Cluster-Driven Hardware/Software Partitioning and Scheduling Approach for a Reconfigurable Computer System

Theerayod Wiangtong[1], Peter Y.K Cheung[1], and Wayne Luk[2]

[1] Department of Electrical & Electronic Engineering,
Imperial College, London, UK
{tw1,p.cheung}@ic.ac.uk
[2] Department of Computing,Imperial College, London, UK
wl@doc.ic.ac.uk

Abstract. To achieve a good performance when implementing applications in codesign systems, partitioning and scheduling are important steps. In this paper, a two-phase clustering algorithm is introduced as a preprocessing step to an existing hardware/software partitioning and scheduling system. This preprocessing step increases the granularity in the partition design, resulting in a higher degree of parallelism and a better mapping to the reconfigurable resource. This cluster-driven approach shows improvements in both the makespan of the implementation, and the CPU runtime.

1 Introduction

Coarse grain partitioning can improve the performance of an implementation by increasing parallelism as reported in [1]. It is therefore not surprising that clustering methods, which tend to increase the granularity of tasks, can be applied to the partitioning problem with good effects. Furthermore, after clustering, the size of the task graph (or problem size) is reduced, and this benefits the runtime of the synthesis process.

In this paper, we introduce an algorithm to solve a multiple-objectives clustering problem, called the *two-phase algorithm*. Our clustering algorithm is further applied as a pre-processing step to the partitioning and scheduling algorithm, which maps and schedules tasks to the target system. System constraints including FPGA resources, shared resources conflicts (such as bus or memory contention), as well as reconfiguration time, communication overhead, processing overhead are all taken into account during the partitioning and scheduling process. We employed the tabu search algorithm for partitioning and list scheduling in the scheduler as previously report in [3]. The results of clustering, mapping, and scheduling are then implemented on the UltraSONIC reconfigurable computing platform [4]. In summary, the contributions of this paper are: 1) two-phase clustering algorithm designed for multi-objectives optimization and 2) integration and evaluation of the two-phase clustering algorithm with partitioning and scheduling in codesign systems.

2 Two-Phase Clustering Algorithm

In our clustering algorithm, the objectives are set to 1) minimize total communication time and execution time of all the tasks in the DAG[1], and 2) minimize critical path of the DAG. These are subjected to constraints including a maximum cluster size, a maximum number of edges on the new cluster, and the resultant graph must be a DAG.

For such a multiple-objective problem, we need to find a method that can achieve both objectives. Based on preliminary experiments, which reveal that 1) in order to minimize delay on the critical path, tasks on the critical path itself have the most impact, and 2) once a minimum critical path is found, further clustering to reduce system cost will increase the critical path delay from its minimum value, the two-phase clustering algorithm is introduced. During the *first phase*, the delay on the critical path is minimized as much as possible, while the *second phase* is responsible for reducing the system cost without increasing the delay on the critical path.

This algorithm clusters tasks in a hierarchical manner. The first phase combines pairs of task nodes for the best fitness value while shortening the critical path delay on each refinement step. In order to prevent being trapped in a local minimum, the delay on the critical path is allowed to hill-climb. This allowance is limited by a threshold value which is decreasing in each iteration step to continuously force critical path to reduce. Two questions arise at this stage, 1) what is the decrement scheme for the threshold value, and 2) how fast is the decrement rate. Too fast a decrement leads to being trapped in a local optimum; too slow a decrement results in long search time and possible failure to find the minimum point. This is essential the idea of employing a *cooling schedule* found in simulated annealing optimization.

In the second phase, we attempt to reduce the system cost (overall communication time and computation time) without increasing the critical path value obtained in the first phase. This is achieved by only considering nodes and clusters outside the critical path as candidates for merging. The second phase is terminated when no further merging satisfies all the constraints or a maximum number of successful clustering attempts have been met.

3 Experimental Results

3.1 Comparing with Greedy Algorithm

We compare the effectiveness of the two-phase clustering algorithm described above with a straight forward greedy algorithm by applying them to randomly generated task graphs with 100, 200 and 400 tasks nodes with different granularity of tasks varying from 0.1 to 1.0. (Task granularity can be defined in many different ways. In this paper, we use the ratio of the average computation time and communication time as defined in [2].)

[1] This is referred to as the *system cost* in the rest of the paper.

Table 1. Comparisons between two-phase approach and greedy approach

Reduction of	Two-Phase algorithm	Greedy algorithm
Critical path delay	30.2%	8.6%
The system cost	14.4%	20.6%
Number of nodes	21.3%	28.1%
Number of edges	16.2%	21.3%
CPU time (PIII 866MHz)	166 sec	362 sec

Table 1 summaries the overall improvement as a result of applying the two clustering methods. From the table, the two-phase method reduces the critical path delay by an average of 30.2%, which is significantly larger than the 8.6% obtained using the greedy algorithm. Since critical path delay has a direct impact on the makespan produced after partitioning and scheduling, this is a significant and useful improvement.

