On Hybrid Abstraction-level Models in Architecture Simulation

A.W. van Halderen A. Belloum A.D. Pimentel L.O. Hertzberger
Computer Architecture & Parallel Systems group, University of Amsterdam

Abstract— **In the life cycle of a hardware design, it is often needed to start first with a gross evaluation, typically performed using abstract simulation models, then refine the design in a step-wise approach towards more detailed and accurate evaluation. This progressive refinement process of the design may bring the system evaluation in a state where parts of the design are expressed in a high level of abstraction, while others are more detailed. This results in hybrid, or mixed-level, architecture simulation models.**

To support hybrid models, designers often apply intermediate components that translate transactions to a unique abstraction level. The process of translating from one abstraction level to another can have some side effects on the accuracy of the model being studied. The study presented in this paper, which has been conducted within the scope of the Artemis project, shows how dramatic the impact of the hybrid modeling can be on the accuracy of the system.

The aim of this study is to point-out the impact of hybrid modeling when applied to very simple and common architecture models. The results show a number of interesting phenomena not expected beforehand. A detailed discussion of the simulation is presented as well as the first steps to solve the stated problems.

Keywords— **hybrid simulations, design space exploration, computer architecture simulation**

I. Introduction

The rapid prototyping of an application is becoming more and more important for a successful design. Tools and methods that allow performance evaluations at an early stage are the key element in the design process. In the top-down design approach the initial versions of the design are very abstract, they mainly describe the functionality of the system. At this stage, the description does not contain any details on the real implementation. Abstraction is a concept used to hide a number of details which allows the system designer to focus only on specific issues. In the RASSP taxonomy [1] abstraction level is defined as: “an indication of the degree of detail specified about how the function is to be implemented”. Most of the simulation environments allow the architect to create and explore a design at different levels of abstraction [2], [3]. In the literature the high abstraction levels are mostly composed of behavioral models which give only a description of the functionality of the systems. A more accurate definition of the high level abstraction levels, is the one proposed in the work of Williams [4] where two distinct high levels of abstraction are identified:

Performance Model: Performance models focus mostly on the flow of the information they do not consider the form nor the value. This modeling level is referred to as *uninterpreted* since the components cannot interpret the information.

Behavioral Models: Behavioral models contain more detail than the performance model. As described by Williams et al. in [4] “behavioral components contain functionality responsible for mapping values at their input to values at their outputs and typically contains more detailed timing and event granularity.”

Having parts of the architecture model described at different levels of abstractions requires, in general, components that translate the information between the different abstraction levels. These hybrid components, called also “hybrid interface” receive information defined at a certain level of abstraction and generate the matching representation of another abstraction level. The data generated by the hybrid components can be just a refinement of the data they receive or completely a different type of data. For example, the hybrid components used in the ADEPT prototyping tool convert the tokens used at the high abstraction level into the type data accepted by the low level of abstraction [5]. In all cases, the hybrid components have to deal with two main challenges: the timing and data abstraction. The more the low abstraction level adds functional details the more the timing and the data abstraction can play an important role in the accuracy of the simulation model. A hybrid simulation model can be executable but not valid; in this study, we distinguish between two types of models, namely: executable simulation models and valid simulation models. An executable model allows the simulation to complete successfully, however, it may not be valid because the performance study performed using this model can be heavily affected by the translation within the hybrid components. Thus, a simulation model is considered to be valid if and only if it is executable and independent from the abstraction interfacing process.

After describing how hybrid simulations will be used, this paper will introduce a very small application and introduce the abstraction levels for it. This application is used to see how a hybrid simulation behaves on a basic architectural construct. The results will point out a combination of two problems, one of which is looked in further in a second experiment.

