Measuring the Quality of an Artificial Hormone System based Task Mapping

Uwe Brinkschulte
Institute for Process Control, Automation, and Robotics
University of Karlsruhe (TH), Germany
brinks@ira.uka.de

Alexander von Renteln
Institute for Process Control, Automation, and Robotics
University of Karlsruhe (TH), Germany
renteln@ira.uka.de

Mathias Pacher
Institute for Process Control, Automation, and Robotics
University of Karlsruhe (TH), Germany
pacher@ira.uka.de

ABSTRACT

The idea of Organic Computing is a trend to counter the problems arising from the fact that computing systems are getting smaller and smaller, and we will soon be surrounded by large numbers of little computers which will be hard to configure, maintain, and control.

We reintroduce an organic middleware - the artificial hormone system (AHS) which can map tasks onto a grid of heterogeneous processing elements while providing the system with self-X properties and even guaranteeing upper bounds for the self-configuration and self-healing.

This paper investigates the quality of task mappings on a grid of heterogeneous processing elements.

An algorithm is proposed to measure the quality of such task mappings. Experiments with randomly generated configurations will show results of mappings done by our artificial hormone system and compare them with ordinary load balancing.

Categories and Subject Descriptors

C.3 [**Computer Systems Organization**]: Special-Purpose And Application-Based Systems—*Real-time and embedded systems*; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*; D.2.10 [**Software Engineering**]: Design—*Methodologies*

General Terms

Algorithms, Measurement

Keywords

artificial hormone system, real-time task mapping, decentralized control loops

1. INTRODUCTION

Today's computational systems are growing increasingly complex. They are build from large numbers of heterogeneous processing elements with highly dynamic interaction. Middleware is a common layer in such distributed systems, managing the cooperation of tasks on the processing elements and hiding the distribution from the application. It is responsible for seamless task interaction on distributed hardware. All tasks are interconnected by the middleware layer and are able to operate beyond processing element (PE) boundaries as if they would reside on a single hardware platform. To handle the complexity of today's – furthermore tomorrow's – distributed systems, self-organization techniques are necessary. The idea to autonomously achieve this desired behaviour is introduced in several papers [10, 15, 11]. Such a system should be able to find a suitable initial configuration by itself, to adapt or optimize itself to changing environmental and internal conditions, to heal itself in case of system failures or to protect itself against attacks.

Middleware is well-suited to realize such self-X features (self-configuration, self-optimization, self-healing) by autonomously controlling and adapting task allocation. Especially for self-healing it is important that task allocation is decentralized to avoid single points of failure.

In earlier publications we introduced our artificial hormone system (AHS) for task mapping on heterogeneous processing elements [3, 4]. The term "artificial hormone system" was chosen, because our approach was highly inspired by the hormone system of higher animals. Several comparable properties are between the hormone system in biology and our technical system. However it has to be stated that our "artificial hormone system" is not a copy of the biological hormone system, but rather inspired by nature and its strategies. In biology, hormones are chemical objects transmitted via chemical processes and reactions. In our approach, the messengers are messages transferred via communication links. However, the effects and principles are similar, hence we dubbed the messengers in our approach "hormones" as well.

A striking problem is that of measuring the behavior of non-deterministic task-scheduling algorithms such as our proposed hormone-based approach. Starting with common metrics, we present quality metrics for such algorithms. The paper is structured as follows: we begin with the related work in Section 2. Then we introduce our quality measurement in Section 4. In Section 5 we present and discuss the test series. The paper finishes with a conclusion in Section 6.

2. RELATED WORK

Several approaches for clustered task allocation in middleware exist. In [2], the authors present a scheduling algorithm distributing tasks onto a grid. It is implemented in the Xavantes Grid Middleware and arranges the tasks in groups. This approach is completely different from ours because it uses central elements for the group-

AUTONOMICS 2008, September 23-25, Turin, Italy

DOI 10.4108/ICST.AUTONOMICS2008.4662

ing: the Group Manager (GM), a Process Manager (PM) and the Activity Managers (AM). Here, the GM is a single point of failure because, if it fails no possibility exists to get group information from this group any more. Our approach does not apply a central task distribution instance and therefore a single point of failure can not occur.

Another two approach are presented in [12]. The authors present two algorithms for task scheduling. The first algorithm, Fast Critical Path (FCP), ensures time constrains are kept. The second one, Fast Load Balancing (FLB), schedules the tasks so that every processor will be used. Using this strategy – especially the last one – it is not guaranteed that related tasks are scheduled nearby each other. In contrast to our approach, these algorithms do not regard failing of processing elements.

In [9], a decentralized dynamic load balancing approach is presented. Tasks are considered as particles which are influenced by forces like e.g. a load balancing force (results from the load potential) and a communication force (based on the communication intensities between the tasks). In this approach, the tasks are distributed according to the resultant of the different types of forces. A main difference to our approach is that we are able to provide time bounds for the self-configuration. Besides, our approach covers self-healing which is not considered by this decentralized dynamic load balancing.

[13] presents a load balancing scheme for task allocation based on local work piles (of PEs) storing the tasks to be executed. The authors propose a load balancing algorithm applied to two PEs to balance their workload. The algorithm is executed with a probability inversely proportional to the length of the work pile of a PE. Although this approach is distributed it does not consider aspects like self-healing or real-time constraints.

