UDC 681.234

A.V. DERGUNOV

Lobachevsky State University of Nizhny Novgorod

SPECIFICATION AND AUTOMATIC DETECTION OF PERFORMANCE PROBLEMS IN MESSAGE PASSING (MPI) APPLICATIONS

Traditional way to analyze performance of message passing (MPI) applications is via visualization of their execution traces. Several tools were developed to aid this activity; one example of such tool is Jumpshot. However, performance analysis with such tools is a complex task due to large traces and complex interactions between processes. In this paper a new way to analyze performance is proposed by automatic detection of performance problems in message passing applications. Performance problem is defined as a set of actions that inhibit good performance and is specified using tracing and analysis rules.

***Keywords**: performance problems specification, performance problems detection, message passing applications.*

Introduction

Performance properties are very important for parallel applications, so several tools were developed to analyze performance. Most popular performance analysis tools for MPI applications perform visualization of execution traces. Example of such tool is Jumpshot [1] which performs visualization using timeline view. But the task of performance analysis using such tools is quite complex, because trace files usually consists of many events and interaction between events is very complex. Performance analysis also requires expert knowledge about MPI implementation.

This paper proposes a new way to do performance analysis which is based on expert methodology of performance problems descriptions.

1. Model of an MPI application

An MPI application [2] is defined as a set of communicating processes $PR = \{PR_1, ..., PR_N\}$. Communication is performed using initiating actions in defined sequence:

$$PR_i \Rightarrow a_{i1}, ..., a_{iM_i}; i = 1, ..., N.$$

Every action is a call of a function defined by MPI standard $F = \{f_k\}; k = 1, ..., K$. Each function f_k has input and output arguments:

$$FA_k^{IN} = (fa_{k1}^{IN}, ..., fa_{kL_k^{IN}}^{IN})$$

$$FA_k^{OUT} = (fa_{k1}^{OUT}, ..., fa_{kL_k^{OUT}}^{OUT}).$$

Thus, every action is represented by the calling function and the values of input and output arguments:

$$a_{ij} = \left\langle f_k, FAval_k^{IN}, FAval_k^{OUT} \right\rangle;$$

$$FAval_k^{IN} = (av_{k1}^{IN}, ..., av_{kL_k^{IN}}^{IN});$$

$$FAval_k^{OUT} = (av_{k1}^{OUT}, ..., av_{kL_k^{OUT}}^{OUT});$$

$$i = 1, ..., N; j = 1, ..., M_i; f_k \in F.$$

Every action a_{ij} has start time t_{ij}^a and duration d_{ij}^a. Loss of performance due to communication is defined as:

$$D^a = \sum_{i=1}^{N} \sum_{j=1}^{M_i} d_{ij}^a.$$

The task of performance improvement is defined as minimization of this value.

2. Model of a performance problem

Performance problem is defined as a set of actions that inhibit good performance, because the actions are not synchronized. Fig. 1 shows synchronous execution of send and corresponding receive actions which results in good performance. Fig. 2 and fig. 3 show cases when send and receive actions are not synchronized thus producing late sending and late receiving problems correspondingly.

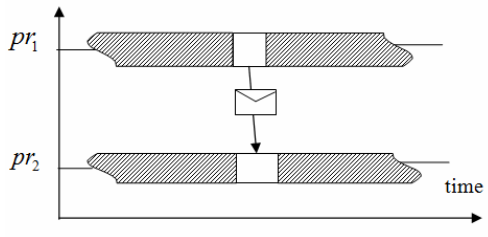

Fig. 1. Synchronous send and receive actions

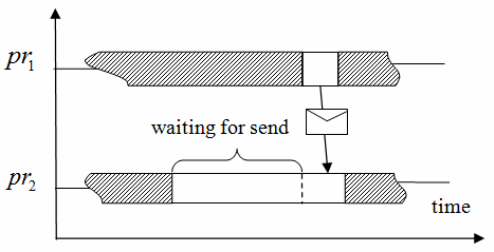

Fig. 2. Problem of late sending

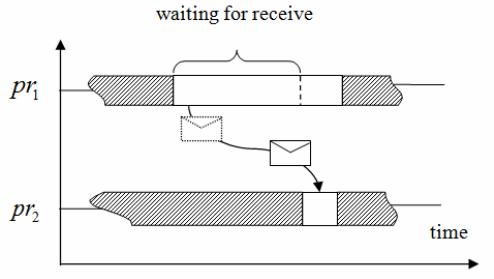

Fig. 3. Problem of late receiving

Formally *performance problem* is defined as:

