

# Post-Game Analysis in a Mixed HW/SW Environment

Jasper VAN LEEUWEN and Herman ROEBBERS

Philips TASS

P.O. Box 80060, 5600 KA Eindhoven, The Netherlands

Phone: +31 (0)40 2755 288 Fax: +31 (0)40 2755419

E-mail: { [Jasper.van.Leeuwen](mailto:Jasper.van.Leeuwen@philips.com) | [Herman.Roebbers](mailto:Herman.Roebbers@philips.com) }@philips.com

*Abstract*—This paper presents an approach to allocate tasks in a mixed hardware-software environment in a (near-) optimal way. Optimal is defined as having a minimal execution time given the size restriction of the programmable hardware. Differentiating factor of the research is the hardware environment, a combination of a general-purpose processor and a Field Programmable Gate Array. An iterative approach is explored based on Post-Game Analysis (see [Yan]). The partitioning of tasks is improved iteratively between execution runs. During execution, measurements are performed. Heuristics are defined to suggest the reallocation of a task. The heuristics are defined on three orthogonal axes, these being reduction of resource contention, exploitation of processing entities and communication reduction. As the location of the neighbors of each task is unknown at the time of coding that task a communication harness is provided. Currently the communication harness and all measurements on the processor side are completed. The measurements on the hardware side are a work in progress.

*Keywords*— Hardware-software co-design, FPGA, Post-Game Analysis, task allocation algorithm

## I. INTRODUCTION

In the race to optimally balance performance and costs in (embedded) computer systems the combination of a general-purpose processor and reconfigurable hardware is an attractive approach as it combines the flexibility of a general-purpose processor with the performance of (reconfigurable) hardware. It is left to the system architect to distribute the tasks over the processing entities while making sure that the non-functional requirements are satisfied and an optimal balance between performance and costs is reached. The distribution of tasks over the different processing entities is however not a trivial task. On the contrary, when the number of tasks increases it becomes infeasible to

manually create a good partitioning of tasks. In this paper, an approach for allocating tasks in a mixed hardware-software environment is described. The goal of the research is to serve as exploration of the issues that arise when deploying tasks on different processing entities. In order to explore those challenges a research project has been started investigating how to allocate tasks in a mixed hardware-software environment in a (near-) optimal way.

In computer science, much research is done on how to (optimally) distribute a set of (communicating) tasks over the processors of a multi-processor machine. Post-Game Analysis (PGA) is an iterative allocation strategy. In this paper, an approach for allocating tasks in a mixed hardware-software environment based on Post-Game Analysis is described.

## II. POST-GAME ANALYSIS

The main goal of Post-Game Analysis is reduction of execution time. Post-Game Analysis (see [Yan] and [Sunter]) is an iterative allocation algorithm. Measurements obtained during execution of the set of tasks are used to generate the next allocation off-line. This has two advantages. Firstly, the algorithm uses measurements to optimize an allocation, so no a-priori information is necessary. Secondly, the actual generation of the next allocation is performed off-line, so the runtime overhead can be limited.

In Post-Game Analysis, the computation is modeled as objects (called “players”). Players exchange data and synchronize only via communication over channels. This model closely resembles CSP (see [Hoare]).

In Post-Game Analysis, the measurements are used (off-line) to calculate so-called indicators. Indicators describe aspects related to the performance. The indicators are grouped into key performance indicators, namely processor contention, load balancing, and

communication reduction. Heuristics are defined (using the indicators) and lead to proposals for changing the allocation of players. When the value of an indicator differs (much) from the average value, the heuristics generate a proposal. There are two types of proposals. The immigration proposals (associated with a processor) suggest that tasks should be moved to that processor. Emigration proposals (associated with a task) indicate that the task should be moved from the current processor. The emigration proposals are counted per task and the immigration proposals are counted per processor. The task with the highest number of (emigration) proposals is moved to the processor with the highest number of (immigration) proposals. This reallocation is repeated a number of times.

The heuristics also provide confidence values for the generated proposals. For each task, the related confidence values are summed. When tasks have the same number of proposals, the sum of the confidence values is used to select the task to move. An example of a confidence value is:

$$Confidence = \frac{indicator\ value - indicator\ average}{indicator\ average}$$

Five veto heuristics are defined to prevent the system from making undesirable moves:

1. Don't "move" if the task is already allocated to that processor.
2. Don't move the task when the capacity of the target processor is low compared to the runtime of that task.
3. If a task has been moved around often during previous iterations, don't move it.
4. If indicators indicate that the task should be moved to different processors, don't move the task (yet).
5. After a certain (maximum) number of moves, don't move anymore.

