Evaluation of Multi-Core Scheduling Mechanisms for Heterogeneous Processing Architectures

Håkon Kvale Stensland[1], Carsten Griwodz[1,2], Pål Halvorsen[1,2]
[1]Simula Research Laboratory, Norway
[2]Department of Informatics, University of Oslo, Norway
{haakonks, griff, paalh}@simula.no

ABSTRACT

General-purpose CPUs with multiple cores are established products, and new heterogeneous technology like the Cell broadband engine and general-purpose GPUs bring an even higher degree of true multi-processing into the market. However, means for utilizing the processing power is immature. Current tools typically assume that exclusive use of these resources is sufficient, but this assumption will soon be invalid because the interest in using their processing power for general-purpose tasks. Among the applications that can benefit from such technology is transcoding support for distributed media applications, where remote participants join and leave dynamically. Transcoding consists of several clearly separated processing operations that consume a lot of resources, such that individual processing units are unable to handle all operations of a session of arbitrary size. The individual operations can then be distributed over several processing units, and data must be moved between them according to the dependencies between operations. Many multi-processor scheduling approaches exist, but to the best of our knowledge, a challenge is still to find mechanisms that can schedule dynamic workloads of communicating operations while taking both the processing and communication requirements into account. For such applications, we believe that feasible scheduling can be performed in two levels, i.e., divided into the task of placing a job onto a processing unit and the task of multitasking time-slices within a single processing unit. We have implemented some simple high-level scheduling mechanisms and simulated a video conferencing scenario running on topologies inspired by existing systems from Intel, AMD, IBM and nVidia. Our results show the importance of using an efficient high-level scheduler.

1. INTRODUCTION

Multi-processor and multi-core systems are quickly becoming mainstream computing resources. Dual-core general-purpose CPUs are established products, but systems including the Cell broadband engine (BE) and general-purpose GPUs bring an even higher degree of true multi-processing into the market. Making use of this parallel processing capacity, however, is still in the earlier stages. Current tools that assist in developing for many-core systems like the Cell and GPUs assume typically that the parallel computing resources can be used exclusively for a single task. We expect that this will become an invalid assumption because the increasing commoditization of parallel processing hardware and an increasing interest in using it for general-purpose tasks. Among the applications that can benefit strongly from this hardware are computing-heavy media processing applications.

Examples of this are multi-party video conferences where participants dynamically join and leave the session, personalized video streaming services and free-viewpoint 3D environments. In the video conferencing scenario, multiple streams have to be merged, adapted, and so on. Contributors and consumers join and leave conferences at arbitrary times and use heterogeneous devices. The necessary media processing operations can then consume much processing and bandwidth resources. Each individual operation can possibly be handled by an individual core without missing any deadlines. However, individual processing units are unable to handle the operations that are necessary for a conference of arbitrary size. Operations must then be distributed over several processing units, and data must be moved between them, all while staying within the application-defined deadlines. The media processing applications are time-dependent and cyclic depending on individual contributors' frame and sampling rates. Because the increasing heterogeneity of end systems, the resource consumption of media processing operations varies widely, as does the amount of data that is needed for communication between processing steps. The challenge is to then find a system scheduler for such dynamically changing workloads that fulfills the processing requirements of the application and at the same time avoids congestion on the interconnects between the processing units. Many multi-processor scheduling approaches exist, but we are not aware of efforts that address the dependencies of long-lived, cyclic, inter-job dependent processing operations that require time-critical communication with neighboring processing units. Our envisioned applications require that a feasible scheduler takes both capacity of processing units and the bandwidth of interconnects between them into account. We believe that feasible scheduling can be performed in two levels, i.e., divided into the task of placing a job onto a processing unit and the task of multitasking time-slices within a single process-

NOSSDAV '08 Braunschweig, Germany

ing unit. Our application provides a considerable amount of long-term knowledge for the long time-scale that can be fed to a high-level scheduler that can reserve resources within each timeslice, while the low-level scheduler for processing units and interconnects can act independently.

