Evaluation of Different Task Scheduling Policies in Multi-Core Systems with Reconfigurable Hardware

Mahyar Shahsavari*, Zaid Al-Ars*,
Koen Bertels*,[1],

* *Computer Engineering Group, Software & Computer Technology Department, Delft University of Technology, Mekelweg 4, 2628 CD, Delft, The Netherlands*

ABSTRACT

Multi-core processing technology is one of the best way of achieving high performance computing without driving up heat and power consumption. In addition, reconfigurable systems are gaining popularity due to the fact that they combine performance and flexibility. These systems allow us to have software tasks running on a General Purpose Processor (GPP) along with hardware task running on a reconfigurable fabric, such as FPGA. An important part of parallel processing in multi-core reconfigurable systems is to allocate tasks to processors to achieve the best performance. The objectives of task scheduling algorithms are to maximize system throughput by assigning a task to a proper processor, maximize resource utilization, and minimize execution time. Task execution on such platforms is managed by a scheduler that can assign tasks either to the GPPs or to the reconfigurable fabric. In this paper, we compare and evaluate different scheduling policies which have been classified into descriptive categories. The various task scheduling algorithms are discussed from different aspects, such as task dependency, static or dynamic policies, and heterogeneity of processors.

KEYWORDS: Multi-Cores; Task Scheduling Policies; Evaluation

1 Introduction

During the last decades before second millennium, shrinking the transistor size and increasing the number of transistors on a chip have responded to demand of fast and more efficient

[1]E-mail: {m.shahsavari,Z.Al-Ars,k.l.m.bertels}@tudelft.nl

processing performance. Increasing the number of transistors on a chip not only leads to more heat increament but also causes more processor power consumption. In addition, reducing transistor size is facing new challenges like physical constraints and high fabrication cost. " Multi-core chips are the biggest change in the PC programming model since Intel introduced the 32-bit 386 architecture," stated Gwennap [Gee05].

An important part of parallel processing in multi-core systems is assigning tasks to processors due to best performance. The purpose of task scheduling in multiprocessor systems are maximizing performance by assigning correct tasks to correct processors, optimizing resource application, minimizing computation time. Due to the wide variety of approaches to different policies, it is complicated to compare various systems meaningfully since there is no uniform means for evaluating them qualitatively or quantitatively. It is not also simple to build upon existing work or identify areas valuable for additional effort without some understanding of the relationships between past efforts. Classifying of approaches to the resource mapping is presented in [KA99] which try to provide a common classification mechanism which is required for addressing this problem. Whether the system is static or dynamic, multiprocessor systems are heterogeneous or homogeneous, and communication delay has considered or not the algorithms will be categorized.

Recently reconfigurable platforms are gaining popularity due to the fact that they combine performance and flexibility. Tasks implemented as a software programs, running on GPP, have the characteristics of high performance. On the other hand, tasks implemented as hardware modules placed on FPGA, demonstrate high performance, but low flexibility and high cost. Task execution on such platforms is managed by a scheduler that can allocate tasks either to the processors or to the reconfigurable platforms. In this paper, we evaluate different scheduling policies for Multiprocessors with reconfigurable hardware. The rest of the article is organized as follows. The next section presents scheduling policies. In Section 3 evaluation on the multi-core reconfigurable platform has been done. Summary and conclusions discussed at the final section.

2 Scheduling Policies

Min-min: This type of heuristic can be applied to schedule a set of tasks without dependencies onto a heterogeneous multiprocessor system. Min-min selects a task with the minimum execution time on any processor from the set U of unmapped tasks, and schedules it onto the processor on which it has the minimum completion time. The Min-min heuristic is very simple, easy to implement and it was one of the fastest algorithms compared [CJW+10].

Chaining: Chaining distributes the task graph between the processors and it does not allow duplication of the tasks. The algorithm begins with a partially scheduled task graph which is the original task graph to which two proposed tasks with zero execution time, p and q are added. Each processor starts by executing the task p and finishes by executing the proposed task q. Other tasks will be executed only once.

HLFET: Highest Level First with Estimated Times algorithm is one of the simplest list scheduling algorithm. First, we calculate the static *b-level* (bottom level) of each node. The *b-level* of a node n_i is the length of the longest path from n_i to an exit node. Then, make a ready list in a descending order of static *b-level*. Initially, the available list contains only the entry nodes. Ties are broken randomly. In third step, schedule the first node in the ready list to a processor that allows the earliest execution, using the non-insertion approach. Update

the ready list by inserting the nodes that are now ready. Go to third step until all nodes are scheduled.

ISH: The Insertion Scheduling Heuristic (ISH) algorithm, improves the HLFET algorithm by utilizing the idle time slots in the scheduling. Virtually, it uses the same approach as HLFET to make available list based on static *b-level* and schedule the first node in the ready list by using the non-insertion approach. The difference is that, once the scheduling of this node creates an idle slot, ISH checks if any task in the available list can be applied in the idle slot but cannot be scheduled earlier on the other processors. Schedule such tasks as many as possible into the idle slot.

DSH: Duplication Scheduling Heuristic (DSH) differs from the HLFET and ISH algorithms that allow no task cloning or duplication. DSH algorithm duplicates some predecessors in different processors so that each child can start as earlier as possible by eliminating communication delay. Once a node creates an idle slot, the algorithm attempts to duplicate as many predecessors as possible into the slot solely if the duplicated predecessors can improve the start time of this node.

Genetic Algorithm: Genetic Algorithm (GA) is a heuristic search technique which allows for large solution spaces to be heuristically searched in polynomial time, by applying evolutionary techniques from nature. It starts with an initial population of individuals, which can either be generated randomly or based on some other algorithm. Each individual is an encoding of a set of parameters that similarly identify a potential solution of the problem. In each generation, the population goes through the processes of crossover, mutation, fitness evaluation and selection. During crossover, parts of two individuals of the population are exchanged in order to create two entirely new individuals which replace the individuals from which they evolved. Each individual is selected for crossover with a probability of crossover rate. Mutation alters one or more genes in a chromosome with a probability of mutation rate [JST08].

