Available online at www.sciencedirect.com
ScienceDirect
Procedia Computer Science 1 (2010) 1971–1981
Procedia Computer Science
www.elsevier.com/locate/procedia

International Conference on Computational Science, ICCS 2010

Agent-based parallel system for numerical computations

Marcin Sieniek*, Piotr Gurgul, Pawel Kołodziejczyk, Maciej Paszyński

Department of Computer Science, AGH Univeristy of Science and Technology, al. A. Mickiewicza 30, 30-059 Krakow, Poland

Abstract

The paper describes a way of applying agent paradigm to hp-adaptive Finite Element Method (hp-FEM). We discuss a choice of classical numerical algorithms suitable for incorporating into an agent-based application, along with an efficient way of adopting them into an agent-based application. We define formally a Computing Multi Agent System (C-MAS) for adaptive 1D FEM based on Smart Solid Agent model and describe tasks executed by hp-FEM agents. Finally, we spare a few paragraphs for numerical experiments performed with an application developed accordingly to the described model.

Keywords: computing multi-agent systems; adaptive finite element method; distributed computations

1. Motivation

Adaptive systems for numerical methods like Finite Element Method or Finite Difference Method grow nowadays to great sizes because of the amount of included components, algorithms and parallelism subroutines combined altogether (typical solutions in this area are still based on low-level communication libraries like PVM or MPI [12]). A usual way of attempting to address complexity issues by applying object- or component-oriented approaches [6, 7] works well with simplifying general design, but seems to struggle where parallelism matters, as this is not the aim these paradigms were designed for.

This is a point where multi-agent system concept comes in handy. Agent paradigm is a high-level, scalable and relatively simple approach of developing distributed applications (for detailed description, see for example [11]). Its usefulness, for not only artificial intelligence but also numerical computations, has already been proven in existing works [4, 5].

The aim of this work is to apply the agent approach to hp-FEM. By doing so, we gain an ability to divide and merge computational tasks in runtime, which is crucial to a vast majority of adaptive algorithms. This allows for a more accurate automatic load balancing, which can be delegated to an underlying agent platform [3].

Being probably the most advanced adaptive mesh-based algorithm intended for solving PDEs, the hp-FEM is what we focus on in this article. The paper can be treated, however, as a more general proof of concept for agent-based adaptive computational solvers, as the same approach applies to Finite Difference Method and presumably other mesh-based algorithms.

* Corresponding author: M. Sieniek; *tel.*: +48-725-567-100, e-*mail address*: qadro@student.agh.edu.pl

1877-0509 © 2010 Published by Elsevier Ltd.
doi:10.1016/j.procs.2010.04.221

2. Self adaptive hp-FEM algorithm

In the following section we introduce the basic ideas of the adaptive FEM and the sequential version of its algorithm, as they have been the starting point for the development of our application. Later on, we demonstrate how they can be directly and efficiently adopted in an agent-based application.

2.1. Approximation space

The FEM involves approximation using Galerkin method, according to which the equation is expressed using a set of basis functions (so called 'shape functions'). In our application these functions are based on polynomials with supports restricted to only several neighbor elements (in 1D – two elements at most). Though, the computational mesh is composed of the following items:

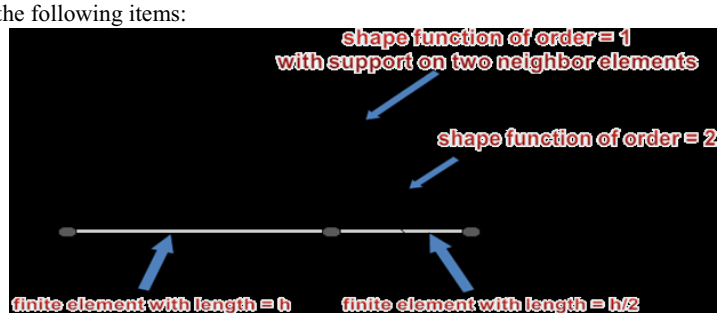

Fig. 1 – mesh items, interpretation of *h* and *p* coefficients

The solution is then represented as a linear combination of these base functions (DoF_{count} stays for the total number of basis functions, $[u_i]_{i=0,DoF_{count}}$ is a solution of the linear equation introduced by the discretization):

$$u(x) = \sum_{i=0}^{DoFcount} u_i e_j(x)$$

Extending the base (either by dividing finite elements into smaller ones (h-adaptation), or by increasing the polynomial order of shape functions (p-adaptation)) results in a larger equation to solve on one hand, but may increase precision significantly on the other. That is why it is essential to choose the mesh carefully.

