

Comparing Parallel Simulation of Social Agents using Cilk and OpenCL

Dominik Moser and Andreas Riener and Kashif Zia and Alois Ferscha

Johannes Kepler University Linz, Institute for Pervasive Computing, A-4040 Linz/Austria

Tel. +43/732/2468-1432, Fax. +43/732/2468-8426

{lastname}@pervasive.jku.at

Abstract—Recent advances in wireless/mobile communication and body worn sensors, together with ambient intelligence and seamless integrated pervasive technology have paved the way for applications operating based on social signals, i. e., sensing and processing of group behavior, interpersonal relationships, or emotions. Thinking in large, it should be apparent that modeling social systems allowing to study crowd behavior emerging from individual entities’ (agents’) condition and/or characteristics is, in fact, a challenging task. To address the heterogeneity, analytical agent-based models (ABMs) are gaining popularity due to its capability of directly representing individual entities and their interactions; unfortunately, ABMs (in which each agent has unique behavior) are not very well suited for large populations, expressed by exponentially rising simulation time. To solve this problem, the questions (i) how does the parallel execution of such models scale with capabilities of both the machine (number of cores, cluster size, etc.) and agents (behavioral adaptation function, interaction extent, etc.) and (ii) what is, in comparison, the performance coefficient applying the approach of model execution on graphical processors (GPUs) with its different pipelining architecture, need answers. To this end, we have performed simulation runs with parameter variation on a real parallel and distributed hardware platform using Cilk as well as on a GPU employing OpenCL. Simulation efficiency for two realistic models with varying complexity on a scale of 10^7 agents has shown the usefulness of both approaches.

Keywords—Parallel distributed simulation; Shared memory architecture; GPU execution; Multi-core cluster; Agent-based modeling; GPGPU

I. MODELING PERVASIVE SOCIAL SYSTEMS

One of the consequences of the success of pervasive computing is the introduction of computing applications which are based on social sciences. According to [1], pervasive communication technology together with sensor technologies is on its way to fundamentally change social networking in local communities. However, sensing and interaction with the environment does not only involve infrastructure elements such as digital signs, interactive walls or smart floors, but, to apply user-adaptive or context-aware behavior, also the users themselves. Since in many cases a user (“agent”) is more than a digital device or entity, e. g., a human being, the inclusion of social behavior into pervasive applications is increasingly gaining importance. The collective paradigm, derived from pervasive computing, social media, social networking, social signal processing, etc., has recently become known as “pervasive social computing” [2].

Recent developments within body worn sensors and ambient intelligence provide new possibilities to contribute to sensing of physical as well as cognitive attributes of human being, and moreover to integrate these into pervasive applications serving a smart environment [3]. The wearable systems to sense the physical characteristics such as presence, location, locomotion, and body postures are already well developed by using accelerometers, gyroscopes, compasses and positioning/orientation sensors. The new generation of wearable systems which could –for the first time– measure the cognitive aspects (e. g., tension, happiness, excitement, etc.) [4] is gaining popularity. Examples of these sensors are EOG, EEG, and ECG sensors as well as galvanic skin response (GSR) sensors or pupil diameter variation sensing. Similarly, the developments in ambient sensors to recognize physical as well as cognitive aspects has progressed well beyond the video and audio streams analysis, and has entered into implicit interaction paradigms [5].

In many of these smart environments the ultimate beneficiary are humans incorporated as social individuals. A prerequisite for the successful application of personalized services (allowing contextualization on single person granularity) is efficient, safe, and unobtrusive user identification and profiling. In addition to a profile which defines a user, the context also involves social relationships which can be woven deep down into a profile (e. g., family members, office colleagues, relationship status, friends) or can be formulated on the fly (e. g., passengers traveling in same train carriage, fans visiting a soccer game, or drivers stucking in a traffic jam). Overlying, the social relation of a person is the social behavior composed of individual preferences and the collective situation. For example, if a user is getting out of a railway station in a hurry and pushing hard within a crowd, he/she may be getting late for work or there is an unusual situation, e. g., a mass panic. What complicates it further are interpersonal connections. For example, the same user may act differently if accompanied with his child and perhaps would require a different assistance from the system.

Realizing the importance of social pervasive applications, it is apparent that modeling of such systems is a challenging task due to the gap between social and technological domains, which may add exceptional complexity to the model. However, before unification of these domains (which is not the focus of this work), two questions need answers.

First, what kind of modeling technique should be used?, and second, how such a model should be examined? Modeling a social system starts with modeling representative individual entities constituting such a system. These entities are heterogeneous with varying character and capabilities. In a social system we cannot model these entities at variable (using structural equations) or system (using differential equations) level. As an analytical method for social systems, the agent-based modeling is rapidly gaining popularity, due to its capability of directly representing individual entities and their interactions [6]. Whereas experiments are the standard way of collecting evidence and analyzing a behavior, in most of the social systems, conducting experiment is impossible, undesirable, or dangerous. Therefore, it is necessary to perform simulations where behavior of individual entities is either extracted from small-scale evidence collected through a focused and less harmful experiment, or through some analogous model or even theoretical understanding. Further, simulating a social phenomena for a pervasive social system is essentially a complex process due to variety of behavior and the scale. In many cases the number of entities involved are quite large (e. g., crowd dynamics). Hence, the complexity of the system increases exponentially with an increase in the number of entities (humans, devices) in the system.

In this paper, we have designed and simulated a framework of large scale social agents (with almost full social/dynamic behavior), with simulation focusing on issues involved in social behavior. We have provided an abstract view of essential social ingredients required to perform agent-based social system simulation. This framework does not resolve any of the PDS specific issues, however the simulation is performed on (i) a real parallel and distributed hardware platform (*multiprocessor system with global shared memory*) using Cilk as well as on (ii) a GPU (*single graphical processor with optimized pipelining architecture*) employing the rarely used framework OpenCL. In this way it is possible to compare the simulation efficiency of popular hardware/software platforms running a simplified, but realistic agent-based social system simulation.