In this paper, we look at the interaction between interconnection topology and workload patterns using simple high-level scheduling mechanisms. The topologies are inspired by the interconnection topologies of the Opteron, the Core 2, the Cell BE and nVIDIA's latest GPUs. The workloads are modeled to reflect a video conferencing scenario where the size of each job is varied under the capacity limit of the single cores and links. Our results show the importance of using an efficient high-level scheduler in order to reduce the overall resource consumption. In particular, means to cluster jobs on cores in close proximity are promising because the bandwidth consumption is usually reduced.

The rest of this paper is organized as follows. The next section describes our example scenario. In section 3, we look at our hierarchical scheduling approach. We describe our simulator in section 4 and present some results in section 5. In section 6, we present some related work before we summarize and conclude in section 7.

2. EXAMPLE SCENARIO

Large-scale media processing applications are coming, and we use here video conferencing systems as a motivating example. Such conferences have multiple senders and receivers as for example possible today using equipment from vendors like Polycom, Tandberg or Lifesize. In this scenario, every conference participant should be able to receive A/V content from all the other participants. Only active participants will produce A/V data that must be processed, merged and sent to all the others. The setup allows a large degree of sharing in the processing graph, but requires expensive individual processing as well. Furthermore, depending on the equipment used by each participant, each stream may have different characteristics. Thus, we end up with a scenario similar to the one as depicted in figure 1 where we cannot assume equally sized jobs.

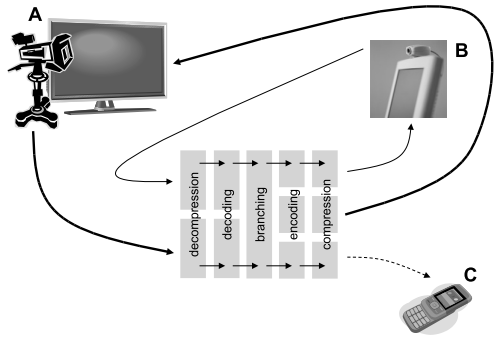


Figure 1: Example scenario: video conferencing

In the example in figure 1, the conference has three participants. The first (A) has equipment that produce and receive HD data. The second user (B) produce and receive SD data whereas the last participant (C) is only following (listening) the conference using a mobile phone displaying only video in CIF resolution. In this situation, each stream could be processed according to the layers in figure 1: 1) decompression, 2) decoding, 3) branching (e.g., to filter and adapt the data to the available resources and the receiving devices), 4) encoding and 5) compression. In addition, the system must handle dynamics in terms of users joining or leaving the session, users becoming active and oscillating resource availability in other parts of the system. Finally, each system often manages many concurrent sessions further complicating the task of finding available resources.

The processing resources required for handling these real-time tasks often (at least when the conferences are large) exceed the processing capacity provided by a single processing unit. Consequently, the imposed workload must be distributed to a number of processing units for parallel execution, for example using multi-core processors, multi-chip computers, processing clusters or distributed nodes over the Internet. However, parallel processing of a large number of real-time tasks introduces challenges related to orchestrating and multiplexing the resource usage in a timely manner. We are therefore addressing the challenge of finding feasible dynamic schedulers for a given set of (possibly heterogeneous) processing units and their corresponding links (like busses or network links).

3. TWO-LEVEL SCHEDULING

In the application scenarios that we imagine here, long-lived dynamic sessions of media operations must be mapped to resources of interconnected processing units. The data flows that are processed are sent from remote machines and are subject to packet loss and jitter. We can therefore not assume that a fine-grained, detailed reservation mechanism at cycle granularity is reasonable for the scheduled processing units. Rather, long-term, high-level decisions must be made to allocate resources of processing units, such that they are available for arriving data with high likelihood. To separate this from accurate low-level scheduling that is concerned with the allocation of resource shares, we consider two-level scheduling as a reasonable option.

First, a high-level scheduler acting on the timescale of signaling protocols must select processing units for tasks in such a way that each task has its required resources and that it can communicate with those jobs from which it receives streaming data and with those to which it forwards streaming data. This scheduler must know profiling information of processing stages and know approximately the processing demands of the expected workload. It must then choose processing units taking both remaining processing capacity and remaining interconnect bandwidth into account. When the jobs have been placed on a processing unit with available capacity, a low-level algorithm working on the timescale of operating system time slices must handle the individual jobs on this unit in order to find a schedule where each job meets its deadline.

