International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume6(2014) pp.323-332
© MIR Labs, www.mirlabs.net/ijcisim/index.html

CUDA code support in Multiagent platform JADE

Lukáš Zaorálek[1] **and Petr Gajdoš**[2]

[1]VŠB - Technical University of Ostrava, 17.listopadu 15
Ostrava, 708 33, Czech Republic
lukas.zaoralek@vsb.cz

[2]VŠB - Technical University of Ostrava, 17.listopadu 15
Ostrava, 708 33, Czech Republic
petr.gajdos@vsb.cz

Abstract: **This paper describes a new feature of the multi-agent framework that enables the execution of GPU kernels. This brings a novel implementation of the JADE CUDA Agent that accepts PTX/CUDA-C code and runs this code on GPUs. All such agents can receive PTX code or CUDA source code via ACL messages. Beyond the integration of CUDA support, this paper is also focused on the performance measurement of the proposed solution and the evaluation of final results.**

Keywords: CUDA, PTX code, JADE, Multi-Agent System, ACL messages

I. Introduction

There are several approaches dedicated to parallelism on different levels, such as threads, processes, and distribute parallel computing. One of the most important challenges consists in migration parallel tasks across heterogeneous platforms, such that the number of platforms is variable in time. A solution for this problem can be seen in the intersection of the Multi-agent System and standard GPU programming. The Multi-agent system JADE was chosen for this purpose and GPU platform with nVidia CUDA. JADE is a multiplatform framework written in Java language and its JADE API [1] is easy to use. The main goal is to implement a JADE CUDA Agent that accepts PTX/CUDA-C code and runs this code on GPUs. The concept of such solution can be described as follows:

1. The JADE CUDA Agent accepts PTX/C code from another agent (a sender)

2. It compiles the PTX/C code into a binary code with respect to a given GPU

3. It executes the binary code on the CUDA platform

The CUDA Agent can send a result of the process back to the sender after the previously described steps are completed. The following text describes the idea in more detail. Next several performance tests were done. The performance of the running code was measured, and the overall time of sending the code and receiving results back to the sender agent were measured as well. The main advantage of the CUDA Agent lies in a platform-independent GPU computing with emphasis on solving computational and time consuming tasks in a multi-agent environment.

II. Multi Agent Systems (MAS)

Multi Agent Systems will be introduced in this section. Before presenting a general definition of a MAS, basic units of such systems will be introduced first.

A. Agents

An agent represents an essential part of all MASs. There are several definitions of an agent. Two important and commonly used definitions are as follows:

Definition 1 *An agent is a programme process that implements own autonomy and communication capability. [2] Agents communicate by the usage of a communication language Agent Communication Language (ACL). [3] An agent has to have at least one owner, i.e. one organization or user, and has to have a defined identity Agent Identifier (AID) that is unique. This identity helps the user to exactly identify an agent in the whole system. [4] [5]. Listing 1 shows the ACL message structure.*

Listing 1: The ACL message structure

```
(request
  :sender (agent-identifier
                  :name alice@mydomain.com)
  :receiver (agent-identifier
                  :name bob@yourdomain.com)
  :ontology travel-assistant
  :language FIPA-SL
  :protocol fipa-request
  :content
  ""((action
  (agent-identifier
                  :name bob@yourdomain.com)
  (book-hotel :arrival 15/10/2006
  :departure 05/07/2002 ... )
  ))""
)
```

Definition 2 *The term agent describes a software abstraction, an idea, or a concept, similar to Object Oriented Programming terms such as methods, functions, and objects. [6] The concept of an agent provides a convenient and powerful way to describe a complex software entity that is capable of acting with a certain degree of autonomy in order to accomplish tasks on behalf of its user. But unlike objects, that are defined in terms of methods and attributes, an agent is defined in terms of its behavior. [7]*

As mentioned above, there are many other definitions of an agent. However, we can find some common points:

- An agent is situated in a specific environment and can to accept or respond to the outer stimuli.
- An agent can work autonomously.
- It can move, clone itself or remove itself from the environment.
- It can communicate with other agents.

One can also meet another notion, namely that of an Intelligent Agent. Such agent has a few additional features that extend its skills:

- The ability to work in a quickly changing environment [8].
- Flexibility; agents can respond to the current situation and they can cope with changes.

So an agent is a computer programme that is situated in some environment and it can work autonomously. [9] However, autonomy is a very difficult notion. In the context of MAS it means that an agent can work independently of other, and if a particular agent malfunctions the whole system does not collapse. [10]

Now the basic types of agents can be described.
A *Reactive agent* is the simplest type of agent. Such an agent can receive some request and then perform a specific action only. The agent does not keep the whole history (what was happened in the system) and it cannot anticipate future actions. In particular, the behavior of such an agent can be programmed by a set of rules [11]. On the other hand, it has several advantages:

1. The system is very fast.
2. The system is simple.
3. The behavior of all agents is predictable.

Most of the current multi agent systems use this kind of agent. Their implementation is very easy, and they can be represented by finite automata or they can be defined by a PTX/C code as in our case. [12] However, such systems have many disadvantages too. In practice, we cannot represent each agent by a set of rules in full. Especially in the case of a real world simulation, it seems to be an impossible task. The agents cannot anticipate their influence on the whole state of system [11] [13].

A Deliberative agent is based on the techniques of Artificial Intelligence and Expert systems [14]. The agent can make a plan to reach its own objective, and it can make a decision according to the stock of knowledge it has. The problem consists in the speed of the decision making process. [15] Usually it is very difficult to create an agent that can make a decision in real time. The Artificial Intelligence (AI) process of deduction is usually very time-consuming [11]. However, such agents can respond to unpredictable situations, make plans and cope with changes.