Tabu Search (TS): TS is a neighborhood search technique that tries to avoid local minimal and attempts to guide the search towards a global minimum. Tabu search starts with an initial solution, which can be obtained by applying a simple one-pass heuristic, and scans the neighborhood of the current solution. For the multiprocessor task-scheduling problem, a move consists of moving a task from one processor to some other processors, or changing the order of execution of a task within the list of tasks scheduled to a processor. This technique considers all the moves in the immediate neighborhood, and accepts the move which results in the best makespan.

Simulated Annealing (SA): Simulated Annealing is a search method based on the physical process of annealing, which is the thermal process of getting low energy crystalline states of a solid. Firstly, the temperature is increased to melt the solid. If the temperature is slowly reduced, particles of the melted solid arrange themselves locally, in a stable ground state of a solid. SA theory proposes that if temperature is lowered sufficiently slowly, the solid will reach thermal counterbalance, which is an optimal state. The thermal balance is an optimal task-machine mapping (optimization goal), the temperature is the total completion time of a mapping (cost function), and the change of temperature is the process of mapping change. We use this physical based technique as a solution in task scheduling problems.

A* Search Algorithm: The A* algorithm is a best-first search algorithm, originally from the area of artificial intelligence. It is a tree search algorithm that starts with a null solution, and proceeds to a complete solution through a series of partial solutions. The root or the null solution means none of the tasks are allocated to any processor. The tree is expanded

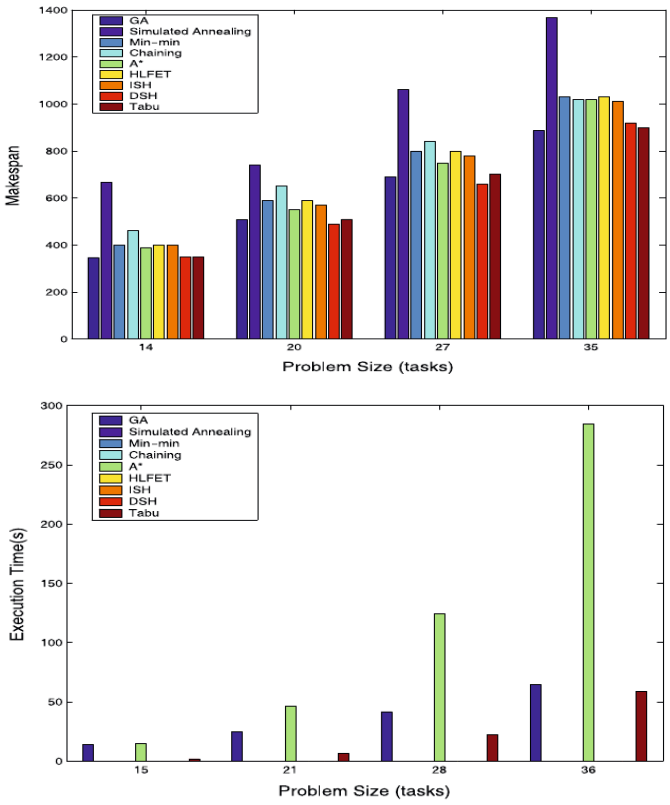

Figure 1: Makespans and execution time of variable task sizes of scheduling algorithms.

by selecting a task and allocating it to all possible processors. Each allocation is a different partial solution; therefore, each node has p children. At any node, the partial solution has one more task mapped than its parent node. The total number of nodes in the tree is limited to a predetermined constant in order to avoid an exponential execution time.

3 Evaluation

We have run all the algorithms on various sizes of factorization problem. In our simulation, we assume the number of processors is four. The computations and communication time are 40 s/task and 100 s respectively. The number of tasks that we choosed are 15, 20, 30, and 40. For GA implementation Population size, Mutation rate, and Crossover rate are 30, 0.01, and 0.7 respectively. The makespan of the obtained solutions and execution time of the algorithms are represented on Figure 1.

4 Summary

In this report, we presented a survey of algorithms for task scheduling problem. In the DAG model, a node represents an atomic program task and an edge represents the communication beside data dependency. Each node is labeled a weight to show the amount of execution time that tasks require. An edge is also labeled a weight, presents the amount of communication time required. It is notable as a comparison output that most of recent algorithm

literatures refer to list heuristic algorithms. Duplication Scheduling Heuristic (DSH) had provided short scheduling time and schedules with the shortest makespan. We conclude that DSH and ISH are the best solution, but their deployment need to be subject of a careful cost-benefit analysis. The next group, scheduling algorithms based on iterative search such as genetic algorithms, simulated annealing, tabu search, and A* require an order of longer execution time, but have better solutions with a shorter makespan. In this group, the best solutions were obtained by genetic algorithms and tabu search.

References

[CJW+10] Haijun Cao, Hai Jin, Xiaoxin Wu, Song Wu, and Xuanhua Shi. Dagmap: efficient and dependable scheduling of dag workflow job in grid. *J. Supercomput.*, 51:201–223, February 2010.

[Gee05] David Geer. Industry trends: Chip makers turn to multicore processors. *Computer*, 38:11–13, 2005.

[JST08] Shiyuan Jin, Guy A. Schiavone, and Damla Turgut. A performance study of multiprocessor task scheduling algorithms. *The Journal of Supercomputing*, 43(1):77–97, 2008.

[KA99] Yu-Kwong Kwok and Ishfaq Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv.*, 31:406–471, December 1999.