3.1 Mapping media operations

The first step of the scheduler in our scenario is to perform the mapping of media operations to the processing units. Here, a decision is necessary for every join and leave operation, i.e., every change in the workload. We approach this as a timeless problem. At this level, we ignore the necessity to schedule units, and instead reason about capacities. Future work must also take the difference between timeslice cycle length and preemptable workload blocks into account. For each media operation, only the resource consumption is considered, resulting in a timeless flow model.

Likewise, the load that can be handled by a processing unit, the bandwidth between processing units and the communication requirements between operations are expressed by a single capacity value. The high level scheduler's task is then to find a mapping that neither violates the capacity limits of any processing unit nor any communication channel.

Migration of media operations between processing units is in principle possible in our scenario. However, we consider migration highly undesirable. There is no particular need for avoiding it in symmetric multiprocessing systems; however, most of the topologies that we investigate are either NUMA or non-shared memory architectures that would require active movement of the operations' state between the processing units. This would be very costly in terms of the interconnection bandwidth that is required for moving the complete processing state that is stored in one processing unit to the new processing unit. Additionally, migration takes time and may cause disruption of streams, and in a worst case, break the dependency if interconnection bandwidth is exhausted during migration, resulting in a complete rescheduling. Consequently, media operations are here mapped and pinned to processing units by the high level scheduler.

3.2 Managing timeslices

The scheduling of tasks at a particular processing unit is handled by a low level scheduler. The low level scheduler must support the flow abstraction, and the worst case performance and overhead must be known. This provides the capacity value available to the high level scheduler. For the low-level scheduler, media operations are independent of each other. This implies, that communication between tasks and the resources consumed for inter-task communication are not considered. Each task has a period and in every period a certain amount of resources is consumed.

Nevertheless, at this level, old basic schedulers such as Rate Monotonic, Earliest Deadline First, Heits, AQUA, RT Upcalls, SMART, etc. are useful schedulers, and earlier work (e.g., [7]) has shown that these schedulers are able to find a schedule for each individual processing unit as long as there in total is enough available resources (which is an admittance check performed by the high-level scheduler). We are therefore currently planning for independent single-core real-time schedulers to solve our need for low-level scheduling, but do not consider it any further in this paper.

4. SIMULATOR

To understand the interaction between interconnection topologies, different workload patterns and high-level scheduling algorithms, we have written a simulator to evaluate the schedulability of dynamic workloads. It allows us to look at metrics like bandwidth consumption and scheduling failure rates in a controlled environment, and we here introduce the topologies, workloads and algorithms that we have used.

4.1 Topologies

Current multi-core architectures take a variety of approaches for memory handling and interconnection between cores. We are currently not considering memory at all, but assume the need for data forwarding from one processing node the next. This is an approach that is not typically used in the current Intel architecture, but rather typical for specialized media processors, and also applicable for NUMA architectures. Interconnects come as switch, bus, or point-to-point approaches. We model all of them as switches and point-to-point links at this time. Furthermore, the architectures that we use in the simulations in this paper are inspired by the topologies of the Intel Core 2, AMD's Opteron, the STI Cell BE and nVIDIA's GPUs (see figure 2). Since we intend to examine the effects of topologies and not real-world processor capabilities, we have provide all models with the same total processing capacity. The processing capacity is however, distributed over different numbers of processing units (heterogeneous in case of the Cell-inspired topology). Similarly, the bandwidth of interconnects is modeled in abstract performance numbers according to their role in the multi-core architecture that inspired each topology.