A Hybrid agent is basically a combination of the two previous types of agents. It merges the advantages of both. A reactive subsystem handles all the time-consuming operations. The planning and reasoning are in the hands of the deliberative subsystem. The problem consists in determining the border between reactive and deliberative parts of such an agent. [16] [17]

The design of a multi agent system which is exclusively reactive or proactive is an easy task. However, when attempting at a combination of both and their proper balance, we meet problems.[18]

Agents can achieve their objectives systematically. We do not want agents to blindly perform a sequence of procedures or functions. We need agents to react dynamically and not try to do something that is already impossible. [19]

The last aspect of agents' intelligence consists in cooperation, as was mentioned in the work of Zambonelli [20]. It is not only data exchange. It is a common term that includes communication, negotiation and cooperation. The whole problem of agents' cooperation is the most challenging task, and it has still not been solved in a satisfactory way.

B. Multi Agent System

The skills of intelligent agents are limited by their knowledge, computer resources and surrounding behavior; agents are resource-bound. A single agent is usually not able to perform a complex real-world process. We need a couple of agents to simulate such processes. [21] Groups of cooperative agents make up a system called *Multi Agent System.* [22] [23]

MAS is dynamic; its components are not known in advanced and can be created or removed from the system. In the case of MAS, we usually speak about distributed systems. It means that agents can exist within different software and hardware platforms, and communicate through a communication protocol. [24]

MAS is a hot topic of current research. Because these agents are apt for application in many areas. Here is a list of some advantages of MAS:

- MAS provide the solution for problems that are too extensive and time-consuming for classic realized system.
- There is a possibility for connection and cooperation between several systems.
- They work with distributed information, e.g. sensor monitoring.
- The agents can be mobile within a system. They can interrupt their activity, move into another place within the current system, and then continue the work in a new locality.

The environment of the real world is inaccessible, non-deterministic, dynamic and continuous. An agent can never have complete knowledge of the whole environment (inaccessible), results of particular actions are not foreseeable, and the actions can fail (non-deterministic). The environment is changing in time independently of the agents (dynamic) and it has an infinite set of states (continuous). [25] [26]

For obvious reasons, modeling such an environment is extremely difficult. That is why it is necessary to simplify the environment for the needs of modeling and simulations. [27]

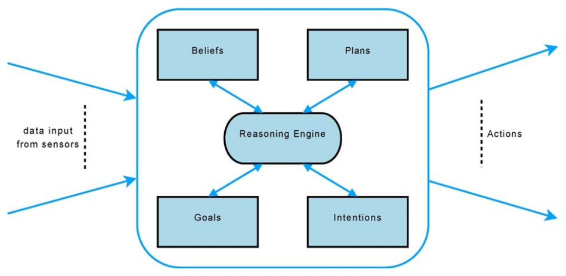

Figure. 1: BDI model

Thera are exist several architectures within a multiagent system:

- *Logic-based (symbolic)* architectures draw their foundation from traditional knowledge-based systems techniques in which an environment is symbolically represented and manipulated using reasoning mechanisms.

- *Reactive* architectures implement decision-making as a direct mapping of a situation to action and are based on a stimulus-response mechanism triggered by sensor data. Unlike logic-based architectures, they do not have any central symbolic model, and therefore do not utilize any complex symbolic reasoning.

- *BDI (Belief, desire, intention)* architecture describes beliefs as the representation of the agents knowledge about the current world/environment and messages from other agents, as well as internal information. Beliefs represent the information an agent has about its environment, which may be incomplete or incorrect. Desires represent the tasks allocated to the agent, and so correspond to the objectives or goals it should accomplish. Intentions represent desires that the agent has committed to achieving. Finally, plans specify some courses of action that may be followed by an agent in order to achieve its intentions. Figure 1 illustrates the BDI model scheme.

An ontology in the multi agent system means a set of vocabulary that an agent has understood. Consequently, the agent has certainly described the domain of a problem based on this set of vocabulary and shares it with each other. In other words, the ontology is a central pivot of the communication act in the multi agent system, and gives us a tool how to communicate with agents. The main idea of the ontology is way how agents can share knowledge, describe the domain of a problem and exchange tasks and ask or answer each other. [28]

A programmer can use the basic ontology defined by the FIPA organisation, or implement his/her own ontology within the JADE platform. Moreover the programmer has implemented its own ontology that is based on the FIPA onotology. In other words, ontology can be a hierarchical structure to share the agent knowledge, as we can see in Figure 3. The FIPA ontology standard has defined three basic kinds of ontologies. There are top-level ontologies, domain ontologies and task and application ontologies in the standard. The top-level ontologies contain several very basic general concepts such as space, time, matter, object, event, action. The domain ontologies and task define a vocabulary that describe some specific domain of a problem. Figure 2 describe the basic concept of the ontology defined by the FIPA standards.

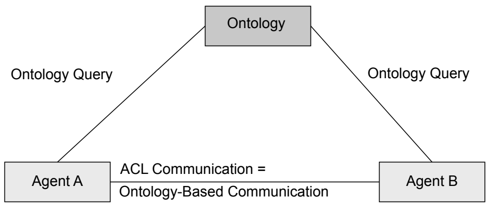

Figure. 2: The FIPA ontology

A performative is defined as the type of communication act between agents. In other words, an agent has to know which type of message is received in order to provide a correct reaction to the received message. There are several basic performatives that FIPA has defined: Confirm, Disconfirm, Failure, Inform, Request, Request When, Request Whenever, Not understood, Query If, Query ref. Confirm means a sender agent believes that the subject of a communicative act is true where the receiver agent is uncertain about the subject. Disconfirm means a sender agent believes that the subject of a communicative act is false where the receiver agent is uncertain about the subject. The subject of the communicative act was not completed by a sending agent. Inform means a sender agent informs that the subject of the communicative act is true. Request means the sender agent wants to perform some action. Request when performative means the sender agent is asking the receiver agent if a subject is true or not. If the agent wants to perform some action whenever a subject is true, then a request is sent whenever it is performative. At the end, not understood means either the sender agent did not understand the received message from the other agent, or the sender agent informs that it did not understand the performed act (the receiver agent performed an act which was not understood to the sender agent).

