12

An Artificial Hormone System for Self-Organizing Real-Time Task Allocation in Organic Middleware

Uwe Brinkschulte, Mathias Pacher, and Alexander von Renteln

University of Karlsruhe, Bldg. 40.28, Engler-Bunte-Ring 8, 76131 Karlsruhe, Germany.
brinks@ira.uka.de, pacher@ira.uka.de, renteln@ira.uka.de

Summary. This article presents an artificial hormone system for a completely decentralized realization of self-organizing task allocation. We show that tight upper bounds for the real-time behavior of self-configuration, self-optimization and self-healing can be given. We also calculate the communication load produced by the hormone system and find it acceptable.

Key words: Decentralized control loops , real-time task allocation, task clustering, hormone simulator

12.1 Introduction

Today's computational systems are growing increasingly complex. They are built from large numbers of heterogeneous processing elements with highly dynamic interaction. Middleware is a common layer in such distributed systems, which manages the cooperation of tasks on the processing elements and hides the distribution from the application. It is responsible for seamless task interaction on distributed hardware. Like shown in figure 12.1, all tasks are interconnected by the middleware layer and are able to operate beyond processing element boundaries as if residing on a single hardware platform. To handle the complexity of today's and even more tomorrow's distributed systems, self-organization techniques are necessary. These systems 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. These so-called self-x features are essential for the idea of Organic Computing. Middleware is well-suited to realize such self-x features. By autonomously choosing an initial task allocation, which means finding the best initial processing element for each task, middleware can configure the distributed system. By changing the task allocation, middleware can optimize the system in case of changing environmental

R.P. Würtz (ed.), *Organic Computing*. Understanding Complex Systems,
doi: 10.1007/978-3-540-77657-4_12,

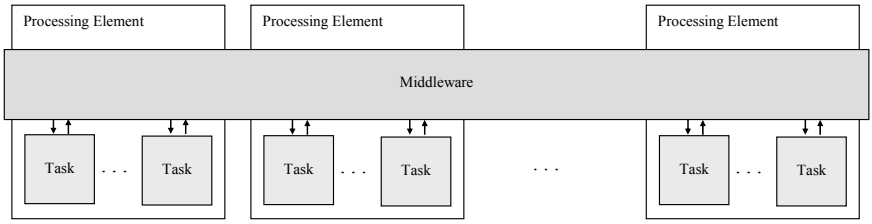


Fig. 12.1. Middleware in a distributed system.

conditions and heal it in case of processing element or task failures. Especially for self-healing, it is important that these organic features are decentralized to avoid single points of failure. This work presents an artificial hormone system for task allocation to heterogeneous processing elements. The proposed approach has the following properties:

- It is **completely decentralized**. There are no central decision making instances to determine the task allocation. Each processing element (PE) in the heterogeneous distributed system decides which tasks to take on the basis of simple local rules and information received from other processing elements.
- It is **self-organizing**. There is no external organization instance which influences the task allocation. This is done by the interaction of the PEs only.
- It is **self-configuring**. The presented approach determines an initial task allocation, which takes into account the capabilities (e.g. computational power, memory, etc.) and the state (e.g. operation temperature, energy level, etc.) of the heterogeneous PEs.
  The artificial hormone system is also able to respect related tasks (which often have a high communication rate) in order to cluster them close together, thus forming "organs".
- It is **self-optimizing**. The task allocation autonomously adapts to changing environmental conditions and states of the PEs (e.g. decreasing energy level, increasing temperature) during operation. Self-optimization also includes the assignment of newly arriving tasks to PEs.
- It is **self-healing**. Due to the lack of central instances and due to the capability of self-optimization the presented approach automatically compensates the effects of failing tasks or PEs by reordering the task allocation.
- It is **real-time** capable. There are tight upper time bounds for self-configuration and self-optimization. This bounds are partially valid for self-healing, too.
- It produces **limited communication overhead**, which is reasonable for embedded applications.

The term "artificial hormone system" was chosen because our approach was highly inspired by the hormone system of higher animals. There are several comparable properties between the hormone system in biology and our technical system:

- In biology, chemical signals called messengers or hormones are unspecifically spread to certain regions of the body or the whole body to cause some effects. The messengers (or hormones) of our artificial hormone system are also not addressed to a specific processing element (PE); rather they are spread in the neighborhood of a processing element or over the whole processor grid.
- The reaction of a cell to a hormone depends on the cell itself. In the same way, the reaction of a PE to a messenger in our system depends only on the specification of the PE itself (see properties mentioned above).
- A PE is able to react to a received messenger in different ways: It starts, stops, continues or quits the execution of a task. In reaction to this, the PE itself is also able to spread messengers over the system establishing a closed control loop, which stabilizes the system. Such loops can also be found in nature: the hormones T3 and T4 of the thyroid implement a closed loop controlling the body temperature.
- Like in the biological hormone system, these closed loops are completely decentralized. As for cells, removing PEs from the loop does not harm the system as long as there are enough PEs left to execute tasks and send or receive messengers.
- The hormones of higher animals are reduced by their metabolism, so they are not effective after some time (unless new ones are produced). In our implementation of the artificial hormone system, the effectiveness of the messengers is bounded by time stamps. If not renewed the messengers of our system expire, too.

It has to be stated that our "artificial hormone system" is not a copy of the biological hormone system, but it has been inspired by nature and its strategies. In biology, hormones are chemical objects transmitted via chemical processes and reactions. In our approach, the messengers are bits and bytes transferred via communication links. However, the effects and principles are similar. This is why we have called our messengers hormones as well.

