Resource-aware High Performance Scheduling for Embedded MPSoCs With the Application of MPEG Decoding

Parth Malani, Ying Tan, Qinru Qiu
Department of Electrical and Computer Engineering
Binghamton University, State University of New York
Binghamton, New York 13902, USA
{parth, ytan3, qqiu}@binghamton.edu

Abstract – In this paper, we propose a scheduling algorithm to minimize the resource contentions and the processing latency for applications running on a multiprocessor system-on-chip (MPSoC) platform. The scheduling algorithm is applied on an MPSoC MPEG decoder to improve the system performance. Application specific task partition and mapping techniques are further investigated. The experimental results show average improvement of 17% in total latency when comparing to the ad-hoc scheduled method.

I. INTRODUCTION

Multiprocessor system-on-chip (MPSoC) platform has become more and more popular in the design of embedded systems. Due to the parallel architecture, the MPSoC provides significant performance improvement compared with the traditional single processor SoC.

The hardware components in an MPSoC, such as memory, I/O and processing elements (PEs), are usually referred as resources. If certain resources are shared among the PEs, there will be a potential resource contention. For example, shared memory architecture is widely used in multiprocessor system. CELL processor from SONY, TOSHIBA and IBM, Daytona [1] from Lucent and Piranha [2] from DEC/Compaq are all single chip multiprocessor using shared memory architecture. In these systems, the contention for bus and memory access may be the performance bottleneck of the applications that require frequent memory read and write.

As the number of PEs in the MPSoC increases, the potential resource contention also increases. The conventional way to resolve resource contention such as bus contention or memory contention is to use an arbiter. For example, the function of a bus arbiter is to prioritize the bus requests and to grant the bus access to the device that has the highest priority. This increases the hardware cost. Furthermore, a PE is idle while it is waiting to be granted for bus access. This wastes the processing bandwidth. An efficient scheduling algorithm should be able to reduce the time that a PE spent waiting for any shared resource. The traditional scheduling algorithms cannot solve this problem because they assume that each operation occupies only one resource. However, in an MPSoC system, an operation that uses a shared resource must also occupy one of the PEs at the same time. Therefore, two or more resources must be allocated to one operation.

This paper presents a modified force directed scheduling algorithm that solves the above mentioned problem. Although this algorithm is designed for general multi-resource scheduling problems, here we apply it particularly to the scheduling problem for an MPEG video decoder implemented on an MPSoC with shared memory. We use an example of CELL architecture here to show how our approach can fit onto this type of systems.

The proposed scheduling algorithm starts with task partition and distribution. In most image processing applications there is a repetitive sequence of operations that are performed independently on tiny samples of the whole job. For example, in MPEG decoder the whole frame is decoded by applying the same decoding procedure on an 8x8 pixel area called *block*. These blocks do not depends on each other and require only information from past frames that are stored in the memory. The similar processing nature is observed in other DSP applications as well, such as Edge Detection in an image. We consider these small blocks as a task unit and distribute them among available processors so that there is no communication between processors when decoding the same video frame. Each task unit is further divided into subtasks, which will be the input of the scheduling procedure. Compared with traditional task partition and mapping, which distributes operations, such as Run Length Decoding, Inverse Discrete Cosine Transform (IDCT) and motion compensation onto different processors on an MPSoC platform [3], the proposed approach significantly reduces inter-processor communication. The force-directed scheduling algorithm prioritizes and schedules the subtasks based on the resource utilization probability in each cycle. The algorithm schedules the subtasks so that the resource utilization probability will be distributed evenly in each control cycle. This implies a balanced resource usage during the task execution.

There are many works focusing on performance optimization of MPSoC running DSP applications especially MPEG. Many of them divide different decoding function of MPEG video decoder to different PEs to achieve performance improvement [4]-[6]. However, the only work in MPSoC behavioral synthesis which focuses on reducing inter-processor-communication (IPC) is suggested in [7]. It assumes a hardware architecture that has the point-to-point communication between the processors, which is different from the architecture that is interested here. In this paper we present an approach which totally eliminates the inter processor communication for a given application and a scheduling algorithm and mapping techniques to obtain minimum latency with resource contention avoidance.

The rest of this paper is organized as follows. Section II introduces the targeted hardware architecture and scheduling problem in MPEG decoding. Section III presents our resource-aware multiprocessor scheduling algorithm for general purpose. We present our experimental results and discussion in Section IV. Finally, the conclusions are given in Section V.