II. Hybrid abstraction models

When in the design process the designer descends from a high level of abstraction in his design to a lower level, it is a benefit if this can be done gradually. This may imply that one part of the architecture is refined, while leaving other parts more abstract. In this way the designer can concentrate on the features of a specific part of the architecture without being bothered by other parts and receiving the benefit of a potentially faster simulation because the model is partially still more abstract. Especially in exploring architectural paths this comes as a real feature, pruning the behaviour of individual paths in the architecture. In such simulations two or more abstraction levels are present within a single model, which inevitably connect and interact with each other.

When model components are developed, they are not developed completely independently. There is a certain strategy involved, such that the components agree upon a common protocol to interact. Included in this are not only the events sent from one component to another, but also a common understanding on how resources are claimed and latencies are accounted for.

When model components from different abstraction levels are mixed, the coherence in the model which would make a simulation consistent and valid may not be there. The problems that occur in making hybrid models of different abstraction levels within a single simulation setup can be illustrated and investigated using some simple architectural constructs, which commonly occur. In this study we will use two cases where two different abstraction levels are combined within a single model.

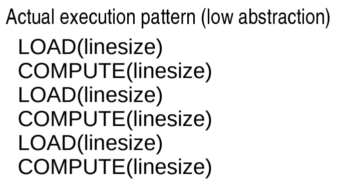
III. Application and its abstraction levels

In the first example, data is passed from one producer component to another component —the consumer— by an intermediate component, which is a FIFO buffer memory. This is an example of a data-flow oriented network, where the modeled components receive, process and forward data. No complex interaction takes place in these systems, and the abstraction level in such systems can be defined simply by the size of the data being handled. A typical application which is represented by this model is a filter operation on images.

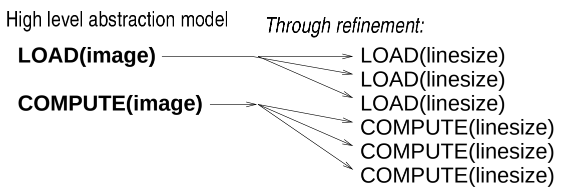
On a relatively high abstraction level we define operations on the whole image: read image from source, filter image, forward image to next component. In reality, such an application would never be developed this way. Instead of using coarse grained operations, it would fetch only part of the image, process it and pass it out and only then fetch the next part of the image. The number of parts into which the image would be split depends on the application and/or the architecture. For now we will call the size of a part of the image the *line-size*, which leaves undecided whether this is application or architecture dependent.

As a frame of reference we assume that the application using the finer grained operations, also shown in figure 1(a), embodies the real-world application. Any deviance in behavior which the higher abstraction or hybrid models show are therefore the error made by these models.

In a design process, one would first model the application on the coarse grained level, yielding a high abstraction level and would end up after one or more steps with the more detailed model. Now our application is relatively simple, and we could easily go from the high to the low abstraction level in one step. However the process of detailing will become more important in this paper to warrant pursuing this issue. A common method for lowering the abstraction level is using stepwise refinement. Using this method each component or operation on a high level of abstraction is individually decomposed into smaller units. The operation to retrieve an image from a source will thus be decomposed into operations to retrieve line-sized parts of the image.



(a) Expected execution pattern.



(b) Refinement of High-level abstraction.

Fig. 1. Discrepancy between model derived using refinement and actual execution pattern.

In figure 1(b) this process is depicted. The high-level load and compute operations are decomposed into three sequences of operations each. This pattern does however

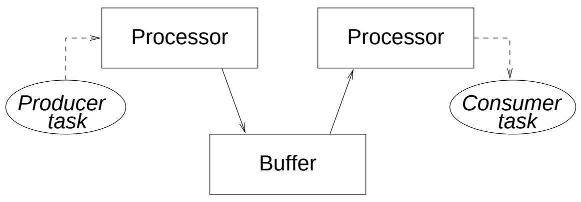


Fig. 2. Producer-consumer architecture.

not conform to the actual low-level sequence being modeled (see figure 1(a)). Through stepwise refinement we cannot capture the desired scheduling. Still it is a logical and often used method in modeling and hybrid simulation.

