Reducing the Communication Overhead of an Artificial Hormone System for Task Allocation by a Task Window

Uwe Brinkschulte
Institut für Informatik
Johann Wolfgang Goethe Universität Frankfurt, Germany
Email: brinks@es.cs.uni-frankfurt.de

Abstract

The Artificial Hormone System (AHS) is a completely decentralized operation principle for a middleware which can be used to allocate tasks in a system of heterogeneous processing elements (PEs) or cores. Tasks are scheduled according to their suitability for the heterogeneous PEs, the current PE load and task relationships. The AHS also provides properties like self-configuration, self-optimization and self-healing by task allocation. The AHS is able to guarantee real-time bounds for such self-X-properties.

If a large number of PEs applies for a large number of tasks a considerable amount of hormone communication is produced to assign these tasks to PEs. Until now this could be only circumvented by limiting the number of tasks a single PE applies for. However, this reduces the self-optimization and self-healing properties because the number of possible PEs to execute a tasks is limited in the same way. In this paper we present the concept of the task window as a new approach to significantly reduce the communication overhead without affecting self-optimization and self-healing properties. We additionally show that the task window can even improve the real-time behavior of task allocation.

Keywords: *Artificial Hormone System, communication overhead, task window, real-time, self-organization*

1. Introduction

Embedded systems are growing more and more complex because of the increasing chip integration density, larger number of chips in distributed applications and demanding application fields (e.g. in cars and in households). Self-organization is a key feature to handle this complexity.

The Artificial Hormone System (AHS) was developed as a powerful tool to handle the complexity of assigning tasks to processors or processor cores (so called processing elements, PEs) in a self-organizing and completely decentralized way. Features like self-configuration (initial autonomous assignment of tasks to PEs), self-optimization (autonomously moving tasks to more suited PEs) and self-healing (autonomously recovering failing tasks and PEs) allow an autonomous and robust system operation without user intervention, while the decentralized approach avoids single-points-of-failure. The task assignment is done by the AHS in real-time using different kinds of artifical hormones to build up distributed closed control loops. While so-called eager value hormones indicate the suitability of a PE for a given task, antagonistic suppressor and accelerator hormones balance the task distribution. Suppressors are spread globally in the system and reduce the impact of eager values thus limiting the number of tasks assigned. Accelerators increase the impact of eager values and are spread locally in the neighborhood of a PE in order to cluster cooperating tasks.

For initial self-configuration, each PE sends eager value hormones for each task it applies for. If a large number of PEs applies for a large number of tasks, then a large number of eager values will have to be sent and this would lead to a considerable communication overhead. So far, the only solution to reduce such an overhead is to limit the number of tasks a single PE applies for. As a disadvantage, this reduces the self-optimization and self-healing properties because the number of possible PEs to execute a task is limited in the same way. In this paper we present the novel concept of the task window to significantly reduce the communication overhead without affecting self-optimization and self-healing properties. We additionally show that the task window can even improve the real-time behavior of task allocation in the self-configuration phase.

The paper is structured as follows: After the motivation we present related work in section 2. Section 3 sketches the basic ideas and operation principles of the AHS. The concept of the task window to reduce the communication overhead is presented in section 4. Furthermore, the impact of the task window on worst-case time bounds for task allocation and different variants to handle the task window are discussed there. An evaluation is conducted in section 5, while section 6 concludes this paper.

2. Related Work

Self-organization has been a research focus for several years. Publications like [1] or [2] deal with basic principles of self-organizing systems, like e.g. emergent behavior, reproduction, etc. Regarding self-organization in computer science, several projects and initiatives can be listed.

IBM's and DARPAS's Autonomic Computing project [3], [4] deals with self-organization of IT servers in networks. Several so called self-X properties like self-optimization, self-stabilization, self-configuration, self-protection and self-healing have been postulated. The MAPE cycle consisting of *M*onitor, *A*nalyze, *P*lan and *E*xecute was defined to realize these properties. This MAPE cycle is executed in the background and in parallel to normal server activities similar to the autonomic nervous system.

The German *Organic Computing* Initiative was founded in 2003. Its basic intention is to improve the controllability of complex embedded systems by using principles found in organic entities [5], [6]. Organization principles successful in biology are adapted to embedded computing systems. The DFG priority programme 1183 "Organic Computing" [7] has been established to deepen research on this topic.

Self-organizing and organic computing is also followed on an international level by a task force of the IEEE Computational Intelligence Society (IEEE CIS ETTC OCTF) [8]. Several other international research programs have also addressed self-organization aspects for computing systems, e.g. [9], [10].

So far, there are several approaches for clustered task allocation in middleware.

The authors of [11] present a scheduling algorithm for distributing tasks onto a grid. It is implemented in the Xavantes Grid Middleware and arranges the tasks in groups. Their approch is completely different from ours because it uses central elements for the grouping: the Group Manager (GM), the Process Manager (PM) and the Activity Managers (AM). The GM is a single point of failure. If it fails, there will be no possibility of obtaining group information from this group anymore. In our approach we do not use a central task distribution instance and therefore single point of failures are avoided.

Another approach is presented in [12]. The authors propose two algorithms for task scheduling. The first algorithm, Fast Critical Path (FCP), ensures that time constrains are kept. The second one, Fast Load Balancing (FLB), schedules the tasks equally on processors according to current loads. Different from our approach, task relationships are not regarded to allocate cooperating tasks closely together. Furthermore, these algorithms do not consider failing processing elements.

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

Other approaches of load balancing are presented in [14], [15], [16], [17], [18]. None of them cover the whole spectrum of self-X properties, task clustering, and real-time conditions like our approach.

A research project regarding self-organizing task allocation with respect to real-time properties is the CAR-SoC project [19], [20]. This project uses agent based principles and an auction mechanism to achieve self-X features. This approach is not completely decentralized like the AHS, since an auction manager is responsible for a certain set of tasks.