4.2 Workloads

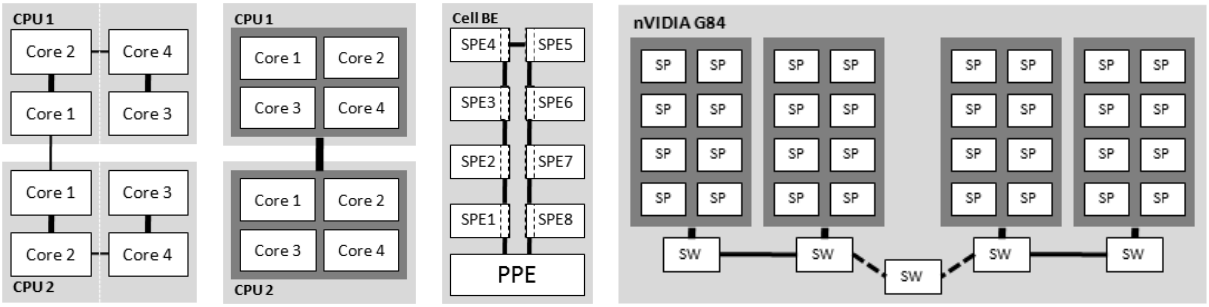
The workload is specified as graphs of interconnected media processing operations which represent the need for processing capacity. Bandwidth requirements between media processing operations represent the consumed interconnect bandwidth that is required when neighboring processing stages in the graph are not processed by the same processing unit. Figure 1 shows a very simple example for such a processing graph, and where for example the initial conference members are users (A) and (B), and the conference is later dynamically extended to accommodate user (C). Moreover, to test various scenarios, we used a workload generator with the ability to create dynamically changing media processing demands with well-known parameters. The workload generator creates conferences with negative exponentially distributed duration, and with either Poisson-distributed inter-arrival times or at a constant number of concurrent conferences. Within each conference, uniformly distributed sets of participants arrive and depart according to a Poisson process, where each of the media operations claims a processing power and bandwidth from a uniform distribution up to a given maximum value.

4.3 Scheduling Algorithms

Developing high-level scheduling algorithms for arbitrary inter-dependent workloads is a future goal, along with real-world implementations. A one-for-all algorithm appear to be out of the question, and there is no reason why a real-world implementation should be able to adapt dynamically to wide ranges of different hardware. In any case, this is out of the scope of this paper. Here, we use strategies that are inspired by packing strategies for passive operating system resources, but extended with functionality to check link availability. The selected results use the following strategies:

First Fit (FF): Assign every processing unit an index. For every media operation that requires a certain amount of processing capacity, start searching at the processing unit with index 0 for available processing capacity. If the unit with index 0 does not have sufficient resources, use a breadth-first-search (BFS) approach through the topological neighbors of the unit, until a unit with sufficient capacity is found, or until the scheduling fails.

Next Fit (NF): Similar to FF, but instead of starting at index 0 (first processing unit) for every new job, NF starts its search (in BFS order) from the node where the previous media operation was successfully scheduled. If subsequent media operations are interconnected, this can achieve a high degree of packing and saves interconnection bandwidth.



(a) Dual Core2 (b) Dual Opteron (c) Cell BE (d) nVidia G84

Figure 2: Tested processor topologies

Random Start (RS): In contrast to NF, RS keeps separate indices for each conference to start the search for appropriate processing units for newly arriving media operations belonging to that conference. This is meant to achieve better clustering and thus, less interconnection bandwidth consumption than NF. For a newly starting conference, a random processing unit is chosen.

Worst Start (WS): WS is similar to RS, but instead of randomly finding a processing unit for a newly starting conference, WS starts at the processing unit with the highest remaining free capacity (thus the name worst fit). This is done in the hope that the highest initial packing can be achieved for the new session.

We have also tested several others like plain first-, best-, worst-fit, etc., known from old memory management systems placing data elements in memory, but they do not take bandwidth into account so we have only used these as basis and benchmarks.

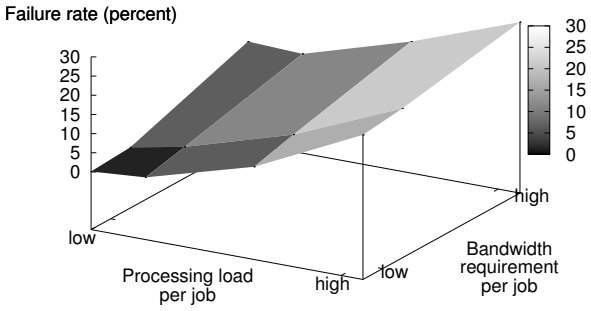


Figure 3: Failure rate versus processing and communication costs (FF on nVidia G84)

5. RESULTS

This section presents the results from our simulations including some selected plots and our main findings. Using the workload generator mentioned above, we have generated several workloads over the same set of parameters to be able to extract statistics. We have varied the core and link load and the number of concurrent conference sessions and participants. The same workloads have been used for all algorithms on all topologies.