In [5], several heuristics for dynamic task mapping are proposed. They start with simple heuristics like *First Free* and *Nearest Neighbour* and go on to more powerful heuristics like *Minimum Average Channel Load* and *Path Load*. The authors evaluate the heuristics only considering link load whereas we evaluate our AHS considering more aspects like CPU share, suitability and communication distance.

[14] describes an AHS for self-organization of Networked Nodes. So called digital hormones are used to optimize the task ("service") distribution on nodes and they are piggy-backed on the outgoing communication. Other aspects of organic computing like self-healing which are included in our approach are not addressed here.

Other approaches of load balancing are presented in [1, 6, 7, 8, 16]. None of them cover the whole spectrum of self-X properties, task clustering, and real-time conditions like our approach.

Our artificial hormone system maps the tasks on the processing elements paying attention to the following three aspects: the load of the processing elements, the suitability of a processing element to execute a task and the minimal communication distance between tasks which heavily communicate with each other. The last aspect inspired us to name such tasks which heavily communicate with the term: related tasks.

Because none of the aforementioned task mapping algorithms addresses all 3 aspects of load, suitability and communication distance we therefore cannot perform a proper comparison with our artificial hormone systems. To properly rate the quality aspects, we developed a new way of rating task mappings which will be presented in the following section.

3. THE ARTIFICIAL HORMONE SYSTEM

This chapter gives a short introduction to the Artificial Hormone System (AHS). Its function as well as functionality will be explained.

Given is a grid of heterogeneous processing elements (PEs). The AHS is used to distribute the upcoming tasks to all available processing cells. As the task requirements as well as the PEs are typically not identical, the quality of the assignment of a task on a processing element also differs. This assignment quality can be measured by e.g. the execution time of the task, the consumption of energy or the possibility of the task to be executed on the processing element at all. The execution time e.g. can be influenced by the number of tasks that are executed on the processing element - therefore a good and even distribution of the tasks on the processing elements is also necessary.

To be able to decide where to place the tasks on the processing cell grid, messages have to be exchanged between the processing elements. Messages are used to exchange the information which is necessary for each PE to decide if it is the best suited PE for a special task.

For task allocation, three types of hormones are used:

Eager value: This hormone determines, how suited a PE is to execute a task. The higher the hormonal value, the better a PE can execute the task.

Suppressor: A suppressor represses the execution of a task on a PE. Suppressors are subtracted from eager values. They can be used to limit task execution and to indicate a degrading PE state.

Accelerator: An accelerator favors the execution of a task on a PE. Accelerators are added to eager values. They can be used to cluster related or cooperating tasks in the neighborhood (thus forming organs) or to indicate an improved PE state.

Figure 1 sketches the basic control loop used to assign a task T_i to a processing element. This closed control loop is executed for every task on every processing element. It determines based on the level of the three hormone types, if a task T_i is executed on a processing element PE_γ or not.

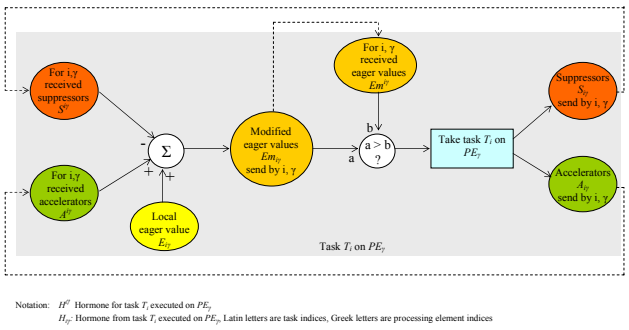

Figure 1: Hormon based control loop

The local static eager value $E_{i\gamma}$ indicates how well task T_i executes on PE_γ. From this value, all suppressors $S^{i\gamma}$ received for task T_i on PE_γ are subtracted and all accelerators $A^{i\gamma}$ received for task T_i on PE_γ are added. The result of this calculation is a modified eager value $Em_{i\gamma}$ for task T_i on PE_γ. The modified eager value is send to all other PEs in the system and compared to the

modified eager values $Em^{i\gamma}$ received from all other PEs for this task. Is $Em_{i\gamma}$ greater then all received eager values $Em^{i\gamma}$, task T_i will be taken by PE_γ (in case of equality a second criterion, e.g. the position of a PE in the grid, is used to get an unambiguous decision). Now, task T_i on PE_γ sends suppressors $S_{i\gamma}$ and accelerators $A_{i\gamma}$ to the others. This procedure is repeated periodically.

In this point, we emphasize that the initial strength of the hormone values is set by the applicants who want to influence the task allocation. The organic middleware evaluates the hormones to allocate the different tasks, but it does not set their initial strength.

3.1 Notations

Now we will define some basic indices and sets which will be used frequently in the following sections. To allow an easy distinction, we use Latin lower case letters for task indices and Greek lower case letters for processing element indices (like already used in figure 1). Accordingly, we use upper case Latin letters for task sets and upper case Greek letters for sets of processing elements. Let

$\mathbf{\Omega}$ be the set of all processing elements in the system.