Jade ontology represents a java class. There are several base classes such as a Predicate, Concept, Primitive, Aggregate or AgentAction that can extend our ontology class in the JADE framework. In other words, each of our ontology classes must be extended from a base ontology class that we can see in figure 3. The following list describes basic ontology classes in the JADE framework:

- The predicates are expression that can be true or false and describe the status of the world.

- The Term is a generic entity that agents can talk.

- Concepts are entities that have complex structures.

- The Agent action is an action that can be performed by agents.

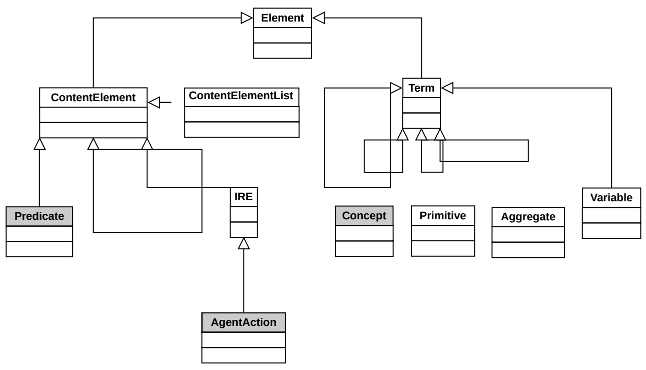

Figure. 3: The base JADE ontology classes

- The Primitive contains a primitive value such as integer, float, long, short, boolean or byte.
- The Aggregate holds several elements together (similar to list).
- The Variable represents a variable expression that is not known yet.

III. GPU and CUDA

Architecture of GPUs (Graphics Processing Units) is suitable for vector and matrix algebra operations. That leads to the wide usage of GPUs in the area of information retrieval, data mining, image processing, data compression, etc. [29]. There are two graphics hardware vendors: ATI and nVIDIA. ATI develops technology called ATI Stream, and nVIDIA presents nVIDIA CUDA. Nowadays, programmers usually choose between OpenCL which is supported by ATI and nVIDIA [30], and CUDA which is only supported by nVIDIA [29]. An important benefit of OpenCL is its platform independence; however, CUDA still sets the trends in GPU programming. This article is not focused on a detail comparison of these two approaches; we utilize CUDA in our experiments.

CUDA (Compute Unified Device Architecture) is a general purpose parallel computing architecture. GPUs utilized in our experiments are based on the Fermi architecture [31], which is still one of the most common GPU architectures since the original G80. Currently, new architecture called Kepler has been introduced by nVIDIA.

GPUs of the Fermi architecture include a number of Streaming Multiprocessors (SM) with 32 cores, e.g. nVIDIA Tesla 2050 provides 14 SM with 448 CUDA cores. [32] A CUDA program calls parallel kernels. A kernel executes in parallel across a set of parallel threads. The programmer or compiler organizes these threads in the thread blocks and grids of the thread blocks. Each thread within a thread block executes an instance of the kernel. [33] The GPU instantiates a kernel program on a grid of parallel thread blocks. The simplified arrangement of threads is illustrated in Figure 5. A thread block is a set of concurrently executing threads that can cooperate among themselves through a barrier synchronization and shared memory. [34] A grid is an array of thread blocks that execute the same kernel, read inputs from global memory, write results to global memory, and synchronize

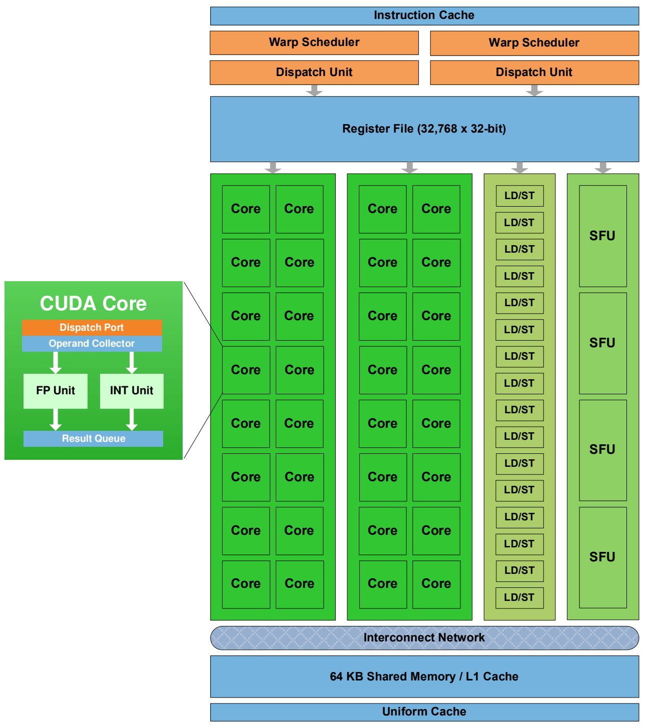

Figure. 4: Single Fermi Streaming Multiprocessor (SM) [31]

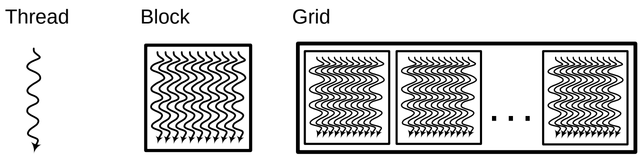

Figure. 5: A schema of the CUDA threads arrangement [31]

kernel calls. For more detail we refer to [31].