Because the model obtained through refinement is expressed in operations that are of a finer grain size that the abstract model, one could argue that the refined model is more detailed than the high-level model. In a classification this means that both the refined model and the low-level abstraction model are hierarchical below the high-level model. However the refined and low-level models both use the same grain size of operations, just the schedule is different. This makes it questionable that the refined model represents the low-level model at a higher abstraction level.

IV. Producer-consumer experiment

A fundamental construction (figure 2) in many architectures is where one processing component inserts data into a buffer and another component retrieves the data. An abstract model of a producer-consumer model may represent the following actual low-level implementation:

```
Producer task:                 Consumer Task:   fetch part
                                                compute
          compute                               fetch part
          store part                            compute
          compute                               fetch part
          store part                            compute
          compute part                          fetch part
          store part                            compute
            next image                            next image
```

In this case we see the producing task subdivides the image in three parts and the consuming task processes the same image in four parts, the line-size ratio is therefore 3:4. Since high-level operations operate on whole images this distinction is not present in a high-level model. When such a high-level model is lowered in abstraction these details are uncovered, opening a range of possible implementations.

We map this application to the simple architecture of figure 2 where we can vary the delays for computation and storing or retrieving data from the buffer. For easy comparison the total communication and computation for a single processing component is kept constant, and we can express our results for different communication to computation ratios. For simplicity we also assume the delay for computation at the producer to be the same as for the consumer, and likewise for latencies of storing and retrieving data from the buffer.

The performance of this example system can be evaluated using the metric *throughput.* Over a period of time, the number of whole images that have been processed by the consumer is counted.

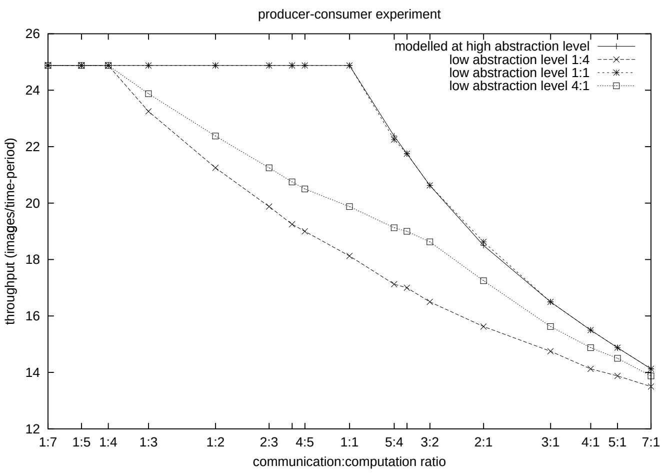


Fig. 3. Throughput of the producer-consumer system as measured in images that have been processed entirely at the consumer side over a fixed interval. The top two lines (which almost coincide) are the results of the high-level model and of a low-level model where the producer and consumer transfer equally sized blocks. The other two lines are results where either the consumer of producer uses a line size-ratio which is four times larger than the other (and thus uses four times less operations to process an entire image). This is the consumer:producer ratio. On the x-axis an increasing communication to computation ratio is set, however for comparison purposes the total of the communication plus computation latencies is kept constant. Ideally the result should be a horizontal line, and the drop in performance is caused by the contention for the shared resource.

Figure 3 shows the throughput as a function of the communication to computation ratio for both the high-level abstraction model and the detailed model. The detailed model shows that for three different combinations of line-sizes we obtain different results. This should be no surprise. Higher abstraction levels do not consider certain factors and therefore we only get an approximation of the intended result.

From the results we can furthermore see that as the communication to computation ratio shifts to an increasing communication importance, performance will logically drop, because both tasks access a shared resource which becomes the limiting factor. However the influence of the line-size ratio in a detailed model is not immediately apparent. Figure 4 provides clarity in this respect. It shows that for certain “right” combinations of reader/writer line-size ratios the system is more effective, especially when communication is about as important as computation in the architecture. For these combinations, the communication with the buffer of both processors can be interleaved, while for others both processors try to access the buffer at the same time. The contention creates stalls and the communication of one processor cannot be efficiently be overlapped by the computation of the other.