$$\text{pb} = \left\langle \begin{matrix} \text{pd}, \text{dur}, \text{TRRULES}, \\ \text{ANRULES}, \text{REC}(\text{A}^{\text{INFO}}) \end{matrix} \right\rangle,$$

where pd – textual description of the problem; dur – duration of the problem; TRRULES – trace rules for actions that introduce the problem; ANRULES – analysis rules to recognize the problem in sequence of events in trace file; REC – recommendations to fix the problem; $\text{A}^{\text{INFO}} = \{\langle \text{f}_i, \text{t}_i, \text{d}_i, \text{pr}_i, \text{cs}_i \rangle\}$ – description of actions that introduced the problem (where f_i – the function that was called, t_i – time when the function was called, d_i – duration of the function execution, pr_i – process that initiated the call, cs_i – the call site represented, for example, by source file name and line number in MPI application).

3. Performance Expert system

The task of automatic detection of performance problems in MPI applications turns out to be quite complex and it is hard to solve it using formal methods. Thus, expert methodology was proposed and implemented in Performance Expert system.

The workflow of Performance Expert system is illustrated in fig. 4.

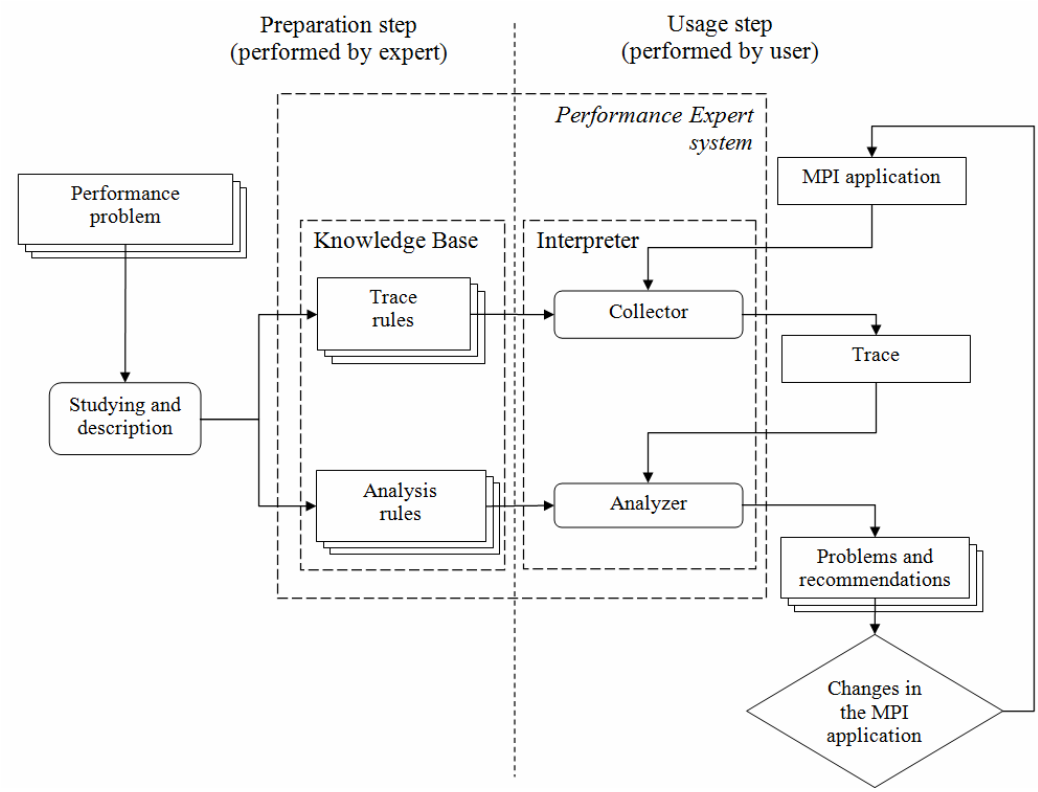

Fig. 4. Performance Expert system

The usage of the system supposes two steps:

Preparation step. This step is performed by expert in performance analysis of MPI applications. The task of the expert is to describe performance problems that she encounters using *trace and analysis rules*. The trace and analysis rules are described using languages described later in this paper and they constitute *Knowledge Base* of the system which represents knowledge about all performance problems known so far. At the moment Knowledge Base of Performance Expert contains description of 10 typical performance problems of MPI applications [3].