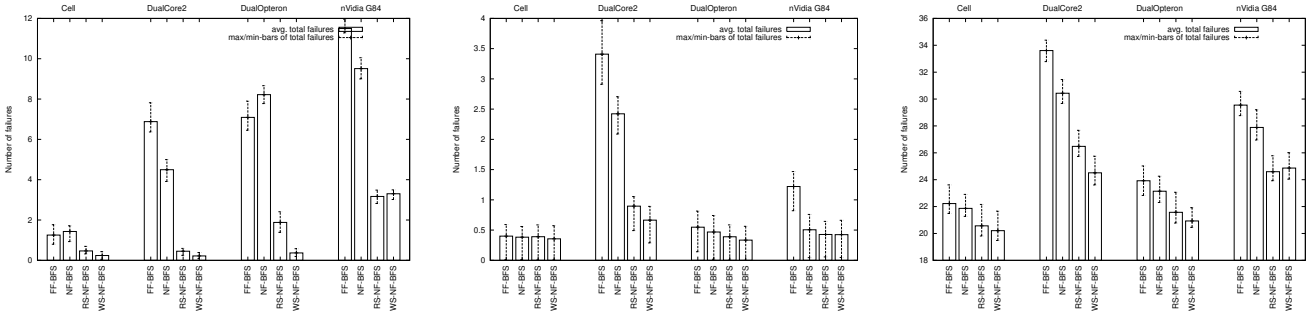
5.1 Scheduling Ability

We first look at how well different means help to find a schedule. As expected, the failure rates increase for all algorithms if either of the processing or communication cost increase (see figure 3 for a representative example). Furthermore, in figure 4, we compare the tested algorithms with respect to the failure rate. If we do not use BFS but a random ordering of the processing unit, and apply FF and NF as they are known from memory management systems that are not concerned with bandwidth, the performance drops significantly. There is usually also a failure rate reduction if we try to place related and dependent jobs in close proximity in order to minimize communication, as done by NF in contrast to only finding the first available as in FF. However, in our set of algorithms, we see the largest performance gain if we search for a suitable place (least loaded node) to start a new session like in RS and WS. These approaches are beneficial because they allow dynamically joining media operations to be packed on processing units that are topologically close to other jobs in the same session. When resource fragmentation grows, this saves considerable interconnect bandwidth.

5.2 Bandwidth Consumption

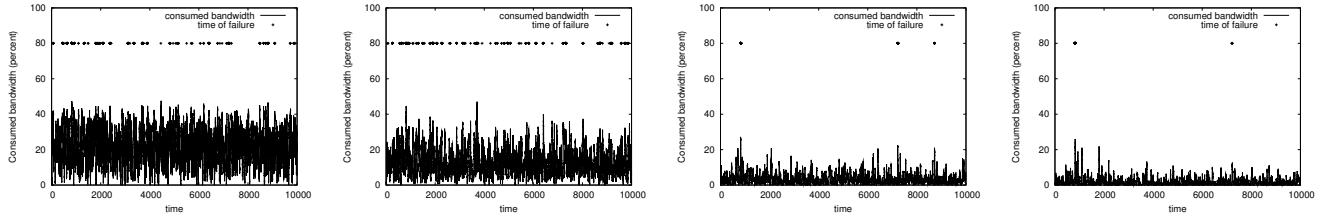
One other important metric in finding a good schedule is a measure for the communication required between processing units. Such communication both takes time and consumes bandwidth. A good schedule for a time-dependent scenario should often try to pack jobs on a cluster of processing cores to minimize communication. To see the bandwidth required for the found schedules, we monitored the communication needs during all the simulations. Figures 5 and 6 show representative examples where the workload is executed on Cell- and DualOpteron-based topologies. The plots show the consumed bandwidth on the time line and also a plot on which times the algorithm failed to find a schedule for a new job.