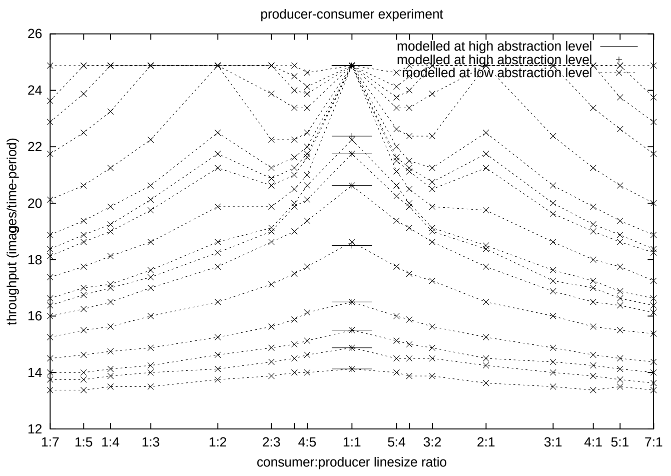


Fig. 4. Throughput plotted against consumer to producer line-size ratio. Otherwise the same experiment as in figure 3. Results from the high-level model which has no consumer to producer line-size ratio is represented by the small lines in the center of the plot. The other lines show the throughput for higher communication to computation ratios as the throughput gets overall smaller.

The abstract model and the more detailed are very much alike. In principle, the only actual difference in the simulation implementation is the size of the data being transferred. One operates on whole images while the other on parts. It is easy to merge both models, where the buffer is able to service requests for whole images or parts alike. This model is executable, however the question should rise whether it is valid. For this reason, we apply a rough classification of architecture simulation models:

Functional model A simulation is functional when the operations and operations modeled are actually performed. When a multiplication is modeled, the result is actually computed, and an operation to retrieve a memory address actually returns the value previously stored.

Operational model Operational correctness is required for any simulation and involves that the operations are performed in the right circumstances and with the correct protocol. When a processor stores information in a memory, it needs to contact the bus component and instruct it to pass the information to the memory component. Retrieving information from a buffer needs to take into account the possibility of an empty buffer.

Behavioral model Correct behavior involves introducing the right latencies to operations and the allocation of resources at the proper sequence.

A correct operational level typically involves model verification, while correct behavior typically involves model validation.

We regard performance models to be either functional or non-functional. The functional level is not of very much interest for our study since it is not a factor in this simulation outcome. The simulation runs identical to whether it is functionally implemented or not. Operationally the simulation of hybrid models has been made correct too. A buffer can be designed in such a way that both images can be served as well as parts of them. By keeping track of the amount of data in the buffer in bytes, the buffer can operate independently of whether incoming requests are in images or line-sized parts. A request for a whole image can be queued when only a part of an image is present in the buffer.

Valid behavior of a model is however a more difficult concept. We have composed a system for which it is more difficult to define which behaviour would be correct. The behaviour of the system is determined by the interaction of both the producer and consumer task. Now these have been modeled at different abstraction levels and thus with different behaviour. For now we make the intuitive statement that the behaviour of the mixed-level system should be somewhere between the high level abstraction and the low level abstraction.

A buffer is intrinsically suited to make the translation of one abstraction level to another. This is because the requests modeled at different abstraction levels are not directly related to each other. Effectively the requests on the abstraction levels are uncoupled and the connection is made in an internal data transfer in the buffer. But, as we will demonstrate, there is still a different behaviour in the hybrid model evaluation compared to both high and low-level abstraction models.

In figure 5 the results of some hybrid simulations are shown for different communication to computation ratios. In the simulation one processor processes data at the grain size of images, while the other processes data on the line-size grain size. We can clearly see that the hybrid simulation result can differ quite from the high- or low-level model simulation results.