In [Sunter], Post-Game Analysis is applied in a Transputer environment and the algorithm has proved to be successful.

### III. PGA IN A MIXED HW-SW ENVIRONMENT

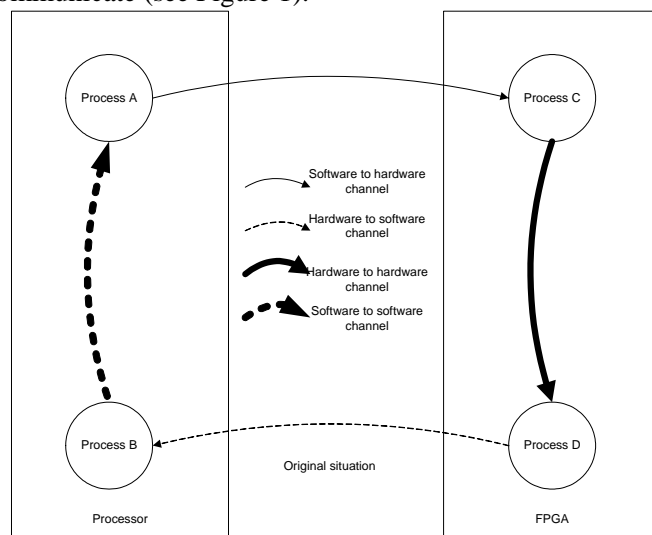
Recent advances in technology have made it possible to program Field Programmable Gate Arrays (FPGAs) with a high level language (similar to the C language). This opens the opportunity to implement algorithms in a mixture of hardware and software tasks. Of course, this leads to the question of where to allocate which task.

#### A. Communication harness

In Post-Game Analysis, the computation is modeled as

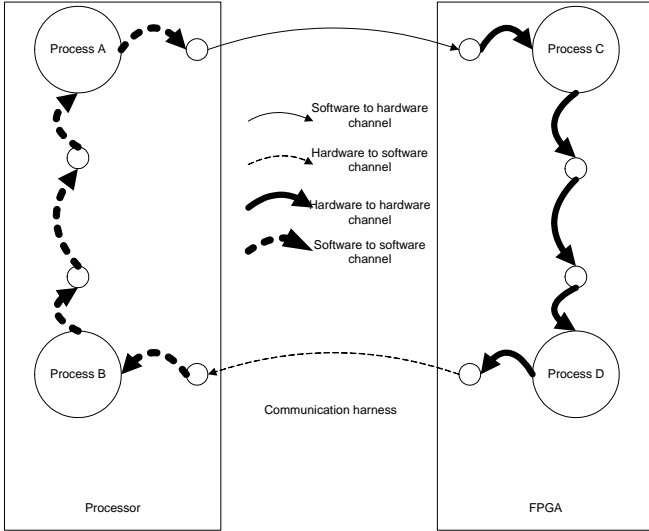
players that exchange data only via channels. As players can be re-allocated it should be possible to execute each player on all available processing entities. At the moment two implementations for each player should exist; one implementation for the processor and one implementation for the FPGA. The necessity to have multiple implementations for each player is costly. In our project we used Handel-C (see [Celoxica]). Handel-C is a C-like language, which supports most of the ANSI-C constructs. Handel-C is fully synthesizable, allowing direct implementation of C algorithms onto FPGA. The use of Handel-C facilitates fast implementation of both implementations of each player. Eventually, the arrival of a language is expected which can target both the processor and the FPGA.

When making a fixed partitioning of tasks over processing entities, the available communication channels are used to communicate with the neighbors. The API of the communication channels is used to communicate (see Figure 1).



**Figure 1 - Fixed partition**

As players can be reallocated, the location of the neighbors is not known when coding the tasks. For this reason, a communication harness is provided. With the communication harness, a task can communicate with its neighbors independently of where these neighbors are located (see Figure 2).