A. Related work

[7] has focused on the problem of latency hiding in the parallelization of ABM simulations. Preliminary results have been shown for multi-core clusters (*pthreads*) and multi-GPU systems (CUDA). Nevertheless, they omitted ABM specific features, such as agent mobility or neighborhood adaptation in their models, which is a considerable restriction in model dynamicity. In [8] the problem of replication, in order to make (stochastic) simulations statistically significant, was addressed. The authors propose to compute the individual simulation jobs in parallel in order to reduce simulation runtime. The distribution of jobs over several computers is controlled by a grid-inspired approach. In [9], the replication mechanism is refined, allowing to adapt itself to a

given problem in order to find the best configuration used in the replicated simulations. The replication approach may be useful when using many-/multi-core CPU or cluster systems, but with availability of powerful frameworks for utilizing GPUs its advantage is invalidated, as GPUs are suited much better for massive parallelization and simultaneous execution of many jobs (as shown in the work at hand). Recently, [10] used a CA approach to simulate ABMs on a GPU. The authors showed promising performance gains, but concluded that additional research is needed to address data parallel issues for more generalized ABM execution; preventing artificial bias is one of the most important factors that needs to be considered when parallelizing ABM. [11] has achieved a speed-up of 72 using a CUDA model compared to single CPU execution. Richmond *et al.* [12] went a step forward by presenting FLAME GPU, an extension to the FLAME framework providing a flexible agent-based architecture entirely on the GPU and showed a direct performance increase of 250 between models executed in the two frameworks. [13] reported speed improvements with a single GPU (CUDA) of around 200 and concluded that even further gains should be possible with code optimization. [14] has implemented “SugarScape”, a framework with rich and complete support for simulation on GPUs, and showed that a GPU can be used to simulate large scale ABMs efficiently. [15] have used an extension of CUDA to study agent navigation. They have found spatial hashing as being extremely beneficial, leading to almost linear scale of running time with increased number of agents. Restrictively, simulation results were shown on a relative low number of 20,000 agents only.

In contrast to our approach, most of the recent simulation papers are focusing on (i) the GPU based approach and (ii) using CUDA framework for implementing ABMs, e. g., [11], [15], [16], [13], [7]. This is, for the former, most-likely due to the fact that GPUs are available off-the-shelf (consumer graphic cards) [13], which implicates low costs, high availability, ease of access, etc. [14]. For the latter, the main reason is its object-oriented architecture and long history; nevertheless, OpenCL is a interesting alternative strongly expanding in industry. [13], for instance, explicitly recommends to switch to OpenCL – due to cross-platform availability and availability of powerful API’s. More important, most of the related work reviewed discusses only a single approach, simulation either on a multi-core system or on GPU(s) (with few exceptions, e. g., [7], [17], [18]); some uses parallel execution only for the (required) replication of simulation runs, and not for parallelizing the model itself.

Outline: The rest of the paper is organized as follows. Section II discusses agent-based modeling and how specifications of a social agent can be abstracted focusing on problems like heterogeneity, complexity of space, etc. arising in large scale systems. Section III gives an overview of available and suitable (software) technologies for high-performance simulation and concludes with a paragraph

describing the techniques utilized in this paper. Section IV gives detailed insight into development and execution of two models, a (coarse-grained) cluster behavior model used in preliminary studies, and a agent individual behavior model to derive more specific results, e. g., on potential performance, developed later. The final section V concludes the paper and gives some references to prospective work.

II. POTENTIAL OF ABMS IN SOCIAL SYSTEMS

Agent-based modeling, the simulated interaction of many agents most likely equipped with individual behavior (heterogeneity), is widely used in engineering and science to model large, complex dynamic systems. It has been, for example, successfully used in engineering for transport logistics [19] or production technology [20], and applied to the biology domain for modeling the effects of climate and ecosystem change [21]. Recent advances in both processing power (e. g., GPGPU, PDS) and developments in cognitive social modeling allows for close-to-reality simulation of (i) social phenomena like group formation, cultural transmission, combat, or trade emerging from the interaction of individual agents [22], (ii) neural mechanisms in the human brain [23], (iii) collective phenomena [24], and (iv) even for policy analysis [25]. The possibility for further model up-scales allows ABM in these days to explain the emergence of higher order patterns – movement dynamics in traffic jams, behavioral patterns in global social networks, and social segregation across populations, to name a few.

A. Social agents

Agent-based models (ABMs) provide appropriate features on agent level that could define a social entity. These features are, for instance, (i) autonomy: ability to make its own decisions without a central controller, (ii) social ability: ability to interact with other agents, (iii) reactivity: ability to react to a stimulus, and (iv) proactivity: ability to pursue its goal on its own initiative. Each agent in a system may have its own version of implementing these features. Additionally, an ABM allows multiple scales of social structures culminating naturally at a macro or societal level. None of other modeling approaches (for modeling a social system) comes as natural as the agent-based modeling approach. More formal, *agent-based modeling is a computational method that enables a researcher to create, analyze, and experiment with models composed of agents that interact within an environment. The challenge is usually not to limit the rationality of agents, but to extend their intelligence to the point where they could make decisions of the same sophistication as it is commonplace among real people [6].*

Some of the features of ABM making it an attractive choice for social modeling and simulation are:

- there can be **one-to-one correspondence** between real world actors and virtual agents which makes it easier

and natural to design an agent and also to interpret the simulation results,

- possible **heterogeneity** in agents behavior advocate the usage of ABM in social systems,
- possibility to **represent the space** in which agents are acting directly into the ABM which makes modeling easier in an integrated environment,
- using ABM, agents can **interact** with each other **at different granularities** thus introducing the core social building blocks of communication, grouping, etc.,
- ABM are furthermore able to **implement learning/adaptation** at local as well as on global scale,
- many models implicitly assume that the **individuals** whom they model are rational; Simon [26] criticized this and suggested that people should be **modeled as boundedly rational**, i. e., as **limited in their cognitive abilities** and thus in the degree to which they are able to optimize their utility [27]. ABM makes it easy to create such agents.

B. Simulating and modeling social agents at large scale

A large scale social system cannot be modeled in a traditional way – the difficulties arises due to at least three broad reasons, (i) agents’ heterogeneity, (ii) overlapping granularity of interacting entities, and (iii) complex space models. In the scientific computing (or computational science) community, a generic term for aspect (ii) is multi-scale modeling which can be narrowed down as *social organization in systems addressing social phenomena*.