In the following we will present our approach in detail and we will discuss and prove the enumerated properties.

12.2 An artificial hormone system for a decentralized realization of the self-x-properties

For task allocation, three types of hormones are used:

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

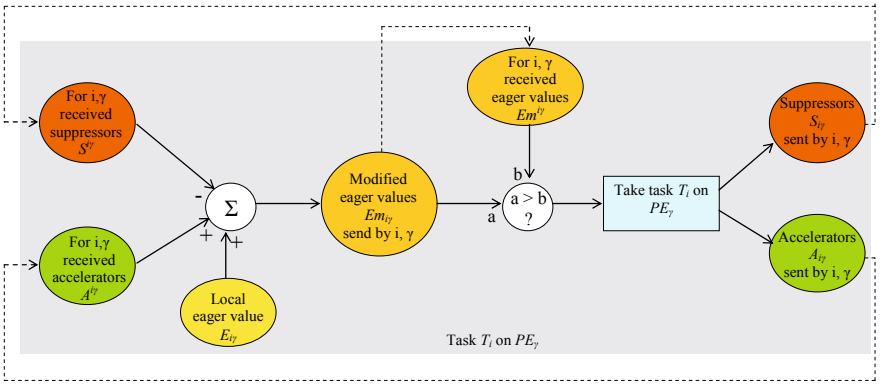


Fig. 12.2. Hormone-based control loop.

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 12.2 sketches the basic control loop used to assign a task T_i to a processing element. The notation scheme is as follows: $H^{i\gamma}$ means a hormone for task T_i executed on PE_γ and $H_{i\gamma}$ means a hormone from task T_i executed on PE_γ. Latin letters are task indices and Greek letters are processing element indices. This closed control loop is executed for every task on every processing element. Based on the level of the three hormone types it determines if a task T_i is executed on a processing element PE_γ or not. 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 sent 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. If $Em_{i\gamma}$ is greater than 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.

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

12.2.1 Notation

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 done in figure 12.2). Accordingly, we use upper case Latin letters for task sets and upper case Greek letters for sets of processing elements.
Let

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

$$\omega = |\Omega|$$

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

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

Thus, we obtain the set of all processing elements as

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

Φ_γ be the set of processing elements which are neighbored to processing element PE_γ. Notice that this relation is reflexive.
Neighbored 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.

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

M be the set of all tasks in the system.
m be the number of all tasks in the system.

$$m := |M|$$

IM be the set of indices of all tasks.

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

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

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

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 extend the notation from figure 12.2 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.

12.2.2 Different kinds of hormones

Using the notation introduced above we now describe the used hormones and their function in detail and start by explaining the eager values:

Local eager value $E_{i\gamma}$: This value states the initial suitability of PE_γ for task T_i. It assures that task allocation is adapted to 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 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. Thereby it 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 one 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. E.g., the lower the energy level or the higher the temperature of a PE, the stronger this suppressor becomes.

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. The stronger the accelerator the stronger is the attraction. The basic idea behind this is that related tasks work on common problems and have to communicate frequently, making short communication distances useful. Related tasks form a kind of 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. This leads to stable task allocation in the context of self-configuration. But to allow self-optimization, the possibility of changes in 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 now more suitable. Such a task migration introduces costs expressed by the stay accelerator by means of favoring the stay of task T_i on PE_γ. It is sent from task T_i on PE_γ to itself (i, γ). The stronger the stay accelerator, the 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 task allocation.
- It is **self-configuring**, as 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 want to be taken. This is done by sending suppressors. Of course, the 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.

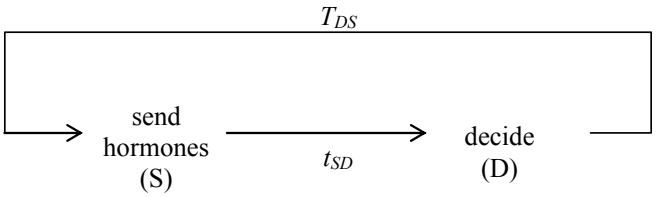


Fig. 12.3. Hormone cycle.

- The **self-optimization** is done by offering tasks. The point of time for such an offer is determined by the task or by the PE itself. It can be done periodically or at a point in time when 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 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 with 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 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 following sections. The communication overhead introduced will be analyzed there, too.

12.3 Dynamics of the artificial hormone system

In this section, the dynamics of the artificial hormone system and the conditions and rules for its correct operation will be presented. Figure 12.3 shows the cyclic sequence of sending out hormones and deciding on task allocation. The sequence starts with "send hormones" (S) to create the knowledge base for the first decision. At least the eager values need to be available. At time t_{SD} after sending the hormones, a decision (D) to allocate tasks is made based on the received hormones. This process is repeated after a time of t_{DS}.