**Figure 2 - Communication harness**

### B. Heuristics

The heuristics and philosophy of PGA are aimed at multi-processor machines. We, however, want to get an optimal execution time in a mixed HW/SW environment. This includes quite drastically different processing entities, such as processors and programmable logic (in our case a Field Programmable Gate Array). On a processor, only one task can run at any time. On a programmable logic device, each task is implemented (directly) in hardware and the tasks can run (potentially) truly in parallel.

Although the direct application of PGA is not possible, we propose more general indicators to improve an allocation in a mixed HW/SW environment.

We observe that tasks may need to use resources of a processing entity in order to make progress. Examples of resources are dedicated hardware multipliers on an FPGA, various IP cores for the FPGA and the processor on the software side. In our definition, a task makes only use of the locally available resources. A resource can be scarcely available on a processing entity or in abundance. When a resource is available in abundance, no task (on that processing entity) will ever have to wait to obtain an instance of that resource. When a resource is scarce, a task might have to wait because all instances are occupied. Two indicators model this observation. “*Resource wait (p)*” defines how long task  $p$  waits for resources (on the current processing entity). “*Resource competition (p,q)*” shows how long the pair of tasks  $p$  and  $q$  is competing for resources. When applying PGA in a mixed HW/SW environment, all scarcely available resources should be considered and not only the

processor.

As the processing entities are not homogenous, we introduce indicators that characterize the different aspects of the processing entities. In our case, we have an FPGA and a processor. Characteristic for the FPGA is that it poses a size restriction on the tasks that can be executed on it (as an FPGA has a limited number of gates). Of course, we would like to use the FPGA as optimal as possible. As indicator, we define “*GainPerGate(p)*”. This is the gain in execution time (gained by executing the task on the FPGA instead of on the processor) per used gate (for that task). Another property of an FPGA is that tasks can (potentially) execute concurrently. We introduce two indicators for this characteristic. “*Total Concurrency (p)*” indicates how concurrent the task  $p$  can execute with respect to all other tasks on the processing entity. If a task  $p$  can execute “very concurrently”, then it is a good candidate for execution on the FPGA. The other indicator is “*symmetric concurrency (p,q)*”. This indicator shows how concurrently the pair of tasks  $p$  and  $q$  executes.

Besides executing (and using resources while executing), tasks can communicate. We define the indicator “*prox (p,q)*”, the ratio of number of communications, and the time spent calculating. In addition, we have one indicator called “*affinity (p,t)*”. This indicator is defined as the ratio of number of communications of task  $p$  with tasks on processing entity  $t$ , and the time task  $p$  spent calculating.

Again, the heuristics generate a proposal when the value of an indicator differs (much) from the average value. Of course, the veto heuristics now prevent moving a task to programmable logic device when there is not enough space.

At the moment, no heuristics for choosing a particular processing entity are defined (as we have only two processing entities). When more processing entities are incorporated, the target processing entity is selected based on the same principles (resource usage, communication behavior and the characteristics of the different processing entities).

### C. Measurements

The indicators are derived from a basic set of measurements. The measurements should be implemented such that no unacceptable overhead is introduced in the system. The measurements are introduced in Table 1.

Measurement <sup>1</sup>	Description
Tap(t, p)	Time that task $p$ spends actively processing when placed on processing entity $t$ .
Conc(p,q)	Fraction of the active processing time of task $p$ where task $q$ is also actively processing or is waiting in the ready queue.
Reswait(p,r)	Time that task $p$ spends waiting to obtain resource $r$
Twait(p,q)	Time that task $p$ waits for task $q$ , because task $q$ has a resource that task $p$ wants to use.
FpgaSize(p)	Area (size) that task $p$ would take when placed on an FPGA.
C#(p,q)	Number of messages that task $p$ sends to task $q$

**Table 1 – measurements**

The indicators are defined using the measurements in the following way:

#### Resource contention

- Resource wait(p): total amount of time spent by task  $p$  waiting for resources (on the current processing entity).  
(SUM  $r$ :  $r$  is a resource on site(p): Reswait(p,r))<sup>2</sup>
- Resource comp(p,q): competition between tasks  $p$  and  $q$  (for using resources on the current processing entity).  
((Twait(p,q) + Twait(q,p))/2)