In a large scale agent-based system, the individual agents are typically **heterogeneous** in nature. There is a variety of behavior for each agent in chemical, biological, economic or engineering systems; however, a social system addressing the cognitive aspects of participating entities (human beings) is much more complex. The following aspects highlight the challenges involved:

- **Individualism:** Each individual agent can be as physically and behaviorally different as the humans are,
- **Functional complexity, learning:** The process of social decision making is not a simple one as it may involve unlimited options to explore. Even a single decision may involve complex formulations. It is not practical to formulate a complete rule set before hand. The decision making rules evolve all over the time as an agent learns from previous decisions as well as from decisions taken by others; most decisions are not independent and depends on parameters from influencing entities,
- **Behavioral adaptivity:** This can be distinguished from behavioral learning as it targets the change in behavior due to dynamics of the other entities in interaction whereas learning describes the cause-affect relation which changes the rule-based with experience.

The **concept of space** has two meanings, ontological and physical. In the first, space is taken as room or corridor connecting two rooms. The conceptual definition of space in this way guarantees a more convenient behavioral analysis granularity focusing on a conceptual basis of analysis rather than unnecessary physical details of a space (like coordinates, etc.). In the second, space is described in a physical domain which is necessary for agent's behavioral implementation at agent level. Many observational evidence can only relate behavior of an individual on ontological level. For example an interviewee can only relate his experiences to a contextual space, or the observations reported by an ambient device may report a flow of people through an exit. This makes it difficult to extract the physical space from contextual space. Additionally a true representation of space in modeling also derives its complexity from complexity of the environment itself. Most real environments cannot easily be represented in a true-to-scale and representative "digital clone".

Considering that an agent's specification includes the spatial aspects of the environment, the decision making of an agent in a social system is absolutely dependent on group size (scale). Within a scale it is not necessary that all the agents would or should be communicating with each other as agent's individualism describes the desire or capabilities to communicate. Based on the influences from the group and agent's own expectations and experiences, an agent can perform an action or adapt its behavior. This simple specification of a social agent can intuitively be derived from ABM specifications discussed above, and is simulated using kind of artificial workload.

III. HIGH PERFORMANCE SIMULATION TECHNOLOGIES

Agent-based simulation has shown its potential as tool for small- to medium-sized problems [28]. A timely simulation of socio-technical systems on large scale (saying 10^6 – 10^7 agents) is, however, not possible using a self-contained environment on a single workstation – as it by far exceeds its (processing) capabilities. Accordingly, a system is required that scales with growing problem size. The objective is to develop and support methods for scalable simulation of ABMs which considers the specifics of agent technology. For distributed/parallel computing, generally two approaches are followed today, (i) multithreaded parallel computing using similar CPU cores and (ii) execution on heterogeneous platforms consisting of CPUs, GPUs, and other processors.

Compute unified device architecture (CUDA) is the parallel computing architecture developed by NVIDIA as the way to use GPUs for general purpose parallel processing (GPGPU). (Unlike CPUs however, GPUs have a parallel throughput architecture that emphasizes executing many concurrent threads slowly, rather than executing a single thread very quickly.) The (meanwhile) standardized open computing language (OpenCL) is the more hardware independent approach utilizable on CPUs and GPUs [29].

On the other side, Cilk [30] has been developed as an extension of the C language. The two keywords "*spawn*" and "*sync*" are all that are needed to start using the parallel features of Cilk. The biggest design principle of Cilk is, that the programmer is responsible for exposing the parallelism and identifying elements that can safely be executed in parallel. The run-time environment, particularly the scheduler, decides during execution how to actually divide the work between processors.

Comparison: Cilk and OpenCL/CUDA seem to be emerging as two important, rather distinct approaches to parallel programming. In some ways, Cilk and OpenCL can be seen as lower-level languages, where the programmer has to worry about identifying particularly places where threads should be spawned (in Cilk), or worry about extracting kernels from code that operates on an array or stream (in OpenCL).

Cilk makes it easy to spawn threads off to compute an arbitrary expression, and then uses a set of *worker processes* to service the threads, where threads spawned by a given worker are serviced LIFO, but when a worker runs out of threads, it steals from other workers using a FIFO strategy. OpenCL, on the other hand, is focused on data parallelism, where many data items are all being manipulated in nearly the same way. It is roughly a SIMD approach, though some variation in control flow is permitted, meaning that it is somewhat closer to the SPMD approach. In both OpenCL and CUDA the per-data-item code is bundled into a kernel, which is essentially a function that operates on a single data item in a large array/stream of such items¹.

For the simulation studies explained below, we have implemented models for both Cilk language and OpenCL framework. We performed simulation runs with parameter variation for *coarse-grained* (cluster behavior) and *fine-grained* (individual agent behavior) models to study influence of parameters like cluster size (agent group size) and connectivity (rate of communication in a group) on agent-based behavior.

To reduce the model complexity two restrictions have been taken for both the coarse- and the fine-grained model. First, each social agent is able to interact only with other agents that are in the same cluster; second, each cluster is static with respect to its size (and potential of reconfiguration), which implicates that agents can not move to another cluster during the simulation. This scenario is not very realistic, but is very suitable for benchmarking analysis. A more realistic implementation, allowing intra- and inter-cluster communication as well as dynamically reorganization of clusters, will be presented in the future, after understanding basic differences in the code generation and limitations/performance disparities of the two approaches.

¹Tuck, "Designing ParaSail, a new programming language", URL: <http://parasail-programming-language.blogspot.com/2011/01/cuda-opencl-cilk-and-parasail.html>, last retrieved June 8, 2011.

IV. SIMULATION OF BEHAVIORAL MODELS

A. Abstraction from individual behavior (coarse-grained)

The cluster behavior model is designed to provide an abstract view of a high performance parallel agent-based model (a cluster is here used synonymously to a group of agents). This high level of abstraction provides the possibility to investigate the runtime behavior and performance of the model in a very abstract manner, i.e., to analyze model behavior with respect to the quantity of agents, their properties, as well as the influence of cluster dimensions and communication behavior on model performance. A cluster in the here used manner has two major parameters to steer its behavior (and the behavior of agents assigned it):

- **Cluster size:** is the number of agents abstracted into a single cluster; in the basic (coarse-grained) model, the chosen cluster size applies to all clusters in the model,
- **Connectivity:** is the communication rate of agents within a cluster. It has to be noted that, in general, each agent in the cluster has the ability to directly interact/communicate with all the other agents in the cluster.