The DoDOrg project [21] researches the use of bio-inspired principles to build a new, self-organizing robust processor architecture. Within this project, we invented a first version of the AHS to assign software tasks to distributed processing cells. In [22] we improved this first approach to distribute time dependent tasks in a distributed system.

The contribution of the paper now presented here is a further improvement of the AHS. Indroducing the new approach of a task window significantly reduces the communication overhead without affecting the self-healing and self-optimization properties. It even can improve the real-time bounds for self-configuration.

3. The Artificial Hormone System

In the following, we will briefly present the operation principle of the AHS. A detailed description can be found in [23] and [24]. The aim of the AHS is to

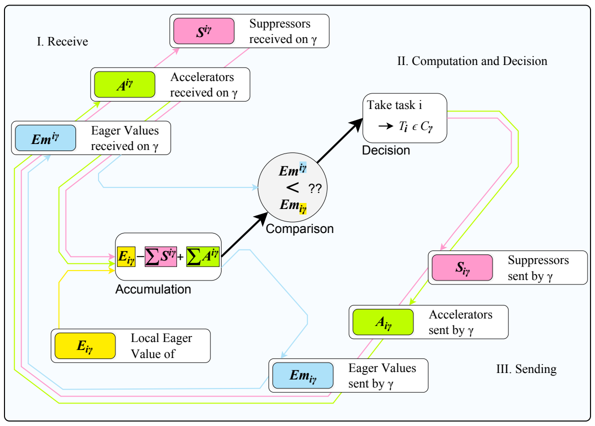

I. Receive
$S^{i\gamma}$ Suppressors received on γ
$A^{i\gamma}$ Accelerators received on γ
$Em^{i\gamma}$ Eager Values received on γ
II. Computation and Decision
Take task i → $T_i \in C_\gamma$
Decision
$Em^{i\gamma}$ < ?? $Em_{i\gamma}$
Comparison
$E_{i\gamma} - \sum S^{i\gamma} + \sum A^{i\gamma}$
Accumulation
$S_{i\gamma}$ Suppressors sent by γ
$A_{i\gamma}$ Accelerators sent by γ
$E_{i\gamma}$ Local Eager Value of
$Em_{i\gamma}$ Eager Values sent by γ
III. Sending

Figure 1. Hormone based control loop

assign tasks to processing elements (PEs) in a self-organizing way. In order to achieve this, three main types of hormones are used:

- **Eager value:** This hormone type determines the suitability of a PE to execute a task. The higher the hormonal value the better is the ability of the PE for executing the task.
- **Suppressor:** This hormone type lowers the suitability for task execution on a PE. Suppressors are subtracted from eager values. There exist several subtypes of suppressors e.g., the *task suppressor* to prevent duplicate task allocation, the *monitoring suppressor* to indicate a deteriorating PE state and the *load suppressor* to indicate the current load of a PE caused by the executed tasks. While the task suppressor is broadcasted to all PEs in the system, the monitoring and load suppressors are only used locally to limit the number of executed tasks on a PE.
- **Accelerator:** This hormone type favors the execution of a task on a PE. Accelerators are added to eager values. Like for suppressors there exist several subtypes of accelerators. The most important one is the *organ accelerator* which is used to cluster related tasks in the neighborhood. This accelerator is multicasted to neighboring nodes to form so called 'virtual organs' of cooperating related tasks. Another subtype is the *monitoring accelerator*, which is used locally to indicate improved PE capabilities. Accelerators and suppressors are antagonistic hormones.

We have to distinguish between received hormones and hormones to be sent and also between tasks and PEs. Therefore, we use Latin letters such as i as task indices and Greek letters such as γ as PE indices. A hormone of any type denoted by $H^{i\gamma}$ with superscripted indices signifies that this hormone is dedicated to and will be received by PE γ and task T_i. A hormone of any type denoted by $H_{i\gamma}$ signifies that this hormone is sent by PE γ and task T_i to other PEs.

The task assignment is carried out in the following way: Each PE periodically executes the hormone based control loop presented in Figure 1. In the receive stage of the loop, PE γ receives the modified eager values $Em^{i\gamma}$, suppressors $S^{i\gamma}$ and accelerators $A^{i\gamma}$ for each task T_i from each PE in the network. In the compute and decision stage, PE γ computes its own modified eager values $Em_{i\gamma}$ for all of its tasks in the following way: The local static eager value $E_{i\gamma}$ indicates how suited PE γ is to execute task T_i. From this value, all suppressors $S^{i\gamma}$ received for task T_i are subtracted, and all accelerators $A^{i\gamma}$ received for task T_i are added:

$$Em_{i\gamma} = E_{i\gamma} - \sum S^{i\gamma} + \sum A^{i\gamma} \qquad (1)$$

The modified eager value $Em_{i\gamma}$ of each task T_i is then distributed to the other PEs in the send stage.

To decide on the allocation of a task T_i, a PE compares its own modified eager value $Em_{i\gamma}$ with the received modified eager values $Em^{i\gamma}$ from all other PEs. If $Em_{i\gamma} > Em^{i\gamma}$ is true for all received modified eager values, PE γ will decide to take the task[1]. Otherwise another PE has a higher modified eager value for task T_i, so PE γ will decide not to take it.

If a task T_i is taken on PE γ, it also will distribute suppressors $S_{i\gamma}$ dedicated to the same task on all other PEs. This limits the number of allocations of the tasks. Depending on the strength of this suppressor, the task is either taken only once or several times. Furthermore, the PE distributes an accelerator $A_{i\gamma}$ to its neighboring PEs to attract tasks cooperating with task T_i to neighboring PEs thus forming clusters of related tasks. Additionally, a local load suppressor $Sl^{i\gamma}$ indicates the load task T_i produces on PE_γ and limits the number of tasks taken by the PE. This is a completely decentralized approach providing self-configuration in terms of finding an initial task allocation by exchanging hormones, self-optimization by task reallocation when hormone levels change, and self-healing by automatic task reassignment due to missing hormones in case of failures. It allows task allocation with respect to real-time bounds. As shown in [25], an individual PE takes at most one task per hormone cycle while in the entire system at least one task per hormone cycle is taken. This results in a

1. In case of equality, a second criterion e.g., the smallest position identifier of the PEs, is used to get an unambiguous decision.