12.3.1 Dynamics of task allocation

Let PE_γ be a processing element willing to run a task T_i. We need to distinguish three cases:

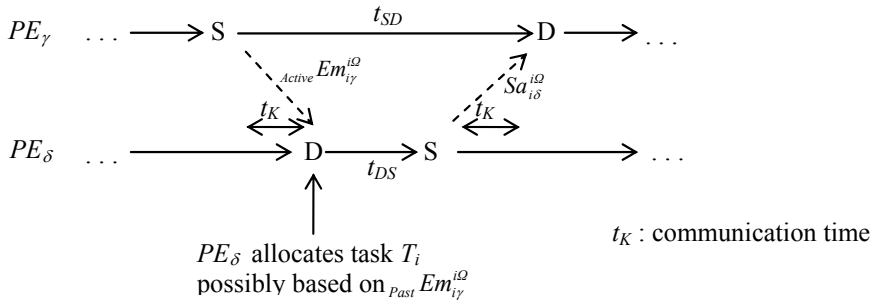


Fig. 12.4. Worst-case timing scenario of the hormone exchange for task allocation

Case 1: All eager values $Em_{i\gamma}^{i\Omega}$ of all processing elements $PE_\gamma \in \Omega$ for task T_i are constant and spread over the entire system. Thus, the system is in a steady state and all PEs make their decisions based on up-to-date and constant values. Then, PE_γ can allocate a task if it has the highest eager value or, in the case of equal eager values, a higher priority.

Case 2: The eager value $Em_{i\gamma}^{i\Omega}$ of processing element PE_γ for task T_i declines (e.g. by suppressor influence), i.e. $_{Active}Em_{i\gamma}^{i\Omega} < {}_{Past}Em_{i\gamma}^{i\Omega}$. In this case PE_γ may allocate the task T_i if the declined eager value $_{Active}Em_{i\gamma}^{i\Omega}$ is still sufficient. All the other PEs will not allocate the task, as they know either $_{Active}Em_{i\gamma}^{i\Omega}$ or $_{Past}Em_{i\gamma}^{i\Omega}$, and PE_γ wins with both values.

Case 3: The eager value $Em_{i\gamma}^{i\Omega}$ of the processing element PE_γ for task T_i increases (e.g. by accelerator influence), i.e. $_{Active}Em_{i\gamma}^{i\Omega} > {}_{Past}Em_{i\gamma}^{i\Omega}$. This case is critical if PE_γ becomes the winner by the increased eager value $_{Active}Em_{i\gamma}^{i\Omega}$, because other PEs might not yet know this increased eager value and therefore decide wrongly. Thus, PE_γ may only allocate the task T_i after the new eager value $_{Active}Em_{i\gamma}^{i\Omega}$ has successfully been submitted to all PEs and until PE_γ itself has received a possible acquisition suppressor $Sa_{i\delta}^{i\Omega}$ from another PE_δ ($\gamma \neq \delta$), which allocated the task T_i based on the old, lower eager value $_{Past}Em_{i\gamma}^{i\Omega}$.

But how long is the waiting time for PE_γ? Figure 12.4 shows the worst-case scenario, in which PE_δ allocated the task T_i just before the new eager value $_{Active}Em_{i\gamma}^{i\Omega}$ from PE_γ has been received. PE_γ may not come to a decision until it has received the possibly incoming suppressor $Sa_{i\delta}^{i\Omega}$ from PE_δ. The communication time t_K needed by a hormone to spread the whole system is very important. It is possible to establish the following rule for task allocation for increasing eager values as well as conditions for the times t_{DS} and t_{SD}.

Rule: If a processing element PE_γ is able to allocate a task T_i only by the increased eager value $_{Active}Em_{i\gamma}^{i\Omega}$ then it may not decide before the next communication cycle to allow the new eager value $_{Active}Em_{i\gamma}^{i\Omega}$ to

spread and to wait for potentially incoming suppressors. This is true if (follows directly from figure 12.4):

$$t_{SD} \geq t_{DS} + 2t_K$$

Thus, the cycle time results in:

$$t_C = t_{SD} + t_{DS}$$

Of course, the cycle time should be kept at a minimum, therefore
1) t_{DS} should be as small as possible, ideally 0.
2) $t_{SD} \geq t_{DS} + 2t_K$, ideally with $t_{DS} = 0$: $t_{SD} \geq 2t_K$

Conclusion: For the allocation of a task T_i by a processing element PE_γ the following cases can be distinguished:

1) ${}_{Past}Em_{i\gamma}^{i\Omega} = {}_{Active}Em_{i\gamma}^{i\Omega}$: The task can be allocated, if ${}_{Past}Em_{i\gamma}^{i\Omega} = {}_{Active}Em_{i\gamma}^{i\Omega}$ qualifies the processing element.
2) ${}_{Past}Em_{i\gamma}^{i\Omega} > {}_{Active}Em_{i\gamma}^{i\Omega}$: The task can be allocated, if ${}_{Active}Em_{i\gamma}^{i\Omega}$ qualifies the processing element.
3) ${}_{Past}Em_{i\gamma}^{i\Omega} < {}_{Active}Em_{i\gamma}^{i\Omega}$: The task can be allocated, if ${}_{Past}Em_{i\gamma}^{i\Omega}$ qualifies the processing element. Otherwise the decision has to be postponed until the following cycle.

♦

12.3.2 Self-configuration: worst case timing behavior

Figure 12.5 shows the detailed cycle of the hormone distribution and interpretation based on figure 12.3. First the hormones (eager values, suppressors and accelerators) for all tasks PE_γ is interested in are emitted by PE_γ. Therefore, we define

$$M_\gamma := \{T_j \mid T_j \in M \text{ and } PE_\gamma \text{ is interested in } T_j\}$$

After waiting the time t_{SD}, the decision for a task $T_i \in M_\gamma$ is made. Afterwards i is incremented and the next cycle starts ($t_{DS} = 0$). This way, in each cycle the hormones for all relevant tasks are emitted and the decision for exactly one task is made. To decide on only one task per cycle allows the hormones to take effect. If task allocation took place all at once for all available tasks, the accelerators emitted when a task is allocated would not have a chance to make an impact as all the tasks would already be allocated in the first cycle.