2.2. HP-adaptive 1D algorithm

The initial mesh tends to prove too simple for the majority of industrial problems. That is why, different kinds of adaptive algorithms are used to improve the mesh increasing the number of degrees of freedom in a smart way, until the results become sufficient in terms of an arbitrary norm.

In our implementation we have chosen the *hp-adaptive* algorithm described in [1]. In its simplest form it is presented with the pseudo code below (alg. 1).

The error decrease rate err_K on an element K may be defined as follows:

$$err_K = \left\| u_{fine} - u_{coarse} \right\|_K^{H_1}$$

where $\|f\|_K^{H_1} = \sqrt[2]{\int_K ([f(x)]^2 + [f'(x)]^2)dx}$ is a norm in the Hilbert space H_1.

```
function adaptive_fem( mesh_initial , err_desire )
  mesh_coarse = mesh_initial
  repeat
    u_coarse = solve the problem on mesh_coarse using frontal solver
    //section #1 - refinement
    mesh_coarse = copy mesh_coarse
    divide each element K of mesh_fine into two new elements (K1, K2)
    increase polynomial order of shape functions on each element of mesh_fine by 1
    //section #1 ends
    u_fine = solve the problem on mesh_fine using frontal solver
    //section #2 - adaptation error estimation
    err_max = 0
    for each element K of mesh_fine do
      err_K = compute relative decrease error rate on K //defined below
      if err_K > err_max then
        err_max = err_K
      end if
    end do
    //section #2 ends
    //section #3 - error evaluation and adaptation
    mesh_adapted = new empty mesh
    for each element K of mesh_coarse do
      if err_K > 0.33 * err_max then
        add K1 (left child of K) and K2 (right child of K) from mesh_fine to mesh_adapted
      else
        add K from mesh_coarse to mesh_adapted
      end if
    end do
    mesh_coarse = mesh_adapted
    //section #3 ends
    output u_fine //section #4
  until err_max < err_desired
  return ( u_fine , mesh_fine )
end function
```

Alg. 1 – hp-adaptation

2.3. Equation solver

For our application we adopted a distributed high-performance frontal direct solver, described in [8,9]. The solver is based on Schur complement concept [10] (see fig. 2). The recursive version of this algorithm is presented by the following pseudo code (alg. 2).

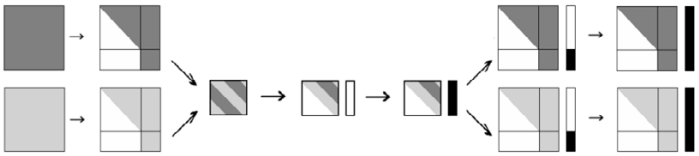

Fig. 2 – parallel solving of two, partially dependent equations; only their Schur complements need to be merged and eliminated together; then, the common part undergoes backward substitution; in the end, the solution of the common part is propagated back to son nodes, which continue backward substitution in parallel

```
function recursive_solver(tree_node)
  if tree_node has no son nodes then
    //section #1 - leaf nodes elimination
    eliminate leaf element stiffness matrix internal nodes
    return Schur complement sub-matrix
    //section #1 ends
  else if tree_node has son nodes then
    for each son of tree_node do
      son_matrix = recursive_solver(tree_node_son)
      merge son_matrix into new_matrix     //section #2
    end do
    //section #3 - non-leaf nodes elimination
    decide which unknowns of new_matrix can be eliminated
    perform partial forward elimination on new_matrix
    return Schur complement sub-matrix
    //section #3 ends
  end if
end function
```