ω be the number of all processing elements in the system.

$$\omega = |\Omega|$$

$\mathbf{I\Omega}$ be the set of indices of all processing elements.

$$I\Omega := \{1, ..., \omega\}$$

Thus, we obtain the set of all processing elements as

$$\Omega = \{PE_1, ..., PE_\omega\} = \{PE_\gamma \mid \gamma \in I\Omega\}.$$

$\mathbf{\Phi}_\gamma$ be the set of processing elements which are neigbored to processing element PE_γ. Notice that this relation is reflexive. Neigbored processing elements are able to communicate directly (hop count=0 or 1).

$$\Phi_\gamma := \{PE_\delta \mid \delta \in I\Omega \text{ and } PE_\delta \text{ neighbored to } PE_\gamma\}$$

φ_γ be the number of processing elements neighbored to PE_γ.

$$\varphi_\gamma := |\Phi_\gamma|$$

$\mathbf{M}$ be the set of all tasks in the system.

$\mathbf{m}$ be the number of all tasks in the system.

$$m := |M|$$

$\mathbf{IM}$ be the set of indices of all tasks.

$$IM := \{1, ..., m\}$$

Thus, we obtain the set of all tasks in the system as

$$M = \{T_1, ..., T_m\} = \{T_i \mid i \in IM\}.$$

$\mathbf{V_i}$ be the set of all tasks related to task T_i. Related tasks work on common problems and therefore have to cooperate closely.

$$V_i := \{T_j \mid j \in IM \text{ and } T_j \text{ related to } T_i\}$$

$\mathbf{v_i}$ be the number of all tasks related to task T_i.

$$v_i := |V_i|$$

$\mathbf{E}_\gamma$ be the set of tasks executed on processing element PE_γ.

$$E_\gamma := \{T_j \mid T_j \in M \text{ and } T_j \text{executed by } PE_\gamma\}$$

$\mathbf{e}_\gamma$ be the number of all tasks executed on PE_γ.

$$e_\gamma := |E_\gamma|$$

In the following sections, we describe the hormones in more detail. Several kinds of eager values, suppressors and accelerators have to be distinguished. Therefore, we use an extended notation compared to figure 1 to specify the hormones:

$H_{i\gamma}^{j\delta}$**:** Hormone from task T_i running on PE_γ to be sent to task T_j running on PE_δ.

Hormones can be also sent to several tasks or PEs simultaneously. In that case, indices are replaced by the associated sets, e.g.:

$H_{i\gamma}^{M\Omega}$**:** Hormone from task T_i executed on PE_γ to be sent to all tasks on each processing element.

3.2 Different Kinds of Hormones

Using the notation introduced above we now describe the used hormones and their function in detail.

We begin explaining the eager values:

Local eager value $E_{i\gamma}$**:** This value states the initial suitability of PE_γ for task T_i. It takes care that the task allocation is oriented on the capabilities of the PEs.

Modified eager value $E_{i\gamma}^{i\Omega}$**:** This value is calculated by adding the received accelerators for task T_i on PE_γ and subtracting the received suppressors for task T_i on PE_γ from the local eager value $E_{i\gamma}$. It is sent to task T_i on all other PEs.

We used the following suppressors for the artificial hormone system:

Acquisition suppressor $Sa_{i\gamma}^{i\Omega}$**:** This suppressor is sent to task T_i on all other PEs in the system, as soon as PE_γ has taken task T_i. Therefore, this suppressor determines how often task T_i will be allocated in the overall system. A very strong acquisition suppressor enforces that task T_i is taken only once, while a weaker suppressor enables a multiple allocation of this task.

Load suppressor $Sl_{i\gamma}^{M\gamma}$**:** This suppressor is sent only locally to that PE_γ which has taken task T_i. It affects not only task T_i, but all tasks on this PE. Therefore, the load suppressor determines how many tasks can be taken by a PE. A very strong load suppressor enforces, that a PE can take only one task, while a weaker suppressor allows multiple tasks to be allocated on this PE.

Monitoring suppressor $Sm_{M\gamma}^{M\gamma}$**:** This suppressor is sent locally to a PE by local monitoring and affects all tasks on this PE. Thereby, the common state of a PE influences task allocation. As lower e.g. the energy level or as higher the temperature of a PE, as stronger gets this suppressor.

We also used different kinds of accelerators for the artificial hormone system:

Organ accelerator $Ao_{i\gamma}^{V_i\Phi_\gamma}$**:** This accelerator is sent to all tasks V_i related to task T_i on the PEs Φ_γ neighbored to PE_γ, if PE_γ has taken task T_i. Thereby, this accelerator attracts tasks related to task T_i to settle on the same or neighbored PEs. As stronger the accelerator as stronger is the attraction. The basic idea behind this is that related tasks work on common problems and have to communicate frequently. Therefore, short communication distances are useful. Related tasks form kind of a virtual organ which works on a bigger problem.