II. MULTIPROCESSOR MPEG DECODER

This section provides a brief background on MPEG decoding algorithm and discusses the design optimization of a shared memory MPSoC MPEG decoder. The targeted hardware platform has a CELL liked architecture [10] which is a heterogeneous chip multiprocessor with a CPU core and a set of eight special purpose processing units also known as Synergistic Processing Elements (SPEs). A CPU is a POWERPC core that can run two threads and has L1 and L2 cache. The SPEs do not have any cache but have a small size (256kb) local storage (LS) instead. The assumed MPSoC

architecture in this paper is shown below in Figure 1. For simplicity, a three-processor case is shown in the figure. We assume that our scheduling and mapping algorithm resides in CPU and the actual computation is carried out in the PEs utilizing the system potential to the maximum. Each PE is connected to an on-chip coherent bus and they use Direct Memory Access (DMA) technique to communicate with shared memory.

The MPEG video stream has a hierarchical layered structure. It is a sequence of *GOPs* (*Group of Pictures*), each one of which comprises of several *frames*. There are three types of frames defined in MPEG standard. **I**-frames or *intra-coded* frames are encoded as a whole image i.e. it does not depend on any other picture. **P**-frames or *predictive coded* frames are encoded using past I or P frame as a reference. Finally there are **B**-frames also called as *bi-directionally predictive coded* frames that use both past and future I or P frames as references. Each frame is further divided into vertical strips called slices. Each slice contains several *macro blocks* that are a 16 by 16 pixel area of the image. There are six *blocks* per macro block amongst which four are luminance (Y) and two are chrominance (Cr and Cb) blocks.

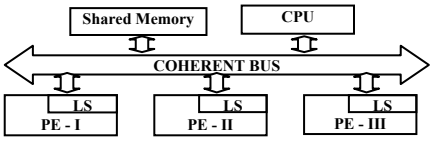


Figure 1 The MPSoC architecture

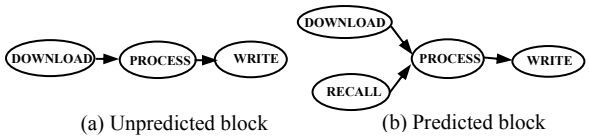
The MPEG decoding consists of operations such as *parse*, *decode*, *motion compensation* (*MC*), *inverse quantization* (*IQ*), *IDCT*, *variable length decode* (*VLD*), *display* etc [3]. The smallest data unit on which these operations are performed is a block. Based on how they are encoded, the blocks can be categorized into 8 types [11]. Each type of blocks goes through certain number of decoding steps or operations, which are summarized in Table 1. Note that the 8th type of block is skipped block. It is directly copied from the reference frame. No further processing is needed. Those blocks that require *motion compensation* at decoder need to recall the reference image from the memory. This operation is called as *recall*. Besides the four operations listed in Table 1 and the *recall* operation, two operations are must for all blocks. One of them is *download*, which downloads the encoded block from the main memory. The other one is *write*, which stores the decoded image back to the memory.

Table 1 Decoding steps for 8 types of MPEG blocks

	1	2	3	4	5	6	7	8
Forward MC		√	√			√	√	
Backward MC				√	√	√	√	
IDCT	√		√		√		√	
VLD, IQ	√	√	√	√	√	√	√	

The *read*, *download*, and *write* operations require two resources, the PE and the memory subsystem. Those four operations given in Table 1 only require the PE and they are always performed after *recall* and *download* but before *write*. Therefore we merge these four operations together and call them *process*. These four operations (i.e. recall, download, write and process) are referred as *subtasks* in the rest of the paper. Based on the number of subtasks that are involved in the decoding procedure, the blocks can also be classified into categories *unpredicted* and *predicted*. The dependencies among the subtasks are illustrated in Figure 2.

One important fact is that there is no dependency between two subtasks that belongs to different blocks of same frame. So in homogeneous MPSoC it is advantageous to decode an entire block on a single processor to reduce the inter-processor communication.



(a) Unpredicted block (b) Predicted block

Figure 2 Task set created from MPEG decoder

The execution time for the subtasks can be estimated before the actual decoding started. The *recall* and *write* operations always have fixed execution delay. The execution time for *download* operation can be predetermined based the size of the encoded block. The *process* nodes have variable execution delay depicting the actual time spent decoding a particular block. Based on the block type information, which can be extracted from the frame header, the process time can be estimated fairly accurate [11].

The MPEG video frame comprises of hundreds of blocks that belong to any of the two categories shown in Figure-2. These blocks will be distributed to different processors for decoding. In addition to the computation resources, the *recall*, *download*, and *write* operations require shared resources such as the bus and memory. The proposed scheduling algorithm arranges run time of each block to avoid resource contention and achieving minimum latency while maintaining QoS.

