Developing distributed multi-core and many-core architecture using java agents

Jelena Tekić, Predrag Tekić, Miloš Racković
University of Novi Sad, Faculty of Sciences, Novi Sad, Serbia
{radjenovic, tekic} @uns.ac.rs, rackovic@dmi.uns.ac.rs

***Abstract*—In this paper java agent architecture for utilizing available multi-core and many-core hardware is presented. Architecture is developed using JADE (Java Agent DEvelopment Framework) and OpenCL standard for programming multi-core and many-core devices. A JADE-based system can be distributed across available machines and OpenCL promises portability of the developed code between heterogeneous devices. OpenCL is an open, royalty free, standard developed by Khronos group for parallel programming of heterogeneous devices (CPU's, GPU's, …) from different vendors. Developed java agents communicate with each other in accordance with FIPA specification and are independent from operating system of the specific machine they run on. In this paper two java agents are presented. First agent inspects available hardware with OpenCL support and sends message to the second agent which runs simulation utilizing all the available devices with OpenCL support discovered by the first software agent.**

Keywords: JADE, OpenCL, multi-core, Java Agent, GPU

I. INTRODUCTION

Recently, there has been a breakthrough in computer processor technology. Multi-core and many-core processors are replacing single core processors. Graphics Processing Units (GPUs) have an important role in today's high performance computing applications. High performance computing was a privilege of small group of scientists with budget to fund expensive large computer clusters or specially manufactured High performance computers. Nowadays, with massively produced multi-core and many-core processors it is available to almost every commodity desktop/personal computer.

Large processing power potential of new multi-core and many-core processors (CPUs and GPUs) has attracted researchers and developers to start considering to utilize power of those new processors in solving their specific scientific problems.

There are number of scientific fields which are profiting from this trend in computer hardware industry, among them is computational fluid dynamics (CFD). This field of science in recent years, is starting to make use of new and more powerful multi-core and many-core processors in order to solve more and more complex numerical simulations.

In this paper we have proposed software solution which will be able to automatically detect, and make use of any device available for large scientific calculations and therefore reduce time needed for specific scientific calculation. Proposed solution should create, at runtime, a cluster of hardware devices able to perform parallel numerical computations and execute specific task on all of the available devices in that cluster.

In the following pages we will explain software technology we have chosen for our software solution and the architecture of the proposed software solution.

A. FIPA

The Foundation for Intelligent Physical Agents (FIPA) is an international organization helping to promote the industry of intelligent agents. FIPA is developing specifications supporting interoperability among agents and agent-based applications.

FIPA Abstract Architecture includes:

- Distributed computing platforms or programming languages,
- Messaging platforms,
- Security services,
- Directory services, and,
- Intermittent connectivity technologies.

Also FIPA Abstract Architecture defines the following concepts:

- A model of services and discovery of services available to agents and other services,
- Message transport interoperability,
- Supporting various forms of ACL representations,
- Supporting various forms of content language, and,
- Supporting multiple directory services representations.

FIPA abstract architecture specification describes how to make the agent system, agents and services that rely on the system, in Figure 1. Mapping FIPA-abstract architecture on the concrete realization is shown.

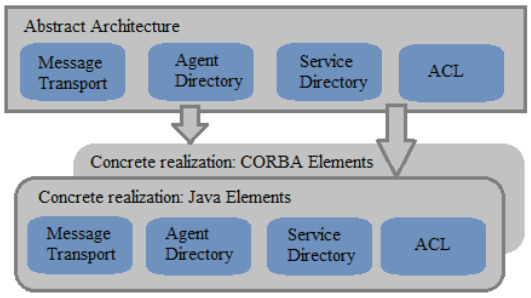

Figure 1. Mapping FIPA-abstract architecture on the concrete realization

Agents living in one environment can communicate by sending messages to each other. Messages are encoded in an Agent-Communication-Language.

Services provide support services for agents. Services are generally implemented as software that is accessed via method invocation. Service root is provided to an agent on start-up, service root will provide a set of service-locators such as: message-transport-services, agent-directory-services and service-directory-services.

Agent-directory-service provides a location where agents register their descriptions as agent-directory-entries. Agent-directory-entry consists of at least two key-value-pairs Agent-name and Agent-locator. Agent-name is globally unique name and Agent-locator consists of one or more transport-descriptions. Each transport-descriptions contain transport-type, transport-specific-address and zero or more transport-specific-properties. Also agent-directory-entry may contain other descriptive attributes (services offered by the agent, cost of using the agent, restrictions on using the agent, etc.)