There are several new important features like GigaThread, concurrent kernel execution and Nvidia Parallel DataCache in Fermi architecture. The GigaThread is Fermi thread scheduler (second generation), which dramatically improves the speed of context switching, thread throughput, and thread block scheduling. Another significant feature of the Fermi architecture is concurrent kernel execution that allows applications to run a number of small kernels to utilize the whole GPU. [35] [36] In other words, on the Fermi architecture, different kernels of the same CUDA context can execute concurrently, allowing maximum utilization of GPU resources. The Fermi architecture also has improved configurable shared memory, Warp scheduler, and Streaming Multiprocessor (SM). Scheme of the Streaming Multiprocessor is illustrated in Figure 4. The scheme includes Special Function Unit (SFU), load/store units, shared memory, and several cores. Each Streaming Multiprocessor has 16 load/store units allowing source and destination addresses to be calculated for sixteen threads per clock. The Special Function Unit designed for efficient calculation of math operations such as sin, cosine, reciprocal, and square root. [37]

The main advantage of CUDA technology consists in the power of the different architecture of graphics processing units. There are a number of tasks that were solved on GPU rather than CPU such as Fourier transform and convolution, matrix multiplication, neural network or data mining algorithms etc. We refer to [29] for more information.

IV. Amdahl's Law

For the prediction of the theoretical maximum speed up of a parallel application we used Amdahl's Law.[38] Amdahl's Law is a model for the relationship between the expected speedup of the parallelized implementations of an algorithm relative to the serial algorithm. [39] The maximum speedup S is defined as follows:

$$S = \frac{T_{old}}{a_{new}} = \frac{r_s + r_p}{r_s + \frac{r_p}{n}} = \frac{1}{r_s + \frac{r_p}{n}} \qquad (1)$$

where r_s is the amount of time that the program spends in parts that can by run sequential only, r_p is the amount of time that the program spends in parts that can be parallelized. Let r_s and r_p be normalized such that $r_s + r_p = 1$. The art is to find for the same problem an algorithm that has a large r_p. Algorithms for which r_p=1 are called "embarrassingly parallel". [40]

V. CudaAgent

CudaAgent is an agent that handles CUDA kernel source code (hereinafter only kernel) in the form of C or PTX (Parallel Thread Execution) code and runs it on the GPU. [41] The CudaAgent has its own ontology which is described below in section V-B, and it has its own implementation of inner behavior. In other words, CudaAgent offers to another agent its own service to compile and run the kernel. Such kernel must be in the form of C or PTX source code. The example of PTX code can be seen in listing 3.

Figure 6 illustrates the process of acceptance of the kernel by CudaAgent. [42] If the kernel is in the form of C source code then the CudaAgent tries to compile it. [43] If the kernel is received and possible compilation of the kernel source code is successful, then the kernel is integrated into the inner execution queue of the agent with respect to priorities. The CudaAgent sends the ACL message as a result after the kernel is processed. The result can contain data or some message information on the kernel failure/success. Note that a part of kernel transfers also contains also running parameters, such as data types and values. A parameter should be a primitive type such as float, integer, double and/or other. Moreover, the parameter can be an array of these types.

The sender agent expects an ACL message, which should contain kernel execution result. Otherwise, the sender agent could receive an ACL message informing the sender agent about kernel process failure, e.g. kernel transformation failure, compilation failure, kernel run failure, etc. Each kernel is sent to the CudaAgent through an ACL Message. The kernel code (including input parameters) is converted into byte array before being sent. The kernel code is converted back when it is received by the CudaAgent.

Listing 2: An example of the C cuda kernel

```
// Loop over all the sub-matrices of A and B
// required to compute the block sub-matrix
for (int a = aBegin, b = bBegin;
         a <= aEnd;
         a += aStep, b += bStep) {

    // Declaration of the shared memory array As used
         to
    // store the sub-matrix of A
    __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];

    // Declaration of the shared memory array Bs used
         to
    // store the sub-matrix of B
    __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];

    // Load the matrices from device memory
    // to shared memory; each thread loads
    // one element of each matrix
    AS(ty, tx) = A[a + wA * ty + tx];
    BS(ty, tx) = B[b + wB * ty + tx];

    // Synchronize to make sure the matrices are
         loaded
    __syncthreads();

    // Multiply the two matrices together;
    // each thread computes one element
    // of the block sub-matrix
    for (int k = 0; k < BLOCK_SIZE; ++k)
        Csub += AS(ty, k) * BS(k, tx);

    // Synchronize to make sure that the preceding
    // computation is done before loading two new
    // sub-matrices of A and B in the next iteration
    __syncthreads();
}

// Write the block sub-matrix to device memory;
// each thread writes one element
int c = wB * BLOCK_SIZE * by + BLOCK_SIZE * bx;
C[c + wB * ty + tx] = Csub;
```

Listing 3: An example of the compiled PTX kernel

```
        mul.wide.s32    %rl10, %r71, 64;
        mov.u64         %rl11,
             __cuda_local_var_15874_39_non_const_As;
        add.s64         %rl6, %rl11, %rl10;
        mul.wide.s32    %rl12, %r70, 4;
        add.s64         %rl4, %rl6, %rl12;
        ld.param.u32    %r69, [matrixMul_param_5];
        .loc 2 96 1
        mad.lo.s32      %r10, %r71, %r69, %r70;
        mov.u64         %rl13,
             __cuda_local_var_15878_39_non_const_Bs;
        add.s64         %rl14, %rl13, %rl10;
        add.s64         %rl5, %rl14, %rl12;
        add.s64         %rl7, %rl13, %rl12;
        mov.f32         %f55, 0f00000000;

BB0_2:
        .loc 2 95 1
        add.s32         %r19, %r9, %r75;
        mul.wide.s32    %rl15, %r19, 4;
        add.s64         %rl16, %rl3, %rl15;
        ld.global.f32   %f5, [%rl16];
        st.shared.f32   [%rl4], %f5;
        .loc 2 96 1
        add.s32         %r22, %r10, %r74;
        mul.wide.s32    %rl17, %r22, 4;
        add.s64         %rl18, %rl2, %rl17;
        ld.global.f32   %f6, [%rl18];
        st.shared.f32   [%rl5], %f6;
        .loc 2 99 1
        bar.sync        0;
        .loc 2 105 1
        ld.shared.f32   %f7, [%rl7];
        ld.shared.f32   %f8, [%rl6];
        fma.rn.f32      %f9, %f8, %f7, %f55;
        ld.shared.f32   %f10, [%rl7+64];
        ld.shared.f32   %f11, [%rl6+4];
        fma.rn.f32      %f12, %f11, %f10, %f9;
        ld.shared.f32   %f13, [%rl7+128];
        ld.shared.f32   %f14, [%rl6+8];
        fma.rn.f32      %f15, %f14, %f13, %f12;
        ld.shared.f32   %f16, [%rl7+192];
        ld.shared.f32   %f17, [%rl6+12];
```