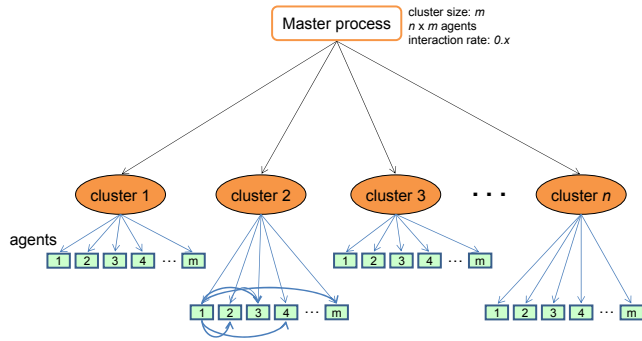


Figure 1. Structure of the cluster behavior model with its clusters, agents assigned to clusters, and communication between agents within a cluster.

Clustering agents provides the potential to let a specified subgroup of agents defined in the (entire) model communicate with other agents in the model (see Figure 1). This is assumed to represent basic behavior of people in real social systems, as for instance only part of the people in a waiting area (=model) can (directly) communicate with other people in that room/area (=cluster). A more technical example would be the interaction or communication of mobile devices (phones) using Bluetooth technology; all the devices in the network (e.g., 256 in a piconet) are represented as a cluster in our model. As, under normal circumstances, not all of the people in a waiting area are communicating with each other, and only some devices in a certain network have the ability to communicate with other nodes (in a Bluetooth piconet, for example, only eight devices can be active at the same time), the cluster behavior model also needs a parameter to express this limited

connectivity. This parameter allows to control the rate of interaction (“communication pathways”) within a cluster in the range $[0, 1]$ (0...communication free environment, 1 full communication, i.e., each agent is communicating with any other agent). A more detailed specification of communication, e.g., which agent is interacting with which other agent, is not provided for the coarse-grained model. Beside communication, real agents (people, (A)MI devices) furthermore exhibit individual behavior and movement. In the cluster behavior model, however, both agent behavior and movement models are merged into a single function generating artificial, hypothetical workload (see Figure 2) for each and every agent. This abstraction is sufficient in the first step to study the impact of group size and/or interaction rate on the general performance of large scale, parallel simulation. In the simulation experiments discussed below, the model was executed on a scale of 10^6 and 10^7 agents.

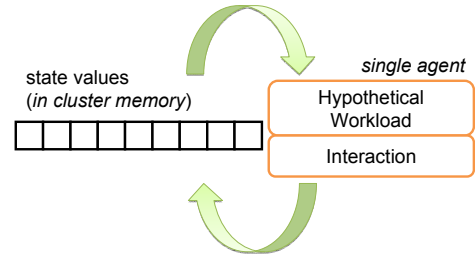


Figure 2. In the coarse-grained (cluster behavior) model, the “hypothetical workload” generalizes the workload generated by movement, cognitive decisions, etc. that real, complex models would generate.

Implementation: The cluster behavior model was implemented in both Cilk [30] and OpenCL [29] language. Both frameworks fully support the models properties as introduced above, therefore simulation results should be directly comparable.

The implementation in Cilk, supposed to be executed on multi-core shared-memory machines, follows the architecture as highlighted in Figure 1. A single master process generates for each an every cluster in the model a new thread; each of this cluster threads is able to access the global memory. Each cluster itself obtains (“loads”) all required information from the global memory (number of assigned agents, interaction rate, etc.), and spawns for each and every agent in the model a new agent thread. Agent threads are representing single agents (with no individual behavior but hypothetic workload instead of), and have only limited memory access (a thread can access its local memory as well as the common cluster memory, to which all of the agents in the cluster have access to).

A second instance of the model was developed using the OpenCL framework. This framework takes advantage of the massive processing power available on the latest generation of graphic cards (graphical processing unit, GPU). Nevertheless, before the optimized structure of the GPU is usable

for executing models like that described before, a more or less extensive code redesign/reimplementation is necessary (due to special hardware restrictions on graphics cards). The final implementation in OpenCL follows the structure as described for the Cilk case, with the main thread (master process) executed on the CPU of the host computer. One additional step is necessary here, transferring parameters and data to the GPU. In the next step, the master process executes all agent threads simultaneously, assigning them to their corresponding clusters. Agent threads have access to both the global memory and the local cluster memory. Upon completion of a simulation run, the results have to be transferred back to the host system to make them accessible for the master thread to process it further.

For the Cilk implementation, the synchronization strategy used is stepwise in the manner that, within a cluster, agent code for each agent (representing its behavior) is executed independent, and a synchronization mechanism at the end of each iteration locks finished threads until completion of each and every remaining thread within the cluster. For the given model behavior, inter-cluster synchronization is not necessary, thus not secured.

For the OpenCL implementation, a flat hierarchy of clusters and agents is used, meaning that clusters are only logical units established for comparison reasons only, and all the agents exists on a single tier. Synchronization is here not performed on cluster granularity, but each and every agent is synchronized with other agents as described above for the cluster level. This sync. strategy is far from optimum and will be replaced later with a more realistic policy, but is, for now, well suited to perform performance analysis.

Model execution: During simulation, each and every agent can be identified with its unique ID. Data intended for a certain agent is not directly exchanged between master process and individual agent, but is, due to performance reasons, handed over via the cluster memory and under control of the cluster thread (see Figure 3). (Once a new agent is instantiated, it synchronizes itself with the cluster memory).

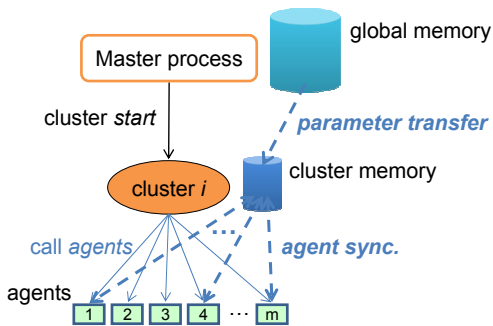


Figure 3. Model execution and synchronization of agents in the cluster behavioral model.