Stay accelerator $As_{i\gamma}^{i\gamma}$**:** As soon as PE_γ has taken task T_i, this assignment is initially fixed. It is not questioned any more in the beginning. This leads to a stable task allocation in the context of self-configuration. But to allow self-optimization, the possibility of changes in the task allocation is necessary. Therefore, a task assigned to a PE can offer itself periodically for reallocation. To achieve this, the task suspends the transmission of its acquisition suppressor $Sa_{i\gamma}^{i\Omega}$ and starts sending its modified eager value $E_{i\gamma}^{i\Omega}$ again. This enables other PEs to take this task, if they are more suitable meanwhile. But, a task migration introduces costs. The stay accelerator expresses these costs by favoring the stay of task T_i on PE_γ. It is sent from task T_i on PE_γ to itself (i, γ). As stronger the stay accelerator, as better another PE must be suited for task T_i to be able to take it from PE_γ.

Monitoring accelerator $Am_{M\gamma}^{M\gamma}$**:** This accelerator is sent locally to a PE by local monitoring and affects all tasks on the PE. It is the opponent of the monitoring suppressor. Therefore, the local monitoring can strengthen a PE if it is currently very powerful, e.g. due to a high energy level (solar cell in plain sun).

The described approach is completely decentralized, each PE is responsible for its own tasks, the communication to other PEs is realized by a unified hormone concept. Furthermore, it realizes the described self-X properties:

- The approach is **self-organizing**, because no external influence controls the task allocation.
- It is **self-configuring**, an initial task allocation is found by exchanging hormones. The self-configuration is finished as soon as all modified eager values become zero meaning no more tasks wants to be taken. This is done by sending suppressors. Of course, these suppressors have to be chosen strong enough to inhibit an infinite task assignment (the suppressors must be stronger then the accelerators), otherwise the system would become instable.
- The **self-optimization** is done by offering tasks. The point of time for such an offer is determined by the task respectively the PE itself. It can be done periodically or at a point in time where the task or the PE is idle. Furthermore, an offered task continues its operation on the old PE as long as it is not taken by a new PE.
- The approach is **self-healing**, in case of a task or PE failure all related hormones are no longer sent, especially the acquisition suppressors. This initiates an automatic reassignment of the task to the same PE (if it is still active) or another PE. The only additional requirement is a hormone $H_{i\gamma}^{j\delta}$ sent from task T_i on PE_γ to task T_j on PE_δ has an expiration time. If task T_j on PE_δ receives no new hormone value within this expiration time, the old value is discarded. This enables the detection of missing hormones after the expiration time.

A detailed discussion of the real-time behavior, especially of upper time bounds for self-configuration, self-optimization and self-healing can be found in the next sections. The communication overhead introduced will be analyzed there, too.

4. QUALITY MEASURE TO RATE THE TASK MAPPING

To measure the quality of a task mapping the following three aspects have to be combined:

- the load of the single processing elements
- the suitability of the processing element to execute a certain task
- the communication distances between related tasks (i.e. tasks which need to communicate with each other)

We will first define quality measures for single tasks and how suitable they are on a certain processing element. As an outcome of this we will then define a quality for a processing element and afterwards the quality of the task mapping for the whole system.

4.1 Quality measure for tasks, processing elements and the whole system

To rate the mapping of a task T_i on a processing element PE_γ we defined the following quality measure:

$$QU_{i\gamma} = \frac{w_{SH} * SH_{i\gamma} + w_{EV} * EV_{i\gamma} + w_{CD} * CD_{i\gamma}}{w_{SH} + w_{EV} + w_{CD}}$$

$with:$

$QU_{i\gamma}$	quality of t_i on PE_γ, range $[0 \dots 1]$
$SH_{i\gamma}$	cpu share of PE_γ available to t_i, range $[0 \dots 1]$
$EV_{i\gamma}$	suitability of t_i on PE_γ, ratio of the EagerValue of this PE for task t_i to best EagerValue of a PE for task t_i, range $[0 \dots 1]$
$CD_{i\gamma}$	communication distance of T_i on PE_γ to all related tasks, range $[0 \dots 1]$
w_{SH}	weight of $SH_{i\gamma}$
w_{EV}	weight of $EV_{i\gamma}$
w_{CD}	weight of $CD_{i\gamma}$

The weights w_{SH}, w_{EV} and w_{CD} were included to be able to shift the importance of single quality attributes. Normally these weights are chosen equal hence all quality attributes are rated equally. Based on this quality measure for a task T_i on a processing element PE_γ it is possible to calculate an average mapping quality for all tasks on a processing element:

$$QU_\gamma = \frac{\sum_{T_i \in E_\gamma} QU_{i\gamma}}{e_\gamma}$$

$with:$

QU_γ	quality of PE_γ
$QU_{i\gamma}$	quality of T_i on PE_γ
E_γ	set of all tasks running on PE_γ
e_γ	number of all tasks running on PE_γ

Finally we can define an overall quality for the whole system by averaging out the qualities of all tasks on all processing elements. It is not recommendable to use the average quality of all processing elements for the overall quality of the system as this would eliminate the information of the number of tasks on the different processing elements. The following example will illustrate this:

100 tasks are mapped onto 10 processing elements in such a way that one processing element executes 91 tasks and the remaining 9 processing elements each execute one task. With this mapping

configuration one processing element has a very bad quality while the other 9 processing elements have an ideal quality. The average quality of the processing elements would be quite good although the quality of 91 tasks would be very bad and only 9 tasks would have a good quality. Since our interest lies on the quality of the task execution this result would be incorrect. The following formula will accommodate this by averaging the qualities of all tasks in the system and not the qualities of the processing elements. In our example the overall quality results to a bad mapping which is correct.