A. CudaAgent Implementation

CudaAgent implementation can be divided into three main parts:

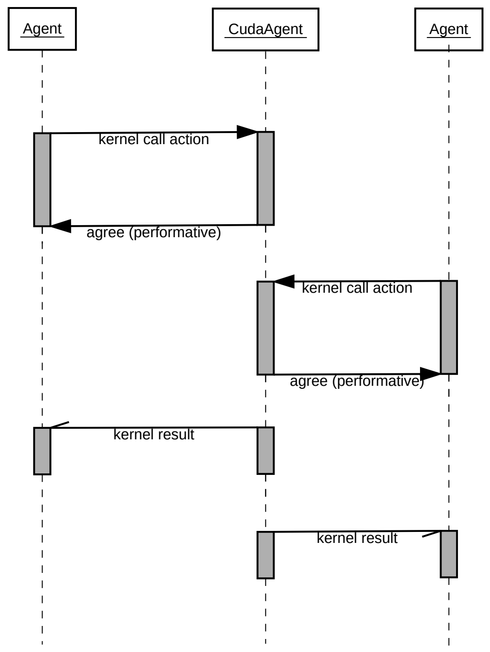

Figure. 6: The cuda kernel transfer

1. Handling ACL messages
2. Kernel management
3. CudaAgent behaving

The first part is represented by several java classes that manage the handling and processing of ACL messages: MessageParser and MessageFactory, then Kernel, KernelParameter and KernelBase as part of Kernel management. [44] An initial CudaAgent behavior is implemented as a java class ReceiveMessage and KernelCallManager. There are also other classes, see table 1. [45]

B. Ontology and Performatives

The important part of a CudaAgent is its own ontology and performative definition. [46] [47] There are several ontology schemes: KernelParameter, KernelBase, KernelCallAction, KernelRegisterAction, KernelUnregisterAction, CudaAgentDescription and CudaResult.

An agent needs to register some kernel by the CudaAgent when the agent wants to process it. There are two ways the agent can make the registration. First, the agent sends an ACL message as a kernel register action (an instance of the KernelRegisterAction class). After that, the agent will be informed about successful kernel registration and will also get a kernel ID in this ACL message. After receiving the kernel ID, the agent can send a request with the kernel ID for the kernel process as a kernel call action (an instance of a KernelCallAction class). The second way is to send an ACL message directly as a kernel call action that contains the whole kernel code with appropriate kernel parameters. [48]

The CudaAgent supports other actions such as KernelUnregisterAction and GetCudaAgentDescription. The KernelUnregisterAction is dedicated to unregister CUDA kernels. It has to contain a kernel ID. This is part of the memory

Class name	**Description**
CopyDirection	Define if a parameter is input or output.
CudaAgent	The main class of the CudaAgent project.
CudaAgentOntology	CudaAgent ontology include schemas for a class KernelParameter, a KernelBase and kernel actions.
CudaAgentVocabulary	Interface as vocabulery list used in the CudaAgent ontology.
CudaResult	CudaResult has used as an answer.
CudaResultParser	Helper able to parsing result.
dim3w	Wrapper for a JCuda class dim3 because the class dim3 is not serializable.
GetCudaAgentDescription	An action for short description about the CudaAgent such as number of GPU, performance and other.
KernelBase	A concept for the CUDA kernel.
KernelCallAction	An action dedicated to running a CUDA kernel.
KernelCallManager	A behaviour class dedicated to maintaining a CUDA kernel (running kernel, copying in/out data).
Kernel	Helper for easier manipulating a CUDA kernel. This class extends the class KernelBase.
KernelParameter	A concept for the kernel parameter.
KernelPriorityComparator	The class implements a Comparator interface because of kernel comparation.
KernelRegisterAction	An action dedicated to registering kernel by the CudaAgent.
KernelUnregisterAction	An action dedicated to unregistering kernel by the CudaAgent.
MessageFactory	The class implements Factory pattern to create ACL message such as reply.
MessageParser	The MessageParser is able to parse ACL message specific for the CudaAgent.
ReceiveMessage	A behaviour class dedicated to receiving ACL message.

Table 1: The CudaAgent classes

management system of CudaAgents. If the agent wants to know some basic information on the CudaAgent, e.g. the number of devices or its GFlop/s, then the agent sends to the CudaAgent the ACL message as the GetCudaAgentDescription. [49]

The CudaAgent supports these performatives: request, inform, unknown, not understood, agree and failure. Every agent must understand these performatives if it wants to communicate with the CudaAgent. [50]

C. Compilation and Data Transfer

The list of behaviors of CudaAgents also contains compileKernel, prepareKernel, runKernel and finalizeKernel methods. Table 2 describes the mentioned methods. There are also implemented behaviors like KernelCallManager and ReceiveMessage. These behaviours are responsible for the management of the kernel and receiving ACL messages respectively. Algorithm 1 describes the method responsible for calling kernels on the GPU.

Method name	**Description**
prepareKernel	The method initializes kernel, copies necessary input parameters on the global memory space of GPU.
runKernel	It runs the kernel.
finalizeKernel	It frees allocated memory on the GPU.
prepareCudaAgentInfo	It makes a set of basic information on the CudaAgent, e.g. the number of GPU devices.
setup	Bootstrap method of the CudaAgent.
getMessageParserInstance	The method returns an instance of MessageParser, it also overrides handle-like methods of ACL message management.