#### Exploitation of processing entities

- Gain per Gate (p): execution time gained by placing task  $p$  on the FPGA divided by the size (in gates) of task  $p$ .  
(Tap(cpu,p)-Tap(FPGA,p))/fpgaSize(p)
- Symmetric concurrency (p,q): The amount of concurrency between tasks  $p$  and  $q$ .  
(Conc(p,q)+Conc(q,p))/2
- Total concurrency(p): Measure of the concurrency of task  $p$  with all other tasks (on the same processing entity).  
(SUM  $q$ :  $q$  on site(p) and  $q \neq p$ : conc(q,p)\*Tap(q))

#### Communication reduction

- Proximity(p,q): proximity of tasks  $p$  and  $q$ , ratio of the number of messages sent to each other and the time spent calculating.  
(C#(p,q)+ C#(q,p))/(Tap(p)+ Tap(q))

<sup>1</sup> “p” and “q” denote tasks, “r” denotes a resource and “t” indicates a processing entity (either CPU or FPGA).

- Affinity(p,t): ratio of number of communications task  $p$  has with processor  $t$  to the time task  $p$  spent calculating  
(SUM  $q$ :  $q$  on  $t$ : (C#(p,q)+ C#(q,p))/Tap(site(p),p))

## IV. IMPLEMENTATION

The communication harness and measurements need to be implemented on both the processor side and the FPGA side. The processor has as operating system Microsoft Windows 2000 Professional. Microsoft Windows 2000 Professional is a pre-emptive operating system, so the available processor time is divided among all the tasks that need it. There are 32 different scheduling priorities. The task with the highest scheduling priority is allowed to run and tasks with the same priority are scheduled in a round-robin manner. In order to measure  $Tap(t,p)$ ,  $Conc(p,q)$ ,  $Reswait(p,r)$ , and  $Twait(p,q)$  with sufficient accuracy on the processor, the tasks in the task queue are sampled. Sampling is not done at fixed time intervals, but through a function that is executed when a context swap is performed (when the scheduler changes the running task). To implement this functionality, a device driver is designed. Upon installation, the device driver will insert a custom function that will be executed when a context swap is executed. The custom function will register the execution time, executing task and the tasks that are ready for execution. This will provide the necessary information. In order to measure the communication behavior, the communication harness is extended with the necessary measurement.

The measurements should also be implemented on the FPGA. Besides the communication harness, an API is provided in order to make resource reservations. Both API’s are extended with measurement flags. Measurement tasks use these flags to perform the required measurements. Finally, the fpga size of each task can be obtained from the compiler tools.

## V. CURRENT STATE

At the moment of writing, the communication harness is implemented on both the hardware and the software side. In addition, the measurements are fully implemented on the software side. Furthermore, half of the hardware measurements has been implemented.

In order to test drive the system, a simple test application consisting of three tasks is implemented. With this application, different processing and communication behavior can be simulated. The results

<sup>2</sup> “Site(p)” denotes the processing entity executing task  $p$

are not yet available as not all the hardware measurements are implemented.

## VI. FUTURE WORK

In the near future, the hardware measurements will be finished. When all measurements are implemented, the indicators can be calculated and the heuristics can be used to suggest re-allocation of tasks.

Next, the quality of the obtained partition should be checked by optimizing an industry-size application. To show the quality of the algorithm, the improvement of the execution times with respect to the number of performed iterations should be shown. In addition, the optimum found should be compared to the “real” optimum. And finally, it should be affirmed that the measurements do not impact the application-under-investigation too much.

It is expected that the algorithm can be tuned for certain types of applications (for instance with weighing factors for the indicators). First the different classes of applications need to be determined, after which the applicability and effectiveness of the algorithm can be researched.

## REFERENCES

- [Celoxica] Celoxica, *Handel-C*,  
<http://www.celoxica.com/methodology/handlc.asp>
- [Hoare] C.A.R Hoare, *Communicating Sequential Processes*,  
*Communications of the ACM*, v21 n.8, p.666-667, Aug  
1978, Association for Computing Machinery.
- [Sunter] J. Sunter, *Allocation, Scheduling & Interfacing in  
Real-time Parallel Control Systems*, ISBN 90-  
9007161-X, 1994
- [Yan] J. Yan, S. Lundstrom, *The Post-Game Analysis  
Framework – Developing Resource Management  
Strategies for Concurrent Systems*, IEEE Log Number  
8930632, *IEEE Transactions on Knowledge and Data  
Engineering*, 1(3) pp. 293-309, 1989