To calculate the worst case timing behavior of this allocation process, we make the following basic **assumption**: All tasks (m tasks) have to be distributed on all PEs and all PEs are interested in all tasks.

First we make a further assumption to simplify the scenario: Let all eager values be constant, i.e., there are no accelerators and suppressors. Then, all

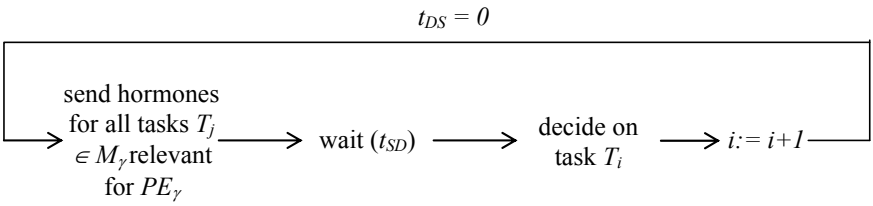


Fig. 12.5. Cycle of the hormone distribution and decision by PE_γ

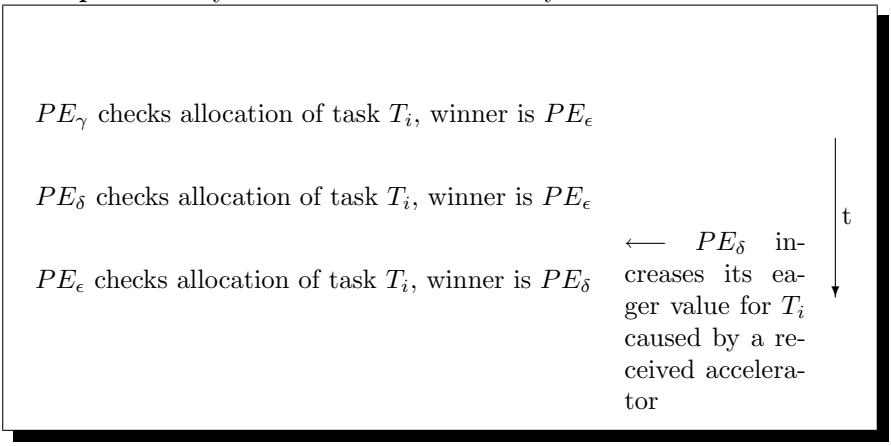
tasks have been handled by all PEs and have been allocated after m cycles and it follows:

$$\text{Worst Case Timing Behavior } = m \text{ cycles} \tag{12.1}$$

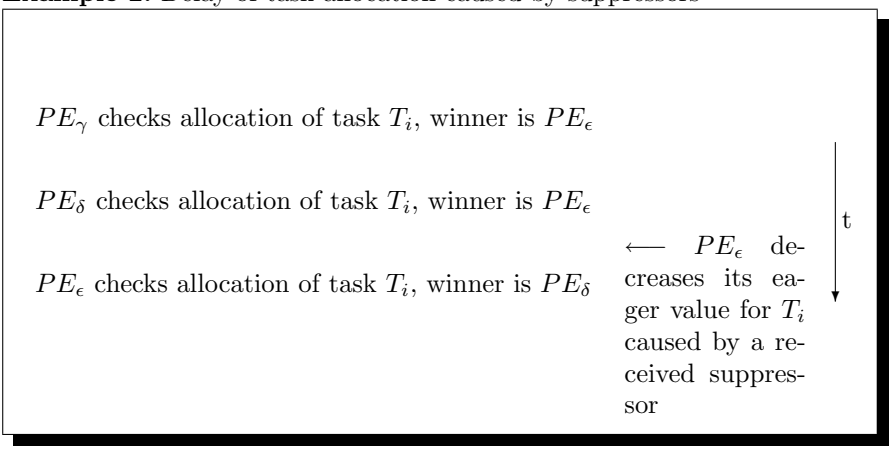
♦

In the following we remove the simplifying assumption of constant eager values and allow accelerators and suppressors. Now some tasks may not have been allocated after m cycles. This can be caused by accelerators and suppressors as shown in the following examples. In the first example three PEs are checking one after another the possibility to allocate task T_i. While PE_γ and PE_δ are checking, PE_ϵ still is the winner. After PE_δ has checked, it increases its eager value caused by a received accelerator. If afterwards PE_ϵ checks for allocation, PE_δ becomes the winner. However, PE_δ will not check again for allocation within the next m cycles. The second example shows a similar scenario, this time caused by an eager value decreased by a suppressor.

Example 1: Delay of task allocation caused by accelerators



Example 2: Delay of task allocation caused by suppressors



At worst in both cases task T_i will not be re-checked until a complete cycle of all other tasks, thus after m cycles. Afterwards, the same scenario could occur again. However, the maximal number of cycles is limited: A change of the eager value by suppressors or accelerators only takes place if a task has been allocated somewhere in the system (Assumption: Monitoring accelerators and suppressors are constant during the initial self-configuration). It follows that in each allocation cycle at least one task will be allocated. Thus, in the case of a variable eager value we get the following worst case timing behavior for the self-configuration:

$$\text{Worst Case Timing Behavior } = m^2 \text{ cycles} \tag{12.2}$$

♦

12.3.2.1 Improvement of the worst case timing behavior

By refining the algorithm presented in figure 12.5 it is possible to improve the timing behavior of the worst case scenario.