In general, always starting a search on the same node (as FF) effectively packs data on a small area of the topology. This is fine as long as there is enough processing power, but as the load increases, the system benefits from a more distributed clustering between sessions, i.e., packing each session on different places. A better approach is therefore to start every new jobs where the last job in the same session was placed (as NF). Furthermore, if the starting point is varied for each new session the results are increased further, where the best results are achieved by searching for the least loaded node with WS. As a consequence of a reduced bandwidth consumption, we see again that the number of failed



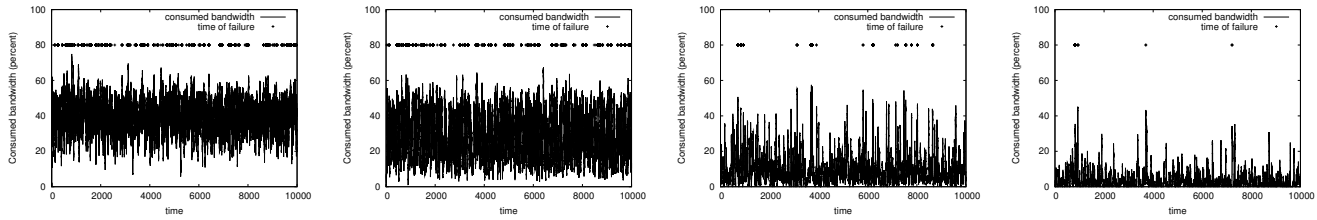
(a) Core: low, Link: high (b) Core: medium, Link: low (c) Core: high, Link: high

Figure 4: Scheduling failure rate for different algorithms on different topologies when varying the load



(a) FF (b) NF (c) RS (d) WS

Figure 5: Consumed bandwidth resources on a Cell topology.



(a) FF (b) NF (c) RS (d) WS

Figure 6: Consumed bandwidth resources on a DualOpteron topology.

schedules are reduced using RS and WS.

5.3 Discussion

There are several metrics that are important with respect to resource scheduling. In order to scale and support as many inter-dependent jobs (conference participants) as possible, important metrics include scheduling ability and efficient resource utilization. We have here shown that there are large differences for different kind of scheduling means, and there are also some differences based on the topology characteristics they are used on. In this respect, our results show that mechanisms like performing job clustering, using BFS and finding a high capacity place to start a new session, are promising because the bandwidth consumption is usually reduced. This is especially important if bandwidth is a scarce resource or the workload is time-critical due to added communication delay.

Other interesting approaches in terms of compacting the workload is to analyze the workload and try to make larger computing blocks ahead of running the scheduler and then recursively split on the cheapest link is no schedule is found. This is efficient as long as there are much available processing capacity, but when the load increase and the large blocks must be re-divided, this approach increase the probability that large blocks will be placed further from each other, i.e., consuming bandwidth on several links. Thus, in our test, the failure rate on a loaded system is greatly increased meaning that more logic must catch and avoid the long-distance placement of neighboring jobs.

In this preliminary study, we have focused on scheduling ability and bandwidth consumption only. However, there are other metrics that are relevant. For example, the time to find a schedule or scheduling overhead will in some scenarios be important, but they will be implementation and architecture dependent and are thus not considered in the simulator. Load-balancing properties of the algorithm is also in some cases important, but in the conferencing scenario discussed here, the reduction of bandwidth consumption and load balancing is contradictory in terms of link overhead, and we therefore here look at the ability to compact the workload to save bandwidth.

In our work on efficient scheduling algorithms for inter-job dependent workloads, we aim for a Linux implementation. Here, it will be important to also integrate the low- and high-level schedulers for example to exchange information about available resources. However, as a first step, we look

at what kind of means are most promising based on tests in our scheduling evaluation framework.