Service-directory-service provides discovering of services and a location for registration of service-directory-entries. Service-directory-entry consisting of at least one key-value-pairs: Service-name, Service-type and Service-locator. Service-locator contains one or more key-value tuples that consists of a signature type, service signature and service address. Also service-directory-entry may contain other descriptive attributes (cost of using the service, restrictions on using the service, etc.)

Agent message consists of sender name, receiver names and message content. Every message must have one sender and zero or more receivers. Message content can be determined by the ontologies, in Figure 2 message structure is shown.

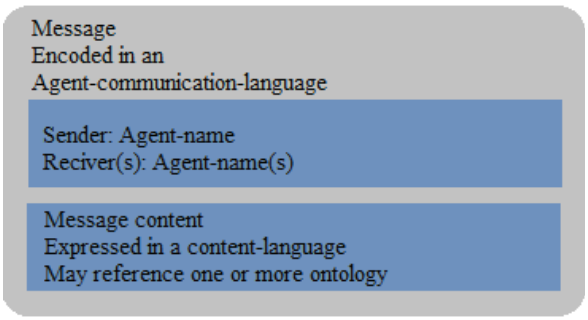

Figure 2. Message structure

Before transport message is encoded into a payload and then included in a transport-message. Transport-message consists of payload and envelope. Envelope defines transport-description: sender address, receiver address, transport protocol, encoding-representation etc.

In order to obtain the parameters needed to transport messages agent communicate with directory agent. The parameters that are needed are placed in the agent locator. Communication begins by a search of the directory service for the agents (1 : Query) . Based on the name of the destination agent, the starting agent from the directory agents obtains agent-directory-entry which contains all the necessary parameters for communication (Transport - type, Transport- specific -address , Transport- properties) . The initial agent can send a message by the first type of transport (e.g. , HTTP protocol) , and then change the type of transport (e.g. SMTP protocol) if the destination agent is able to communicate using both.

FIPA specification supports two types of security mechanisms: message validation and encryption of the messages. Validation of the message includes the ability to detect changes of the messages that occurred after sending the message. If some of the protection mechanisms are applied in the description of the message (envelope) additional parameters must be placed. These additional parameters will allow the use of protective mechanisms. [1,2]

B. JADE

JADE (Java Agent DEvelopment Framework) is free software Framework that is complied with the FIPA specifications and implemented in the Java language.

JADE was initially developed by the University of Parma. Copyright holder and distributor for JADE software is Telecom Italia.

JADE consists of run-time environment, a library of classes (used by programmers for developing agents) and graphical tools for administrating and monitoring the activity of running agents.

Container is running instance of the JADE runtime environment, it can contain several agents. Platform contains several containers; one container in platform is special container called Main container. Main container is always active in a platform, all other containers register with it; it contains special agents AMS (Agent Management System) and DF (Directory Facilitator). The AMS (Agent Management System) provides the naming service and represents the authority in the platform. DF (Directory Facilitator) provides service for finding agents.

JADE provides communication between agents by use of asynchronous message passing, The JADE asynchronous message passing paradigm is shown in Figure 3. For each agent JADE runtime posts messages sent by other agents in the agent message queue and notify agent that he got message. When the agent will process message is completely up to the programmer. [3,4]

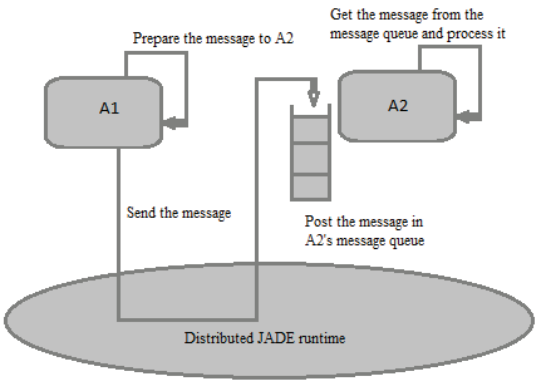

Figure 3. The JADE asynchronous message passing paradigm

C. OpenCL and JOCL

Open Computing Language (OpenCL) [2] is an open, royalty free, standard developed by Khronos group [3] for parallel programming of heterogeneous devices (CPUs, GPUs, DSPs) from different vendors. The execution model for OpenCL consists of the controlling host program and kernels which execute on OpenCL devices.[8]

JOCL is library that provides Java binding for OpenCl. JOCL is very similar to original OpenCL API. JOCL functions are written as static methods. Semantics and signatures of JOCL methods are consistent with the original OpenCL library functions, except for the language-specific limitations of Java. [5]

II. IMPLEMENTATION DETAILS

In this chapter we will focus on describing implementation of java software agents which are used to inspect hardware devices and start numerical simulation. Benchmark problem that is used for the numerical simulation is well known and often used in CFD, lid driven cavity flow.[6,7] Implementation details of this benchmark simulation problem will not be subject of this paper.