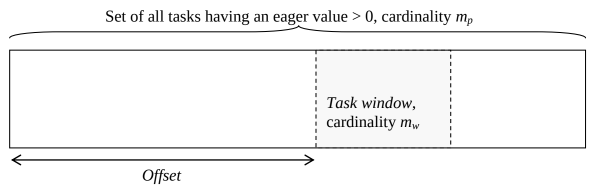

Set of all tasks having an eager value > 0, cardinality m_p
Task window, cardinality m_w
Offset

Figure 2. Task window

guaranteed upper time bound of m hormone cycles for the assignment of m tasks.

To validate this approach a hormone simulator [23] and a hormone based middleware [24] have been created.

4. Task window

If PEs apply for a large number of tasks, they will have to emit and evaluate eager values for all these tasks during self-configuration. Let us look at an example: If 100 PEs apply for $m = 1000$ tasks and each PE is able to execute each task, $100 \cdot 1000 = 100000$ eager values have to be emitted and evaluated. In worst case, self-configuration lasts for $m = 1000$ hormone cycles as stated in the previous section.

In its current form the AHS can reduce this self-configuration complexity only by limiting the number of tasks a PE applies for. Unfortunatly, this limitation also restricts self-optimization and self-healing. This means in case of PE failures not all PEs can stand for all tasks.

To reduce the self-configuration complexity without affecting self-healing and self-optimization, we now propose to introduce a task window. Let a PE have m_p eager values $Em^{i\gamma} > 0$ meaning this PE can execute m_p tasks. A *task window* defines a subset of $m_w < m_p$ tasks. This subset consists of the first m_w tasks having an eager value $Em^{i\gamma} > 0$ starting from an arbitrary constant *offset*, as shown in figure 2. The PE now applies only for tasks within this task window.

4.1. Effects on self-configuration

For each task assigned in the task window the corresponding eager value $Em^{i\gamma}$ eventually drops to 0. Therefore, this task vanishes from the set of tasks having an eager value $Em^{i\gamma} > 0$ and consequently this task also vanishs from the task window. Since the size of the task window is kept to constant value of m_w tasks (as long as at least m_w tasks are left to assign), a new task enters the window from the right side. In the representation of figure 2 this means the overall set of unassigned tasks (m_p) is shrinking

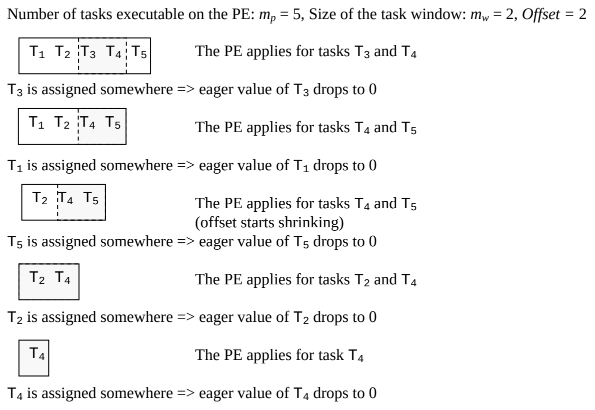

Number of tasks executable on the PE: m_p = 5, Size of the task window: m_w = 2, Offset = 2
T_1 T_2 T_3 T_4 T_5
The PE applies for tasks T_3 and T_4
T_3 is assigned somewhere => eager value of T_3 drops to 0
T_1 T_2 T_4 T_5
The PE applies for tasks T_4 and T_5
T_1 is assigned somewhere => eager value of T_1 drops to 0
T_2 T_4 T_5
The PE applies for tasks T_4 and T_5 (offset starts shrinking)
T_5 is assigned somewhere => eager value of T_5 drops to 0
T_2 T_4
The PE applies for tasks T_2 and T_4
T_2 is assigned somewhere => eager value of T_2 drops to 0
T_4
The PE applies for task T_4
T_4 is assigned somewhere => eager value of T_4 drops to 0

Figure 3. Example for a task window

while the task window moves to the right due to its constant offset. As soon as the task window reaches the right border, the offset is reduced. Therefore, the task window constantly contains m_w tasks as long as enough tasks are left to assign. Figure 3 gives a simple example, where a PE is able to execute 5 tasks ($m_p = 5$ tasks with eager value $Em^{i\gamma} > 0$) and the size of the task window is $m_w = 2$ with an offset of 2.

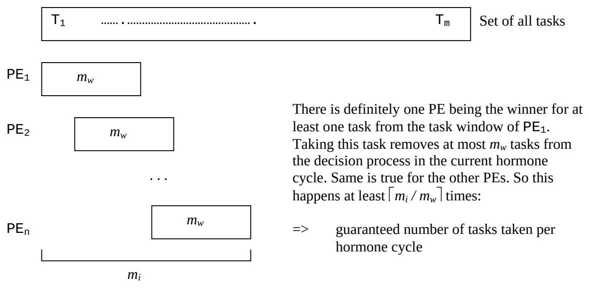
By introducing the task window the communication overhead and computational complexity regarding the transmission of eager values is reduced by the factor

$$m_p/m_w \tag{2}$$

because eager values are emitted and have to be evaluated for only m_w tasks instead of m_p tasks. In the example from figure 3 this would be a reduction of factor $5/2 = 2.5$. Taking the example given at the beginning of section 4 with $m = m_p = 1000$ tasks and using a task window of size $m_w = 10$, the resulting reduction would be $1000/10 = 100$. So the task window can significantly reduce the overhead and complexity of self-configuration. The real-time bounds for self-configuration are not deteriorated but can even be improved by the task window as we will see in section 4.2. One drawback of the task window concept is that the best possible PE for a task may initially not be found. The task might be assigned to a suboptimal PE. This will happen if a task is not in the initial task window of the best suited PE. However, during the process of self-optimization this will be corrected as explained in section 4.3 and shown in the evaluation.