Usage step. This step is performed by user. She executes her MPI application under *collector*. Collector uses trace rules to produce trace file containing events that correspond to actions executed by MPI application. Then the trace file is examined by *analyzer*. Analyzer detects performance problems in MPI application using analysis rules and generates recommendations for performance improvement. The user is able to follow the recommendations and implement the changes in her MPI application and then repeat usage step until good performance is achieved.

The data flow in Performance Expert system in illustrated in fig. 5:

1. MPI application executes a sequence of actions. Trace rules are used to describe which events should be generated and saved in the trace file.
2. Analysis of the trace file is performed in two steps:

a) composite events are constructed from simple events of the trace file. Composite event construction rules are used to perform this step;

b) performance problems are identified among the constructed composite events. Performance problem detection rules are used to perform this step. If a problem is identified, recommendations are produced which are related to the exact locations in source code of MPI application (this is done by extracting information about actions that produced the problem).

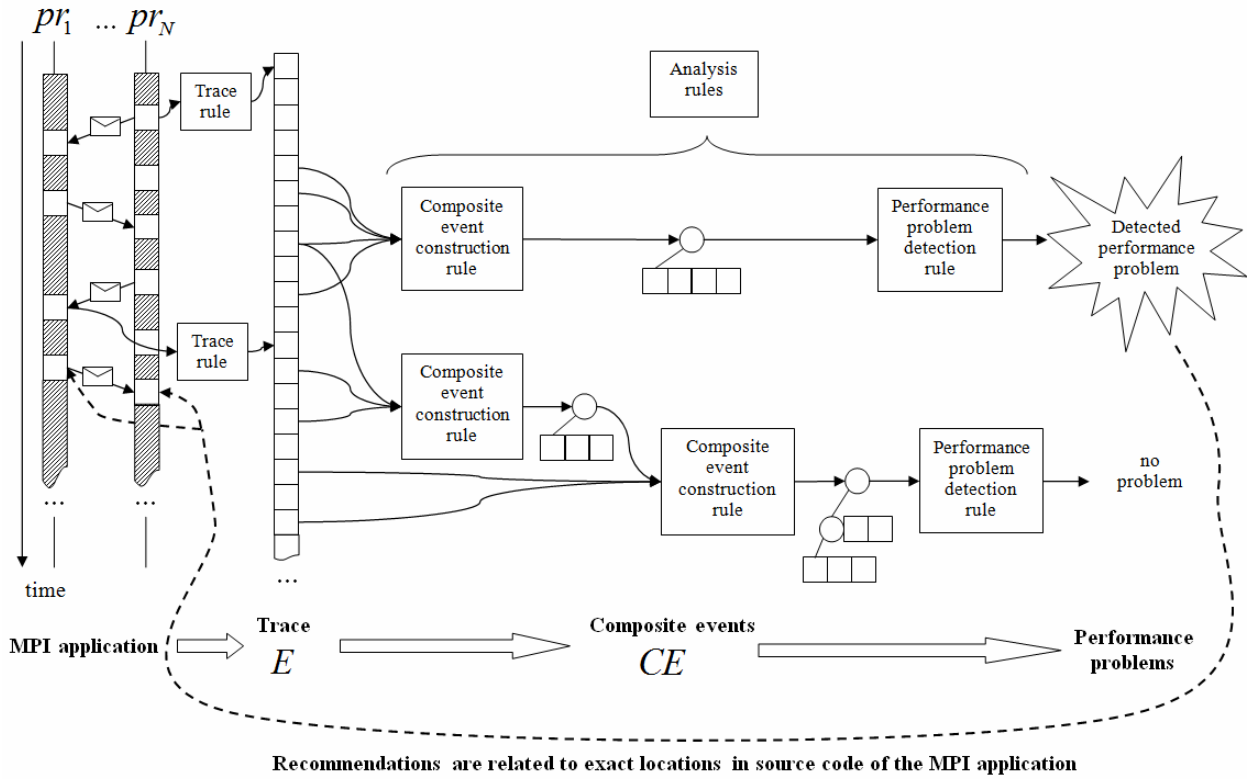

Fig. 5. Data flow in Performance Expert system

4. Tracing model

Simple events (or just events) are stored in trace file generated by collector. Each simple event is represented as:

$$e = \langle f, et, EPval, t, d, pr, cs \rangle,$$

where:

$f \in F$ – function that was called;

et – type of event which describes event parameters $EP = (ep_1, ..., ep_K)$;

$EPval = (v_1, ..., v_K)$ – values of event parameters;

t – event occurrence time;

d – event duration;

pr – process which generated event;

cs – call site.