$$QU = \frac{\sum_{PE_\gamma \in \Omega} \sum_{T_i \in E_\gamma} QU_{i\gamma}}{m} = \frac{\sum_{PE_\gamma \in \Omega} e_\gamma * QU_\gamma}{m}$$

$with:$

QU	overall quality of the task mapping
QU_γ	quality of PE_γ
$QU_{i\gamma}$	quality of T_i on PE_γ
Ω	set of all PEs in the system
m	number of all tasks running in the system

In the following we will show how to calculate the three different quality parts: $SH_{i\gamma}$ - the CPU share of a task, $EV_{i\gamma}$ - the suitability of a task, and $CD_{i\gamma}$ - the communication distance of a task to all its related tasks.

4.2 Calculating the part of the CPU share

To calculate the CPU share which is available to a task, we first have to determine the load produced by a task running on a processing element. This is necessary as a task will not always fully load a processing element, but will only need a part of the full capacity to run optimally. If e.g. two tasks both only need 50% of the computing capacity of a processing element then these two tasks can run simultaneously without reducing their quality. The overall load of a processing element $PE_{i\gamma}$ results as the sum of all loads $LD_{i\gamma}$ of all tasks T_i running on the processing element. The average share of PE_γ which is available to a running task can be calculated as follows:

$$SH_{i\gamma} = \begin{cases} \frac{1}{\sum_{T_i \in E_\gamma} LD_{i\gamma}}, \text{for } \sum_{T_i \in E_\gamma} LD_{i\gamma} \geq 1 \\ 1, \text{otherwise} \end{cases}$$

$with:$

$LD_{i\gamma}$	load of T_i on PE_γ
E_γ	set of all task running on PE_γ

($SH_{i\gamma}$ is limited to a maximum of 1 as a task running on a processing element maximally has one processing element available.)

As a measurement for the load of task running on a processing element we can e.g. use the quotient of the local eager value and the load suppressor, as the eager value symbolizes the suitability of processing element to execute this task and the load suppressor symbolizes the load produced by its execution. The load of a PE_γ caused by a task T_i can be stated as follows:

$$LD_{i\gamma} = \begin{cases} \frac{Sl_{i\gamma}^{M\gamma}}{E_{i\gamma}}, \text{for } Sl_{i\gamma}^{M\gamma} \leq E_{i\gamma} \\ 1, \text{otherwise} \end{cases}$$

$with:$

$LD_{i\gamma}$	load of PE_γ executing T_i
$Sl_{i\gamma}^{M\gamma}$	load suppressor of T_i on PE_γ
$E_{i\gamma}$	local eager value of PE_γ for T_i

($LD_{i\gamma}$ is limited to a maximum of 1 as a task running on a processing element will, in a worst case scenario, fill up a processing element.)

One very simple possibility is to assume that each task will fill up the processing element completely. This means:

$$LD_{i\gamma} = 1$$

The share of a PE_γ which is available to T_i can be easily calculated as follows:

$$SH_{i\gamma} = \frac{1}{\sum_{T_i \in E_\gamma} LD_{i\gamma}} = \frac{1}{\sum_{T_i \in E_\gamma} 1} = \frac{1}{e_\gamma}$$

$with:$

e_γ	total number of task running on PE_γ

It is left to note that the share $SH_{i\gamma}$ is always equal for all tasks T_i running on PE_γ and therefore is independent from i. This simplifies the calculation of the quality of a processing element. For the calculation of the formula $QU_\gamma = \frac{\sum_{T_i \in E_\gamma} QU_{i\gamma}}{e_\gamma}$ the term $\sum_{T_i \in E_\gamma} SH_{i\gamma}$ can be simplified to $e_\gamma * SH_{i\gamma}$.

4.3 Calculating the part of the suitability

This part is quite easy to calculate, it results from the ratio of the eager value for the task T_i on PE_γ to the best possible eager value of a processing element for this task.

$$EV_{i\gamma} = \frac{E_{i\gamma}}{\max_{PE_\delta \in \Omega}(E_{i\delta})}$$

$with:$

$E_{i\gamma}$	local eager value of PE_γ for T_i
Ω	set of all PEs in the system

4.4 Calculating the part of the communication distance

The quality of the communication results from the communication distance between a task T_i running on a processing element PE_γ and all its related tasks. For this purpose the communication distance will be defined as follows:

$$KD_{i\gamma j} = 1 + \text{number of PEs between } PE_\gamma \text{ executing } T_i \text{ and the PE executing } T_j$$

$with:$

$KD_{i\gamma j}$	communication distance between T_i on PE_γ and T_j

For two tasks on the same processing element as well as on processing elements directly next to each other there will be a communication distance of 1. If an additional processing element is between the processing element executing T_i and the processing element executing T_j the communication distance will be 2 and so on.

