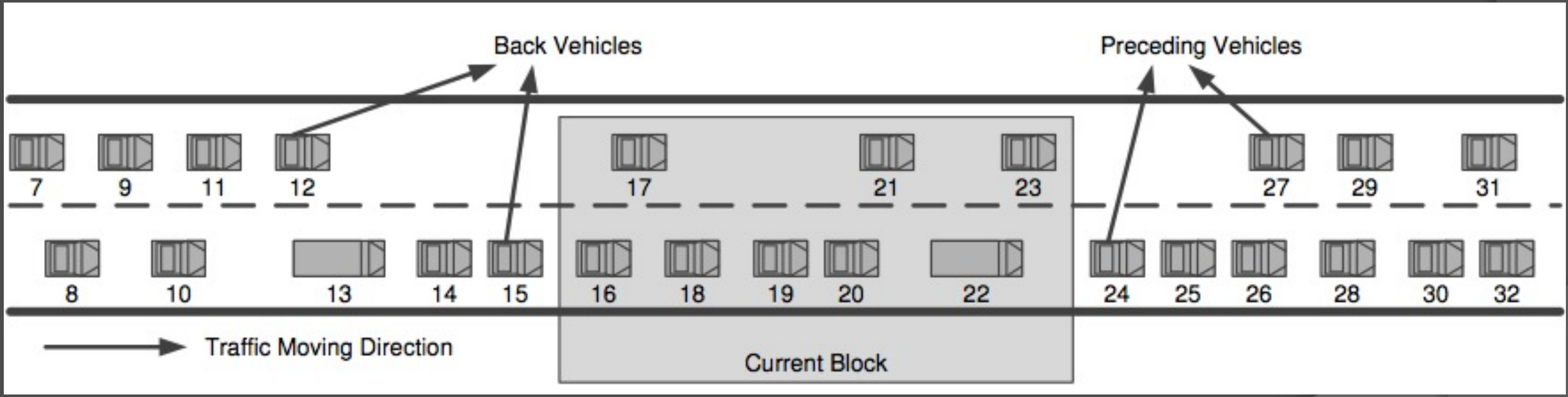
Figure 8. Execution time with varying population sizes for the Java implementation used in AGiLESim [21] and our OpenCL implementation on various processors. The execution times of AGiLESim were measured on an Intel Xeon X5650.

Traffic Simulation (Wang et al.(2013))

- Heterogene Architektur
- AMD APU
- Vollständig in OpenCL umgesetzt

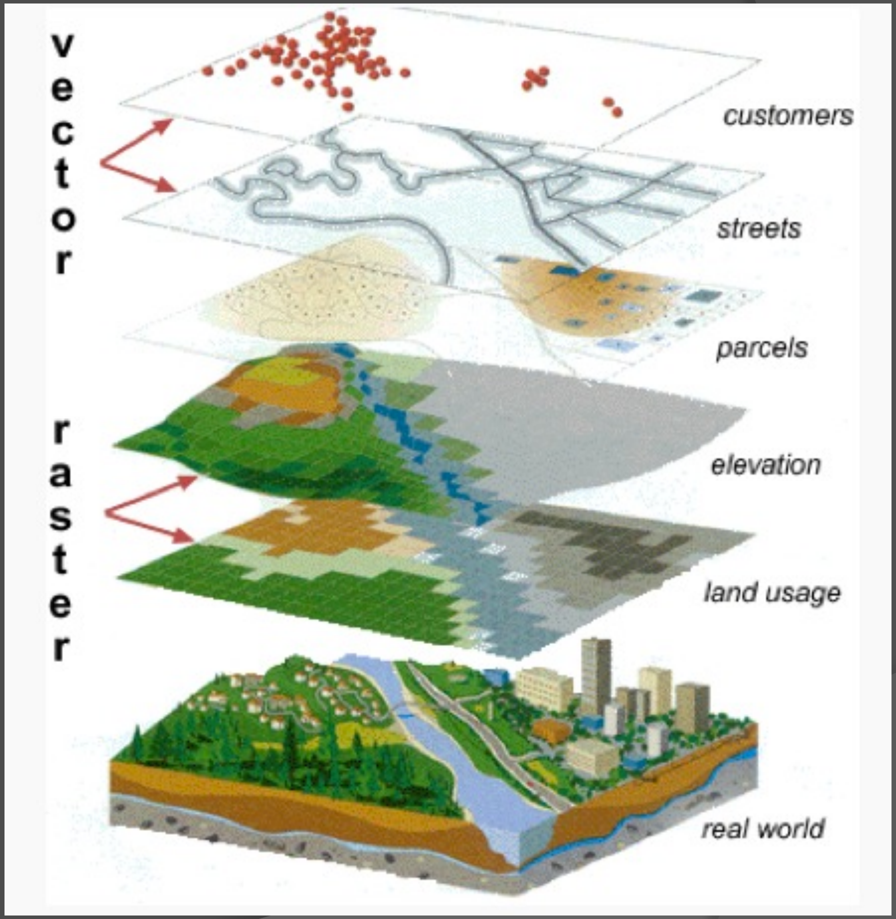
Traffic Simulation (Wang et al.(2013))