III. Resource-aware scheduling in MPSoC

In this section, we propose a minimum latency resource constrained scheduling algorithm for multi-processor system with shared resources. The new algorithm improves the traditional scheduling algorithm by allowing each operation to occupy more than one resource.

The input of the scheduling algorithm is a data flow graph $G(V, E)$, which models the data/control dependency of the operations that have been mapped to different processors. A vertex v in the graph corresponds to an operation. An edge from vertex v to vertex w indicates that w cannot start until v has finished. A set of available resources is given, which is denoted as A_{res}. Each vertex v in the graph is associated with a set of resources A_v, $|A_v| \geq 1$. Operation v cannot start until all of the resources in A_v are available. Further, each operation v takes D_v cycles to execute acquiring its resources for this duration. For example, Figure-2 (a) and (b) show the data flow graphs of the decoding procedure for one MPEG block. We assume that the mapping between operations and PEs is given.

Our scheduling algorithm is a modified force-directed list scheduling algorithm [9]. The major difference between the algorithm proposed here and the classical force-directed algorithm is that our resource set consists of processors and memory and there are some operations that require more than one resource, while the classical force-directed algorithm only considers single resource type per operation.

The proposed scheduling algorithm prioritizes the tasks to be scheduled using their *total-force*, which can be calculated using the following equation.

$$total-force(v,l) = \sum_{k \in A_v} self-force(v,l,k) + \sum_{\substack{v' \in \text{successor} \\ \text{or predcessor of v}}} PS-force(v',l)$$

In the above equation, $self-force(v,l,k)$ relates operation v that requires resource k to cycle l and $PS-force(v',l)$ gives the changes of self-forces of the successors/predecessors of v given the

condition that v is scheduled at cycle l. The following equations calculate the self-force and PS-force.

$$self\_force(v,l,k) = \sum_{m=est_v}^{lst_v} Q_k(m)\cdot(\delta_{lm} - P_v(m)), k \in A_v ,$$

$$PS\_force(v',l) = \sum_{k\in A_{v'}} \sum_{est_{v'} \le m \le lst_{v'}} Q_k(m)\cdot(P_{v'|t_v=l}(m) - P_{v'}(m)) ,$$

where est_v and lst_v are the earliest and latest start time of v, δ_{ml} = 1 if $m = l$ and 0 otherwise, $P_v(t)$ is the probability of an operation v being scheduled at any given cycle t, it is calculated as

$$P_v(t) = \begin{cases} 1/(lst_v - est_v) & \text{if } est_v \le t \le lst_v \\ 0 & \text{otherwise} \end{cases},$$

$P_{v'|t_v=l}(m)$ is the $P_{v'}(m)$ given the condition that operation v is scheduled at cycle l. $Q_k(l)$ is the *operation type probability*. It indicates the usage of given resource at particular time l, therefore $Q_k(l) = \sum_{v\in V, k\in A_v} P_v(l)$.

From the definition we can see that, the operations requiring multiple resources has multiple self-forces. However, all of them will be added to count the total force.

A force-directed scheduling algorithm picks the operation and control cycle pair (v, t) which has the least total-force to be schedule because such scheduling produces a more balanced resource usage in each cycle. In this work, we extended the traditional algorithm to consider multiple resource requirements. We not only changed the definition of self-force, PS-force and total-force, but also modified the scheduling algorithm to consider multi-resource contraints. Figure 3 shows the modified force-directed list scheduling algorithm. Here S_k is a set of all operations that requires resource k and U_{lk} are those operations whose time frame includes the current scheduling cycle. T_{lk} is a set of the operations that were scheduled before and are still running at cycle l.

FD_LIST_SCHEDULING_MPSoCs (G(V,E),a) {
1. *Given set S of all operations,*
2. repeat *{*
3. *Compute Time Frames of unscheduled operations in S;*
4. *Compute operation and type probabilities;*
5. *Compute self-forces and predecessor/successor forces;*
6. for *each resource type k = 1,2,….,A_{res} {*
7. *Determine candidate operations $U_{lk} \in \{S_k\}$;*
8. *Determine unfinished operations T_{lk};*
9. *Select $V_k \in U_{lk}$ vertices, such that $|V_k| + |T_{lk}| \le 1$ for all k;*
10. *Schedule the operation V_k with the least total force at step l by setting $t_i = l$ and update its time-frame;*
11. *Make all the resources A_v busy for D_v;*
12. *}*
} until (all operations are scheduled);
return (t);
*} //**End FD_LIST_SCHEDULING_MPSoCs***