4.2. Influence on worst-case task allocation time

As stated in section 3 and derived in [25], the following two conditions hold true for task assignment:



T1
Tm
Set of all tasks
PE1
mw
PE2
mw
...
PEn
mw
mi
There is definitely one PE being the winner for at least one task from the task window of PE1. Taking this task removes at most mw tasks from the decision process in the current hormone cycle. Same is true for the other PEs. So this happens at least ⌈mi / mw⌉ times:
=> guaranteed number of tasks taken per hormone cycle

Figure 4. Minimum number of tasks taken per hormone cycle

- For each task with eager value $Em^{i\gamma} > 0$ in each hormone cycle there exists a PE having the highest eager value (the winner) and therefore taking the task. So at least one task having an eager value $Em^{i\gamma} > 0$ is taken per hormone cycle.
- A PE takes at most one task per hormone cycle. So if a PE applying for m_w tasks takes a task in the current hormone cycle, it can remove up to m_w tasks from the decision process in this cycle[2].

First of all it follows that in each hormone cycle there will be at least one PE being the winner for a task in its task window. Therefore at least one task is taken per hormone cycle. Let m be the overall number of tasks in the system. These tasks will be assigned after m hormone cycles at the latest. So the task window does not deteriorate the worst case timing behavior of self-configuration. In fact, it can even improve the worst case timing:

Let m_w be the size of the task window for all PEs in the system. Furthermore, let m_i be the number of tasks covered by the initial task windows of all PEs in the system[3]. From the two conditions above it directly follows that at least $\lceil m_i/m_w \rceil$ tasks will be taken per hormone cycle. Figure 4 illustrates this.

Task assignment using the task window can be divided into three phases. Let m_r be the number of tasks not assigned yet (eager value $Em^{i\gamma} > 0$).

Phase 1: $m_r > m_i \rightarrow$ all task windows are filled and overall cover m_i tasks (figure 4).

Therefore, at least $\lceil m_i/m_w \rceil$ tasks will be taken per hormone cycle. Let h_1 be the number of hormone cycles encountered since the start of phase 1. Then it follows:

$$m_r(h_1) \leq m - h \lceil m_i/m_w \rceil$$

Phase 1 is finished when $m_r(h_1) \leq m_i$. This is true if $m_r(h_1) \leq m - h_1 \lceil m_i/m_w \rceil \leq m_i$ or $h_1 \geq (m - m_i)/\lceil m_i/m_w \rceil$. So the maximum number of hormone cycles for phase 1 can be calculated to:

$$h_1 = \lceil (m - m_i)/\lceil m_i/m_w \rceil \rceil \qquad (3)$$

Phase 2: $m_i \geq m_r > m_w \rightarrow$ all task windows are filled and overall cover $m_r < m_i$ tasks (figure 5a, the task windows are contracting).

Therefore, at least $\lceil m_r/m_w \rceil$ tasks will be taken per hormone cycle. Let h_2 be the number of hormone cycles encountered since the start of phase 2. Then the number of tasks in phase 2 can be recursively defined:

$$m_r(0) \leq m_i$$

$$m_r(h_2) \leq m_r(h_2 - 1) - \lceil m_r(h_2 - 1)/m_w \rceil$$

Estimating the upper bound of remaining tasks by $m_r(h_2 - 1) - (m_r(h_2 - 1)/m_w) \geq m_r(h_2 - 1) - \lceil m_r(h_2 - 1)/m_w \rceil$ and resolving the recursion leads to:

$$m_r(h_2) \leq m_i(1 - 1/m_w)^{h_2}$$

Phase 2 is finished when $m_r(h_2) \leq m_w$. This is true if $m_r(h_2) \leq m_i(1 - 1/m_w)^{h_2} \leq m_w$ or $h_2 \geq log_{1-1/m_w}(m_w/m_i)$. So the maximum number of hormone cycles for phase 2 can be calculated to:

$$h_2 = \lceil log_{1-1/m_w}(m_w/m_i) \rceil \qquad (4)$$

Phase 3: $m_r \leq m_w \rightarrow$ all task windows start to run dry and overall cover $m_r \leq m_w$ tasks (figure 5b, the task windows are shrinking).

Therefore, at least $\lceil m_w/m_w \rceil = 1$ tasks will be taken per hormone cycle. Let h_3 be the number of hormone cycles encountered since the start of phase 3. Then the maximum number of hormone cycles for phase 3 calculates to:

$$h_3 = m_w \qquad (5)$$

Combining the phases 1, 2 and 3, we get the maximum number of hormone cycles h to assign all tasks:

2. If the PE is the winner for all m_w tasks, it will take exactly one of them in the hormone cycle. The remaining $m_f - 1$ tasks can not be taken by other PEs in the same cycle. So overall m_f tasks are removed from the decision process in the current cycle.

3. $m_i \geq m_w$ due to different offsets on each PE.