- Bilden von eigentständigen Clustern



MARS LIFE

- Layer Architektur
- Jeder Layer ein Agententyp
- Layer werden als Plugin eingefügt



Abgrenzung

- Entwickeln eines Layers mit OpenCL Agenten
  - Optimieren der Synchronisation
  - Aufteilen des Layers in Cluster

- Schnittstelle / Framework für nutzerfreundliche Entwicklung eines OpenCL Layers

Bildquellen

- Bild 1: http://www.nscc-tj.gov.cn/en/resources/resources_1.asp#TH-1A
- Bild 2: http://gamma.cs.unc.edu/CA/
- Bild 3: http://www.pcper.com/files/imagecache/article_max_width/review/2011-12-18/slide26.jpg
- Bild 4: http://www.guru3d.com/articles-pages/amd-radeon-hd-7970-review,5.html
- Bild 5,7,8: Scarpino, M. (2012). *OpenCL in action*. Shelter Island, NY: Manning.
- Bild 6: https://www.khronos.org/assets/uploads/developers/library/overview/opencl_overview.pdf
- Bild 9,10,11: Klaus Kofler, Gregory Davis, and Sandra Gesing. 2014. SAMPO: an agent-based mosquito point model in OpenCL. In *Proceedings of the 2014 Symposium on Agent Directed Simulation* (ADS '14). Society for Computer Simulation International, San Diego, CA, USA, , Article 5 , 10 pages.
- Bild 12: Jin Wang, Norman Rubin, Haicheng Wu, and Sudhakar Yalamanchili. 2013. Accelerating simulation of agent-based models on heterogeneous architectures. In *Proceedings of the 6th Workshop on General Purpose Processor Using Graphics Processing Units* (GPGPU-6), John Cavazos, Xiang Gong, and David Kaeli (Eds.). ACM, New York, NY, USA, 108-119.
- Bild 13: http://www.seos-project.eu/modules/agriculture/agriculture-c03-s01.de.html

Quellen

- Moser, D., Riener, A., Zia, K., & Ferscha, A. (2011). Comparing Parallel Simulation of Social Agents Using Cilk and OpenCL (pp. 88–97). Presented at the DS-RT '11: Proceedings of the 2011 IEEE/ACM 15th International Symposium on Distributed Simulation and Real Time Applications, IEEE Computer Society. doi:10.1109/DS-RT.2011.12
- Scarpino, M. (2012). *OpenCL in action*. Shelter Island, NY: Manning.
- Wang, J., Rubin, N., Wu, H., & Yalamanchili, S. (2013). Accelerating simulation of agent-based models on heterogeneous architectures (pp. 108–119). Presented at the GPGPU-6: Proceedings of the 6th Workshop on General Purpose Processor Using Graphics Processing Units, New York, New York, USA: ACM Request Permissions. doi:10.1145/2458523.2458534
- Laville, G., Mazouzi, K., Lang, C., Philipppe, L., & Marilleau, N. (2013). Using GPU for Multi-Agent Soil Simulation. *2013 21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2013)*, 392–399. doi:10.1109/PDP.2013.63
- Kofler, K., Davis, G., & Gesing, S. (2014). SAMPO: an agent-based mosquito point model in OpenCL. Presented at the ADS '14: Proceedings of the 2014 Symposium on Agent Directed Simulation, Society for Computer Simulation International.
- Hüning, C. (2013). *Konzeption und Entwurf einer Architektur zum Einsatz von Multi-Agenten-Simulationen in der ökologischen Systemmodellierung.* B.Sc. Hamburg University of Applied Sciences. Germany
- J. Ferber, *Multi-Agent Systems. An Introduction to Distributed Artificial Intelligence.* Addison Wesley, 1999.
- Richmond, P., Walker, D., Coakley, S., & Romano, D. (2010). High performance cellular level agent-based simulation with FLAME for the GPU. *Briefings in Bioinformatics*, *11*(3), 334–347. doi:10.1093/bib/bbp073
- Sano, Y., & Fukuta, N. (2013). A GPU-based Framework for Large-scale Multi-Agent Traffic Simulations (pp. 262–267). Presented at the Advanced Applied Informatics (IIAIAAI), 2013 IIAI International Conference on, IEEE. doi:10.1109/IIAI-AAI.2013.75