Trace rule is represented as:

$$a \xrightarrow{trrule} e;$$

$$a = \langle f, FAval^{IN}, FAval^{OUT} \rangle;$$

$$e = \langle f, et, EPval, t, d, pr, cs \rangle;$$

$$trrule = \langle f, et, EPtemp \rangle,$$

where:

$f \in F$ – function of the rule, a call to that function produces described event;

et – type of event which is generated as a result of the function call;

$\text{EPtemp} = \{\langle \text{ep}_i, \text{Expr}_i, \text{kind}_i \rangle\}$ – description of generated event parameters and the way to calculate their values, where:

ep_i – event parameter;

$\text{Expr}_i : \text{FAval}^{\text{IN}} \times \text{FAval}^{\text{OUT}} \to \text{v}_i$ – function to calculate value of event parameter (using argument values of the called function);

$\text{kind}_i \in \{\text{in}, \text{out}\}$ – kind of that parameter (in – calculated before function call; out – after the call).

An XML-based language was developed to declare trace rules. Trace rules are used by collector generator to produce wrappers of original functions. Function wrappers are used by PIN system [4] for dynamic instrumentation of MPI applications (PIN is a third-party component, a framework for dynamic instrumentation).

5. Analysis model

Composite event is represented as:

$$\text{ce} = \langle \text{cet}, \text{CEPval}, \text{ME} \rangle,$$

where:

cet – type of composite event which describes parameters $\text{CEP} = (\text{cep}_1, ..., \text{cep}_K)$;

$\text{CEPval} = (\text{cev}_1, ..., \text{cev}_K)$ – values of composite event parameters;

$\text{ME} = \{\text{e}_i\} \cup \{\text{ce}_j\}; \text{i} = 1, ..., \text{N}; \text{j} = 1, ..., \text{M}$ – set of simple and composite events that are members of this composite event.

Composite event construction rule is:

$$\left.\begin{array}{l} \text{E} \xrightarrow{\text{ET}} \text{E}^{\text{R}} \\ \text{CE} \xrightarrow{\text{CET}} \text{CE}^{\text{R}} \end{array}\right\} \xrightarrow{\text{cerule}} \text{ce} = \langle \text{cet}, \text{CEPval}, \text{ME} \rangle;$$

$$\text{cerule} = \langle \text{ET}, \text{CET}, \sigma, \text{cet}, \text{CEPtemp}, \text{ET}^{\text{S}} \rangle,$$

where:

$\text{ET} = \{\text{et}_i\}$ – set of simple event types to select a subset of relevant simple events:

$$\text{E}^{\text{R}} = \{\langle \text{f}_i, \text{et}_i, \text{EPval}_i, \text{t}_i, \text{d}_i, \text{pr}_i, \text{cs}_i \rangle\} \subseteq \text{E};$$

$\text{CET} = \{\text{cet}_j\}$ – set of composite event types to select a subset of relevant composite events:

$$\text{CE}^{\text{R}} = \{\langle \text{cet}_j, \text{CEPval}_j, \text{ME}_j \rangle\} \subseteq \text{CE}.$$

Let us denote:

$$\text{EPV}_i = \langle \text{f}_i, \text{EPval}_i, \text{t}_i, \text{d}_i, \text{pr}_i, \text{cs}_i \rangle;$$

$$\text{P} = \text{EPV}_1 \times ... \times \text{EPV}_N \times \text{CEPval}_1 \times ... \times \text{CEPval}_M;$$

$\sigma : \text{P} \to \{1, 0\}$ – Boolean condition for constructing the composite event;

cet – type of the composite event to compose;

$\text{CEPtemp} = \{\langle \text{cep}_k, \text{Expr}_k \rangle\}$ – description of the composite event parameters and the way to calculate their values, where:

cep_k – parameter of the composite event;

$\text{Expr}_k : \text{P} \to \text{cev}_k$ – function to calculate parameter value;

If rule conditions are satisfied (i.e., subsets of relevant events ET^{R} and CET^{R} of the specified types exist and Boolean condition for their parameters σ is satisfied), then composite event ce of type cet is constructed, where:

$\text{CEPval} = (\text{cev}_1, ..., \text{cev}_K)$ – parameter values;

$\text{ME} = \text{E}^{\text{R}} \cup \text{CE}^{\text{R}}$ – set of member events.