We can study this difference using a small experiment. In this experiment we take the architecture and:

- make the buffer large enough to fit an entire image plus largest line-size;

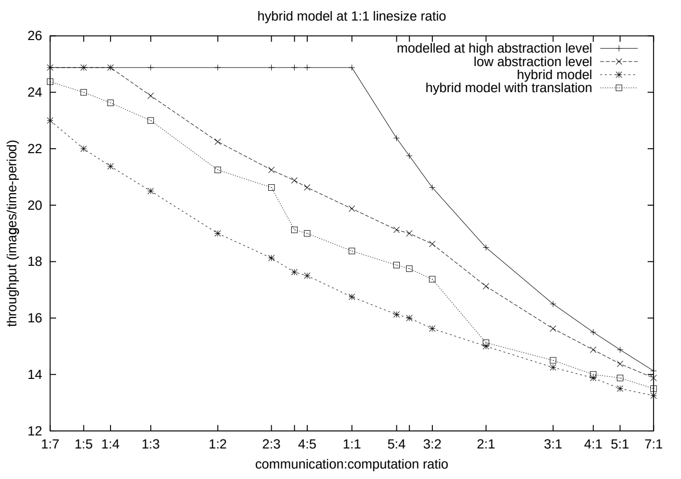


Fig. 5. Evaluation of a hybrid model of the producer-consumer system. As a reference the low-level model of the 1:1 line-size ratio (as in figure 3) is also shown. The naive hybrid model uses a buffer which is accessed by the producer using high-level (image) transfers, while the consumer uses low-level operations, fetching line sized parts. Many approaches to hybrid abstraction-level simulations use a translation from high-level operations using the refinement process (see figure 1(b)), for which the results as shown are much better than the naive hybrid simulation. However the inability of the system to predict the performance knee is a substantial deficiency.

A large buffer ensures that the consumer never has to wait for the producer to supply enough data for it to continue.
• modify the buffer such that access to it is no longer exclusively, i.e. make the buffer dual ported.
Therefore the consumer never has to wait for the producer because of the access to the resource is serialized.

Under these circumstances all differences disappear. This points to two issues why hybrid models are working operationally perfectly fine, but cannot be considered as valid without concern. Interaction takes place between the different abstraction levels at two ways:

1. The synchronization between the tasks which are modeled at a different abstraction level. When the buffer is empty, the consumer task blocks until the producing task, operating at a higher abstraction level, has progressed.
2. When two tasks access the same resource (the buffer in this case), even through there is no synchronization between the tasks, the shared resource schedules the requests. Scheduling is a different form of synchronization.

In each of these cases one of the abstraction levels has a coarser notion of time in the simulation than the other. Synchronization involves sequencing in time, and therefore the simulation is invalid because different notions of time frames are confused.

V. Resources that are accessed on multiple abstraction levels

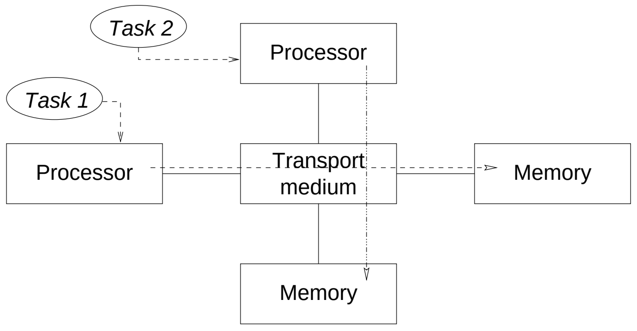


Fig. 6. Shared resource architecture.

In a different architectural construct we can highlight the effect of multiple abstraction levels which access a shared resource at their different levels of abstraction. Two tasks which do not communicate with each other, have their individual memory to access. The processors executing the tasks do need to access their memories through a shared transport medium (see configuration in figure 6).

