A Hardware Resource Management System for Adaptive Computing on Dynamically Reconfigurable Devices

Toshiyuki Ito, Kazuya Mishou, Yuichi Okuyama, and Kenichi Kuroda
The University of Aizu, Graduate School of Computer Science and Engineering
E-mail: {d8061202, m5101211, okuyama, kuroken}@u-aizu.ac.jp

Abstract

This paper proposes a realization method of the computer system with dynamical hardware-resource allocation on dynamically reconfigurable devices. The system consists of two or more parts and they can change the number of processing units according to each processing load. In the system, there is a competition problem between these parts. In order to solve this problem, we investigate required functions of resource management units on a simple processing model. This model is an adapted load balancing model consisting of an upper management unit, two management units and processing units shared by them.

1. Introduction

Dynamic reconfigurable devices, DRP [4], DAP/DNA [5], PCA [1], and self-reconfigurable FPGA [6] have been used in the area of signal or image processing. These devices can dynamically create and delete logic circuits replying to requisition.

As these devices can construct the feature-based architecture with application of processing during operation, they have been used to improve computational efficiency [7][8]. They have advantages of the processing acceleration by parallel operation and the device area reduction by multiple usages of hardware resources. Autonomous distributed processing has been proposed as the method of data processing with their reconfigurability [2][3]. This method can reduce loads by distributed the management tasks.

Implementation of an adapted load balancing process has been proposed as a model using dynamic reconfigurable devices. A load balancing process makes whole loads equal by using hardware resources effectively. This model executes parallel processing by distributing the computation loads. Each computation terminates its execution at the same time. Based on this concept, there are two reports, a initiator - target model and shared computation models [2][3].

Each model automatically estimates each situation of computation-loads by itself, and determines the number of using processing-units. Therefore the number of using processing units can not be set by external factor like user requests. If these models adopt functions which reflect external requests to resource management, we think that this model's advantage can increase.

In this paper, we investigate about functions required for the hardware (processing units) resource management of adaptive processing. When hardware resources run short in these processing, they have a resource competition problem. The competition increases the number of reconfiguration on the processing systems. Therefore, the influence of competition decreases performance of the processing. In order to solve this problem, we try to design optimum hardware resource management system for adaptive computing. We adopt a simple model like these load-distribution models described above for the investigation.

The simple model has two computation parts. Both computations scramble for hardware resource according to each state of computation loads. Each computation tries to equalize processing time. Moreover, this model has priority, limitation of the number of processing units, defined by users. Users can change their processing speeds. First, the features of the proposed adaptive load balancing model are explained, and Section 2, 3 show its architecture. Section 4 discusses its verification. Finally, this paper's conclusion is explained.

2. Load balancing process on dynamically reconfigurable devices

2.1. Recent Works

Generally, the hardware resources required by a processor depend on the applications. In a system where two or more applications are running in parallel, the kinds and number of hardware resources required for them are determined before the runtime of applications. In the case of a conventional processing system, they can be determined only by simulation at the hardware design stage.

However, since the required hardware resources differ in each application, the required resources may run short or be wasted compared to prepared ones. One solution to this problem is to use dynamic reconfigurable devices, which supply the required hardware resources for each application with reconfiguration. In our proposed method, the hardware resources are managed by the waiting situation (state of load) for processing. In this way, the system can use hardware resources efficiently according to processing demands. We have studied one of such load distribution models [2]. However, this system can't manage multiple processes.

2.2. Load balancing process

This paper shows a model that a number of processes use the limited processor resources. This paper clarifies the concept of existing load balancing model as a method managing resources between processes. This concept terminates all processes simultaneously and aims the achievement of 100% system efficiency. This method also implies a concept of dynamic scheduling. When this concept applies, this model will need a concept of priority depending on the intended use of the system, so we apply the concept of priority load balancing process with interprocess priority.

This model needs the change of various points if this model takes into the concept of priority. When a process finished the data processing, the other process doesn't always finish the processing. Then it's preferable that working process uses the space that a process that is finished used. This research implements the termination detection, and sends an end signal after data end. If a process gets the end signal, the process minimizes the area of use after finished processing.

2.3. The composition of this model

Fig.1 shows the model of an adaptive load balancing processor. This model adopts the centrally managed master-slave model to unify management of one process. A Management unit (MU) is the master, and a processing unit (PU) is slave. PUs request a task to MU.