Refinement 1: If a PE_γ sends an eager value for a task T_k which was increased by an accelerator and this increased eager value is higher than all other values received for task T_k so far, then PE_γ exits the regular sequential decision cycle and checks for T_k instead.[1]

By using this refinement the following timing behavior results for the eager values increased by accelerators:

The worst case scenario is as follows: Assume, task T_i would be allocated by processing element PE_γ at the m^{th} cycle. Exactly in this cycle, the corresponding eager value $Em_{i\gamma}^{i\Omega}$ is incremented by an accelerator of another task.

[1] This is conform to the rule from section 12.3, that an interval of $t_{SD} \geq t_{DS} + 2t_K$ will be waited between sending and decision making.

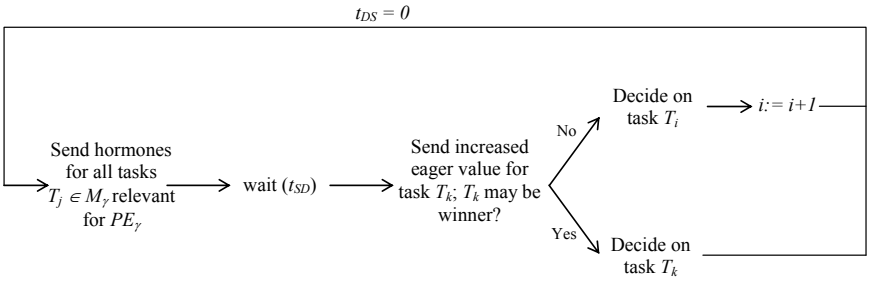


Fig. 12.6. Cycle of the hormone distribution and decision making for a PE_γ using the first refinement.

One of $m-1$ other tasks may be responsible for sending this accelerator by being allocated somewhere. As a result, T_i will not be allocated on PE_γ and will be re-checked in cycle $m+1$.

Further delay will arise if another accelerator is be sent in cycle $m+1$ and the eager value $Em_{i\gamma}^{i\Omega}$ for task T_i will be increased another time. Now, $m-2$ tasks may be responsible, one of which has been allocated. It follows that all tasks are assigned no later than $m+(m-1) = 2m-1$ cycles, which is also shown in the following scheme:

Cycle 1 :	$T_1\ T_2\ ...\ T_{m-2}\ T_{m-1}\ T_m$	
Cycle 2 :	$T_1\ T_2\ ...\ T_{m-2}\ T_{m-1}\ T_m$	
...	... ...	
...	... ...	
Cycle m :	$T_1\ T_2\ ...\ T_{m-2}\ T_{m-1}\ T_m$	$\leftarrow T_m$ assigned, accelerator sent
Cycle $m+1$:	$T_1\ T_2\ ...\ T_{m-2}\ T_{m-1}$	$\leftarrow T_{m-1}$ assigned, accelerator sent
Cycle $m+2$:	$T_1\ T_2\ ...\ T_{m-2}$	$\leftarrow T_{m-2}$ assigned, accelerator sent
...	... ...	...
...	... ...	...
Cycle $2m-2$:	$T_1\ T_2$	$\leftarrow T_2$ assigned, accelerator sent
Cycle $2m-1$:	T_1	$\leftarrow T_1$ assigned, accelerator sent

As a conclusion, we notice assuming that refinement 1 holds:

$$\text{Worst Case Timing Behavior } = 2m-1 \text{ cycles} \tag{12.3}$$

♦

Now we define a similar refinement for delays caused by suppressors:

Refinement 2: If a PE_γ receives an eager value for a task T_k which was decreased by a suppressor and therefore the own eager value is higher than

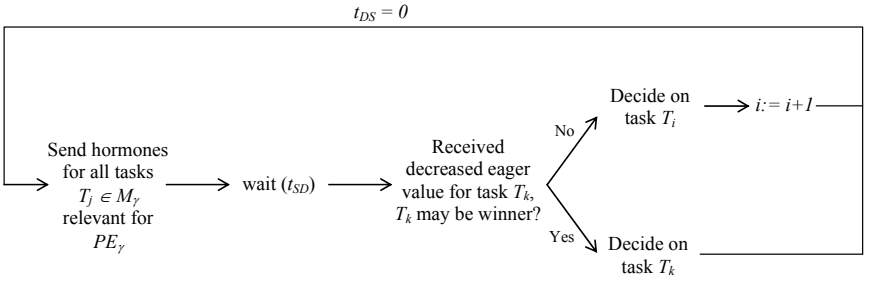


Fig. 12.7. Cycle of the hormone distribution and decision making for a PE_γ using the second refinement.

all other values received for task T_k so far, then PE_γ exits the regular sequential decision cycle and checks for T_k instead.

As a suppressor (similar like an accelerator) results from a task which has been allocated, the same worst-case timing behavior of $2m-1$ cycles will result from the same consideration as before. It should be noted that in our application refinement 2 can be omitted, because suppressors only affect the same tasks that created them. Therefore, a suppressor for a task is only emitted if this task has already been taken somewhere and need not be taken in the same cycle again.

