Hierarchical task scheduler for interleaving subtasks on heterogeneous multiprocessor platforms

Zhe Ma[*], Francky Catthoor[†], Johan Vounckx
IMEC/DESICS
Leuven, Belgium
(Email: {mazhe, catthoor, vounckx}@imec.be)

Abstract— Nowadays, the System-on-a-chip (SoC) has integrated more processors onto a single chip. Applications are also consisting of multiple (sub)tasks that are presented as different source code which can be partly executed concurrently. However, the subtask-level parallelism inside a single task is often too limited to fully utilize all the parallel processors and results in many slacks on processors. To better use the processors, subtasks of multiple tasks will have to be executed in an interleaving fashion. This paper proposes design-time algorithms to interleave subtasks based on the separated schedules of tasks. This interleaver can be considered as part of a hierarchical scheduler to steer the code generation of very complex applications with many tasks. The scheduling experiments show that the execution time can be shortened by 20%–30% when interleaving two tasks against the sequential execution without subtask interleaving. Moreover, the differences between the solutions given by our scheduling algorithm and the optimal solutions are less than 6% for up to 20 subtasks.

I. Introduction

Today, in an up to date heterogeneous platform, usually one or more programmable components, either general-purpose or DSP processor cores or ASIP's, are all integrated into the same chip. However, existing design technologies for complex IT applications fall behind these advances in processing technology. A consistent system design technology that can cope with such characteristics and with the ever shortening time-to-market requirements is greatly needed. It should allow to map these applications cost-efficiently to the target platform while meeting all real-time and other constraints.

Recent papers[5, 8] have shown that the modern multimedia systems contain multiple concurrent tasks in the original specification. These tasks can then be further divided into subtasks by system designers (see [7]). If these subtasks are analyzed and scheduled to generate an interleaved version of source code, the processing capabilities of multiprocessor SoCs can be better utilized. Moreover, the interleaved source code generated at design-time can maximally reduce the context switch and scheduling overheads at run-time. Unfortunately, due to the NP-hardness nature of most scheduling problems, it would take tremendous amount of time to schedule a single huge subtask frame constructed by merging multiple original tasks. This makes it useful to consider a hierarchical scheduling framework. That is, the subtasks in each single task are firstly scheduled separately; then a top-level task scheduler will generate a global schedule for a cluster of single tasks based on their individual schedules. In this paper, we propose optimal and fast heuristic algorithms to schedule task clusters based on interleaving subtasks. In other words, our algorithm will take into account the situation when multiple tasks are required to run concurrently and interleave their separated subtasks schedules to generate a new united schedule. The result of our interleaving algorithm will improve the overall performance compared with the sequential case and hence provide larger room for other techniques such as Dynamic Voltage Scheduling (DVS) and Dynamic Power Management (DPM) to further reduce the global energy consumption.

The rest of this paper is organized as below: Firstly, a motivational example is given to illustrate the concept of interleaving in Section 2. Section 3 presents the interleaving algorithms. Some experimental results are shown in Section 4. Finally, the paper is concluded in Section 5.

II. Motivational example

Before the design-time scheduling, a system should be modeled as a set of subtask frames. The boundaries of subtask frames are typically given by the algorithm designers in terms of functions. A *subtask frame* is equivalent to a *task* of the conventional models, and these two terms will be used interchangeably in this paper. However, rather than a black-box model as the conventional tasks, a subtask frame contains a set of subtasks and a control-data flow graph defining the control/data dependencies among subtasks. System designers can identify these subtasks by analyzing and profiling the source code(see [7]). The subtask frame model can express the

[*]Also Ph.D. student of K.U.Leuven/ESAT, Belgium
[†]Also professor of K.U.Leuven/ESAT, Belgium