An upper management unit (UMU) manages two MUs. The UMU controls the hardware resource for PUs which to use the processes, and the UMU aims to equalize the computation load between processes. Using dynamically reconfigurable device, the number of PU can increase or decrease during operations.

(1) Processing Unit (PU)

Each PU receives the task from the MU, and output the task to a Re-ordering Unit (RU) after task processing. When PU receives copy/elimination command from the MU, and execute the commands: PU can copy itself to the next free space if PU receives copy command, and PU releases the using area if PU receives elimination command. During PU duplication, other PUs can operate. Therefore, this model can hide PU creation time.

A PU has a line structure, as shown Fig. 2, and 3. When a PU isn't having a task, the PU tries to get and execute the task. If a PU gets a task during processing, PU sends the task to the next the PU. If the PU finished the task processing, the PU requests a task.

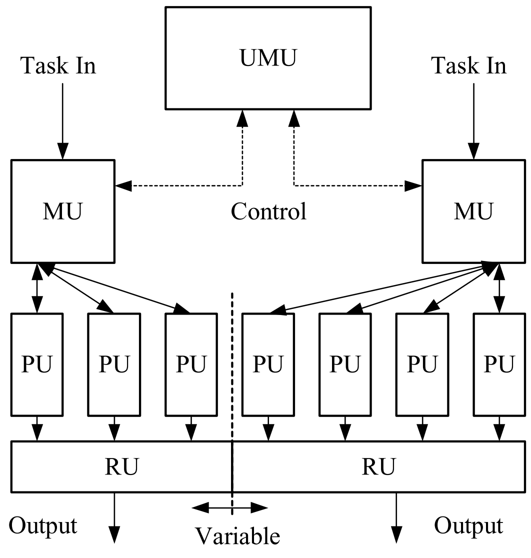


Figure 1. Proposed adapted load balancing model

(2) Management Unit (MU)

A MU realizes a load distribution. The MU receives tasks from outside, and sends the tasks to waiting PUs that for a processing. If the MU decreases the frequency of getting tasks from PUs, the MU detects a load. Then the MU issues an instruction that increases the number of PUs. The MU knows the maximum number of PUs that can create. If the MU detects the load and PUs cannot increase, the MU requests the permission for creating PUs from a UMU. In addition, if the MU has a light load, it sends a message to the UMU having to spare PUs. If the MU receives a change signal of the number of PUs from UMU, it changes the maximum number of PUs, thus it can lend the space for a PU to the other MU. The MU measures input interval of data, and calculates a task loads with a processing time and the number of PUs. The MU sends a result of the task loads to the UMU. If the MU receives an end signal, it gets rid of unwanted PUs for another unfinished process

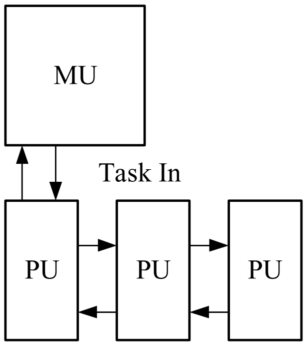


Figure 2. Master-slave load balancing model using line structure

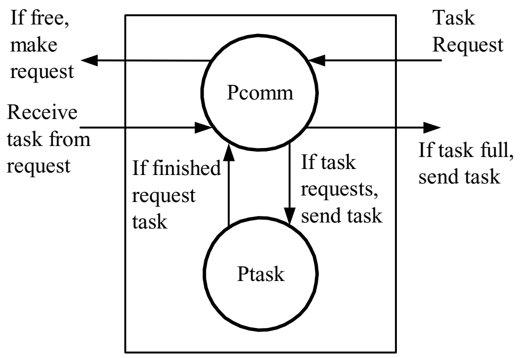


Figure 3. I/O relation with other functions of PU

(3) Upper Management Unit (UMU)
An UMU plays a coordinating role of two MUs. The UMU changes the maximum number of PUs, and sends a signal to MUs. It operates with the following three parameters.

(A) Load detection by the task estimation
(B) Load detection by processing
(C) Priority

First, the UMU allocates the number of PUs between processes by (C). If each process detects the load in processing, each process increases enough to the number of PUs by MUs. However each MU cannot exceed the limit allocated by the UMU. If the UMU receive the state of (A) and (B), it send the restructuring instruction on demand to each MU.