After that, agents are performing some tasks, i. e., decision making, movement, etc. (the coarse-grained model performs for this only hypothetical work) and are interacting with other agents. Interaction stands for exchange of data between two agents, again not in a direct way but via the shared cluster memory (each of the agents in a cluster has its state variables there; see Figure 2). In order to avoid invalidated data/states after synchronization of an agents internal variables with its state variables, locking strategies have to be employed. In the given case, agents are equipped with kind of mailbox communication behavior. A sending agent, intending to distribute information via the shared memory to other agents, has to wait until the memory region is unlocked (meaning that all the receiver agents have read the content at the recent synchronization point), while the receiver agents are not blocked in their operation; quite the contrary, they are receiving, during processing, state variables from the write protected shared (cluster) memory. This mechanism protects from causality violations and repeatability problems, which would invalidate simulation results.

Simulation results: Both instances of the cluster behavior model were executed on their designated infrastructure with similar parametrization on a scale of 10^6 and 10^7 agents. In order to study the impact of cluster size and connectivity on the model performance, both parameters were varied in a rather large range, cluster size was varied continuously from 2^1 to 2^5 agents while the communication rate was increased from 0.0 to 1.0 (full communication of agents within a cluster) in steps of 0.2. Below, the results of the executed simulation runs are given, exemplarily for connectivity 1.0 (or 100%), as acceleration in comparison to the single core execution.

cores	10E6 agents		10E7 agents	
	time [sec.]	acceleration	time [sec.]	acceleration
1	125	0	1252,3	0
10	17,56	7	172,3	7
25	14,05	9	135,4	9
50	3,66	34	35,02	36
100	14,54	9	102,4	12
GPU	0,233	536	2,33	537

Table I
ACHIEVED ACCELERATION RATES FOR THE CLUSTER BEHAVIOR MODEL (VARYING CLUSTER SIZE, FIXED CONNECTIVITY OF 100%). (ROWS 1-5: CILK CODE, ROW 6: OPENCL CODE.)

As shown in Table I, the achieved acceleration is, at least for the OpenCL model executed on a single GPU, considerable high. It seems that the hardware architecture of the GPU (cache size, etc.) perfectly fits the structure of the cluster behavior model; in comparison to a model execution on a single CPU we achieved acceleration rates of more than 500. The Cilk implementation executed on a cluster machine, with different numbers of cores utilized, provides different results. While the acceleration is quite

high for some of the tabulated cases (7 for 10 cores, 34/36 for 50 cores) it drops significantly for others (9 for 25 cores, 9/12 for 100 cores). This strange result has its origin in the utilization of the cluster machine. As we had no exclusive access to it, other computationally intensive calculations lead to this heavy drop in acceleration. With exclusive machine access we should have obtained acceleration rates of ≈ 20 for 25 cores and 70 for 100 cores. Nevertheless, achievable acceleration rates for the shared memory machine are even in the case of hundred available nodes (cores) very low compared to model execution on a single GPU. What furthermore can be derived from Table I is the fact that the achievable acceleration rate is independent from the model size (number of agents) (*acceleration* columns for the cases 10^6 , 10^7 agents).

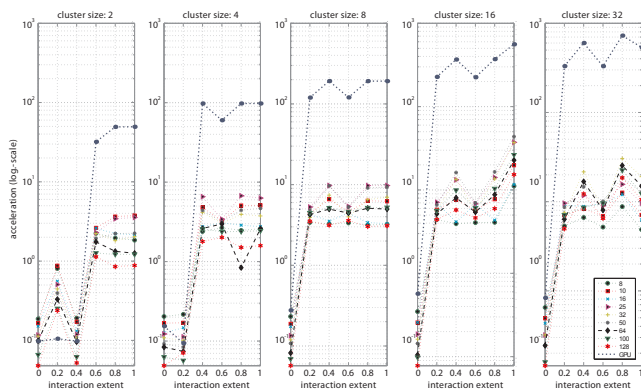


Figure 4. Simulation results for 10^7 agents on a shared memory architecture (8 to 128 cores) in comparison to single GPU execution. The lower the interaction between agents, the lesser the achievable acceleration (indicated by the reversed spike at communication rate “0”).

Figure 4 gives a visual representation of simulation results. Five subdiagrams are shown, corresponding to a variation of the cluster size (i.e., the number of agents assigned to a cluster) from 2 (leftmost) to 32 (rightmost diagram). In each subdiagram, the x-axis corresponds to the connectivity of agents (i.e., communication pathways in a cluster), while the y-axis represents, on a logarithmic scale, the performance increase in comparison to single CPU execution (=reference value 1.0).

Dashed lines (different colors/markers; see legend) represents achieved performance on execution of Cilk code on the shared memory machine using varying number of cores (from 8 to 128) compared to execution of OpenCL code on a single GPU. In more detail, it can be clearly seen that the cluster size has a direct impact on the load of the master process, as this process has to spawn *total agents/cluster size* cluster threads (e.g., $5 * 10^6$ cluster threads in case of cluster size 2 and only 312,500 threads in case of cluster size 32). Furthermore, the larger the cluster size, the higher the

performance/acceleration for both parallel execution using Cilk code and single execution on a GPU using the OpenCL implementation. The parameter connectivity shows a special behavior if set to zero (agents are operating isolated; no communication at all). In this case, it is almost impossible to achieve enhanced performance with increasing number of cores. Figure 4 indicates this effect as minimum values at communication rate 0 (reversed spikes), independent from the number of cores or the size of clusters. For connectivity > 0 we see a steady rising acceleration, driven by higher number of cores and larger cluster size.

It has to be noted here again, that in the cases of higher cores used on the parallel machine (> 50), simulation results are in most instances distorted as we had no exclusive access, and the cores used in our simulation experiments where, at the same time, partly used for other computations (which lessens the available processing power on the one or other core). Scalability shown on 8 to 50 cores should therefore continue also on higher number of CPU’s used (64, 100, 128, etc.), different to the results currently shown in the figure(s).

Simulation runs with the cluster behavior model have shown to what extent the model is affected by parameter variation (cluster size, connectivity). Furthermore, results have shown that the (fixed) size of the clusters has a wide influence on the maximum performance achievable with a certain model. The reason for this is that quantity of agents and cluster size directly affects the number of clusters to be spawned by the master process. This issue will be solved in the future, as clusters will be then dynamically reorganized in order to optimally cover model properties. The parameter connectivity, if greater than zero, has almost no influence on the overall simulation performance as the model only deals with *hypothetical workload* with high idle times for communication attempts. Furthermore, as long as there is no need for the application of extensive data locking strategies, e.g., for information exchange between an agents local memory and the shared state variables in the cluster memory (which would be definitely the case in “real” interaction), the performance increase scales almost linear with the number of cores. The most promising result was derived from a comparison of Cilk and OpenCL implementations, which emphasizes the superiority of the GPU approach in relation to a multi-core parallel computer, and justifies its usage for future simulation studies.