In order to start numerical simulation on the available hardware devices, first we need to inspect available hardware resources in the specific environment. Java software agent *InspectPlatformAgent* has been created for that purpose.

InspectPlatformAgent examines all available devices on the specific platform and verifies which devices have support for OpenCL specification and print that information to the console. Software agent creates message, which will be sent to another agent, with information about devices which have support for the OpenCL specification (no device – *NO*, all devices – *ALL*, only GPU – *GPU*, only CPU - *CPU*). In order to create agent in JADE agent framework, agent needs to inherent *jade.core.Agent* class and implement *setup()* method. Code that is used to examine available devices implemented as a part of *setup()* method. Implemented code uses JOCL library and OpenCL API functions to inspect hardware devices. *InspectPlatformAgent* has inner class which provides functionality of sending messages to another agent. Created class listing is shown in Figure 4. This class creates new *behaviour*, *TickerBehaviour* have been used, which periodically (in this example every 10 seconds) repeats task that need to be carried out.

```
TickerBehaviour  loop = new TickerBehaviour(this,10000) {
    public void onTick() {
        ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
        msg.addReceiver(new AID ("simulationExample", AID.ISLOCALNAME));
        msg.setContent(mesContent);
        myAgent.send(msg);
        ACLMessage msgRecive = myAgent.receive();
        if (msgRecive != null) {
         doDelete();
        }
    }
} ;
addBehaviour(loop);
```

Figure 4. InspectPlatformAgent class listing

InspectPlatformAgent have a task to send ACL message. Message is created by setting message type (ACLMessage.INFORM in this case) local name of the agent that receives message and content of the message.

Another java software agent have been created *SimulationExampleAgent* to carry out numerical simulation on the discovered devices. This agent receives previously created message about platform from *InspectPlatformAgent.* After message have been processed it sends notification to *InspectPlatformAgent* which triggers *doDelete()* command which will destroy *InspectPlatformAgent* software agent.

SimulationExampleAgent software agent receives message from the first agent, and based on that message starts numerical simulation on all of the available hardware devices with OpenCL support and prints out time needed to carry out simulation on each device.

A. Java agent environment

In order to start software agent we need to compile it with following commands:

javac -classpath lib\jade.jar;lib\JOCL-0.1.9.jar -d classes src\examples\inspectPlatformAgent\ SimulationExampleAgent.java

and

javac -classpath lib\jade.jar;lib\JOCL-0.1.9.jar -d classes src\examples\inspectPlatformAgent\ InspectPlatformAgent.java

Following command:

java -cp lib\jade.jar;lib\JOCL-0.1.9.jar;classes jade.Boot –gui

is used to start JADE user interface with dependencies need in this example (*lib\JOCL-0.1.9.jar*). In the Figure 5 JADE user interface is shown.

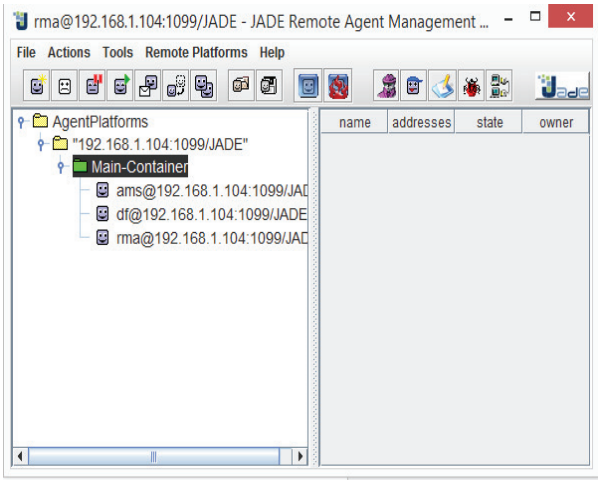

Figure 5. JADE user interface

Upon selecting main container, software agents can be started. In the list of the available (compiled) agents, previously created and compiled container *examples.exampleAgent.InspectPlatformAgent* is selected as shown on Figure 6.