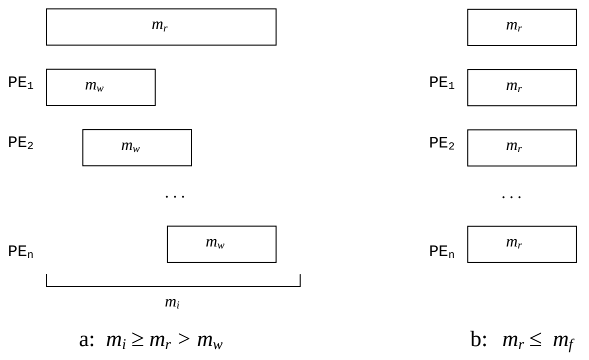


Figure 5. Contracting task windows

$$h = h_1 + h_2 + h_3 = \lceil (m - m_i)/\lceil m_i/m_w \rceil \rceil + \lceil log_{1-1/m_w}(m_w/m_i) \rceil + m_w \quad (6)$$

Let us have a look at some border cases:

- $m_i = m_w = m$: no task window $\rightarrow h = m$
- $m_i = m$: initial task windows cover all tasks (no phase 1) $\rightarrow h = \lceil log_{1-1/m_w}(m_w/m_i) \rceil + m_w$
- $m_i = m_w$: no offset (no phase 2) $\rightarrow h = m$
- $m_w = 1$: task window size 1 (phase 2 and 3 together condense to one hormone cycle[4]) $\rightarrow h = \lceil (m - m_i)/\lceil m_i/m_w \rceil \rceil + 1$
- $m_w = 1, m_i = m$: task window size 1, initial task windows cover all tasks $\rightarrow h = 1$

These bordercases show that by chosing appropriate values for m_w and m_i the real-time behavior of task allocation can be scaled in a wide range. The possible results of formula (6) are in the range of $1..m$. The smaller the task window and the larger the number of tasks covered by the initial task windows the faster the task allocation becomes. The price to pay for speed is allocation quality. In the border case of $m_w = 1, m_i = m$ each PE applies for a single task and takes it instantly. There is no more competition. In the other direction, the larger the task window and the smaller the number of tasks covered by the initial task windows the slower gets task allocation. In worst case, we retrieve the original real-time behavior of m hormone cycles.

4.3. Effects on self-healing and self-optimization

Due to the constant size and the resulting shift of the task window, each PE can cover each task. If all tasks are assigned (the task windows on all PEs are empty) and a PE fails, the eager values $Em^{i\gamma}$ of the m_f tasks running on the failing PE rise above 0 due to the failing suppressors from this PE. Now these are the only tasks in the system with an eager value $Em^{i\gamma} > 0$. Therefore, the task windows of all PEs automatically start to cover these tasks. If $m_w < m_f$, the first m_w failing tasks will be covered first. By assigning these tasks the task window will automatically shift to cover the remaining tasks. Since each hormone cycle at least one task is assigned, all failing tasks will be reassigned after m_f hormone cycles at the latest. So the task window does not deteriorate the real-time bounds of self-healing. In case of self-optimization, tasks are periodically offered for reassignment. Like for self-healing, the task windows of all PEs will automatically start to cover these offered tasks due to their eager value above 0. So the same time bounds as for self-healing hold true. During this process, a potential suboptimal task assignment in the self-configuration phase due to the task window will be corrected. Self-optimization periods are afflicted with an offset so not all tasks are offered for reassignment at the same time. Therefore, if a task is offered the task window of the best suited PE will very probably cover this task. This leads in a task reassignment to the best suited PE. The results given in the evaluation section confirm this mechanism.

4.4. Variants of the task window

4.4.1. Variant 1: variable task window offset. In the approach described above the offset of the task window refers to tasks the PE applies for ($Em^{i\gamma} > 0$). This guarantees a constant offset as long as enough tasks are left for assignment (phase1). Figure 6a gives an example. So the first task in the window has to be dynamically determined in each hormone cycle. A more simple version would be to refer the offset to tasks the PE is able to execute ($E^{i\gamma} > 0$). Since this doesn't change during runtime, the offset can be statically calculated before the first hormone cycle. As a drawback, the offset is no longer constant, as shown in 6b. It can shrink starting with the first hormone cycle. In the worst case, all offsets vanish before a single task beyond m_i is assigned (see figure 7). This means phase 1 will be ommitted completely. Task assignment immediately starts with phase 2, which can last up to $h_2 = \lceil log_{1-1/m_w}(m_w/m_i) \rceil$ hormone cycles according to formula (4). Then, phase 3 starts to assign the remaining $m_w + m - m_i$ tasks with a rate of at least one task per hormone cycle. Therefore, the maximum number of hormone cycles to assign all tasks using

4. All remaining tasks will be taken within a single hormone cycle at the beginning of phase 2, since $\lceil m_r/m_w \rceil$ calculates to m_r.

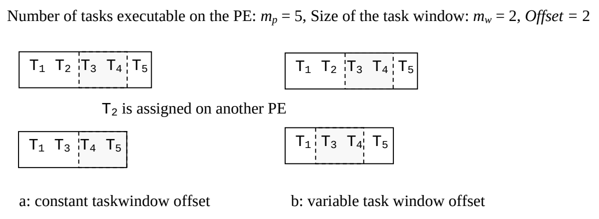

Number of tasks executable on the PE: m_p = 5, Size of the task window: m_w = 2, *Offset* = 2
T1 T2 T3 T4 T5
T1 T2 T3 T4 T5
T2 is assigned on another PE
T1 T3 T4 T5
T1 T3 T4 T5
a: constant taskwindow offset
b: variable task window offset

Figure 6. Constant and variable offset

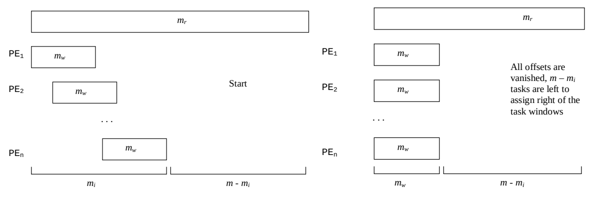

m_r
PE1
PE2
PEn
m_w
Start
m_i
$m - m_i$
All offsets are vanished, $m - m_i$ tasks are left to assign right of the task windows