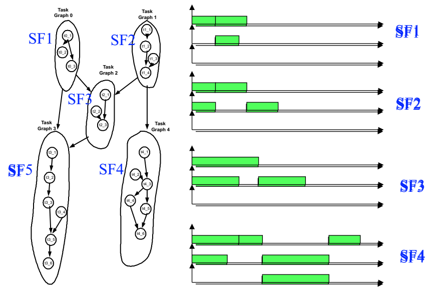

Fig. 1. Subtask frames and their schedules

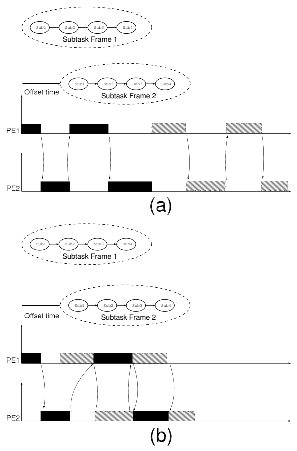

Fig. 2. Non-interleaving *vs.* interleaving

intra-task parallelism explicitly, which is crucial to obtain a high performance schedule with a low energy budget on multiprocessor platforms. Because the computation complexity is too huge, the single subtask frame scheduler has to work inside the frame, that is, this scheduler will explore most possible schedules for a subtask frame. Fig.1 illustrates the concept of subtask frames as well as the schedules which will be explored by the design-time scheduler. On the left side, each large bubble represents a subtask frame; the smaller nodes inside subtask frames are subtasks. On the right side, a number of schedules on a three processor platform are shown; each block in the schedules represents a subtask. Note that although we only show one schedule for one subtask frame in the figure, one subtask frame may have many different schedules with different makespans and energy consumptions. In general, a larger time budget will allow the system running with a schedule with lower energy consumption.

Suppose the next subtask frame can be started before the completion of the ongoing frame, the previous runtime scheduler [8] will have to delay the start of the next subtask frame till the end of the ongoing subtask frame. This is illustrated in Fig.2(a). Since a single subtask frame often does not have sufficient parallelism to fully exploit all the processing resources and therefore has many slacks in its schedules, it is beneficial to interleave two subtask frames and thereby shorten the execution time of the two frames, as indicated in Fig.2(b). The shorter execution time will then allow other subtask frames have larger time budgets and use schedules with lower energy consumptions.

III. Subtask interleaving technique

Scheduling tasks with non-uniform execution times on multiple processors is well-known for its intractability[2]. In fact, Hoogeveen *et al*[3] have proved that even for three processors, scheduling tasks with fixed processor allocations is a NP-hard problem. Still, for not too large tasks, an exact algorithm can be applied. We have developed a branching-and-bound algorithm for the interleaving problem. An outline of this branching-and-bound algorithm is given in Algo.1.

Algorithm 1 Branching-and-bound algorithm for interleaving

```
1: BnB()
2: INPUT: $status; upper\_bound$
3: OUTPUT: makespan
4: if $makespan\ of\ status > upper\_bound$ then
5:    return $makespan\ of\ status$
6: end if
7: if all subtasks are scheduled then
8:    print $status$
9:    return $makespan\ of\ status$
10: else
11:    $schedulable\_subtasks \leftarrow precedence\_free\ subtasks$
12:    for all subtask $i$ in $schedulable\_subtasks$ do
13:       $new\_status \leftarrow status$
14:       schedule subtask $i$ and update $new\_status$
15:       $makespan \leftarrow$ BnB(new_status, upper_bound)
16:       if $makespan < upper\_bound$ then
17:          $upper\_bound \leftarrow makespan$
18:       end if
19:    end for
20:    return $upper\_bound$
21: end if
```

For larger subtask frames (with more than 20 subtasks in our experiments) that are too expensive to handle by the straightforward branching-and-bound approach, a divide-and-conquer heuristic strategy can be applied to process partitions separately. That is, the frames are split into subframes such that the number of subtasks inside each individual subframe can be handled by the branching-and-bound approach. This comes at the cost of sub-optimality (see Section IV)

It is still interesting though to have fast algorithms that can handle more subtasks. Therefore an effective heuristic algorithm has been developed to interleave multiple subtask frames. This heuristic must be fast to construct a valid schedule so that the designer can evaluate multiple schedules which have been provided by preceding individual task scheduler.

We propose an interleaving heuristic based on the list scheduling algorithm[4]. The basic idea is to keep a list for each processor and sort the subtasks allocated to the corresponding list based on their priorities given by the heuristic. Then for each processor, this algorithm will scan the list from left to right. Once a scanned subtask has all of its predecessors completed, it will be added to the ready list and scheduled onto the current processor.

The heuristic calculates the priority values based on the Earliest Start Time (EST) of each subtask in the original single task schedules. We have also adapted the subtasks' priorities according to the execution times of its immediate successors and the successors of immediate successors. This adaption is to compensate the greedy behaviour of the list scheduling.

The entire heuristic algorithm is presented in Algo.2. Once the interleaved schedule is generated, we can use it to steer the code generation by using the code merging technique presented in [6]. The resulting code can then be executed on the multiprocessor platform.