12.3.2.2 Further improvements

Consequently, we narrow the scenario to the influence of accelerators on timing behavior and the worst-case timing behavior can be specified more precisely:

An accelerator is only sent to related tasks. Therefore a task T_i can not receive an accelerator from all the other $m-1$ tasks but only from the v_i tasks it is related to ($v_i \leq m-1$). Then, task allocation will be completed at the latest after

$$m + v_i \leq 2m - 1 \text{ cycles.}$$

Considering all tasks we get the following result:

$$\text{Worst Case Timing Behavior } = m + v_{\max} \text{ cycles} \tag{12.4}$$

where

$$v_{\max} := \max_{T_i \in M} \{v_i\}, \qquad \text{the largest number of related tasks.}$$

♦

Example 3: Differences caused by the improvements

We assume there are 10 tasks to be distributed in the system:

10 tasks: T_1 ... T_{10}, thus $m = 10$
Related tasks: T_1 ... T_4 and T_5 ... T_{10}, thus

$$v_{\max} = \max\{v_1, ..., v_{10}\} = \max\{4,\ 4,\ 4,\ 4,\ 6,\ 6,\ 6,\ 6,\ 6,\ 6\} = 6$$

The result is

- with $2m - 1 = 20 - 1 = 19$ cycles as an upper limit for the self-configuration (without any assumption about the largest number of related tasks).
- with $m + v_{\max} = 10 + 6 = 16$ cycles as an upper limit for the self-configuration (including the information about the largest number of related tasks).
- Furthermore it may happen that not all PEs apply for all tasks, but e.g. PE_1 and PE_2 for T_1 ... T_5 and PE_3 and PE_4 for T_6 ... T_{10}. Then

  $$m_{\max} = \max\{m_1,\ m_2,\ m_3,\ m_4\} = \{5,\ 5,\ 5,\ 5\} = 5$$

  Then, the upper limit for the self-configuration is

  $$m_{\max} + v_{\max} = 5 + 6 = 11 \text{ cycles.}$$

Further reductions of the worst-case timing behavior may take place, if not all PEs apply for all m tasks. If we release this basic assumption, the following timing behavior results for a PE_γ applying for a task T_i: The task allocation is finished at the latest after

$$m_\gamma + v_i \text{ cycles}$$

where

$$m_\gamma = |M_\gamma|, \qquad \text{the number of tasks for which } PE_\gamma \text{ applies.}$$

Extending this result to all tasks we obtain:

$$\text{Worst Case Timing Behavior } = m_{\max} + v_{\max} \text{ cycles} \tag{12.5}$$

where

$$m_{\max} := \max_{PE_\gamma \in \Omega}\{m_\gamma\}, \qquad \text{the largest number of tasks a PE is applying for.}$$

♦

Example 3 illustrates the differences of these improvements.

12.3.3 Self-optimization: worst case timing behavior

Self-optimization means the relocation of a task T_i from a processing element PE_γ to another processing element PE_δ. This relocation takes place only if PE_γ currently executing T_i offers this task. Thus, PE_γ chooses the point in time for the optimization, e.g., periodically or when T_i is idle. Additionally, PE_γ operates the task until it is completely transferred to PE_δ. Notice that even $\gamma = \delta$ is possible, which means the task execution is continued on PE_γ. This has the following consequences:

- There is no blackout time (except the time used to transfer the task state).
- Real-time behavior is guaranteed.

The time interval from offering the task until to the completion of the transfer is bounded. Thus, the time for self-optimization is also bounded.

In the worst case, all processing elements offer all tasks for self-optimization, which leads to the same time bounds like for self-configuration.

$$\text{Worst Case Timing Behavior} = m_{\max} + v_{\max} \text{ cycles} \tag{12.6}$$

♦

Notice that there is no interruption in task execution because relocated tasks are operated by the previous PEs until completely transferred to the new PEs.

If we assume that only one task is offered per cycle the time for self-optimization is decreased considerably. Let's assume PE_γ would offer T_i for self-optimization. Then, the eager value $Em_{i\delta}^{i\Omega}$ would increase on all processing elements $PE_\delta \in \Omega$ applying for T_i as the acquisition suppressor $Sa_{i\gamma}^{i\Omega}$ would be dropped. If we use refinement 1 (see section 12.3.2.1) all eligible processing elements check for T_i in the next cycle. Assuming that only one task is offered in this cycle there are no further modifications of the eager values. Thus, T_i is assigned to a new processing element PE_δ in the next cycle.

$$\text{Worst Case Timing Behavior} = \ 1 \text{ cycle} \equiv const. \tag{12.7}$$

♦

12.3.4 Self-healing: worst case timing behavior

In case of processing element failure, the execution of its tasks fails until they are reassigned. The point in time when the processing element fails is not predictable. Therefore, we only obtain limited real-time behavior. Nevertheless, we are able to compute time bounds for this case. The worst case is that all processing elements are failing simultaneously, which means there is no chance for self-healing. Thus, we exclude this case and assume that there are still enough processing elements operational to execute all tasks. Then,

almost the same upper bound as for self-optimization holds. We only have to add the time the operational processing elements need to recognize that other processing elements failed by the expiration of the hormones.

$$\text{Worst Case Timing Behavior} = m_{\max} + v_{\max} + a \text{ cycles} \tag{12.8}$$

where a: number of cycles after which a not updated hormone is considered to be too old and thus not valid any longer (expiration time, see section 12.2.2). ♦

If only one PE_γ fails and there is no self-optimization in parallel, the tasks $T_i \in E_\gamma$ running on PE_γ will be reassigned due to their vanishing acquisition suppressors. If we use refinement 1 and consider the emission of accelerators when a task is taken, we obtain

$$\text{Worst Case Timing Behavior} = e_\gamma + \max_{T_i \in E_\gamma} \{v_i\} + a \text{ cycles} \tag{12.9}$$

where

$\max_{T_i \in E_\gamma} \{v_i\}$: greatest number of related tasks to the tasks running on PE_γ.