Figure 7. Contracting task windows in variant 1

variant 1 calculates to:

$$h = \lceil log_{1-1/m_w}(m_w/m_i) \rceil + m_w + m - m_i \quad (7)$$

This value is always less equal to m, so task window variant 1 delivers a better or at least the same time bound as without using a task window. Here are some border cases:

- $m_i = m \rightarrow h = \lceil log_{1-1/m_w}(m_w/m_i) \rceil + m_w$, no difference to above
- $m_i = m_w \rightarrow h = m$, no difference to above

4.4.2. Variant 2: drying-out task window. Instead of immediately replenishing the task window after each hormone cycle to keep it always at a constant size, we could also replenish it only after it has run empty. So the task window will only be replenished after drying-out. This significantly simplifies the calculation of the upper time bound for task allocation. If the task windows are not replenished before drying-out, they all will have run empty after m_w hormone cycles at the latest. Then they will be replenished. If m_i tasks are covered by the initial task windows and if we use a constant offset (as in the original task window approach), this sequence will happen m/m_i times. Therefore, the maximum number of hormone cycles to assign all tasks using variant 2 calculates to:

$$h = \lceil m_w m/m_i \rceil \quad (8)$$

This value is always less equal to m, so task window variant 2 delivers a better or at least the same time bound than without using a task window.

4.4.3. Variant 3: drying-out task window with variable offset. Finally we can combine the drying-out task window with the variable offset. Consequently, the offsets might shrink from the beginning. In the worst case, all offsets have vanished after the task windows have run empty for the first time. This takes m_w hormone cycles at most while m_i tasks have been assigned (since all task windows are empty now). The remaining $m-m_i$ tasks are assigned at least at a rate of one task per hormone cycle. Therefore, the maximum number of hormone cycles to assign all tasks using variant 3 calculates to:

$$h = m_w + m - m_i \quad (9)$$

This value is always less equal to m, so task window variant 3 delivers a better or at least the same time bound than without using a task window.

5. Evaluation

To evaluate the task window, its different variants have been implemented in the hormone simulator (see section 3). A grid of 64 PEs with a set of 64 tasks have been used as basic evaluation scenario. The evaluation compares task assignment without the task window to task assignment with the different variants of the task window using different values for the task window size m_w and the number of initially covered tasks m_i (different offsets). Additionally, a comparison to task assignment without task window is done, where each PE applies only on a subset of 8 tasks instead of all tasks. This also reduces communication overhead and speeds up task assignment at the expense of flexibility in self-optimization and self-healing.

Figure 8 shows the development of eager values sent per hormone cycle for some sample scenarios. Without using the task window (blue curve) 4096 hormones (64 PEs x 64 tasks) are sent in the first hormone cycle. A task window size of $m_w = 8$ (pink, green and light blue curves) limits the number to 512 hormones (64 PEs x 8 tasks), a size of $m_w = 2$ (violet curve) to 128 hormones (64 PEs x 2 tasks). This is according to the theoretical predictions in the previous sections. It can also be seen that increasing the number of initially covered tasks m_i to 32 (green curve) or 64 (light blue curve) speeds up the task assignment. The effect of the drying-out task window can be seen at the oszillating green curve. Overall, task assignment using task windows produces a smaller amount of hormones sent for the most hormone cycles. The smallest task window results in the longest time for task assignment.

In the following, the aspects of communication overhead, assignment time, assignment quality, influence of

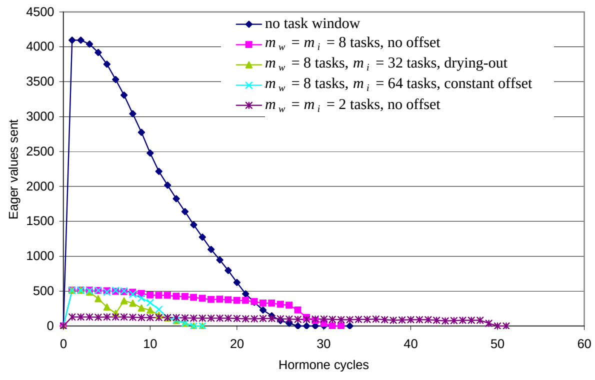


Figure 8. Eager values sent during self-configuration for sample scenarios

self-optimization and robustness are examined using the scenarios listed in table 1. Figure 9 shows the maximum amount of eager values sent per hormone cycle for all scenarios. The results confirm the theoretically derived reduction according to formula (2). Furthermore, the maximum communication overhead for task window size $m_w = 8$ in scenarios B .. K is identical to scenario M, where each PE applies for 8 tasks only. Figure 10 shows the overall number of eager values sent to allocate all tasks. This also confirms the effectivity of the task window. Larger values of m_i lead to less overall overhead since task allocation is finished earlier. The duration of task assignment is examined in figure 11. Measured values are compared to the theoretical upper bounds derived in section 4. It can be seen that all bounds hold. Furthermore it shows that the task window can speed up task allocation significantly. Figure 12 shows the influence of the task window on assignment quality. The quality measure published in [26] taking into account PE suitability, PE load and communication distance is used. For the given PE and task environment, an optimal quality of 0.9 (marked by the orange line in the diagram) can be reached. The measured values show only small variations in quality (0.74 to 0.83) close to the optimal value. It can be seen that the task window only reduces the quality slightly. In this context the influence of self-optimization is interesting, since the quality reduction is mainly caused by tasks not being in the task window of the best suited PE as stated in section 4.1. This can be corrected by self-optimization. Figure 13 examines self-optimization for scenario H, which has the worst quality (0.74) after self-configuration. Self-optimization is done every 64 hormone cycles (indicated by the eager values sent, pink line). It can be seen that the assignment quality (blue line) rises after each self-optimization cycle and finally reaches a value of 0.84. This is even better then the best value without a task window after self-configuration (0.83). Finally, figure 14 examines the system robustness without and with the task window. Here, more and more PEs were destroyed while self-healing tries to keep the system up by assigning the failing tasks to other PEs as long as possible. As maximum load, each PE is able to execute 5 tasks. The blue bars show the number of destroyed PEs that have been completely compensated by self-healing thus leading to no task failures. The red bars show the number of destroyed PEs which cause 50% of the tasks to fail. It can be seen that the task window (scenarios B .. L) does not harm the robustness of the system compared to having no task window[5] (scenario A). Also, the expected degradation of robustness can be observed, when PEs only apply for a limited number of tasks to reduce communication overhead without using the task window technique (scenario M).