Rule parameter $\text{ET}^{\text{S}} \subseteq \text{ET}$ describes event types which are common for the constructed composite event and other event types.

Performance problem detection rules is:

$$\text{ce} \xrightarrow{\text{pbrule}} \text{pb};$$

$$\text{ce} = \langle \text{cet}, \text{CEPval}, \text{ME} \rangle;$$

$$\text{pb} = \langle \text{pd}, \text{dur}, \text{REC}(\text{A}^{\text{INFO}}) \rangle;$$

$$\text{pbrule} = \langle \text{cet}, \varphi, \text{pd}, \text{RECtemp}, \text{L}_{\text{dur}} \rangle,$$

where:

cet – type of the composite event which may represent performance problem;

$\varphi : \text{CEPval} \to \{1, 0\}$ – Boolean condition of the problem occurrence;

pd – textual description of the problem;

$\text{RECtemp} : \text{CEPval} \to \text{REC}$ – recommendation template (A^{INFO} is the description of actions that produced the problem and it is generated by extracting information from all simple events contained in the composite event recursively);

$\text{L}_{\text{dur}} : \text{CEPval} \to \text{dur}$ – function to calculate duration of the problem.

Languages were developed to declare composite event construction rules and performance problem detection rules. Performance Expert system uses CLIPS [5] expert system tool to perform the analysis, so internally these rules are converted into CLIPS rules.

6. Experiment

To investigate the implemented system an experiment was conducted to analyze and improve performance of MPI application which models heart activity [6]. Cells of heart comprise NxN lattice and each cell is

connected to the nearest neighbors. Each cell is described by differential equations and the application performs numerical integration.

The lattice is split into smaller parts MxM (where M < N) which are distributed among processes. Each process performs the following actions in cycle:

1. Calculate values in lattice points.
2. Exchange values on boarders with neighboring processes using MPI_Sendrecv.

After numerical integration is done, the calculated data is sent to the main process using MPI_Gather.

Analysis of this application by Performance Expert system in automatic mode revealed the following performance problems:

- *Late sending*. Cumulative duration of such problems is 28,02% of total execution time.
- *Early receive for "many-to-one" operation*. Cumulative duration is 24,16%.

The second problem relates to application implementation details (main processor initiates call to MPI_Gather which is waiting until data is sent by other processes). So, it was decided to fix the first problem. Performance Expert system provided recommendation to use non-blocking receive operations, so the cycle in application was rewritten:

1. Calculate values in boarder lattice points.
2. Send values on boarders to neighboring processes using MPI_Send.
3. Initiate receive of values on boarders from neighboring processes using MPI_Irecv in non-blocking mode.
4. Calculate values in the rest lattice points.
5. Wait until receiving of values is finished using MPI_Wait.

Fig. 6 shows achieved performance improvements. The best improvement (1,62x) was achieved for 101 processes.

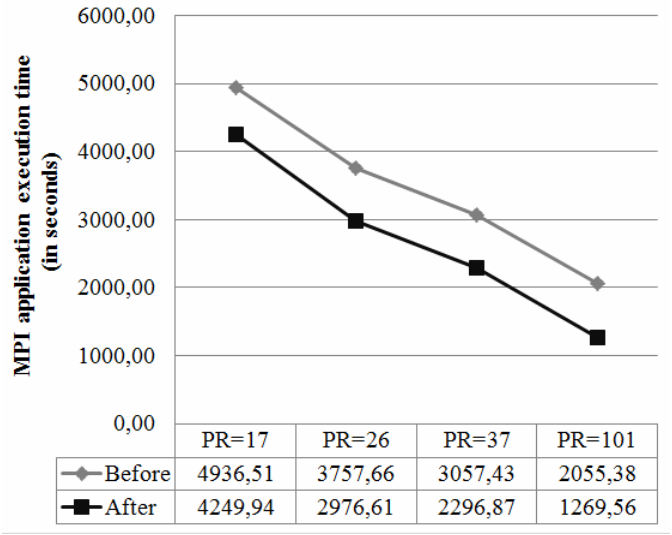

Fig. 6. Performance improvement results (PR denotes processes number)