B. Fine-grained model: Individual agent behavior

The next step towards a more general and realistic social model is to replace the “hypothetical workload” component with functions reflecting real behavior. Therefore, we have extended the cluster behavior model (Figure 2) with specific functions for agent movement and adaptation (Figure 5), all the other model characteristics (e.g., cluster parameters) are inherited from the coarse-grained model. The potential to easily apply changes in the behavior of agents, as adopted

here, guarantees a high degree of versatility also for future developments. In the following, we refer to the model depicted in Figure 5 as the individual agent behavior (or fine-grained) model.

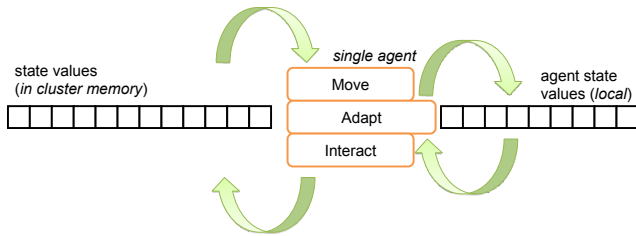


Figure 5. Individual agent behavior: The basic novel structure of a single agent with the methods/actions move, adapt, and interact.

Implementation: The cluster size gives (as in the cluster behavior model) the quantity of agents within a cluster in the range $[1, N]$. It is used to group certain agents with same functionality (cognitive behavior), communication abilities (face-to-face or technological assisted), etc. The extent of communication within a cluster is specified with the parameter connectivity (or communication rate) in the range $[0, 1]$. Figure 6 illustrates extreme cases full communication (1.0), communication free or agents are left on its own (0.0), and the general case 0.5 (uni-/bidirectional interaction between some agents).

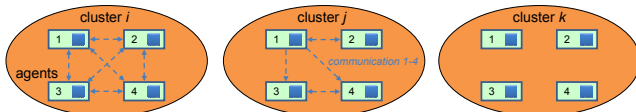


Figure 6. Varying connectivity (1.0, 0.5, and 0.0; from left to right) for a cluster size of 4 (agents).

Based on the cluster behavior model, the fine-grained model specifies concrete methods for agent movement and adaptation. These two functions replaces the “hypothetic workload” component of the coarse-grained model, leading, in the end, to a more realistic agent behavior. The interact(ion) method, already existent in the underlying cluster behavior model, was taken over almost unchanged. Briefly, the behavioral functionality of the individual agent model can be described as follows:

- **Move:** Represents individual movement functionality of an agent,
- **Adapt:** Subsumes all self-adapting methods of an agent,
- **Interact:** Models communication (information exchange) with other agents.

All this three conceptual functions encapsulates, in object-oriented manner, more or less complex underlying behavior functionality; they can –according to the current situation–

contain nested loops or call further methods. To give a concrete example, an agent always calls the top-level “move”-method, regardless the movement capabilities of a certain agent. The “move” method actually called can be either *empty* if the corresponding agent is firmly anchored (e. g., a exit sign) or can contain different commands depending on whether the agent is able to walk, run or jump. The “adapt” function abstracts the self-adaptation functionalities of an agent, i. e. trust or panic behavior; summarized, “adapt” calls all the commands changing the (local) state of an agent. Furthermore, methods that are influenced from changes in the neighborhood, caused either by the agent itself or evoked from the environment, are supposed to be defined in the “adapt” method. Changes in the state of an agent may trigger, during execution of the next “interact(ion)” function, data communication between agent local memory and cluster memory. In reality, data may be transferred using different transmission channels, such as, for instance, a wireless network (WiFi), Bluetooth connection or verbal communication using spoken instructions. The structure of the individual agent behavior model as depicted in Figure 5 is held very general, and can be easily modified to develop and test new functionalities. For example, the model is capable to represent human agents as well as device agents just by checking the agent type in the agent description and registering different methods within the main function.

Model execution: In general, the fine-grained model is executed the same way as the cluster behavior model, but, due to extensions in the model, some modifications were made. The major changes were applied to the thread structure of agents and to the data management component.

In the shared memory model (Cilk), the master process spawns for each and every cluster a new thread. Each of the threads takes the state values of all assigned agents and transfers the data into its (local) cluster memory. After this synchronization task, all cluster threads spawns for each and every assigned agent an agent thread and assigns a unique ID to an agent. This identification number is henceforth used within a cluster for agent communication and data access.

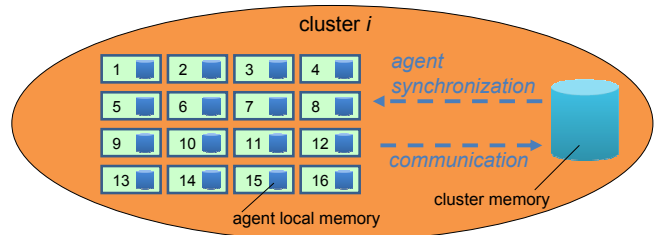


Figure 7. Single cluster representation in both Cilk and OpenCL.

While the functionality provided by the OpenCL implementation is equal to the Cilk model described just now, it uses a slightly different thread hierarchy applicable to single GPU execution. The master process in the OpenCL

implementation transfers at first all data from global memory to the GPU. Subsequently, all agent threads are started by the master process at the same time, assigning them automatically to clusters. GPU memory content is then copied to the cluster memory using a special coalesced reading pattern (see Figure 8, (A) to (C)). Both Cilk and OpenCL implementations are equal in terms of functionality and are operating, from this stage on, in the same way.

At first, agents are synchronizing their local memory with the cluster memory. In the next step, they are executing their “move” method and are moving, based on local state values, to another position. Afterwards, “adapt” recomputes, based on environmental changes caused by the movement of all agents in a cluster, an agents local state values. Finally, “interact(ion)” initiates communication of agent.