♦

12.4 Communication load introduced by the artificial hormone system

Now, we calculate the communication load introduced by the artificial hormone system. A processing element PE_γ is sending per cycle:

Broadcast to all other PEs : 1 modified eager value $Em_{j\gamma}^{j\Omega}$ for each task T_j PE_γ applied for

1 acquisition suppressor $Sa_{i\gamma}^{i\Omega}$ for each task T_i PE_γ has taken

Multicast to neighbors : 1 organ accelerator $Ao_{i\gamma}^{V_i \Phi_\gamma}$ for each task related to a taken task T_i

All other kinds of hormones are sent or used locally (see section 12.2).

We also need to know the sender information additionally to the eager values, suppressors and accelerators to be able to refresh the hormone values of a sender. Thus, we propose the following structure for a sent hormone:

$$\underbrace{\text{Type of hormone}}_{\text{Eager value, accelerator, suppressor}}, \underbrace{\text{PE-identification, Task-identification}}_{\text{Sender information}}, \text{Value}$$

12.4.1 Hormone communication load per processing element

Each PE_γ causes the following hormone communication load:

Broadcast to all other processing elements:

$$Db_\gamma = De * k_\gamma + Ds * e_\gamma \tag{12.10}$$

where

Db_γ : broadcast data load caused by PE_γ
De : data load to send an eager value
Ds : data load to send a suppressors
k_γ : number of tasks PE_γ applied for, which are not yet completely taken in the system

Multicast to neighbors:

$$Dm_\gamma = Da * \sum_{T_i \in E_\gamma} v_i \tag{12.11}$$

where

Dm_γ : multicast data load caused by PE_γ
Da : data load to send an accelerator

Following from this, we obtain the hormone communication load from any processing element PE_γ at the beginning of the task allocation (self-configuration) and in the steady state (all tasks are allocated, only self-optimization and self-healing take place):

$$\left.\begin{aligned} {}_{\text{Start}}Db_\gamma &= De * k_\gamma \\ {}_{\text{Start}}Dm_\gamma &= 0 \end{aligned}\right\} \quad e_\gamma = 0 \text{ at the start} \tag{12.12}$$

and

$$\left.\begin{aligned} {}_{End}Db_\gamma &= Ds * e_\gamma \\ {}_{End}Dm_\gamma &= Da * \sum_{T_i \in E_\gamma} v_i \end{aligned}\right\} \quad k_\gamma = 0 \text{ at the end} \tag{12.13}$$

♦

12.4.2 Overall hormone communication load

The overall hormone communication load introduced by the artificial hormones at any processing element PE_γ results from the sum of the multicasts to its neighbors and the sum of broadcasts of all processing elements:

$$D_\gamma = \sum_{PE_\delta \in \Omega} Db_\delta + \sum_{PE_\delta \in \Phi_\gamma} Dm_\delta \tag{12.14}$$

where D_γ : overall communication load of PE_γ

As a result, we can compute the overall communication load at any processing element PE_γ at the beginning of the task allocation and in the steady state:

$$\begin{aligned} {}_{\text{Start}}D_\gamma &= \sum_{PE_\delta \in \Omega} {}_{\text{Start}}Db_\delta + \sum_{PE_\delta \in \Phi_\gamma} {}_{\text{Start}}Dm_\delta \\ &= \sum_{PE_\delta \in \Omega} De * k_\delta \end{aligned} \tag{12.15}$$

and

$$\begin{aligned} {}_{End}D_\gamma &= \sum_{PE_\delta \in \Omega} {}_{End}Db_\delta + \sum_{PE_\delta \in \Phi_\gamma} {}_{End}Dm_\delta \\ &= \sum_{PE_\delta \in \Omega} Ds * e_\delta + Da * \sum_{PE_\delta \in \Phi_\gamma} \sum_{T_i \in E_\delta} v_i \end{aligned} \tag{12.16}$$

Now, we can calculate an upper bound for the overall communication load at any processing element at the beginning of the task allocation and in the steady state. We estimate the sums of the individual communication load by multiplying the maximal communication load with the number of assigned processing elements.

Considering the broadcast, we estimate the sum of individual broadcast communication load by multiplying the greatest existing broadcast communication load of all processing elements with the number of processing elements.