Furthermore, the degree in which the two tasks are related acts a part as the communication load and frequency will increase with this degree. It is sensible to weigh the communication distance with the degree of relationship. As a result we get a weighted communication distance where the best value however will limited to 1:

$$GKD_{i\gamma j} = \begin{cases} KD_{i\gamma j} * VG_{ij}, \text{ for} \\ \quad KD_{i\gamma j} * VG_{ij} \geq 1 \\ 1, \text{ otherwise} \end{cases}$$

$with:$

$GKD_{i\gamma j}$	weighted communication distance between T_i on PE_γ and T_j
$KD_{i\gamma j}$	communication distance between T_i on PE_γ and T_j
VG_{ij}	degree of relationship between T_i and T_j

The outcome of this we receive a quality measure for the communication distance of task T_i on processing element PE_γ:

$$CD_{i\gamma} = \frac{v_i}{\sum_{T_j \in V_i} GKD_{i\gamma j}}$$

$with:$

v_i	number of task that are related to task T_i
V_i	set of tasks that are related to task T_i
$GKD_{i\gamma j}$	weight communication distances between T_i on PE_γ and T_j

4.5 Finding an upper bound for the quality measure of the task mapping

The upper bound of the quality measure will of course be 1 as an optimum. If however there are more tasks to be mapped than there is computing capacity available the optimal mapping will not be able to reach the value 1. This will always happen if and only if the sum of the load caused by all tasks is greater than the number of processing elements:

$$\sum_{PE_\gamma \in \Omega} \sum_{T_i \in E_\gamma} LD_{i\gamma} > \omega$$

How to calculate the load $LD_{i\gamma}$ can be found in 4.2. Using the simplified version $LD_{i\gamma} = 1$ as stated there the condition above will become:

$$\sum_{PE_\gamma \in \Omega} \sum_{T_i \in E_\gamma} LD_{i\gamma} = \sum_{PE_\gamma \in \Omega} \sum_{T_i \in E_\gamma} 1 = m > \omega$$

In this case it is possible to calculate an upper bound for the quality measure which is smaller then 1:

If the load is as balanced as possible the processing load will be minimal. With an as balanced as possible load and a bottom estimation of the load (upper bound of the quality) the load of a processing element is between:

$$L_{\min} = \left\lfloor \sum_{PE_\gamma \in \Omega} \sum_{T_i \in E_\gamma} LD_{i\gamma} \right\rfloor div(\omega)$$

and

$$L_{\max} = L_{\min} + 1$$

With this distribution (also an bottom estimation of the load)

$$\omega_{\max} = \left\lfloor \sum_{PE_\gamma \in \Omega} \sum_{T_i \in E_\gamma} LD_{i\gamma} \right\rfloor mod(\omega)$$

processing elements will have the load $L_{\max}$ and

$$\omega_{\min} = \omega - \omega_{\max}$$

processing elements will have the load $L_{\min}$.

Transferred to tasks, the share of processing elements which is available to a task will be between

$$SH_{\min} = \frac{1}{L_{\max}}$$

and

$$SH_{\max} = \begin{cases} \frac{1}{L_{\min}}, \text{ for } L_{\min} \geq 1 \\ 1, \text{ otherwise} \end{cases}$$

(Keep in mind that the minimal load $L_{\min}$ will produce the maximal share of $SH_{\max}$ and vice versa.)

The number of tasks with the share of $SH_{\min}$ can be calculated to

$$m_{\min} = \omega_{\max} * L_{\max}$$

The number of tasks with the share of $SH_{\max}$ hence is:

$$m_{\max} = m - m_{\min}$$

With the quality parts EV and CD set to their maximum of 1, the following equation of an upper limit of the overall quality results:

$$\begin{aligned} QU_{\max} &= \frac{m_{\min} * \frac{w_{SH} * SH_{\min} + w_{EV} * EV + w_{CD} * CD}{w_{SH} + w_{EV} + w_{CD}}}{m} \\ &\quad + \frac{m_{\max} * \frac{w_{SH} * SH_{\max} + w_{EV} * EV + w_{CD} * CD}{w_{SH} + w_{EV} + w_{CD}}}{m} \\ &= \frac{m_{\min} * (w_{SH} * SH_{\min} + w_{EV} + w_{CD})}{w_{SH} + w_{EV} + w_{CD}} \\ &\quad + \frac{m_{\max} * (w_{SH} * SH_{\max} + w_{EV} + w_{CD})}{w_{SH} + w_{EV} + w_{CD}} \end{aligned}$$

5. SIMULATIONS AND RESULTS

Having created a quality measure, we started testing different scenarios. As a first step we wanted to check by using this quality measure, how good the mapping of the artificial hormone system would be compared to an ordinary load balancing approach. We analyzed mappings from our artificial hormone system and mappings from an ordinary load balancing approach in regard to the quality over time. The ordinary load balancing worked as follows: all tasks are mapped on the processing elements in such a way that every processing element would execute the same number of tasks (respectively one less, if the number of tasks cannot be divided equally on all processing elements). All tasks had the same load

and were started one after another on the system. For the system configuration we chose a grid of 4 x 4 processing elements as 16 processing elements are a quite realistic number of nodes to map the tasks on and also as our later underlying hardware will also be somewhere around the same size. On this grid 50 tasks had to be mapped. Each task was related to 3 other tasks i.e. each 4 tasks will work and communicate heavily with each other. Although our artificial hormone system would be able to map the tasks on different, heterogeneous processing elements, we wanted to keep it simple and chose all processing elements to be equal hence equally qualified to execute any task. The hormone values were simply chosen in the manner that it would result in a stable outcome with each task being mapped exactly on time onto one processing element. Though we plan to do this in the future, we for now did not use any sophisticated method to chose the hormone values nor did we try to optimize them.