Simulation results: Performance analysis for the individual behavior model was executed with the same parameterization as for the cluster behavior model (10^6 and 10^7 agents, cluster size 2^1 to 2^5 , connectivity 0.0 to 1.0 in steps of 0.2, 8-128 cores, single GPU). Simulation results (see Figure 9, detailed results are given in table II) are, in general, similar to the results obtained with the coarse-grained model, with lower possible acceleration caused by the higher complexity of “move”, “adapt” methods compared to the “hypothetical workload” (lower fraction of arithmetic operations compared to memory access).

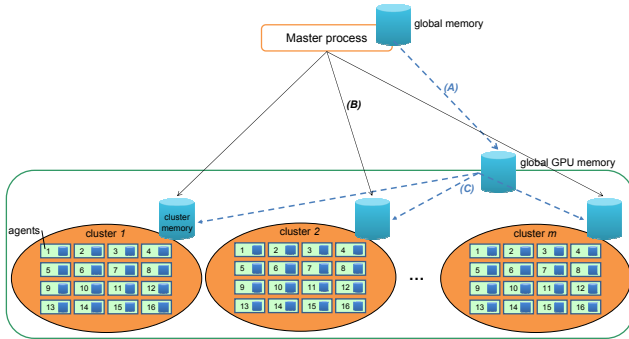


Figure 8. Generalized execution structure for shared memory and GPU models. OpenCL: (A) data is transferred from global memory to global GPU memory, (B) all agents are triggered/“started” simultaneously by the master process, (C) transfer data from global memory to cluster memories; for the Cilk impl., (A) and (C) are fused into a single step, copying data from global memory to local memories of each cluster, (B) clusters are “started” which themselves generating/starting the assigned agents.

The individual behavior (or fine-grained) model has shown that the performance of the simulation is highly dependent on the behavioral functions “move”, “adapt” and “interact(ion)”. Therefore, it is one of the major interest for future developments to optimize these methods (for a given problem).

While for the Cilk model, accelerations achieved are in the same order of magnitude, they are much lower for

cores	10E6 agents		10E7 agents	
	time [sec.]	acceleration	time [sec.]	acceleration
1	822,86	0	8216,3	0
10	84,77	10	848,84	10
25	35,27	23	351,58	23
50	13,38	61	186,2	44
100	13,07	63	212,27	39
GPU	4,675	176	53,0822	155

Table II
ACCELERATIONS FOR THE FINE-GRAINED MODEL (CONNECTIVITY 1.0).

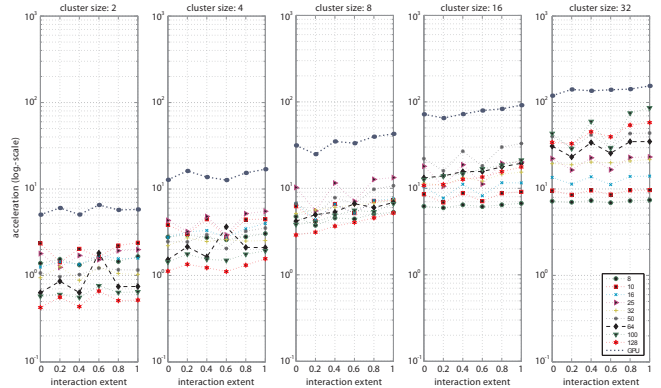


Figure 9. Simulation results for Cilk code executed on the shared memory architecture with 8 to 128 cores in contrast to OpenCL models on a single GPU. The general setting was 10^7 agents with cluster size varying from 2 to 32 and interaction rate between 0.0 and 1.0.

the OpenCL implementation executed on a single GPU. Acceleration collapse (727 for the cluster behavior model, 155 for the individual behavior model) was mainly caused by the complex access to the GPU global memory. Most effort should be therefore spent in optimizing the usage of this memory channel (bottleneck!). As a matter of fact, however, the organization of GPU memory access is very complex and obviously the fine-grained model “does not fit” to this structure as good as the coarse-grained model shown before. Nevertheless, model execution on a (single) GPU can reach, by far, higher performance in comparison with the Cilk implementation executed on a 100+ CPU parallel machine. (GPU architecture allows to execute several million threads simultaneously and exclusively, while the Cilk environment uses a scheduler to execute the cluster and agent threads, which takes (lot of) time).

V. CONCLUSIONS

In order to solve the exponentially rising simulation time for large scale ABMs, we have in this paper shown that it is possible to gain high improvements in acceleration (compared to single CPU execution) when running agent-based social models in parallel on either a distributed shared memory machine or a GPU. The simulation efficiency for two realistic models with varying complexity on a scale of 10^7 agents has proven the usefulness of both PDS and GPU

approaches. Furthermore, it became evident that the GPU approach is, with acceleration improvements of up to 727 for the cluster behavior model (155 for the individual behavior model), superior to the PDS model, even in case of high number of (100) cores used, reaching, e. g., for the individual agent behavior model, only a acceleration factor of 86.

The approach of latency hiding in ABM simulation [7] seems to be promising and may also be applied to our OpenCL model to resolve, in future, the identified problems on acceleration collapse with increasing memory access.

A further issue to solve in future is to release the limitations in terms of model complexity (agents assigned to a particular cluster, fixed cluster size), so to allow more realistic simulation models with inter-cluster communication and dynamic cluster reorganization.

Acknowledgements

This work is supported under the FP7 ICT Future Enabling Technologies program of the European Commission under grant agreement No 231288 (SOCIONICAL).