3.2 Combining with the Existing Partitioning and Scheduling Program

The cluster-partition-schedule approach (called CPS) shown in Fig. 1(a) is compared with the one without clustering step (called xPS) shown in Fig. 1(b). As can be seen in Fig. 2, the CPS strategy yields better makespan (between 13% and 17% improvement) and faster runtime (by around 16%) than that without clustering.

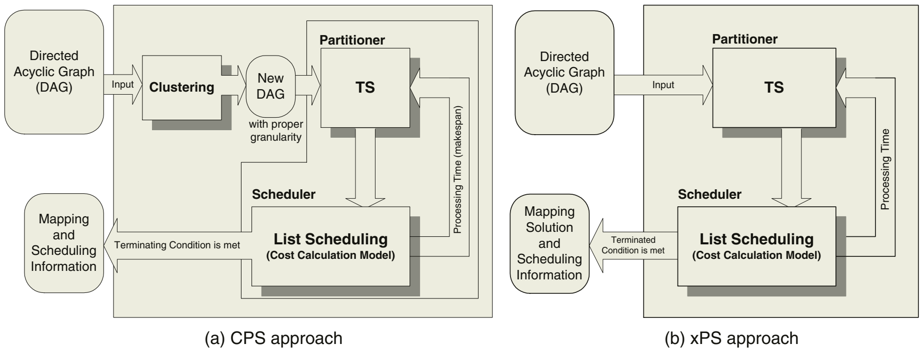

Fig. 1. Algorithm structuring of two different approaches

For a real application, the FFT algorithm is selected as a case study. For each task, the software execution time is obtained by profiling tasks on the PC, while the hardware running time and area are obtained using Xilinx development tools. Parameters such as reconfiguration time, bus speed, FPGA size, are all based on the UltraSONIC platform [4] with two reconfigurable processing elements.

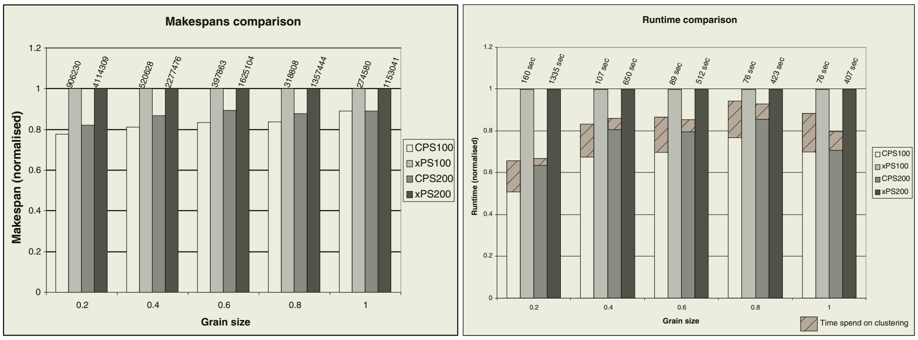

Fig. 2. The comparisons demonstrated in column graphs

Results show that our clustering algorithm can help partitioning and scheduling to reduce the overall makespan of the FFT implementation by around 14%∼15% in 8-point and 16-point FFT implementations respectively. This reduction is comparable to average values of improvement getting from random graphs as described earlier.

4 Conclusions

The two-phase clustering algorithm modifies the granularity of the tasks in the DAG in order to improve the critical path delay and the overall communication time and node computation time. The clustering algorithm presented in this paper is successfully combined with our partitioning and scheduling algorithm for mapping abstract task graphs onto a realistic reconfigurable computing system. Using our algorithm to implement the FFT algorithm onto the UltraSONIC reconfigurable computer shows promising results.

References

1. Srinivasan, V.; Govindarajan, S.; Vemuri, R., "Fine-grained and coarse-grained behavioral partitioning with effective utilization of memory and design space exploration for multi-FPGA architectures", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, pp. 140 -158, 2001.
2. Palis, M.A.; Liou, J.-C.; Wei, D.S.L., "A greedy task clustering heuristic that is provably good", *Parallel Architectures, Algorithms and Networks*, 1994.
3. Wiangtong, T.; Cheung, P.Y.K.; Luk, W., "Comparing Three Heuristic Search Methods for Functional Partitioning in HW-SW Codesign", *International Journal on Design Automation for Embedded Systems*, vol. 6, pp. 425-449, July 2002.
4. Haynes, S.D.; others, a., "UltraSONIC: A Reconfigurable Architecture for Video Image Processing", *Field-Programmable Logic and Applications (FPL)*, 2002.