(4) Reordering Unit (RU)
The tasks that are outputs from PUs are out of order. An RU reassembles the tasks back into their proper sequence, and outputs the ordered tasks.

3. The structure of proposed system

This section describes about the detail of PU, MU, UMU, and RU.

3.1. The detail of processing unit

The configuration of a PU is shown in Fig.4.

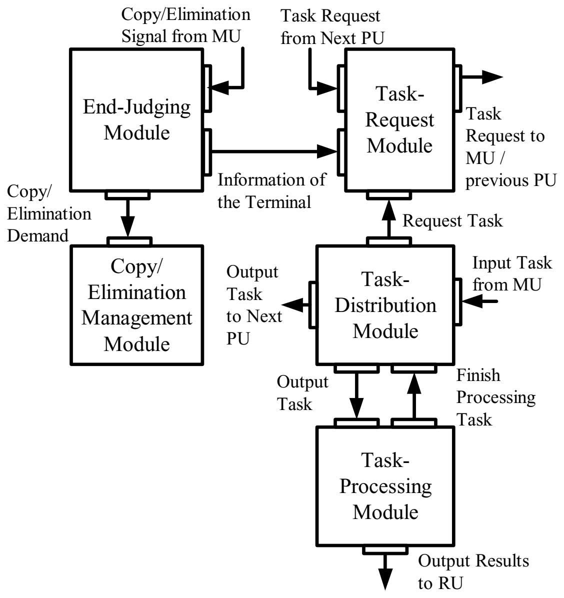


Figure 4. Block diagram of PU

(A) Task-processing module
This module processes a task inputted from an MU. This module consists of a SIMD-type processing module. The module is specific to each application. After this module processes a task, it informs a task request signal to a task-determination module in order to communicate that a PU can get another task.
(B) Task-distribution module
This module distributes the tasks that receive from MU. When this module receives the task and relevant task-processing module is free, this module sends the task to task-processing module. If task-processing module is busy, the task is sent to the next PU. When this module receives the task request signal from task-processing module, this module sends a task request signal to a task-request module.
(C) Task-request module

This module requests a task from MU. This module operates by input from task-distribution module. This module is plugged into another PU in the form of a line. This module add a request of each PU, end sends the result to MU.

(D) End-judging module
This module determines if this PU is terminal, and divides the copy/elimination signal that receives from the MU. The end-judging module is realized using a counter. When the PU is created, the counter set 0: when the counter is set in 0, this PU shows the termination. When this module receives a copy signal and the counter is increased. If the elimination signal is sent, the counter decreases. If this module receives a signal from and counter is 0, this module sends a signal to copy/elimination module. Otherwise this module sends the instruction signal to the next PU.

(E) Copy/Elimination management module
This module outputs an instruction stream for copy or elimination itself on the basis of instruction from end-judging module.

3.2. The detail of Management Unit

The configuration of an MU is shown in Figure 5.

(A) Task-judging module
When the module receives the task, the existence of the task is notified to a scheduler module, and the task is transmitted to a task-issue module. If this module receives an end signal, this module transmits the information to the scheduler module.

(B) Scheduler module
A scheduler module manages and permits the copy/elimination of PUs. The module looks at a state of PU, and transmits the copy/elimination instruction to task-issue module depending on the situation. If PU is lack the resources for it when this module detects an overload, this module requests to UMU in order to lent the resource of other process. When this module receives the end signal, this module transmits the termination of processes to the task-issue module.

(C) Task-issue module
A task-issue module manages and permits the task issue. This module receives the task request from PU, checks the unit waiting for process, and transmits the task to the PU. When this module transmits the task, this module gets an address from an address-issue module. The address adds at the head of task. When this process copies or deletes of PU, this module stops outputting the task to PUs. When this module receives the elimination or end signal, this module transmits the instructions after waiting the end of processing.

(D) Address-issue module
This module issues an address. The address uses the task ordering in RU. The address assigns a number.

(E) Task-estimation module
This module observes an input interval, calculates by the number of PUs and a time of one task processing, and transmits the result to UMU.