REFERENCES

- [1] H. Tirri, "Pervasive Technology that Changed the World," May 2010, keynote at the 8th International Conference On Pervasive Computing.
- [2] J. Zhou, J. Sun, K. Athukorala, and D. Wijekoon, "Pervasive Social Computing: Augmenting Five Facets of Human Intelligence," in *7th Int. Conference on Autonomic Trusted Computing (ATC)*, 2010, p. 6.
- [3] H. Nakashima, H. Aghajan, and J. C. Augusto, Eds., *Handbook of Ambient Intelligence and Smart Environments*. Springer, 2010, ISBN: 978-0-387-93807-3.
- [4] R. Matthews, N. McDonald, P. Hervieux, P. Turner, and M. Steindorf, "A wearable physiological sensor suite for unobtrusive monitoring of physiological and cognitive state," in *Int. Conference of Engineering in Medicine and Biology Society (EMBS 2007)*, 2007, pp. 5276–5281.
- [5] A. Riener, *Continuous Authentication based on Biometrics: Data, Models, and Metrics*. IGI Global, 2012, ch. 7: Sitting Postures & Electrocardiograms: A Method for Continuous and Unobtrusive Driver Authentication, p. 30, ISBN: 978-1-61350-129-0., in press.
- [6] G. N. Gilbert, *Agent-based models*. Sage Publications, Inc, 2008.
- [7] B. G. Aaby, K. S. Perumalla, and S. K. Seal, "Efficient simulation of agent-based models on multi-gpu and multi-core clusters," in *Proceedings of the 3rd Int. ICST Conference on Simulation Tools and Techniques (SIMUTools '10)*, 2010, pp. 29:1–29:10.
- [8] S. Leye, J. Himmelspach, M. Jeschke, R. Ewald, and A. M. Uhrmacher, "A grid-inspired mechanism for coarse-grained experiment execution," in *Proceedings of the IEEE/ACM Int. Symposium on Distributed Simulation and Real-Time Applications (DS-RT '08)*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 7–16.
- [9] R. Ewald, S. Leye, and A. M. Uhrmacher, "An Efficient and Adaptive Mechanism for Parallel Simulation Replication," in *Proceedings of the 23rd Workshop on Principles of Advanced and Distributed Simulation (PADS '09)*. Washington, DC, USA: IEEE CS, 2009, pp. 104–113.
- [10] K. S. Perumalla and B. G. Aaby, "Data parallel execution challenges and runtime performance of agent simulations on gpus," in *Proceedings of the 2008 Spring simulation multiconference (SpringSim '08)*. San Diego, CA, USA: SCSi, 2008, pp. 116–123.
- [11] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, and K. Skadron, "A performance study of general-purpose applications on graphics processors using cuda," *J. Parallel Distrib. Comput.*, vol. 68, pp. 1370–1380, October 2008.
- [12] P. Richmond, S. Coakley, and D. Romano, "Cellular Level Agent Based Modelling on the Graphics Processing Unit," in *High Performance Computational Systems Biology, 2009. HIBI '09. International Workshop on*, oct. 2009, pp. 43–50.
- [13] E. M. Aldrich, J. Fernandez-Villaverde, A. R. Gallant, and J. F. Rubio-Ramirez, "Tapping the supercomputer under your desk: Solving dynamic equilibrium models with graphics processors," *J. of Economic Dynamics and Control*, vol. 35, no. 3, pp. 386–393, 2011.
- [14] R. D'Souza, M. Lysenko, and K. Rahmani, "Sugarscape on steroids: simulating over a million agents at interactive rates," in *Proceedings of Agent2007 conference, Chicago, IL.*, 2007, p. 7.
- [15] A. Bleiweiss, "Multi Agent Navigation on the GPU," *GDC09 Game Developers Conference 2009*, 2008.
- [16] P. Richmond and D. Romano, "Template-driven agent-based modeling and simulation with CUDA," in *GPU Computing Gems*. Boston: Morgan Kaufmann, 2011, pp. 313–324.
- [17] M. Yuffe, E. Knoll, M. Mehalel, J. Shor, and T. Kurts, "A fully integrated multi-cpu, gpu and memory controller 32nm processor," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2011 IEEE International*, feb. 2011, pp. 264–266.
- [18] S. Rybacki, J. Himmelspach, and A. M. Uhrmacher, "Experiments with Single Core, Multi-core, and GPU Based Computation of Cellular Automata," in *Proceedings of the International Conference on Advances in System Simulation*. Washington, DC, USA: IEEE CS, 2009, pp. 62–67.
- [19] P. Davidsson, L. Henesey, L. Ramstedt, J. Trnquist, and F. Wernstedt, "Agent-based approaches to transport logistics," in *Applications of Agent Technology in Traffic and Transportation*. Birkhäuser Basel, 2005, pp. 1–15.
- [20] J. Holmgren, "Multi-agent-based simulation and optimization of production and transportation," Dissertation thesis, Department of Systems and Software Engineering, School of Engineering, Blekinge Institute of Technology, Sweden, 2008, ISBN: 978-91-7295-140-2.
- [21] A. Patt, R. J. Klein, and A. de la Vega-Leinert, "Taking the uncertainty in climate-change vulnerability assessment seriously," *Comptes Rendus Geosciences*, vol. 337, no. 4, pp. 411–424, 2005.
- [22] J. M. Epstein and R. Axtell, *Growing Artificial Societies – Social Science From the Bottom Up*. Brookings Institution Press and MIT Press, 1996, ISBN: 978-0-262-05053-1.
- [23] M. Hoogendoorn, J. Treur, C. van der Wal, and A. van Wissen, "Agent-Based Modelling of the Emergence of Collective States Based on Contagion of Individual States in Groups," *Transactions on Computational Collective Intelligence*, vol. 3, pp. 142–179, 2011.
- [24] A. Sharpanskykh and J. Treur, "Abstraction relations between internal and behavioural agent models for collective decision making," in *Computational Collective Intelligence. Technologies and Applications*, ser. LNCS. Springer Berlin, Heidelberg, 2010, vol. 6421, pp. 39–53.
- [25] Y. Kim, "Enriching policy analysis: The role of agent-based models," in *The 9th Public Management Research Conference*, October 2007.
- [26] H. A. Simon, "A behavioral model of rational choice," *Quart. J. Econom.*, vol. 69, pp. 99–118, 1955.
- [27] D. Kahneman, "Maps of bounded rationality: Psychology for behavioral economics," *The American Economic Review*, vol. 93, pp. 1449–1475, 2003.
- [28] K. Zia, A. Ferscha, A. Riener, M. Wirz, D. Roggen, K. Kloch, and P. Lukowicz, "Scenario based modeling for very large scale simulations," in *14th Int. IEEE Symposium on Distributed Simulation and Real Time Applications (DS-RT 2010)*, Oct. 2010, p. 8.
- [29] Khronos group, "OpenCL – The open standard for parallel programming of heterogeneous systems," [online], 2011, last retrieved June 8, 2011. <http://www.khronos.org/opencl/>.
- [30] M. Davidson, "The Cilk Project," [online], October 8, 2010, last retrieved June 8, 2011. <http://supertech.csail.mit.edu/cilk/>.