Alg. 2 – frontal solver, Gaussian elimination step

3. Agent-based approach

3.1. C-MAS architecture definition

The presented solution can be described altogether as a Computing Multi-Agent System (C-MAS) [2]. To ensure fair separation of roles, this model is based on three layers:

- task definition
- Smart Solid Shell (SSS)
  - API (including routines for grain control, task scheduling) etc.
  - platform-specific implementation
- MAS platform (including migration & communication API)

Namely, in our application these are:

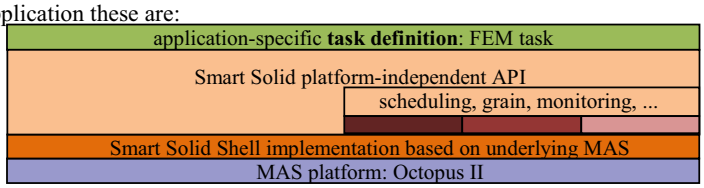

Fig. 3 – Computing Multi Agent System Architecture (based on fig. 1 from [2])

An agent A_i in this model (called Smart Solid Agent - SSA [2]) is donated by a pair $A_i = (T_i, S_i)$ where T_i represents a task assigned to A_i, including all task-specific data, S_i represents the Smart Solid shell, responsible for load management and i stays for a globally unique agent identifier (GUID).

There are a few expectations involving an SSA task: Apart from imperative part (Java code) we are required to provide a declaration of the initial task distribution and a denomination of the expected use of resources. It is also expected that the task is able to divide itself upon a demand from SSS.

3.2. Octopus II agent platform

Although the ideas presented in this article are common and valid for a wide choice of agent platforms, we decided to illustrate them with a concrete application, implemented on Octopus II platform – our Institute's in-house developed framework for agent-based computational applications.

Not only does Octopus include a regular MAS platform but it also provides the SSS layer [5], capable of automatic load-balancing, thanks to the innovative scheduling algorithm based on diffusion, in which computing agents migrate between nodes in a cluster similarly to atoms diffusing in solid state materials [3].

Octopus version 2 [7] is implemented in the Java language and uses Inversion Of Control (IoC) container – Spring [14]. Although the solid shell is currently still being migrated from Octopus 1, the Octopus 2 MAS layer is complete. The following functionality is provided:

- agent management - creating and distributing agents (top-level or children) at startup or runtime, parallel executing agent threads in a thread pool
- global repository of agents meta-data including current physical locations
- communication - based on asynchronous, prioritized messages addressed only with GUID
- migration - forced or on-demand; based on agent object serialization; transactional migration protocol with support for optional policies that control migrating-out of the source node and migrating-in to the target one

Developed communication and migration mechanisms have been proven to perform well, even in cases when nodes are highly loaded with agents that communicate and migrate intensively [7].

3.3. Agent definition (Belief-Desire-Intention)

The computational agent can be defined in terms of its Believes, Desires and Ambitions [15].

To run efficiently, agents in our application do not require much knowledge (believes) about the environment, apart from that provided by the MAS. However, agents are conscious of the global state of the realization of the common goal (but not necessary of details on local agent states) and able to localize their neighbors. Moreover, the SSS requires agents to be capable of forecasting their future resources requirements and communication plans.

The hierarchy of agents' desires is quite simple:

1. stay alive
2. accumulate resources sufficient for task execution
3. compute a solution of the problem
4. compute a solution of the problem with the desired precision

First two of these are controlled by the MAS platform. The other two, which are task-specific, lay in the responsibility of the FEM application. To realize the above desires, agents can demonstrate one of the following intentions: sending a message, requesting migration to a given node and running a stage of its computational task.

4. Task definitions

4.1. Introduction

Like in classical solutions we base our application mainly on a domain decomposition. Each agent manages its own part of the problem description and the computational mesh. We distinguish three types of agent tasks: a Slave task, a Master task and a Root task. To allow for great flexibility and, as a result, for an efficient load balancing, these tasks are capable of dividing themselves upon a request from SSS. Each agent is identified and referred to by a unique id (an integer in our code) rather than its physical location. The platform can be always assumed to be able to localize an agent by its id.