Table 2: The CudaAgent important methods

```
Input  : Kernel kernel
Output: CudaKernel cudaKernel
1 get kernel launcher from kernel;
2 if is not initialize kernel launcher then
3     initialize kernel;
4     call prepareKernel method on cudaAgent instance;
5 run kernel;
6 receive result of the kernel as cudaResult;
7 return cudaResult;
```

Algorithm 1: Kernel call

VI. The Usage of CudaAgents

If some agent wants to use CudaAgents to manage GPU computation, it has to perform several kernel call actions and wait for an adequate response. Generally, such an agent must receive a kernel ID, which means that the kernel is compiled and prepared for execution on the CudaAgent side. After receiving kernel ID, the agent can send an ACL message to the CudaAgent to start the execution of kernel and wait for computation result.

VII. Experiments

Two agents and their codes were performed in the experiment (CudaAgent and CudaAgentSample). The primary goal of the CudaAgentSample is to deliver a CUDA kernel code to CudaAgent, then invoke compilation and execution. The kernel code is a simple matrix multiplication adapted to the parallel CUDA environment. The whole process can be divided into three parts:

1. CudaAgentSample tries to register the kernel on the side of CudaAgent.
2. CudaAgentSample sends to the CudaAgent the kernel and call actions (compile, execute).
3. CudaAgentSample waits for the response in the form of some kernel result from the CudaAgent.

Moreover, the performance of the CudaAgent will be compared with the performance of a non-parallel agent called CpuAgent. The CpuAgent executes the same matrix multiplciation code, but adapted to the CPU environment without parallel execution. The CudaAgentSample tries to execute the kernel code (via kernel call action) several times on the CudaAgent and CpuAgent, respectively.

CudaAgentSample sends an ACL Message as the cuda kernel call action to the CudaAgent. It contains kernel (source code and ptx code respectively) with the kernel parameters. Algorithm 2 describes how a kernel call action is sent. After sending the cuda kernel call action, the CudaAgentSample has to wait until receiving the reply from the CudaAgent with a kernel ID. After receiving the kernel ID, the kernel can be called with pre-set parameters. The kernel does not have to be sent anymore. The kernel ID indicates that the kernel is already compiled and ready to use on the side of CudaAgent.

The second way to send kernel data to the CudaAgent is using the kernel register action. After sending the kernel register action, we receive a reply (same reply as the reply after sending the kernel call action) containing the kernel id via overrided the handle methods of instance of the class MessageParser (described in the next section). This action just sends the kernel with the parameters to the CudaAgent, but the kernel will not be run. If we want to run this kernel, we send another ACL message as the kernel call action with the kernel id. The difference between the kernel register action and the kernel call action is shown in Table 1.

```
Input  : Kernel kernel
Output: void
1 prepare selected kernel;
2 make kernel register action;
3 wait until received a reply contains a kernel ID;
4 create an ACL message with the kernel ID;
5 send the ACL message to execute required kernel;
```

Algorithm 2: Sending the ACL message: CudaAgentSample $\rightarrow$ CudaAgent

Algorithm 3 illustrates the receiving of a result after the kernel is processed. Moreover, each response is identified with the conversation ID to manage subsequent processes. The kernel result data represents parameters that were marked as output (a constant COPY_TO_HOST) parameter in the kernel call action or possibly in the kernel register action. The instance of MessageParser handles the reply of the ACL message and extracts its content as the kernel result data. If a transformation fails, the CudaAgent sends an ACL message with failure performative.

Java emulation kernel has to be implemented to compare the performance of the CUDA technology with performance

```
Input  : void
Output: ACLM ACL message
1 create an instance of MessageParser;
2 override method handleInform;
3 fill the ACL message with respect to process result;
```

Algorithm 3: Receiving the kernel result data: CudaAgent → CudaAgentSample

of the java platform. CudaAgent supports CUDA technology for the kernel call. Another agent called AgentCPU extends the CudaAgent and overrides several methods for kernel calling. It calls the kernel on the java platform instead of kernel call on the CUDA capable device. This kernel is adapted to the java platform, which means that the kernel supports only one thread. Moreover, the kernel is compiled into java byte code.

A. *Performance Tests and Summary Results*

A performance test scenario and summary results are described in the following part. A central point of the performance test consists in the utilization of matrix multiplication algorithm. Two matrices were generated. One is 1200x80 and the other was 80x1600. The multiplication result is saved into a third matrix.

The time spent on a kernel call on CudaAgent and CPUAgent was measured, respectively. Next, the total time of sending a request to run a kernel (as the kernel call action) and back transfer of matrix results to CudaAgentSample was measured. The kernels were called 2000 times. The comparison of measured times can be seen in figures 7 and 8.

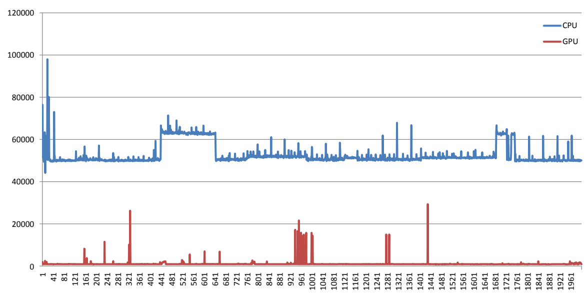

Figure. 7: Time spent on the kernel call [ms]

The figure 7 shows computation times. It is evident that GPU brings significant improvements in comparison with the CPU version. Sometimes, there are some jumps in measured values which caused by system just-in-time workload. The JADE framework runs as a process with pre-set priority, which must be lower than the priorities of system processes. The computation time can be disturbed by requests of display devices, system memory management (swapping), etc.

Figure 8 illustrates transfer times. Even if the transfer times seem to be small and almost the same for CPU and GPU, it is evident, that data transfer within MAS plays an important role in comparison to the computation time. This is a general problem in GPU programming, and it is partially solved in new architectures. On the other hand, some compression algorithm can be used to reduce transfer time, etc.