Figure 3 Proposed force-directed list scheduling algorithm

The algorithm first calculates the time frame (i.e. [est_v, lst_v]) for each task accounting for the data dependencies. A time frame of an operation is a valid period of time cycles inside which it can be scheduled. Once the time frames are computed the probabilities of each subtask within its time frame are computed followed by operation type probability computation. An operation with least total force is scheduled at every cycle l for each resource k by setting its start time $t_i = l$. After scheduling the operation, its updated time frame is now reduced to a time cycle in which it has been scheduled. When there is a tie in the total-force, candidate operations that require more resource will have higher priority to be scheduled. Once any operation is scheduled, resources required for that operation is labeled busy until its execution is finished. On the other hand, if an operation has more than one resource type, it must satisfy all resource constraints to be scheduled. This whole sequence of procedure is repeated until all operations are scheduled.

The proposed force-directed scheduling algorithm is a general purpose scheduling that can be applied to schedule any multiprocessor applications. It can be used to schedule the decoding tasks in a multiprocessor MPEG decoder. In such system, the number of resources is fixed and includes all processor and a shared system bus/memory. A_{res} = {PE_i, M} $0<i<N$. The data flow graph is composed of many subgraphs. Each of these subgraphs has one of the topologies that are given in Figure 2. The subgraphs are disconnected to each other. Each vertex in the DFG corresponds to one of the following subtasks {*download, recall, process, write*}. Among these subtasks, *download, recall* and *write* operations consume two resources, the system bus/memory and a PE. A scheduling algorithm has to take care of this multiple resource case. The other subtask *process* only requires a processor as a resource. A latency upper bound of the decoding procedure needs to be specified in order to run the scheduling algorithm. A natural choice is to set it to the display period of the MPEG video frames. However, this only gives a loose upper bound which increases the scheduling complexity. Here we use the total decoding time on a single processor as an estimation of latency upper bound. Since the blocks are divided amongst available processors, it is highly unlikely that the scheduling algorithm achieves minimum latency greater than this upper bound.

The scheduling algorithm schedules the subtasks in the decoding procedure for minimum latency with resource contention avoidance. The configuration of the L1 cache size imposes some additional constraints to the scheduling problem. In case there is unlimited cache space, no extra handling is needed. If the cache is only large enough to store a single block, then each PE can decode only one block at a time. This constraint signifies that once a subtask of a block is scheduled for any processor, no other subtask of a different block can be scheduled for that processor until all operations of the given block are scheduled. To work with such constraint, the ASAP algorithm, which is used to calculate time frame of each operation in our scheduling algorithm needs to be modified. Any operation that belongs to the blocks that other than the current scheduled block can only start after the current scheduled block finishes its processing.

IV. Experimental Results

We apply our approach to an MPEG video decoder [8]. To divide each block of MPEG video frame into several subtasks, it was necessary to know the execution time of each subtask. We ran various probes on an MPEG-2 video decoder and figured out the processing time for each of the eight block types shown in Table-1. Two MPEG-2 movie clips (*bobo* and *canyon*) were tested for our experiments. The total number of blocks inside each frame of these movies (resolution) was 680 and 378. We tested four different types of frames from both movies. For the experiment we used the architecture with a CPU and three PEs as shown in Figure 1. Three simple mapping techniques are used in our experiments.

Sort_equal: The blocks are first sorted according to their minimum execution time and then distributed to each PE one by one so that the minimum delay of each PE is approximately equal.

Sort_unequal: The block are first sorted according to their minimum execution time and then divided by total number of PEs to distribute them to each PE. The minimum delays of each PE will be different.

Random: The blocks to each PE are distributed randomly without performing sorting procedure.

The effectiveness of our approach is evaluated in terms of overall latency. We compared our algorithm with an ad-hoc scheduling method which let the PE start decoding the next block as soon it is available. Using the ad-hoc scheduling, the selection of the blocks is based on the order of their locations in a video frame. We assume that the system under ad-hoc scheduling has a fixed-priority bus arbiter. To obtain the fair comparison we used the same mapping techniques for both systems.