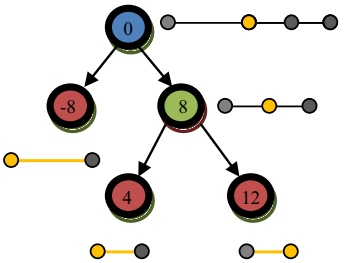

Fig 4 – Exemplary distribution of a computational mesh between 5 tasks: one Root task (blue), one Master task (green) and three Slave tasks (red). Mesh parts managed by a specific task are marked with yellow.

4.1.1. Actions

Each agent runs sequentially its actions (which are connected to the computational application algorithm rather than the system itself). It is not required for the whole action to be executed at once – it may yield after performing a stage of computation. After an action is finished, the task requests the Action Switcher component for the next action to be run. This decision is based on the preceding action of this task and its internal state (expressed by its flags).

4.1.2. Communication

To be compliant with the multi-agent paradigm we restrict our communication to messaging mechanism provided by the Octopus platform. We observed that the overall adaptive algorithm requires only a few patterns of communication – called here ***skeletons***. We implemented the following skeletons:

- ***Gather skeleton*** – Slaves send their data directly to Root; Root performs an operation as soon as it receives data from all Slaves; e.g.: results for outputting (done at Root)
- ***Scatter skeleton*** – the reverse: Root distributes data between Slaves; Slave performs an operation when ready; e.g.: computed global max error decrease rate propagation
- ***Merge skeleton*** – Slaves send data to their parents (usually Masters), Masters merge received data and send merged data to their parents and so on, until the Root receives and merges data from its children; e.g.: Gaussian elimination in Schur complement-based solver

- ***Split skeleton*** – the reverse: Root splits initial data into two parts and propagates it to its children (usually Masters), Master splits received data into two parts and propagates it to its children and so on until Slaves receive finally split data; e.g.: backward substitution in Schur complement-based solver

4.2. Master task description

A Master manages an interface between two neighboring collections of elements. It participates only in two actions, repeated one after another:

- ***Master Eliminate action*** *(responsible for sections #2 and #3 of alg. 2)* – Merges Schur complements received from children. Performs Gaussian elimination on the independent part of the merged matrix. Sends the dependent part of the equation (the Schur complement of the merged matrix) up the hierarchy along the Merge communication skeleton.
- ***Master Substitute action*** – Receives a solution for the dependent part of the equation from a Split. Performs backward substitution on the remaining part of the equation using already determined parts of solution. Forwards the more complete solution down the Split hierarchy.

4.3. Slave task description

A Slave task manages several finite elements and the part of equation associated with this part of the domain. This task contains a few actions:

- ***Slave Eliminate action*** *(responsible for section #1 of alg. 2)* – Generates a local linear equation using the Galerkin method. Performs Gaussian elimination on the independent part of the matrix. Sends the dependent part of the equation (the Schur complement) up the hierarchy along the Merge communication skeleton.
- ***Slave Substitute action*** – Receives a solution for the dependent part of the equation from a Split. Performs backward substitution on the remaining (local) part of the equation using already determined parts of solution.
- ***Slave Refine action*** *(responsible for section #1 of alg. 1)* – Stores the old mesh and the old solution in task-local variables. Performs a deep clone of the mesh. Divides each element of the mesh into two new elements. Inserts additional shape function on each new element. Does not perform any communication.
- ***Slave Error Estimate action*** *(responsible for section #2 of alg. 1)* – Computes an error decrease rate on each element. Determines the maximum error decrease rate. Sends its local max error decrease rate along the Merge skeleton.
- ***Slave Adapt action*** *(responsible for section #3 of alg. 1)* – Receives the globally maximal error decrease rate as a subject of Split pattern. For each coarse mesh element determines, whether to keep the old element or to replace it with two child elements, based on the error rate on it.
- ***Slave Output action*** *(responsible for section #4 of alg. 1)* – Computes the value of solution on its part of the mesh. Sends the corresponding points upwards using Gather skeleton.