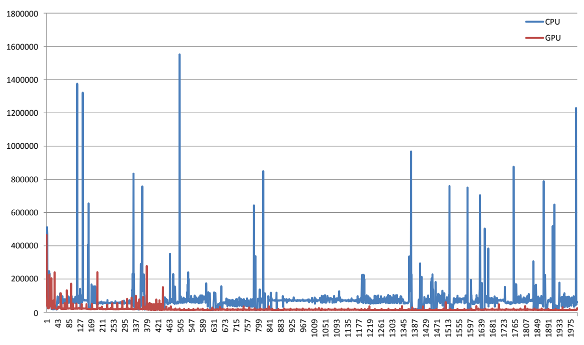

Figure. 8: Time spent on received kernel result [ms]

Matrix size	Time [ms]	copy host to device [ms]	copy device to host [ms]
16 x 16	1	1	1
32 x 32	1	1	1
64 x 64	1	1	1
128 x 128	1	2	1
256 x 256	1	2	1
512 x 512	3	4	2
1024 x 1024	28	11	5
2048 x 2048	250	23	13
4096 x 4096	2164	146	71
8192 x 8192	3971	580	197

Table 3: The matrix multiply using jcuda

VIII. Future research and conclusion

A novel approach to parallel programming in the area of multi-agent system was presented in this article. The implementation of CudaAgent in the Mutli-agent framework JADE was described in more detail. The benefits of utilization of GPU was illustrated in experiments, and the results were compared with the CPU version. It can be seen that GPU can increase performance in the MAS system, and it can be deploy in MAS systems where several tasks are critical and time-consuming. More complex algorithms should be tested in the near future, e.g. signal processing or matrix decomposition, to measure the performance of the proposed solution and to avoid possible divergences in time measurement.

Acknowledgement

This article has been elaborated in the framework of the IT4Innovations Centre of Excellence project, reg. no. CZ.1.05/1.1.00/02.0070 funded by the Structural Funds of the European Union and the state budget of the Czech Republic. The work is partially supported by Grant of SGS No. SP2013/70, VŠB - Technical University of Ostrava, Czech Republic. This work was also supported by the Bio-Inspired Methods: research, development and knowledge transfer project, reg. no. CZ.1.07/2.3.00/20.0073 funded by Operational Programme Education for Competitiveness, co-financed by ESF and state budget of the Czech Republic.

References

[1] S. Sotiriadis, N. Bessis, Y. Huang, P. Kuonen, and N. Antonopoulos, "A jade middleware for grid intercooperated infrastructures," in *AINA Workshops*, 2011, pp. 135–140.

[2] C. Guo, Z. Lin, and K. Guo, "Research and implementation of an enterprise-class mas application development framework-jadeee," in *CSCWD (Selected Papers)*, 2007, pp. 294–303.

[3] M. Baldoni, G. Boella, V. Genovese, R. Grenna, and L. van der Torre, "How to program organizations and roles in the jade framework," in *MATES*, 2008, pp. 25–36.

[4] F. T. F. for Intelligent Physical Agents, "http://www.supercomp.org/sc95/proceedings/-473_MBER/SC95.HTM," 2005.

[5] M. Verdicchio and M. Colombetti, "Communication languages for multiagent systems," *Computational Intelligence*, vol. 25, no. 2, pp. 136–159, 2009.

[6] M. de Weerdt, Y. Zhang, and T. Klos, "Multiagent task allocation in social networks," *Autonomous Agents and Multi-Agent Systems*, vol. 25, no. 1, pp. 46–86, 2012.

[7] J. Ferber, *Multi Agent Systems, An introduction to Distributed Artificial Intelligence*, 1993.

[8] L. Braubach, W. Lamersdorf, and A. Pokahr, "Jadex: Implementing a BDI-Infrastructure for JADE Agents," http://vsis-www.informatik.uni-hamburk.de/papers/pokahrbraubach2003jadex-exp.pdf, September 2003.

[9] N. T. M. Khue and N. V. Do, "Building a model of an intelligent multi-agent system based on distributed knowledge bases for solving problems automatically," in *ACIIDS (1)*, 2012, pp. 21–32.

[10] K. Schelfthout and T. Holvoet, "Object Places: An Environment for Situated Multi Agent Systems," in *International Conference on Autonomous Agents and Multi Agent Systems, AAMAS*. IEEE Computer Society, 2004.

[11] B. M. Namee, "A Proposal for an Agent Architecture for Proactive Persistent Non Player Characters," http://www.cs.tcd.ie/publications/tech.reports/-reports.01/TCD-CS-2001-20.pdf, 2001.

[12] A. Kerr, G. F. Diamos, and S. Yalamanchili, "A characterization and analysis of ptx kernels," in *IISWC*, 2009, pp. 3–12.

[13] C. Nyulas, M. J. O'Connor, S. W. Tu, D. L. Buckeridge, A. Okhmatovskaia, and M. A. Musen, "An ontology-driven framework for deploying jade agent systems," in *IAT*, 2008, pp. 573–577.

[14] E. J. Friedman-Hill, *Jess, The Java Expert System Shell*, Sandia National Laboratories, Livermore, CA, USA, Mar. 1998.

[15] K. Chouchane, O. Kazar, and A. Aloui, "Agent-based approach for mobile learning using jade-leap," in *ICWIT*, 2012, pp. 300–305.

[16] A.-H. Tan, Y.-S. Ong, and A. Tapanuj, "A hybrid agent architecture integrating desire, intention and reinforcement learning," *Expert Syst. Appl.*, vol. 38, no. 7, pp. 8477–8487, 2011.

[17] F. Capkovic, "Cooperation of hybrid agents in models of manufacturing systems," in *KES-AMSTA*, 2011, pp. 221–230.

[18] J. M. Alberola, J. M. Such, V. J. Botti, A. Espinosa, and A. García-Fornes, "A scalable multiagent platform for large systems," *Comput. Sci. Inf. Syst.*, vol. 10, no. 1, pp. 51–77, 2013.