Table 1. Evaluation scenarios

A	no task window
B	$m_w = m_i = 8$ tasks, no offset
C	$m_w = m_i = 8$ tasks, no offset drying-out
D	$m_w = 8$ tasks, $m_i = 32$ tasks, variable offset
E	$m_w = 8$ tasks, $m_i = 32$ tasks, constant offset
F	$m_w = 8$ tasks, $m_i = 32$ tasks, drying out with variable offset
G	$m_w = 8$ tasks, $m_i = 32$ tasks, drying out with constant offset
H	$m_w = 8$ tasks, $m_i = 64$ tasks, variable offset
I	$m_w = 8$ tasks, $m_i = 64$ tasks, constant offset
J	$m_w = 8$ tasks, $m_i = 64$ tasks, drying out with variable offset
K	$m_w = 8$ tasks, $m_i = 64$ tasks, drying out with constant offset
L	$m_w = m_i = 2$ tasks, no offset
M	no task window, each PE applies for a fixed set of 8 tasks

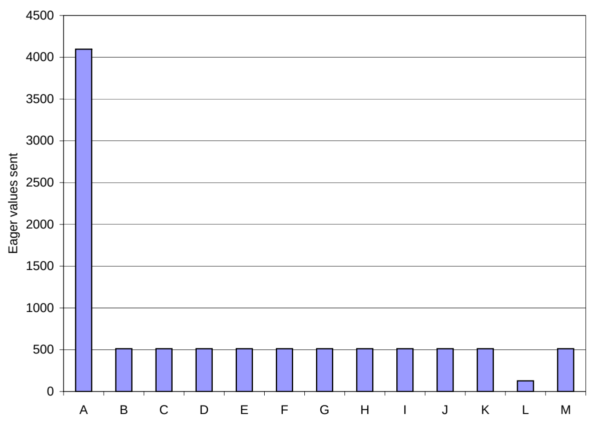


Figure 9. Maximum eager values sent per hormone cycle

5. The slight variations in the blue bars result from random PE destruction and different task locations for each scenario.

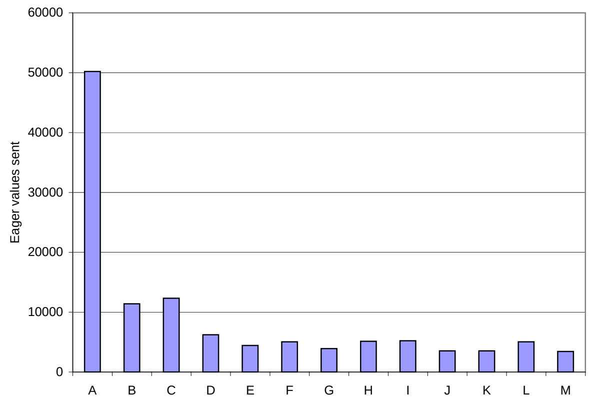


Figure 10. Overall eager values sent to allocate all tasks

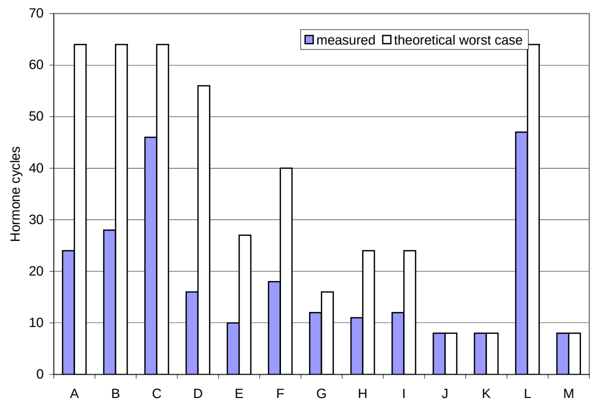


Figure 11. Duration of self-configuration, measured values compared to theoretical worst case