We perform the following calculation to compare the heaviness of load. Tnum is the number of tasks waiting for processing. PUnum is the number of PUs created in current. T is the amount of time to process a task.

$$\frac{Tnum * T}{PUnum} \tag{1}$$

This formula shows a processing time that is spent to execute the entered tasks using the number of current operating units. This value is equalizing these loads to compare.

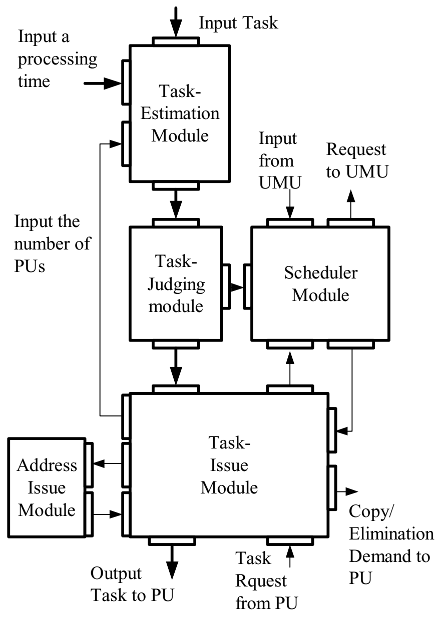


Figure 5. Block diagram of MU

3.3. The detail of upper managing management unit

The configuration of a UMU is shown in Figure 6.

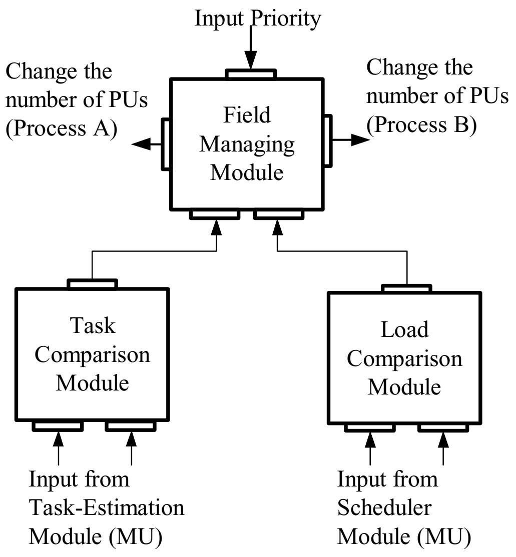
(A) Task-estimation module
This module compares two values that are calculated on the task-estimation module of MUs. The result is transmitted to a field-managing module.

(B) Load-comparison module
This module receives loads of PUs from scheduler module of the MU, and compares two values. The result is also transmitted to a field-managing module.

(C) Field-management module

This module reconstructs the field of a PU by some parameters. This module issues the command which reconstructs the number of PUs. When timing of this command-issue is blunted, the number of vibration produced by competition can be reduced. Therefore, in order to restrict the number of times of the vibration, this module has function of setting threshold value.

Moreover, this module has a function of giving a restriction of the number of PUs. Restrictions of the number of usable PU are given to two MUs. Fundamentally, these MUs must protect the given restriction. In special case, when one processing part has some PUs not to use, the another processing part can borrow these PUs. According to priorities between MUs, this restriction's strength can be changed.



Input Priority
Change the number of PUs (Process A)
Field Managing Module
Change the number of PUs (Process B)
Task Comparison Module
Load Comparison Module
Input from Task-Estimation Module (MU)
Input from Scheduler Module (MU)

Figure 6. Block diagram of UMU

3.4. Details of a reordering unit

A data put into the RU adds an address. RU reorders the data according to the address, and output the data. RU also has the function of increment the port of input when a PU is copied.

4. Verification

We clarify the property of the UMU in the proposed model. This model performs two different calculation processes. In this section, we determine that each process is called A-process and B-process and these processes are executed by two processing parts (A processing part and B processing part). Each processing part consists of a MU and some PUs, and it is managed by an UMU. We implemented a simulator, which can measure processing time when the model executes tasks. The simulator consists of an UMU and two MUs and PUs. This simulator’s time concept is “step”. 1 step is a time which spends to fetch 1 task into the model's each MU. PUs of this simulator don't adopt concrete applications, but adopt delay modules. We set the time needed for 1 task processing into the delay module, and can clarify the property of the model in various operation processing. Moreover this simulator doesn't have concept of reconfiguration's overhead, but only counts the number of reconfiguration's emergence. The environment of an experiment is shown below.