Figure 7 shows the results of simulating this architecture executing tasks like the producer task in the previous experiment. The fact that the high level simulation result is almost the same as that of the low level simulation should not be taken too significantly. What is striking is that with the hybrid simulation, where one of the processors is modeled at high level of abstraction and one of the processors at low level of abstraction yields such a different result.

The explanation can be found in the way the two processors alternate access to the shared resource. Both at the fine grained level and on the coarse abstraction level, the access pattern is exactly interleaved. Granted, at the high abstraction level this interleaving is at the granularity of images while for the low-level of abstraction the interleave pattern is at a line-size granularity (which is also why the fact that both reach an identical result is not self-evident). The drop in performance at relative higher communication to computation ratios can be explained by this too. When a computation cycle at one of the processors finishes earlier that the communication cycle of the other, the communication cycle of the first mentioned cannot yet take place, creating stalls.

The interleaving pattern at the hybrid simulation is however much more different. When the high-level abstraction task obtains access to the shared medium it can transfer an entire image, while the task operating at lower abstraction only transfers a single line-size before it needs to gain access to the shared medium again. This way, the task at the lower level of abstraction is at a disadvantage when com-

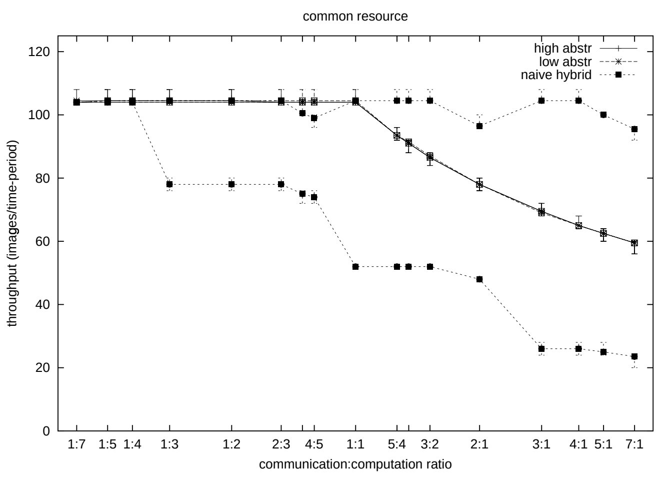


Fig. 7. Experiment using a simulation at high- and low-level abstraction. At each of the two processors the number of images being processed is counted over a fixed period of time. The throughput of both processors is almost equal to each other for the high- and low-level models. The high- and low-level models also give the same result. A naive hybrid simulation is also shown where the processor accessing the shared resource at a higher level of abstraction obtains a much higher throughput and the processor at a lower level of abstraction a much lower throughput.

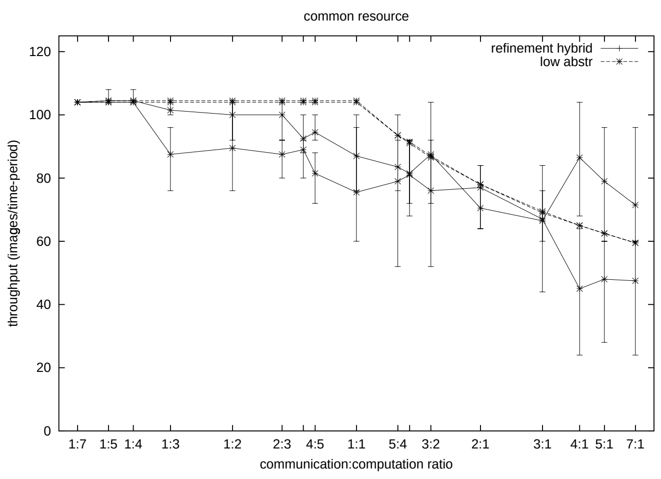


Fig. 8. Simulation results for a hybrid simulation obtained through the process of refinement. One processor is modeled at a low-level of abstraction, and the other at a high-level of abstraction. The latter does however translate the operation to transfer an entire image into smaller individual requests just before accessing the shared resource. However, due to the fact that the processors do not execute the same schedule, the results are not the same as a completely low-level model. Also shown in this figure are the lowest and highest throughput measured at each configuration (shown using the vertical bars). The models using refinement show a far more irregular behavior, which makes it difficult to draw conclusions from these results.

peting for the shared medium. This can clearly be seen in the result: when the communication factor is relatively high, the task modeled at a higher level of abstraction has a higher throughput than the task modeled at the lower level of abstraction within the same simulation.