6. RELATED WORK

Multi-core scheduling attracts a considerable amount of research that aims at load-balancing, cache coherency, and the like. Approaches found in the literature that combine mapping to processing units and scheduling of tasks on the processing nodes in one stage, such as PFair [6], can schedule based on processing capacity alone, since they rely on a unified shared memory architecture. Our problem requires a coordinated scheduling of processing capacity and interconnection bandwidth that is due to a lack of a shared memory-assumption. Perfect scheduling under these conditions is NP-hard, and alternative heuristics must be explored. One of the classical scheduling methods are greedy algorithms. An example of a greedy algorithm can be found in [1]. Here, the tasks of a parallel application are mapped onto different processors based on a the expected run-time of the job. This algorithm does not take job dependencies into account. Another approach is based on the representation of dependent parts of an application as a directed acyclic graph (DAG). DAG can represent both processing and communication demands. Kwok et al. [5] have made a survey of several classes of scheduling algorithms for allocating workload DAGs to a network of processors, e.g., highest level first with estimated times, linear clustering, task duplication and mapping heuristic. DAG-algorithms are also often used to schedule workloads in Grids and several approaches have been suggested in [2]. These solutions are not directly applicable to our case because the do not handle dynamic workloads without full reconfiguration. Another type of algorithms are genetic algorithms [4]. These algorithms are focused around deterministic scheduling problems, meaning that all information about the task and their relations to each other are known in advance. On asymmetric multicore processors there have been several approaches for soft real-time scheduling. In [3], a soft real-time scheduler for Linux has been implemented. Dependencies and communication between the real-time jobs have however not been considered.

7. CONCLUSION

As a step towards better scheduling support in the operating system for inter-dependent jobs on possibly heterogeneous multi-processors without uniform shared memory, we have examined the effects of topologies on the schedulability of dynamic workloads that consist of interdependent long-lived media operations.

We have motivated the distinction between high-level schedulers that act on the timescale of application scheduling and assign processing resources to long-living media operations, and low-level schedulers that managed computing cycles on processing units locally based on assignments of the high-level scheduler and otherwise without global knowledge.

We found that high-level schedulers that try to achieve compact assignment of operations that belong to the same media session to processing units is generally beneficial (as expected), but we have observed extremely strong improvements in topologies that rely on a bandwidth-constraint switched architecture such as the Core 2 or nVidia-GPU inspired topologies. These effects appear to be fairly independent of whether the switches connect few high-capacity processing units or many low-capacity units. On the other hand, we can observe that even the simplest tested scheduling algorithm (first fit) performs quite efficiently on the Cell-inspired ring topology with asymmetric nodes.

In various other tests, we observed also a variety of other effects of scheduler features. One of the more interesting observations was that it is not generally feasible to schedule media operations of a session in larger blocks. This forces the scheduler in many cases to place the block rather far away from earlier parts of the session, consuming excessive interconnection bandwidth and finally, exhausting it. We could also observe that choosing a suitable first node for newly starting sessions has a major effect on performance. An alternative scheduler might generally optimize for minimal bandwidth consumption and processing capacity only as a computational bound.

After these initial observations, we will use our simulator to evaluate a variety of high-level schedulers. Concurrently, we are working on programming tools for Cell and nVidia that circumvent the provided frameworks, which are unable to schedule non-exclusive workloads, as well as porting of media operations to these environments. Given such tools, we will be able to investigate the interaction of high and low-level schedulers experimentally. We find that a final step in this development must integrate this scheduling with the operating system scheduler.

8. REFERENCES

[1] Armstrong, R., Hensgen, D., and Kidd, T. The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions. In *Heterogeneous Computing Workshop (HCW)* (Mar. 1998), pp. 79–87.

[2] Batista, D. M., da Fonseca, N. L. S., and Miyazawa, F. K. A set of schedulers for grid networks. In *ACM Symposium on Applied Computing (SAC)* (New York, NY, USA, 2007), ACM, pp. 209–213.

[3] Calandrino, J. M., Baumberger, D., Li, T., Hahn, S., and Anderson, J. H. Soft real-time scheduling on performance asymmetric multicore platforms. In *IEEE Real Time and Embedded Technology and Applications Symposium (RTAS)* (2007), pp. 101–112.

[4] Chockalingam, T., and Arunkumar, S. Genetic algorithm based heuristics for the mapping problem. *Comput. Oper. Res. 22*, 1 (1995), 55–64.

[5] Kwok, Y.-K., and Ahmad, I. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys 31*, 4 (1999), 406–471.

[6] Moir, M., and Ramamurthy, S. Pfair scheduling of fixed and migrating periodic tasks on multiple resources. In *IEEE Real-Time Systems Symposium (RTSS)* (1999), pp. 294–303.

[7] Wolf, L. C., Burke, W., and Vogt, C. Evaluation of a cpu scheduling mechanism for multimedia systems. *Software - Practice and Experience 26*, 4 (april 1996), 375–398.