Considering the multicast, we estimate the sum of the individual multicast communication load of neighbored processing elements by multiplying the greatest existing multicast communication load with the greatest number of neighbors existing in the scenario. We obtain for each $PE_\gamma \in \Omega$:

$$\begin{aligned} {}_{\text{Start}}D_\gamma &\leq \omega * \max_{PE_\delta \in \Omega} \{{}_{\text{Start}}Db_\delta\} \\ &= \omega * De * k_{\max} \end{aligned} \tag{12.17}$$

and for each $PE_\gamma \in \Omega$ holds:

$$\begin{aligned} {}_{End}D_\gamma &\leq \omega * \max_{PE_\delta \in \Omega} \{{}_{End}Db_\delta\} + \max_{PE_\delta \in \Omega} \{\varphi_\delta\} * \max_{PE_\delta \in \Omega} \{{}_{End}Dm_\delta\} \\ &\leq \omega * Ds * e_{\max} + \varphi_{\max} * Da * e_{\max} * v_{\max} \end{aligned} \tag{12.18}$$

where

$k_{\max} := \max_{PE_\delta \in \Omega} \{k_\delta\}$, maximum of all k_δ

$e_{\max} := \max_{PE_\delta \in \Omega} \{e_\delta\}$, maximum of all e_δ

$v_{\max} := \max_{T_i \in M} \{v_i\}$, greatest number of related tasks, see section 12.3.2.2

$\varphi_{\max} := \max_{PE_\delta \in \Omega} \{\varphi_\delta\}$, greatest number of all neighbored processing elements.

♦

12.4.3 Example

In this section, we calculate the data load introduced by the artificial hormones in a concrete scenario. First, we define the structure of the hormones and the resulting data load.

Eager values, suppressors	2 bit for the type of hormone	
	4 bit x-coordinate of PE 4 bit y-coordinate of PE	(ID of PE (256 PEs at maximum))
	7 bit for the task ID	(128 tasks at maximum)
	7 bit value	(128 nuances of a hormone)
$\sum$	24 bit	

Thus, it follows:

$$De = Ds = 24 \text{ bit}$$

Accelerators	2 bit for the type of hormone	
	4 bit x-coordinate of PE	
	4 bit y-coordinate of PE	
	7 bit for the task ID	
	7 bit for the ID of related tasks 7 bit value	(repeated v_i times)
$\sum$	$17 + v_i * 14$ bit	

To calculate the worst case, we assume $v_i = v_{\max}$. Thus, it follows:

$$Da = 17 + v_{\max} * 14 \text{ bit}$$

Now, we define the values for the number of processing elements, tasks and so on:

$$\begin{aligned} \omega &:= 64 && \text{(Number of processing elements)} \\ \varphi_{\max} &:= 9 && \text{(Number of PEs neighbored to a PE)} \\ k_{\max} &:= 32 && \text{(Maximal number of tasks a PE applied for)} \\ e_{\max} &:= 2 && \text{(Maximal number of tasks taken by a PE)} \\ v_{\max} &:= 8 && \text{(Maximal number of tasks related to a task)} \end{aligned}$$

Using these values, we obtain for each $PE_\gamma \in \Omega$:

$$_{\text{Start}}D_\gamma \leq 64 * 24 * 32 \text{ bit} = 49152 \text{ bit} = 6144 \text{ bytes} \tag{12.19}$$

$$_{End}D_\gamma \leq 64 * 24 * 2 + 9 * 2 * (17 + 8 * 14) \text{ bit} = 674.25 \text{ bytes} \tag{12.20}$$

Let's assume a cycle time of 100 ms ($t_{SD} + t_{DS}$). Then, we can compute the maximal data load caused by the artificial hormones for each $PE_\gamma \in \Omega$:

$$_{\text{Start}}DS_\gamma \leq 10 * 6144 \text{ bytes/sec} = 60 \text{ kBytes/sec} \tag{12.21}$$

$$_{End}DS_\gamma \leq 10 * 674.25 \text{ bytes/sec} \approx 6.58 \text{ kBytes/sec} \tag{12.22}$$

As it can be seen, the data load caused by the artificial hormones is significantly higher at the beginning than in the steady state. However, there is only a small amount of user data to be sent at the beginning because the tasks are not yet assigned. In the steady state, there is more user data to be sent and the data load caused by the artificial hormones is small. In the best case, both effects eliminate each other thus resulting in a constant data load caused by the artificial hormones.

12.5 Related work

There are several approaches for task allocation in middleware. 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 grouping: 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 there is no possibility to get group information from this group anymore. In our approach there is no central task distribution instance and therefore no single point of failure can occur.

Another approach is presented in [7]. The authors present two algorithms for task scheduling. The first algorithm, Fast Critical Path (FCP) makes sure

time constrains to be 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 include the failing of processing elements.

In [6], 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 absolutely not considered by this decentralized dynamic load balancing.

[8] presents a load balancing scheme for task allocation based on local workpiles (of PEs) storing the tasks to be executed. The authors propose to execute a load balancing algorithm between two PEs 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 and real-time constraints.

Other approaches of load balancing are presented in [1, 3, 4, 5, 9]. None of them cover the whole spectrum of self-x-properties, task clustering, and real-time conditions like our approach.

12.6 Conclusion and further work

We presented an artificial hormone system to assign tasks to processing elements within a processor grid. The assignment is completely decentralized and holds self-x features. Besides, we showed that we can guarantee tight upper bounds for the real-time behavior of the artificial hormone system as well as for the data load induced by the artificial hormones.

We implemented a simulator including the presented algorithms and as future work, we will evaluate the time bounds received by the theoretical examinations of the hormone system.

Furthermore, we will investigate additional quality properties of the artificial hormone system like stability of the task assignment and rules for selecting the level of hormone values to, e.g., obtain organs. Another question in this scope is how to find an optimal task assignment (if it exists) by the artificial hormone system.

We will also investigate the artificial hormone system in the scope of a practical example, the DoDORG project, which deals with a grid of processing elements to be organized by an organic middleware using the artificial hormone system.

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. 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.
4. 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.
5. 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.
6. 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.
7. 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.
8. 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.
9. 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.