[19] J. Odell, "The Foundation for Intelligent Physical Agents," http://www.fipa.org/, 2006.

[20] F. Zambonelli, "Developing Multi-Agent Systems: The Gaia Methodology," 2003.

[21] T. P. Filgueiras, L. C. Lung, and L. de Oliveira Rech, "Providing real-time scheduling for mobile agents in the jade platform," in *ISORC*, 2012, pp. 8–15.

[22] A. Akramizadeh, A. Afshar, M. B. Menhaj, and S. Jafari, "Model-based reinforcement learning in multiagent systems with sequential action selection," *IEICE Transactions*, vol. 94-D, no. 2, pp. 255–263, 2011.

[23] W. the free encyclopedia, "http://en.wikipedia.org," 2006.

[24] P. Tichý, P. Kadera, R. J. Staron, P. Vrba, and V. Marík, "Multi-agent system design and integration via agent development environment," *Eng. Appl. of AI*, vol. 25, no. 4, pp. 846–852, 2012.

[25] R. Erol, C. Sahin, A. Baykasoglu, and V. Kaplanoglu, "A multi-agent based approach to dynamic scheduling of machines and automated guided vehicles in manufacturing systems," *Appl. Soft Comput.*, vol. 12, no. 6, pp. 1720–1732, 2012.

[26] C. Boutilier, "Learning conventions in multiagent stochastic domains using likelihood estimates," *CoRR*, vol. abs/1302.3561, 2013.

[27] F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa, "JADE - A White Paper," 1999.

[28] A. Schuldt, J. D. Gehrke, and S. Werner, "Designing a simulation middleware for fipa multiagent systems," in *IAT*, 2008, pp. 109–113.

[29] D. B. Kirk and W. mei W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach (Applications of GPU Computing Series)*. Morgan Kaufmann, Feb. 2010.

[30] Khronos, "Khronos: Opencl," http://www.khronos.org/opencl/.

[31] nVIDIA, “nVIDIA Fermi - White Paper,” http://www.nvidia.com/content/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf, 2012.

[32] D. Kumar and M. A. Qadeer, “Fast heterogeneous computing with cuda compatible tesla gpu computing processor (personal supercomputing),” in *ICWET*, 2010, pp. 925–930.

[33] G. Caggianese and U. Erra, “Gpu accelerated multi-agent path planning based on grid space decomposition,” *Procedia CS*, vol. 9, pp. 1847–1856, 2012.

[34] M. Ciznicki, M. Kierzynka, P. Kopta, K. Kurowski, and P. Gepner, “Benchmarking data and compute intensive applications on modern cpu and gpu architectures,” *Procedia CS*, vol. 9, pp. 1900–1909, 2012.

[35] C. M. Wittenbrink, E. Kilgariff, and A. Prabhu, “Fermi gf100 gpu architecture,” *IEEE Micro*, vol. 31, no. 2, pp. 50–59, 2011.

[36] J. Kurzak, S. Tomov, and J. Dongarra, “Autotuning gemm kernels for the fermi gpu,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 11, pp. 2045–2057, 2012.

[37] G. Tan, L. Li, S. Triechle, E. Phillips, Y. Bao, and N. Sun, “Fast implementation of dgemm on fermi gpu,” in *SC*, 2011, p. 35.

[38] B. H. H. Juurlink and C. Meenderinck, “Amdahl’s law for predicting the future of multicores considered harmful,” *SIGARCH Computer Architecture News*, vol. 40, no. 2, pp. 1–9, 2012.

[39] K. W. Cameron and R. Ge, “Generalizing amdahl’s law for power and energy,” *IEEE Computer*, vol. 45, no. 3, pp. 75–77, 2012.

[40] J. L. Gustafson, “Amdahl’s law,” in *Encyclopedia of Parallel Computing*, 2011, pp. 53–60.

[41] nVIDIA, “Cuda Programming Guide,” http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf, 2012.

[42] I. Vondrák, *Úvod do softwarového inžen´yrstv (in Czech)*. VB–Technical University of Ostrava, 2002.

[43] R. Farber, *CUDA Application Design and Development*, 1st ed. Morgan Kaufmann, 2011.

[44] I. Vondrák, *Methods of Business Modeling*. VŠB-TUO, Ostrava, 2004, vol. I.

[45] C.-J. Su and C.-Y. Wu, “Jade implemented mobile multi-agent based, distributed information platform for pervasive health care monitoring,” *Appl. Soft Comput.*, vol. 11, no. 1, pp. 315–325, 2011.

[46] F. Bellifemine, “Java Agent Development Framework Documentation,” http://jade.tilab.com/, 2005.

[47] Ryerson, “JADE Documentation,” http://www.scs.ryerson.ca/%7edgrimsha/jade/doc/-index.html, 2006.

[48] R. R. de Azevedo, E. R. D. Galvão, R. C. dos Santos, C. M. de Oliveira Rodrigues, F. Freitas, and M. J. Siqueira, “An autonomic multiagent system ontology-based for management of security of information,” in *ICITST*, 2009, pp. 1–9.

[49] A. Sarkar, U. Marjit, and U. Biswas, “A middleware architecture for secure service discovery using ontologies with multiagent approach,” *IJISSC*, vol. 3, no. 1, pp. 47–55, 2012.

[50] V. Dignum, “Ontology support for agent-based simulation of organizations,” *Multiagent and Grid Systems*, vol. 6, no. 2, pp. 191–208, 2010.

Author Biographies

Lukáš Zaorálek is a PhD student at the Department of Computer Science of VŠB-Technical University of Ostrava. His research interests involve soft-computig.

Petr Gajdoš is an Assistant Professor at the Department of Computer Science of VŠB-Technical University of Ostrava. He did his Ph.D. Degree in Informatics and Applied Mathematics at the same university in 2006. Nowadays, his research is primarily focused on parallel programming, soft-computing and application of bio-inspired methods.