Approaches to perform hybrid abstraction level simulations often include the use of modules which translate one level of abstraction to another. The method used is comparable to stepwise refinement. A module between the processor and the shared medium translates an operation at a high level (load an image) to a sequence of operations at a lower level (loads of a line-sized part). This translation makes sure that operations at a shared resource arrive with the same grain size, however as in stepwise refinement the schedule is different.

Figure 8 shows the result of this approach. Although on average the result looks better then the naive hybrid model, the clear transition shown by the detailed simulation at equal computation to computation ratio is not present. In these simulations we would like to know whether our operation has a graceful degradation or a transition.

The figure also shows the deviation from the average throughput measured in the simulation. The high- and low-level model and the first hybrid models showed a very stable simulation result, while in the refinement method many different values where measured. This makes us question a valid interpretation of our simulation, when no stochastic processes where involved.

VI. Towards better hybrid models

It is relatively easy to explain why one gets the results shown in the previous section. However this does not actually tell what the fundamental cause is.

The principal error we have made is that we used two models which have different strategies and placed them together. In each of the strategies there is a different view on the time scale. The high-level model allocates the shared medium to transfer an entire image, and has thus a coarse concept of time. Conversely, low-level models have a finer-grained concept of time.

These concepts should not be mixed and therefore one cannot let both abstraction levels use the same shared component. The high-level abstraction level should access a shared resource modeled at high-level of abstraction, and likewise the processor task at the detailed level should access a low-level shared resource component. But this is a shared component, so the effect of contention for the shared resource would be missing if we would introduce separate instances for the shared resource.

We believe that it is still a good idea to use separate components for the same shared resource for each abstraction level being mixed. This has the advantage that we largely avoid complicated model components which can perform operations at multiple abstraction levels.

Even more important, high-level components continue to be interfaced with the normal high-level (shared resource) components, and low-level components with other low-level components, keeping the distinction between abstraction levels and their corresponding strategies intact. This under the presumption that a mechanism is embedded in the shared resource which performs the correction for contention for the shared resource.

Earlier was stated that the expected result of a hybrid system should be somewhere between the corresponding high-level and low-level models. This is a fairly useless statement in complex systems with more than two metrics or more than two abstraction levels. Furthermore it does not tell anything about how abstraction levels should be integrated. With in mind that we can separate a component where abstraction levels meet into separate components for each abstraction level, we can make the previous statement more precise:

In a hybrid system, a component modeled at a certain abstraction level should be able to observe the same behaviour from another component with which it is interfaced and which is part of the hybrid interface (i.e. the bridge between different abstraction levels) as if all the components in the model would have been modeled at the same abstraction level.

A simulation model of the two abstraction levels with different components for the shared resource for each of the abstraction levels has been built, which includes a method under development for incorporating contention for a shared resource. This mechanism works by influencing the internal latency in the shared resource. Normally when an operations from the processor is received by the shared resource, a delay is introduced by the resource before forwarding the operation to the destination to model the latency of the resource. In the earlier models this delay only depends on the size of the data request. If the shared resource is still processing a request from the other processor, the operation is queued until it has finalized earlier requests. This is the basis for contention and the variable time needed per request. In our implementation the queuing effect does not occur anymore, but instead it is observed how long in the internal latency the resource is also accessed by the other abstraction level. This time represents a possible equally sized delay that could have occurred by the exclusive access to the resource. One method is for the resource to complete the request and return the result, but remain to be blocked for a period of time equal to the time the resource was actually shared by two abstraction levels. Another way is to introduce an additional latency for the same period of time. These two results are shown in figure 9.