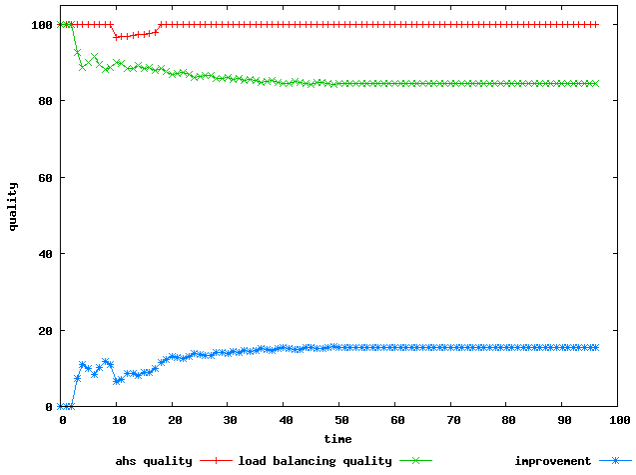

Figure 2: Quality evaluation: 50 tasks on 4x4 PEs

The result can be seen in Figure 2. The line on the top represents the quality of our artificial hormone system. The quality runs from 0 to 100 which represents the percentage (compared to the calculated possible maximum). The line right beneath it is the quality line for the simple load balancing algorithm. The line on the bottom is the difference between these two quality lines. At the end of the simulation, when all task have been mapped on the processing cell grid, the quality of the artificial hormone system is about 15% better than the simple load balancing one.

In the next test series we have investigated how the number of tasks that have to be mapped on the system influences the quality difference between the mapping of our artificial hormone system and the simple load balancing. Once again we used a simulated system with a grid size of 4x4 processing elements. This time we mapped 16, 32 and 64 tasks on this system.

Figure 3 shows the result. To be able to compare the values of the final configuration - when all tasks have been mapped - we continued the lines horizontally. In the scenario with only 16 tasks the quality difference between the mapping of the artificial hormone system and the simple load balancing is a bit over 12% (which can be seen in the bottom line). The line in the middle represents the quality difference with the mapping of 32 tasks which comes to an improvement of about 15%. The mapping of 64 tasks brought an improvement of about 17% as shown by the top line. The more tasks the system had to map the better the quality of the artificial

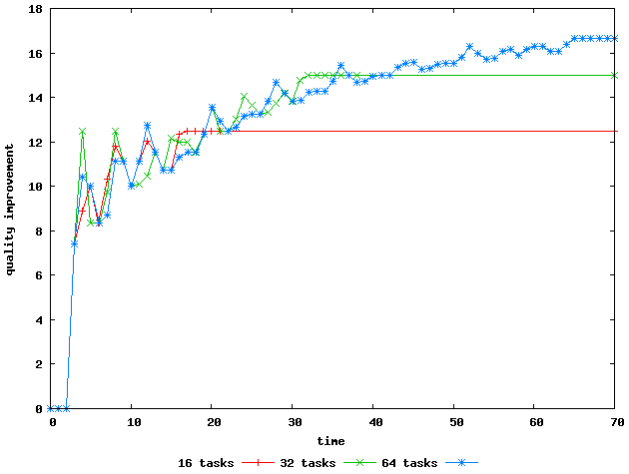

Figure 3: quality over time

hormone system compared to the simple load balancing algorithm became.

In the last test series which will be presented in this paper we varied the number of related tasks. We chose 3 configurations with 2, 4 and 8 related tasks. In each test series we let the system map different numbers of tasks starting from 5 tasks up to 50 tasks in steps of 5 tasks. We then only plotted the final value of the quality difference between the artificial hormone system and the simple load balancing.

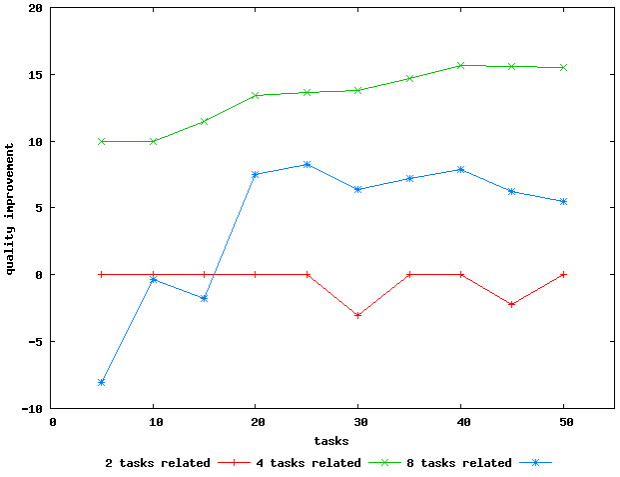

Figure 4: quality over tasks

The results can be seen in Figure 4. The line at the top is the result for the test series with 4 related tasks. It ranges from 10% quality increase up to around 15%. With increased number of tasks to be mapped the quality also rises. As we suspected the line that represents the results of the test series with 2 related tasks did not bring any quality improvement. The simple load balancing algorithm even manages to have a slightly better mapping on 2 occasions. As the load balancing places the tasks evenly onto the processing elements and as related tasks were directly next to each other in the list of tasks that had to be mapped, the simple load balancing algorithm placed these related tasks right next to each other. What came as quite a surprise was the result of the test series