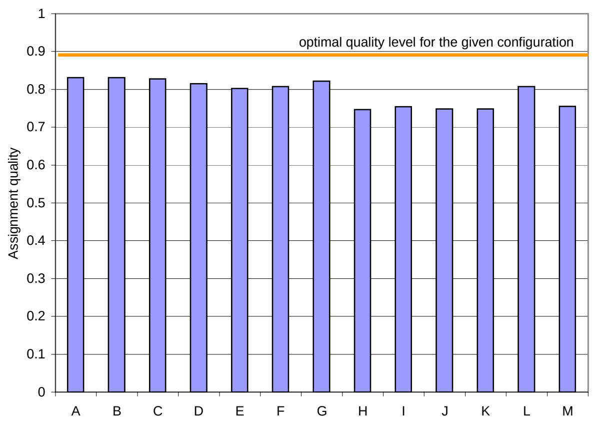


Figure 12. Quality of task assignment

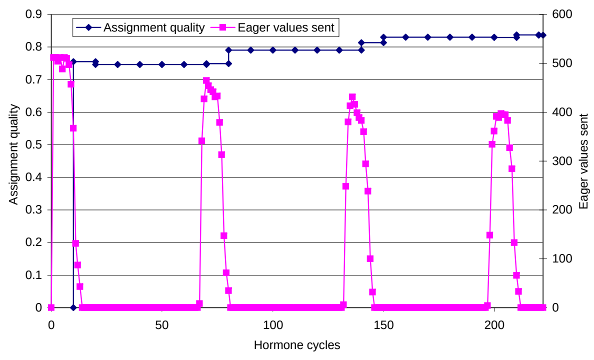


Figure 13. Self-optimization improving assignment quality

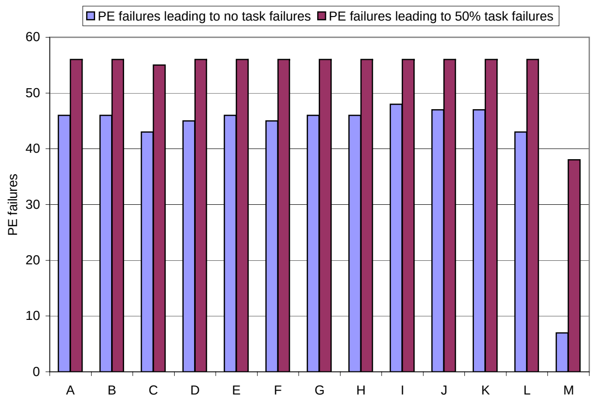


Figure 14. Robustness against PE failures

6. Conclusion

The paper presents and evaluates the concept of a task window to reduce the communication overhead of hormone based task allocation. Several variants have been proposed and examined on a theretical level. The practical evaluation confirms these theoretical results. It shows that the task window technique is an effective way to reduce communication overhead while maintaining assignment quality and robustness. Furthermore, self-configuration times are reduced.

References

[1] G. Jetschke, *Mathematik der Selbstorganisation*. Harry Deutsch Verlag, Frankfurt, 1989.

[2] R. Whitaker, “Self-Organization, Autopoiesis, and Enterprises,” 1995. [Online]. Available: http://www710.univ-lyon1.fr/ jmathon/autopoesis/Main.html

[3] IBM, “Autonomic Computing,” 2003. [Online]. Available: http://www.research.ibm.com/autonomic/

[4] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *IEEE Computer*, pp. 41–50, Jan. 2003.

[5] VDE/ITG (Hrsg.), "VDE/ITG/GI-Positionspapier Organic Computing: Computer und Systemarchitektur im Jahr 2010," *GI, ITG, VDE*, 2003.

[6] H. Schmeck, "Organic Computing - A New Vision for Distributed Embedded Systems," in *8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2005)*, Seattle, USA, May 2005, pp. 201–203.

[7] DFG Schwerpunktprogramm 1183, "Organic Computing," 2007. [Online]. Available: http://www.aifb.uni-karlsruhe.de/Forschungsgruppen/EffAlg/projekte/oc/inhalte

[8] IEEE, "Organic Computing Task Force," 2009. [Online]. Available: http://www.organic-computing.org/ieeetaskforce/

[9] EU, "Program Future Emerging Technolgies FET - Complex systems," 2009. [Online]. Available: http://cordis.europa.eu/ist/fet/co.htm

[10] CSIRO, "Centre for Complex Systems," 2009. [Online]. Available: http://www.dar.csiro.au/css/

[11] 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.

[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*, 1991, pp. 237–245.

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

[15] T. Decker, R. Diekmann, R. Lï¿½ling, and B. Monien, "Universelles dynamisches task-mapping," in *Konferenzband des PARS'95 Workshops in Stuttgart, PARS-Mitteilung 14*, 1995, pp. 122–131.

[16] 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*, vol. 1, 2003, pp. 31 – 36.

[17] J. Finke, K. Passino, and A. Sparks, "Stable task load balancing strategies for cooperative control of networked autonomous air vehicles," in *IEEE Transactions on Control Systems Technology*, vol. 14, 2006, pp. 789–803.

[18] 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*, 1994, pp. 414 – 421.

[19] F. Kluge, J. Mische, S. Uhrig, and T. Ungerer, "CAR-SoC - Towards and Autonomic SoC Node," in *Second International Summer School on Advanced Computer Architecture and Compilation for Embedded Systems (ACACES 2006)*, L'Aquila, Italy, July 2006.

[20] M. Nickschas and U. Brinkschulte, "Guiding Organic Management in a Service-Oriented Real-Time Middleware Architecture," in *6th IFIP Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS 2008)*, Capri, Italy, October 2008.

[21] J.Becker, K.Brändle, U. Brinkschulte, J. Henkel, W. Karl, T. Köster, M. Wenz, and H. Wörn, "Digital On-Demand Computing Organism for Real-Time Systems," in *Workshop on Parallel Systems and Algorithms (PASA), ARCS 2006*, Frankfurt, Germany, March 2006.

[22] M. Pacher and U. Brinkschulte, "Real-Time Distribution of Time-Dependant Tasks in Heterogeneous Environments," in *First IEEE Workshop on Self-Organizing Real-Time Systems (SORT 2010)*, 2010.

[23] U. Brinkschulte, M. Pacher, and A. v. Renteln, "An Artificial Hormone System for Self-Organizing Real-Time Task Allocation in Organic Middleware," in *Organic Computing*. Springer, 2008.

[24] A. von Renteln, U. Brinkschulte, and M. Pacher, "The Artificial Hormone System - An Organic Middleware for Self-organising Real-Time Task Allocation," *Organic Computing - A Paradigm Shift for Complex Systems, Springer Verlag, Basel*, 2011.

[25] U. Brinkschulte and M. Pacher, "An Agressive Strategy for an Artificial Hormone System to Minimize the Task Allocation Time," in *Third IEEE Workshop on Self-Organizing Real-Time Systems (SORT 2012)*, Shenzhen, China, 2012.

[26] U. Brinkschulte, A. von Renteln, and M. Pacher, "Measuring the Quality of an Artificial Hormone System based Task Mapping," in *ACM Autonomics 2008*, Turin, Italy, September 23-25 2008.