The above actions are chosen according to the hp-FEM Slave Action Switcher algorithm, presented on fig. 5.

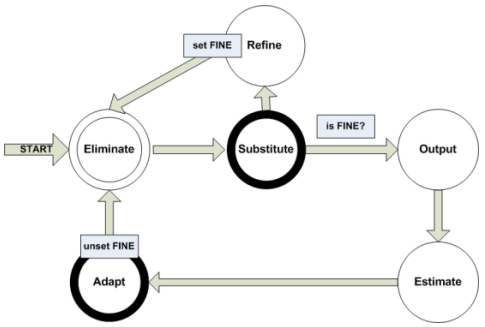

Fig. 5 – state transition diagram for the Slave lifecycle; labels on edges donate special flows of execution

4.4. Root task description

Similarly to a Master, a Root manages a part of interface but also performs some additional operations like collecting and outputting the global solution. Its actions include (see fig. 6 for transitions) :

- ***Root Eliminate action*** *(responsible for sections #2 and #3 of alg. 2)* – Merges Schur complements received from children. Performs Gaussian elimination on the whole of the merged equation.
- ***Root Substitute action*** – Performs backward substitution on the merged matrix. Sends the solution down using the Split skeleton.
- ***Root Error Estimate actio***n – Calculates the global maximum of error decrease rates.
- ***Root Adapt action*** – Propagates the globally maximal error decrease rate using the Scatter skeleton.
- ***Root Output action*** *(responsible for section #4 of alg. 1)* – Receives solution points within the Gather pattern. Puts the collected data, along with the iteration number in a file for further visualizing.

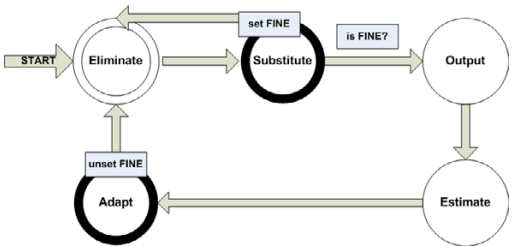

Fig. 6 – state transition diagram for the Root lifecycle; labels on edges donate special flows of execution

The operation of global algorithm is finished upon the signal from the Root. When it discovers that the solution is precise enough, it asks Slaves for the final solution parts to be printed, outputs them and finishes operation.

4.5. Task division algorithm

Although Master and Root tasks tend to be reasonably small (in terms of processing time and memory consumption), a Slave task can easily grow too much as a result of mesh adaptation. In this case the Solid Shell grain control mechanism may demand it to divide itself. Such an operation consists of creating two new Slave tasks and exchanging the old Slave task to a Master task, which handles an interface between the children (e.g. fig. 7).

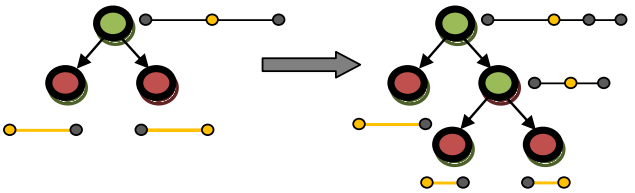

Fig. 7 – Task reproduction upon a request form the granulation-control platform mechanism

If the SSS supported custom task size metrics, the condition for division could be (implemented currently, before Octopus II contains a complete Smart Solid layer), `task.managed_dof > dof_limit`, where `managed_dof` donates the number of degrees of freedom within the lastly computed solution and `dof_limit` is an arbitrary *degrees of freedom per task* limit. Otherwise the decision can be made based on the resources consumption by the SSS like in [4].

5. Exemplary problem

5.1. Brief problem formulation

Step and flash imprint lithography (SFIL) is a patterning process utilizing photopolymerization to replicate the topography of a template onto a substrate [9]. The major processing steps of SFIL include: depositing a low viscosity, silicon containing, photocurable etch barrier on to a substrate; bringing the template into contact with the etch barrier; curing the etch barrier solution through UV exposure; releasing the template, while leaving high-resolution features behind; a short, halogen break-through etch; and finally an anisotropic oxygen reactive ion etch to yield high aspect ratio, high resolution features. Photopolymerization, however, is often accompanied by densification. Densification of the SFIL photopolymer (the etch barrier) may affect both the cross sectional shape of the feature and the placement of relief patterns. We focus on modeling the feature inside the template, after the densification occurred. Occasionally, it happens that during the process, the substrate gets damaged and thus interparticle forces become weaker in one part of the domain. Such a case is presented on fig. 8.