where always 8 tasks were related. With less than 20 tasks the load balancing hat a better performance and the quality improvement of the artificial hormone system compared to the simple load balancing was only between 5 to 10%. We would have suspected that the quality improvement would be at least equally if not a little bit better than the mapping with 4 related tasks. Never the less there is still an improved quality and with a sophistication of the hormone values we hope to achieve improved results.

6. CONCLUSION AND FUTURE WORK

We presented a quality measure to evaluate the mapping of tasks on a grid of processing elements. With this new quality measure we established a mapping measurement which not only evaluates the simple equal distribution of the mapped tasks but also the relationships between the different tasks and the requirements of the tasks combined with the suitability of the processing elements. We presented test results which show that our artificial hormone system is able to achieve a good task mapping and also most of the time a better mapping than a simple load balancing technique. Up to nearly 20% better mapping results compared to the simple load balancing were achieved and at an average still an improvement over 10%.

With this quality measurement we plan to refine our artificial hormone system. With its help we can compare and evaluate different task mappings. Further and more detailed test series will help us improving the adjustments of hormone values for better task mappings. In the near future we want to be able to predict what kind of hormone values for a given system configuration will result in the best possible mapping of the artificial hormone system.

7. REFERENCES

[1] W. Becker. Dynamische adaptive Lastbalancierung für große, heterogen konkurrierende Anwendungen. Dissertation, Universität Stuttgart, Fakultät Informatik, Dezember 1995.

[2] L. F. Bittencourt, E. R. M. Madeira, F. R. L. Cicerre, and L. E. Buzato. A path clustering heuristic for scheduling task graphs onto a grid. In *3rd International Workshop on Middleware for Grid Computing (MGC05)*, Grenoble, France, 2005.

[3] U. Brinkschulte, M. Pacher, and A. von Renteln. Towards an artificial hormone system for self-organizing real-time task allocation. *5th IFIP Workshop on Software Technologies for Future Embedded & Ubiquitous Systems (SEUS)*, 2007.

[4] U. Brinkschulte, M. Pacher, and A. von Renteln. An artificial hormone system for self-organizing real-time task allocation in organic middleware. *Organic Computing - Springer (ISBN: 978-3-540-77656-7)*, pages 261–284, Mar. 2008.

[5] E. Carvalho, N. Calazans, and F. Moraes. Heuristics for dynamic task mapping in noc-based heterogeneous mpsocs. *18th IEEE/IFIP International Workshop on Rapid System Prototyping (RSP'07)*, 2007.

[6] T. Decker, R. Diekmann, R. Lüling, and B. Monien. Universelles dynamisches task-mapping. In *Konferenzband des PARS'95 Workshops in Stuttgart, PARS-Mitteilung 14*, pages 122–131, 1995.

[7] J. Finke, K. M. Passino, and A. Sparks. Cooperative control via task load balancing for networked uninhabited autonomous vehicles. In *42nd IEEE Conference onDecision and Control, 2003. Proceedings*, volume 1, pages 31 – 36, 2003.

[8] J. Finke, K. M. Passino, and A. Sparks. Stable task load balancing strategies for cooperative control of networked autonomous air vehicles. In *IEEE Transactions on Control Systems Technology*, volume 14, pages 789– 803, 2006.

[9] H.-U. Heiss and M. Schmitz. Decentralized dynamic load balancing: The particles approach. In *Proc. 8th Int. Symp. on Computer and Information Sciences*, Istanbul, Turkey, November 1993.

[10] P. I. R. Horn. Autonomic computing manifesto: IBM's perspective on the state of information technology, October 2001.

[11] C. Mueller-Schloer, C. von der Malsburg, and R. P. Wuertz. Organic computing. *Informatik Spektrum*, 27(4):332–336, 2004.

[12] A. Radulescu and A. J. C. van Gemund. Fast and effective task scheduling in heterogeneous systems. In *IEEE Computer - 9th Heterogeneous Computing Workshop*, Cancun, Mexico, 2000.

[13] L. Rudolph, M. Slivkin-Allalouf, and E. Upfal. A simple load balancing scheme for task allocation in parallel machines. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 237–245, 1991.

[14] W. Trumler, T. Thiemann, and T. Ungerer. An artificial hormone system for self-organization of networked nodes. In *Biologically inspired Cooperative Computing, IFIP 19th World Computer Congress 2006, Santiago de Chile, Chile*, 2006.

[15] VDE/ITG/GI. VDE/ITG/GI-Positionspapier Organic Computing: Computer und Systemarchitektur im Jahr 2010, 2003.

[16] C. Xu and F. Lau. Decentralized remapping of data parallel computations with the generalized dimension exchange method. In *Proceedings of Scalable High-Performance Computing Conference*, pages 414 – 421, 1994.