Table 2 shows the comparison between our algorithm and ad-hoc scheduling for three different mapping techniques. The results for modified force-directed algorithm proposed here are indicated under columns marked as FD while those for ad-hoc scheduling are marked as AH. Also shown for each frame is number of operations after task partitioning. The latency for each case is indicated as integral number of time cycles. As seen from the results, the overall improvements achieved by our algorithm for all three mapping techniques are 15.73% for sort_equal, 17.41% for random and 10.42% for sort_unequal. It can be also seen that the improvement for the first frame of each movie is as high as 24%. These frames are I frames which correspond to a high motion video scene. In I frames, all blocks are of same type. The best schedule is achieved if blocks are scheduled one by one for each processor. Since the ad-hoc scheduling system has fixed priority it suffers from latency increase. The other three frames are P or B frames. Also shown for each frame is the number of subtasks (operations). It can be seen that in terms of improvement random mapping achieves the best result while sort_unequal has the least amount of improvement. However, this is mainly because the ad-hoc scheduling system performs the worst under the random mapping and the best under the sort_unequal mapping. The proposed scheduling algorithm has a slightly better performance under the sort_equal and it performs the worst under the random mapping. Though the difference is small, it provides a good hint towards developing a specific mapping algorithm to aid the given scheduling algorithm.

Table 2 Achieved minimum latency (cycles) for different mapping techniques

Movie	Frame	Operations	Mapping Techniques								
			Sort_equal			Random			Sort_unequal		
			FD	AH	Improvement	FD	AH	Improvement	FD	AH	Improvement
Bobo	89	1944	1948	2592	24.85 %	2056	2661	22.74%	1948	2592	24.85 %
	90	2580	3223	3792	15.01 %	3253	3992	17.27 %	3233	3493	7.44 %
	91	2580	3224	3633	11.26 %	3248	3735	13.04 %	3231	3436	5.97 %
	92	2364	2788	3581	22.14 %	2799	3727	24.90 %	2930	3382	13.36 %
Canyon	17	1134	1138	1512	24.74 %	1270	1605	20.87 %	1138	1512	24.74 %
	18	1506	1884	2003	5.94 %	1894	2126	10.91 %	1884	1939	2.84 %
	19	1506	1882	2080	9.52 %	1898	2180	12.94 %	1879	1951	3.69 %
	20	1476	1826	2085	12.42 %	1828	2192	16.61 %	1828	1837	0.49 %
Average			2239	2659	15.73 %	2280	2777	17.41 %	2258	2517	10.42 %

V. CONCLUSIONS

In this paper we presented a scheduling algorithm for an MPEG decoder running on MPSoC to achieve minimum latency eliminating resource contentions. The experimental results show that the proposed scheduling provides 17% of performance improvement in average compared with a system with ad-hoc scheduling. We also investigated several mapping techniques and our experimental results provide strong reasons to consider mapping and scheduling as a combined problem. In future, we see more research efforts focusing on this combined problem especially for embedded MPSoC.

REFERENCES

[1] B. Ackland; et.al, "A single Chip, 1.6-Billion, 16-MAC/s Multiprocessor DSP", *IEEE J. Solid-State Circuits*, March 2000, pp. 412-424.

[2] L. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, B. Verghese, "Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing", *Proceedings of 27th Annual International Symposium on Computer Architecture*, 2000, pp. 282-293.

[3] M. Pastrnak, P. Poplavko, P.N.H. de With, and D.S. Farin, "Data-flow timing models of dynamic multimedia pplications for multiprocessor systems," *Proceedings of 4th IEEE International Workshop on System-on-Chip for Real-Time Applications*, July 2004.

[4] Y. Cho, G. Lee, S. Yoo, K. Choi, N. Zergainoh. "Scheduling and Timing Analysis of HW/SW On-Chip Communication in MP SoC Design," *Conference and Exhibition on Design, Automation and Test in Europe*, 2003.

[5] J. M. Paul, A. Bobrek, J. E. Nelson, J. J. Pieper and D. E. Thomas, "Schedulers as Model-Based Design Elements in Programmable Heterogeneous Multiprocessors", *Design Automation Conference*, 2003.

[6] P. Yang, P. Marchal, C. Wong, S. Himpe, F. Catthoor, P. David, J. Vounckx and R. Lauwereins, "Managing dynamic concurrent tasks in embedded real-time multimedia systems", *Proceedings of the 15th international symposium on System Synthesis*, 2002.

[7] G. Varatkar and R. Marculescu, "Communication-aware task scheduling and voltage selection for total systems energy minimization," *International Conference on Computer Aided Design*, November 2003.

[8] http://bmrc.berkeley.edu/frame/research/mpeg/

[9] P. G. Paulin and J. P. Knight, "Force-directed scheduling for the behavioral synthesis of ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Volume 8, Issue 6, pp. 661-679, June 1989.

[10] Cell Broadband Engine Architecture, www.ibm.com/developerworks/power/cell , Aug. 2005.

[11] Y. Tan, P. Malani, Qinru Qiu and Q. Wu, "Workload Prediction and Dynamic Voltage Scaling for MPEG Decoding, "*Proc. of Asia and South Pacific Design Automation Conference*, Jan. 2006