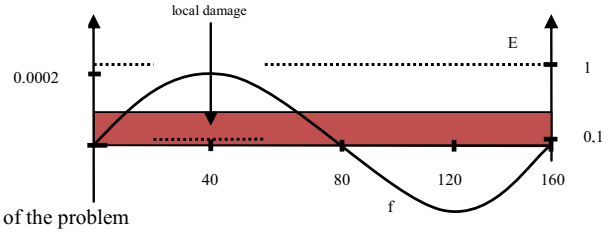

Fig. 8 – Exemplary domain of the problem

Molecular dislocations in the substrate (u) are donated by the solution of the following equation (weak form):

$$\int_0^{160} EAu'v'dx = \int_0^{160} fv\,dx \quad \text{such that } u(0)=0 \text{ and } u(160)=0$$

for each v form the space of base functions (see section 2.1), where E represents Young modulus, A donates cross-sectional area of substrate, and f is a load force – an effect of neighbor layers pressure.

5.2. Operation (case study)

Initially, two Slaves (with ids: -8 and 8) and one Root task (id: 0) were employed to solve the problem. Upon startup, the SSS layer discovered that the task number 8 was overloaded (see section *Task division algorithm*), so it requested this task to divide into pieces (see fig. 7). Two new Slave tasks (number 4 and 12) were created. Later on, the adaptation being most intensive on the left part of the domain (task -8), the SSS requested additional 3 divisions, ensuring fine grain of computations.

5.3. Numerical results

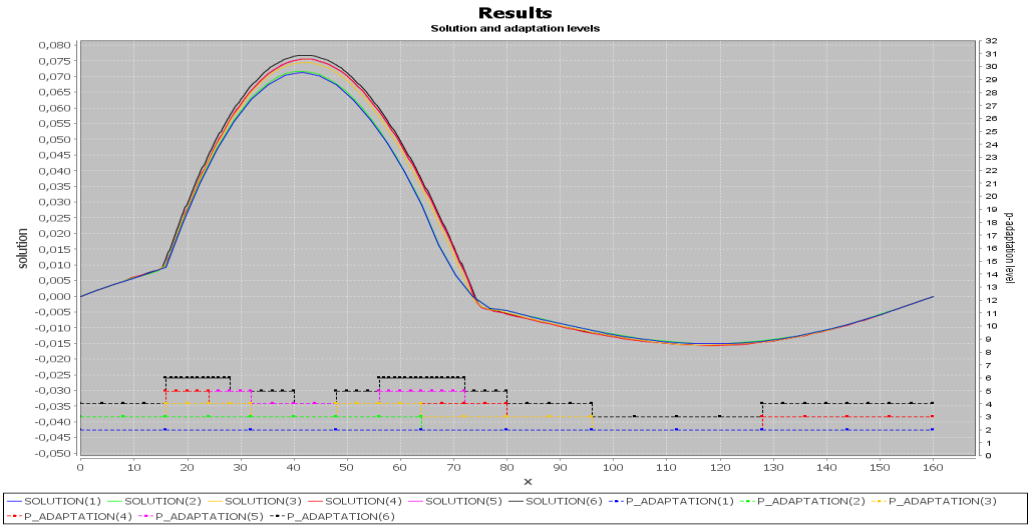

Fig. 9 – numerical results involving 6 iterations of the hp-adaptive algorithm; the actual solution is shown in the upper part of the figure, while the p-adaptation level of elements is shown on the lower part; markers on the lower part donate ends of elements (and thus illustrate the process of the h-adaptation: the denser the markers are placed the higher order of the h-adaptation is)

As the result of running agent-based hp-FEM for the problem described above, we received the solution presented on fig. 9. Target error decrease rate (computed in terms of absolute error in H^1) (0.015) was achieved in 6 iterations (black line).