(1) Task input interval range: 2 ~ 6 steps
(2) Number of PUs: 10
(3) Total tasks: 1000(each processing parts)

In Fig.7,8, we show the experiment results. The explanation about the component of each bar graph in these figures is shown below.

I : The case of fixed processing unit. This case doesn't change the number of processing units. A and B process has each 5 PU in advance.

II : The case of adaptive load balancing model according to load situation. This case doesn't adopt some functions to suppress the vibration produced by competition.

III : The case of adaptive load balancing model adopting a function to suppress the vibration produced by competition. This function is threshold value, and blunts judgment of load.

IV : The case of adaptive load balancing model adopting two functions to suppress the vibration produced by competition. This case is our proposed model. These functions adopt a threshold value and a restriction of the number of PUs. The latter is the function to restrict PU number to be used. At first, restrictions of the number of usable PU are given to A and B processing parts. Fundamentally, these processing parts must protect the given restriction. In special case, when one processing part has some PUs not to use, the another processing part can borrow these PUs.

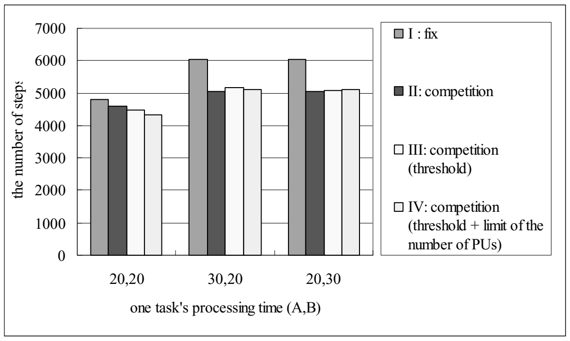


Figure 7. 1000 tasks processing steps

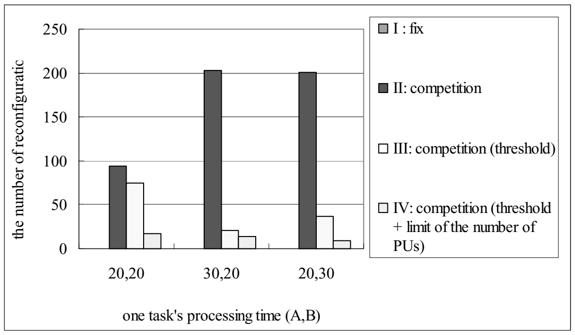


Figure 8. Reconfiguration times when 1000 tasks are processed

In Fig. 7, the result of load balancing model (II) is better than the result of fixed processing unit case (I). But Fig.8 shows that the case of (II) has many times of reconfiguration. If we will implement a proposed model on a reconfigurable device having the high overhead of reconfiguration, the merit of using the adaptive load distribution model could not be found. Concretely, when the reconfiguration overhead of (II) is larger than the task processing advantage steps (the difference of task processing steps of (I) and that of (II)), the cost of realizing adaptive model is wasted.

Therefore we try to reduce steps of task processing and to reduce times of reconfiguration. To fill these reduction demands, we proposed method of (III) and (IV) in this proposed model. When we compare data of (II) with data of (IV) in the case of task processing step (A, B =20, 30), we confirm that the 1000 tasks processing step of (II) is ended 60 steps earlier than that of (IV). However we confirm that the reconfigurable times of (IV) is 192 times (201times – 9times) better than that of (II).

Next, we clarify the property of the proposed model when we change the input interval of a task. The environment of an experiment is shown below.

(1) Fixed task processing time(step): A = 20steps, B = 30steps
(2) Task input interval range: 2 ~ 6 steps, every 50 ~ 500 tasks
(3) Number of PUs: 10
(4) Total tasks: 1000