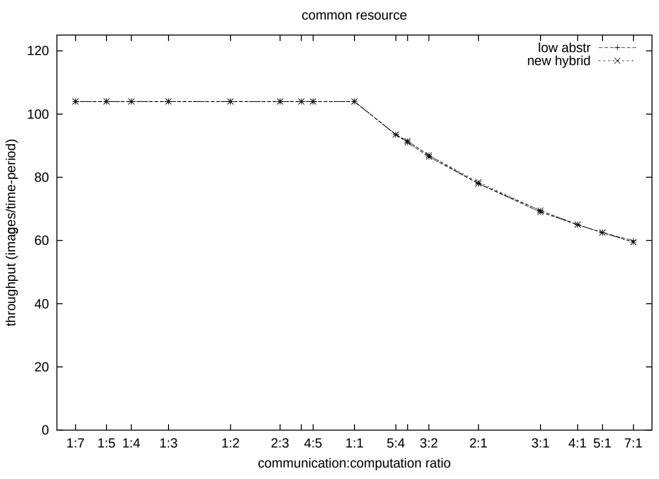


Fig. 9. Comparison between a pure low-level abstraction and the adapted hybrid model. In the new hybrid model the components that model the shared resource for an abstraction level return the result immediately to the processor after performing the request, but can remain blocked for a longer period.

The two methods are actually two extremes of the range of an exploration parameter, which represents the possibility for an abstract model to overlap communication with computation.

VII. Conclusion

In this paper we have looked at some of the problems that occur when building hybrid simulation models, where multiple modeling abstraction levels are placed within a single executable simulation. Although these models were executable, they are not immediately valid. Some very simple architecture constructs were used in this study, which does however not mean that these are just toy-like models. Rather, it concentrates on how multiple abstraction levels can really interact within a single model. One or multiple components will be faced with operations on multiple levels of abstraction. Whether this is a buffer, bus, co-processor or other type of component is rather irrelevant. What is important, is that this hybrid interface is faced with different concepts, especially concerning granularity of time-steps.

This leads to problems when synchronization occurs between components on different abstraction levels. Synchronization because one component has a direct dependency on the other, e.g. it needs data from the other, or synchronization because a common shared resource is being accessed. This paper has mainly focussed on the sequencing and interleaving of operations of the latter and within the Artemis project [6] we are looking into methods to resolve this problem.

REFERENCES

[1] RASSP Taxonomy Working Group, "VHDL modeling terminology and toxonomy," Technical Report Version 3.1, RASSP Taxonomy Working Group, 1999.

[2] P.S. Coe, F.W. Howell, R.N. Ibbett, and L.M. Williams, "A hierarchical computer architecture design and simulation environment," *ACM Transactions on Modeling and Computer Simulation*, vol. 8, no. 4, October 1998.

[3] M Bacis, G Buonanno, F Fummi, L Gerli, and DSciuto, "Application of a testing framework to VHDL description at different abstraction levels," in *Proceedings IEEE Conference on Computer Design*, October 12-15 1997.

[4] Moshe Meyassed, Robert McGraw, James Aylor, Robert Klenke, and Ronald Williams, "A framework for the development of hybrid models," *Proceedings of the 2nd Annual RASSP Conference*, pp. 147–154, July 1995.

[5] Robert H. Klenke, Moshe Meyassed, James H. Aylor, Barry W. Johnson, Ramesh Rao, and Anup Ghosh, "An integrated design environment for performance and dependability analysis," in *Proceedings of the Design Automation Conference*, Anaheim, CA, 1997, ACM.

[6] A.D. Pimentel, P. van der Wolf, E.F. Deprettere, and L.O. Hertzberger, "The artemis architecture workbench," in *PROGRESS2000 Workshop on Embedded Systems and Software*, 2000.