Algorithm 2 Interleaving heuristic

```
INPUT: $st1, st2, ...; pre1, pre2, ....$
OUTPUT: interleaved schedule of subtasks
$time\_tracer \leftarrow 0$
while $unsched\_subtasks > 0$ do
  for all processor $i$ do
    for all subtask frame $j$ do
      if frame $j$ has unsched subtask on processor $i$ then
        if the first subtask of frame $j$ is schedulable then
          add the subtask to the ready list on the processor
        end if
      end if
    end for
    for all subtasks on the ready list do
      $priority \leftarrow EST + accu.\ exec.\ time\ of\ successors$
    end for
    $EST \leftarrow EST$ of the highest priority subtask
    if $time\_tracer < EST$ then
      $time\_tracer \leftarrow EST$
    end if
    schedule the highest priority subtask at $time\_tracer$
    update the schedule
    inform this subtask's start time to its successors
    $unsched\_subtasks \leftarrow unsched\_subtasks - 1$
  end for
end while
```

IV. Experimental results

To evaluate the effectiveness of the interleaving algorithm, we have implemented it in a prototype scheduler with the Python language and firstly conducted the experiments with a set of schedules generated randomly. We then use the Visual Texture Coding (VTC) decoder of the MPEG-4 as our real-life application for the experiment. More experiments have been carried out with the Inverse Discrete Cosine Transform(IDCT) and the Finite Impulse Response (FIR) filter code extracted from the Trimaran

		5×2	7×2	10×2
Sequential Makespan		100	100	100
H	Makespan	65.2	66.3	69.1
	Exec. time (ms)	4.0	9.2	19.8
BB	Makespan	62.6	63.3	65.9
	Exec. time (ms)	$\approx 10^4$	$\approx 10^4$	$\approx 10^7$
		15×2	17×2	20×2
Sequential Makespan		100	100	100
H	Makespan	72.5	75.2	74.0
	Exec. time (ms)	9.2	10.1	13.1
BB+	Makespan	71.6	78.2	76.1
DnC	Exec. time (ms)	$\approx 10^5$	$\approx 10^5$	$\approx 10^7$

TABLE I
Optimality comparison results

benchmarks. We have generated the interleaved source code for IDCT and FIR and simulated the interleaved execution on a heterogeneous multiprocessor simulator.

A. Evaluation of optimality for heuristic interleaving

To evaluate the optimality of the scheduling algorithms, we have tested the branching-and-bound algorithm for two subtask frames. Each subtask frame consists of 5 subtasks, 7 subtasks and 10 subtasks, respectively. For larger frames that are too expensive to handle by the straightforward branching-and-bound approach, we have applied the divide-and-conquer heuristic. Using the branching-and-bound combined with the divide-and-conquer strategy, we have conducted experiemnts for two frames, each of which contains 15, 17 and 20 subtasks, respectively. We have also tested the fast heuristic algorithm for all cases. All the scheduling experiments assume a platform with four heterogeneous processors. For each case, we have repeated the experiment for 100 times and report the average numbers. Tab.I shows the results. H, BB and BB+DnC represent the heuristic algorithm, branching-and-bound algorithm and the branching-and-bound algorithm with the divide-and-conquer strategy. For each algorithm, we report the average makespan and the execution time. Note that we have normalized the makespans. All of the experiments are measured on a Linux PC running at 1.7GHz. It is observed that when dealing with small tasks, branching-and-bound is always better than the heuristic algorithm. This is because branching-and-bound is an exact fast algorithm while heuristic is merely an approximation of the optimal result. However, for large cases, our heuristic can give better results than the branching-and-bound with the divide-and-conquer strategy because that also becomes sub-optimal. Therefore, the system-designer should consider both branching-and-bound with the divide-and-conquer and our interleaving heuristic when scheduling very large subtask frames. Since both are performed at design-time, the time overhead of applying both heuristics is acceptable.