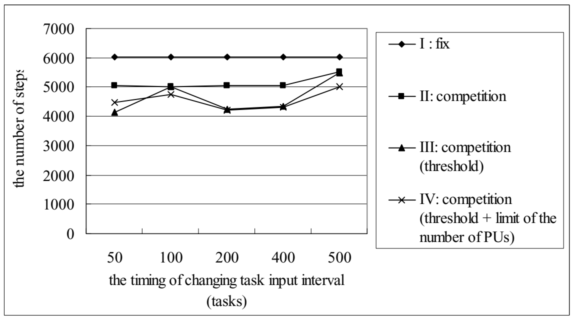


Figure 9. 1000 tasks processing steps

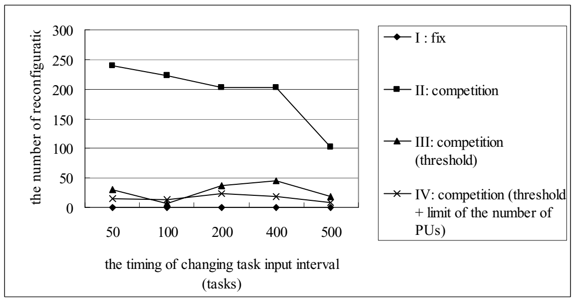


Figure 10. Reconfiguration times when 1000 tasks are processed

In Fig. 9 and 10, the task processing step of (III) and (IV) are fewer than (II), and the number of reconfiguration of (III) and (IV) are better than (II). In this example, after the UMU changes the number of PUs according to the result of load calculation, the UMU does not make it change during a certain period. Based on this result, we confirm that operation efficiency is improved. In other words, this example shows that to use the calculation result of global processing load is better than to use the calculation result of local processing load. However, when a processing situation changes dynamically, it is difficult to judge a load correctly. This example shows one possibility that these methods ((III) and (IV)) can perform effectively under the situation which cannot be predicted.

Based on results of these experiments, we confirmed a possibility that the proposed resource management system, which can change circuit arrangement according to various processing situations dynamically, works effectively.

5. Summary

This paper proposes and verifies the load balancing architecture on dynamically reconfigurable devices. The proposed resource management system can change circuit arrangement according to various processing situations dynamically. Based on the system, we implemented a simulator which can measure time of task-processing and number of competition. We clarify the property of this system using the simulator, and we confirm that the system works effectively. Based on this verification, when we design the adaptive system like this proposal, we confirm that the hardware-resource management functions should optimize the improvement of processing-efficiency and the reduction of number of times of reconstruction.

References

[1] H. Ito, R. Konishi, H. Nakada, K. Oguri, M, Inamori, and A. Nagoya, "Dynamically Reconfigurable Logic LSI-PCA-1, The First Realization of the Plastic Cell Architecture," IEICE TRANS. INF. & SYST., Vol. E86-D, No. 5, pp. 859-867, MAY 2003.

[2] T. Ito, O. Yuichi, J. Kitamichi, and K. Kuroda, "A Master-Slave Adaptive Load-Distribution Processor Model on PCA," The 12th Reconfigurable Architectures Workshop (RAW 2005), April 2005.

[3] K. Tada, S. Kouyama, T. Yuasa, T. Izumi, T. Onoye, and Y. Nakamura, "Adaptive load distribution model on self reconfigurable logic device," The 16th Workshop on Circuits and Systems in Karuizawa, pp. 171-176, April 2003. (in japanese)

[4] T. Sunagawa, K. Ide, and T. Sato, "Dynamically reconfigurable processor implemented with ipflex's dapdna technology," IEICE Transactions on Information and Systems, Vol. E97-D, no. 8, pp. 1997-2003, 2004.

[5] H. Nakano, T. Shindo, T. Kazami, and M. Motomura, "Development of dynamically reconfigurable processor lsi," NEC Technical Journal, Vol. 56, no. 4, pp. 99-102, 2003.(in japanese)

[6] B. Blodget, P. J. Roxby, and E. Keller, "A self-reconfiguring platform,“ FPL 2003, pp. 565-574, 2003.

[7] M. J. Wirthlin, and B. L. Hutchings, "A Dynamic Instruction Set Computer", Field Programmable Gate Arrays (FPGAs) for Fast Board Development and Reconfigurable Computing, John Schewel, Editor," Proc. SPIE 2607, pp. 92-103, 1995.

[8] S. C. Goldstain, H. Schmit, M. Moe, M. Budiu, S. Cadambi, R. R. Taylor, and R. Laufer,"PipeRench: A Coprocessor for Streaming Multimedia Acceleration," Proc. ISCA'99, pp.28-39, May 2-4, 1999.