Data was collected on cluster of Lobachevsky State University of Nizhny Novgorod (it consists of dual-core Intel Xeon 5150 2.66 GHz cores, 4 GB memory, Gigabit Ethernet, Windows Server 2008 x64, Microsoft implementation of MPI library).

Conclusion

A new way to analyze performance is proposed by automatic detection of performance problems in message passing applications. Experiment conducted on a real MPI application shows validity of this approach.

References

1. Toward Scalable Performance Visualization with Jumpshot [Текст] / O. Zaki [et al.] // High-Performance Computing Applications. – 1999. – Vol. 13, № 2. – P. 277 – 288.

2. MPI: A Message-Passing Interface Standard [Электронный ресурс]. – Режим доступа: http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html. – 28.08.2011 г.

3. Дергунов, А.В. Автоматизация выявления причин потери производительности MPI программ на экзафлопсных и других больших суперкомпьютерах [Текст] / А.В. Дергунов // Научный сервис в сети Интернет: экзафлопсное будущее: труды Междунар. суперкомпьютерной конф. – М.: Изд-во МГУ, 2011. – С. 491 – 496.

4. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation [Текст] / C.-K. Luk [et al.] // Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation, Chicago, IL, 2005. – P. 190 – 200.

5. Джарратано, Дж. Экспертные системы: принципы разработки и программирование / Дж. Джарратано, Г. Райли; пер. с англ. К. А. Птицына. – 4-е изд. – М.: Вильямс, 2007. – 1152 с.

6. Моделирование сердечной активности / Г.В. Осипов [и др.] // Суперкомпьютерные технологии в науке, образовании и промышленности. – М.: Изд-во Моск. ун-та, 2010. – С. 35 – 40.

Поступила в редакцию 17.02.2012

Рецензент: д-р техн. наук, ст. наук. співр. В.М. Опанасенко, Інститут кібернетики ім. В.М. Глушкова, Київ, Україна.

ОПИСАНИЕ И АВТОМАТИЧЕСКОЕ ВЫЯВЛЕНИЕ ПРОБЛЕМ ПРОИЗВОДИТЕЛЬНОСТИ В MPI ПРИЛОЖЕНИЯХ

А.В. Дергунов

Традиционно для анализа производительности MPI приложений используются программные средства для визуализации трассы их работы. Для выполнения этой задачи разработано несколько инструментов, примером является Jumpshot. Но анализ производительности с использованием таких инструментов является сложной задачей из-за больших размеров трасс и сложных взаимодействий процессов. В работе предложен новый подход к анализу производительности с использованием автоматического выявления проблем производительности MPI приложений. Под проблемой производительности понимается множество действий, которые негативно влияют на производительность программы. Проблемы производительности описываются с использованием правил трассировки и анализа.

Ключевые слова: описание и выявление проблем производительности, MPI приложения.

ОПИС І АВТОМАТИЧНЕ ВИЯВЛЕННЯ ПРОБЛЕМ ПРОДУКТИВНОСТІ В MPI ДОДАТКАХ

А.В. Дергунов

Традиційно для аналізу продуктивності MPI додатків використовуються програмні засоби для візуалізації траси їх роботи. Для виконання цього завдання було розроблено декілька інструментів, наприклад Jumpshot. Однак аналіз продуктивності з використанням цих інструментів є складним завданням через великий розмір трас та складних взаємодій процесів. У роботі запропонований новий підхід до аналізу продуктивності MPI додатків. Під проблемою продуктивності розуміється певні дії, котрі негативно впливають на продуктивність програми. Проблеми продуктивності описуються з використанням правил трасування та аналізу.

Ключові слова: опис і виявлення проблем продуктивності, MPI додатки

Дергунов Антон Владимирович – аспирант кафедры математического обеспечения ЭВМ Нижегородского государственного университета им. Н.И. Лобачевского, Нижний Новгород, Россия, e-mail: anton.dergunov@gmail.com.