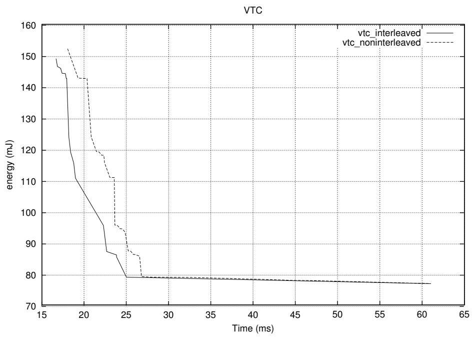

Fig. 3. VTC: interleaved vs. non-interleaved execution

		2 IDCT
System-level estimation	sequential($10^3 cycles$)	1900
	interleaved($10^3 cycles$)	1320
	improvement	30.5%
Cycle-accurate simulation	sequential($10^3 cycles$)	1876
	interleaved($10^3 cycles$)	1520
	improvement	19.0%
		IDCT+3FIR
System-level estimation	sequential($10^3 cycles$)	1700
	interleaved($10^3 cycles$)	950
	improvement	44.1%
Cycle-accurate simulation	sequential($10^3 cycles$)	1696
	interleaved($10^3 cycles$)	946
	improvement	44.2%

TABLE II
Performance improvements comparison

B. Comparisons with sequential cases

We have applied the interleaving algorithm on the schedules of the VTC decoder. We have analyzed the subtask frame model of this decoder and generated the design-time schedules(see [5] for details). In this paper, we assume that two such decoders will run concurrently for different objects. Fig.3 shows the interleaving results of two VTC decoders running concurrently at different decoding speeds. This figure clearly indicates that the interleaved execution times are on average 20% shorter than the non-interleaved execution times in the time range between 20 and 25 ms, which is the interval of the most frequently imposed speed requirements. We have also conducted more experiments with benchmarks based on IDCT and FIR. Firstly, a 8×8IDCT procedure for a QCIF image was modeled as a subtask and we have considered a subtask frame with four such IDCT subtasks. Then, we modeled a 64-tap FIR filter as a subtask and constructed a subtask frame with eight FIR subtasks. After that, we scheduled each of the two benchmarks on a four-processor platform. Finally, we have conducted interleaving for the following two scenarios: two instances of IDCT frames are scheduled to run in the interleaved way; and one instance of IDCT frame and one instance of FIR frame are scheduled to run in the interleaved way. These system-level interleaving results are shown in the Tab.II.

C. Code generation and simulation of benchmarks

In addition to the system-level scheduling, we have generated the interleaved source code for the IDCT and FIR benchmarks. The generated source code is then compiled and simulated on the multiprocessor compiler and cycle-accurate simulator environment called CRISP[1]. In our experiments, CRISP is configured to emulate a four heterogeneous VLIW processors architecture where the processors are synchronized on a shared memory. Tab.II summarizes the average performance improvements against the non-interleaved execution of those subtask frames.

V. Conclusions

This paper presents a fast scheduling technique to minimize the processor slacks for heterogeneous multiprocessor platforms. This technique can interleave multiple frames of subtasks. This interleaving results can then guide the source code interleaving of the original specification. We have implemented the technique in a prototype scheduler and conducted experiments for subtasks generated randomly, the execution time can be shortened by a factor up to 37% (Tab. I). The experiments on a MPEG-4 still image decoder show an average execution time improvement of 20% in the most used decoding speed range (Fig.3). More experiments on the benchmarks have also proved the effectiveness of our heuristic algorithm to steer the code generation for heterogeneous multiprocessor platforms.

References

[1] F. Barat, et al. Low power coarse-grained reconfigurable instruction set processor. In *3th Intl. Conf. on Field Programmable Logic and Applications*, September 2003.

[2] M. R. Garey and D. S. Johnson. *COMPUTERS AND INTRACTABILITY: A GUIDE TO THE THEORY OF NP COMPLETENESS*. W.H. Freeman and Company, New York, 1979.

[3] J. A. Hoogeveen, S. L. V. D. Velde, and B. Veltman. Complexity of scheduling multiprocessor tasks with prespecified processor allocations, 1992.

[4] T. C. Hu. Parallel sequencing and assembly line problems. *Operations Research*, 9:841–848, 1961.

[5] Z. Ma, C. Wong, F. Catthoor, et al. Task concurrency analysis and exploration of visual texture decoder on a heterogeneous platform. In *SiPS*, Seoul, Korea, August 2003.

[6] P. Marchal, , F. Catthoor, and I. Gomez. Optimizing the memory bandwidth with loop fusion. In *CODES*, 2004.

[7] S. Richard, et al. High-level data-access analysis for characterisation of (sub)task-level parallelism in java. In *9th HIPS*, April 2004.

[8] P. Yang, P. Marchal, C. Wong, et al. Managing dynamic concurrent tasks in embedded real-time multimedia systems. In *15th ISSS*, Kyoto, Japan, October 2002.