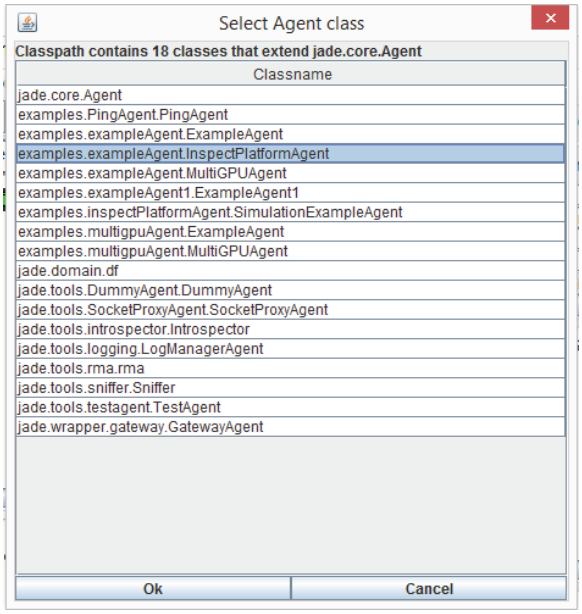

Figure 6. List of the available (compiled) agents

In the next step, each agent needs to be started separately. List of all compiled software agents will appear in user interface, as shown in Figure 7. Result of started agents is console output shown in Figure 8.

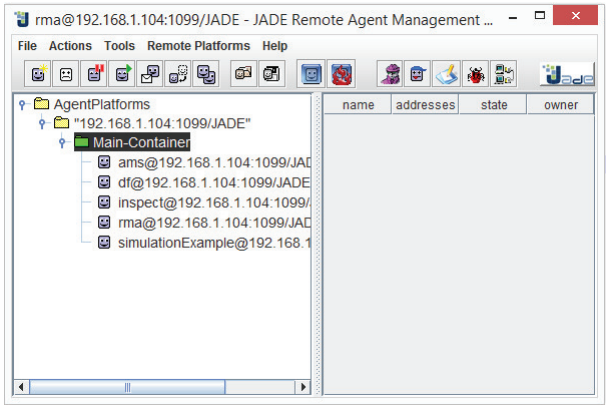

Figure 7. JADE user interface with started agents

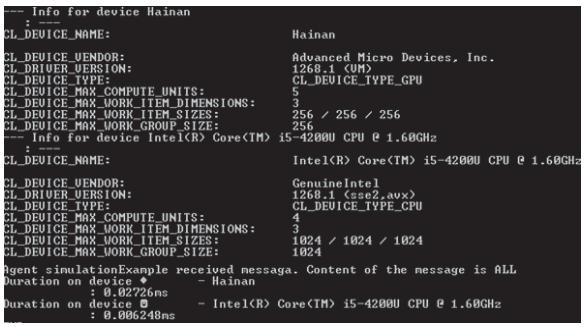

Figure 8. Result of started agents

III. CONCLUSION

In this paper we have described software architecture created to utilize all the available hardware devices with OpenCL support, in order to carry out parallel numerical simulation. Software architecture is based on java agents, using JADE agent framework and java OpenCL library (JOCL) for programming multi-core and many-core devices with support for OpenCL specification. Created software agents are used to inspect hardware devices and after having that information to run numerical simulation on all the available devices, utilizing computational power of all discovered devices.

It has been shown that it is possible to develop software architecture that is both platform and vendor independent which can be used to discover and use all devices with parallel computing potential. This solution can be used to carry out scientific computations by using all the available devices in specific computer network.

ACKNOWLEDGMENT

The work is partially supported by Ministry of Education and Science of the Republic of Serbia, through Project OI174023: ‘‘Intelligent techniques and their integration into wide-spectrum decision support’’

REFERENCES

[1] FIPA http://www.fipa.org/, January 2015.

[2] FIPA specification http://www.fipa.org/specs/fipa00001/SC00001L.pdf, January 2015.

[3] JADE http://jade.tilab.com, January 2015.

[4] JADE Programming Tutorilal http://jade.tilab.com/doc/tutorials/JADEProgramming-Tutorial-for-beginners.pdf, January 2015.

[5] Java OpenCL Library - JOCL, http://www.jocl.org/, January 2014.

[6] Tekić, P., Rađenović, J., Lukić, N., Popović, S., Lattice Boltzmann simulation of two-sided lid-driven flow in a staggered cavity, International Journal of Computational Fluid Dynamics, (ISSN 1061-8562), Vol. 24, Issue 9, pp. 383-390, 2010.

[7] Tekić, P., Rađenović, J., Racković, M., Implementation of the Lattice Boltzmann Method on Heterogeneous Hardware and Platforms using OpenCL, Advances in Electrical and Computer Engineering, (ISSN: 1582-7445), Vol. 12,No 1, pp. 51-56, 2012.

[8] Rađenović, J., Tekić, P., Racković, M., VISUALISATION OF FLOW AND TEMPERATURE FIELD CALCULATED BY LB METHOD IN POST-PROCESSING SOFTWARE PARAVIEW, Proceedings of the International Conference on Information Society Technology and Management (ICIST), pp. 303-306,2014. ISBN: 978-86-85525-14-8