6. Conclusions and future work

In this article we presented a modern approach to designing parallel systems for solving PDEs using the example of hp-FEM. We applied the agent formalism to a typical parallel application to simplify the parallel structure and abstract from technical aspects, which tend to jam down the actual logic in classical solutions. Conforming to the SSA model we delegated tasks such as grain control or load balancing to the agent platform. We employed tools well established in business applications, to ensure that the code is easily manageable and widely understandable.

However, the paper lacks performance measurements as the 1D hp-FEM requires too little computations, to evaluate processing time or scalability and compare with more established solutions. For such experiments, we are going to develop a 2D p-adaptive FEM application. This will also allow us to determine optimal coefficients of agents operation (i.e. the threshold for task reproduction) and to develop strategies of predictive denominating of resources required by a task, which lay at the basis of an effective diffusion-based scheduling.

Acknowledgements

The work reported in this paper has been supported by Polish MNiSW grants NN 501 120836 and NN 519 405737.

References

[1] L. Demkowicz; *Computing With Hp-adaptive Finite Elements*; Chapman & Hall CRC, 2006
[2] M. Grochowski, R. Schaefer, M. Smołka; *Architectural Principles and Scheduling Strategies for Computing;* Fundamenta Informaticae, vol. 71; IOS Press 2006, pp. 15–26
[3] M. Grochowski, R. Schaefer, P. Uhurski; *Diffusion Based Scheduling in the Agent-Oriented Computing System*; Lecture Notes in Computer Science, vol. 3019; Springer 2004, pp. 97–104
[4] J. Momot, K. Kosacki, M. Grochowski, P. Uhruski, R. Schaefer; *Multi-Agent System for Irregular Parallel Genetic Computations*; Lecture Notes in Computer Science, Vol. 3038, Springer 2004, pp. 623–630.
[5] M. Grochowski, R. Schaefer, P. Uhruski; *OCTOPUS – Computation Agents Environment*; Inteligencia Artificial, vol. 9, Polytechnic University of Valencia, 2005, no. 28, pp. 55–62.
[6] P. Gurgul, M. Sieniek, M. Paszyński; *Object-oriented Multiscale HP-Adaptive Finite Element Method*; Computer Methods In Material Science, vol. 9; Akapit 2009, no. 2 pp. 289-295
[7] P. R. B. Devloo; PZ: An object oriented environment for scientific programming, Computational Methods Applied Mechanics and Engineering; 150 (1997), pp. 133-153
[8] M. Paszyński, D. Pardo, C. Torres-Verdin, L. Demkowicz, V. Calo; *A Parallel Direct Solver for Self-Adaptive hp Finite Element Method,* Journal of Parallel and Distributed Computing, 2009 in press.
[9] T. Bailey, B. Smith, B. J. Choi, M. Colburn, M. Meissl, S.V. Sreenivasan, J. G. Ekerdt, C. G. Willson; *Step and flash imprint lithography: Defect analysis*; J. Vac. Sci. Technol. B., 19(6), 2001, pp. 2806-2810.
[10] F. Zhang; *The Schur Complement and Its Applications*; Springer 2005
[11] G. Weiss; *Multiagent Systems – A Modern Approach to Distributed Artificial Intelligence*, 1999
[12] M. Paszyński, J. Kurtz, L. Demkowicz, *Parallel Fully Automatic hp-Adaptive 2D Finite Element Package*; Computer Methods in Applied Mechanics and Engineering, 195, 7-8, 25, pp. 711-741; 2006
[13] M. Paszyński; *Graph Grammar-Driven Parallel Adaptive PDE Solvers*; Polish Academic Publishing House for Science and Didactics 2009
[14] R. Johnson et al.; *The Spring Framework - Reference Documentation (v.2.5.5)*, 2008; http://static.springframework.org/spring/docs/2.5.5/reference/index.html
[15] A. S. Rao, M. P. Georgeff.; *Modeling Rational Agents within a BDI-Architecture*; Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning; 1991