List Based Scheduling

Masrur Jamil Prochchhod
H/W Codesign
Summer Semerster 2024
Department of Electronic Engineering
Hochschule Hamm Lippstadt
masrur-jamil.prochchhod@stud.hshl.de

04 July 2024

Abstract

One of the techniques that is most helpful and flexible in organizing the execution of work over a large number of computing resources is List Scheduling. The purpose of the paper is to introduce in detail the technique of List Scheduling: computational complexities of algorithms, empirical evaluations, and applications in practice. It has also focused on the application domains that range from high-performance computing to real-time systems, cloud computing, and parallel processing. This paper will talk about a listing-based algorithm for OT, proposing innovative solutions in parallel and distributed computing paradigms, empirical assessments and some practical applications

1 Introduction

List scheduling is one of the important task scheduling algorithms, mainly in the environment of parallel and distributed computing systems. It is basically a very important strategy stitching activities into practical execution over some computing resources. The more complex and large a computing system is, the more apparent the need and importance of using the finest work scheduling techniques become. The objective of list scheduling is to maintain a priority-ordered list of tasks ready for execution and allow on-the-fly allocation of available resources, subject to the goal of maximizing throughput, reducing latency, and optimizing resource utilization.

Fundamentally, list scheduling depends on algorithms or heuristics designed to make instantaneous decisions concerning the assignment of tasks to available resources. The decisions are usually based on variables such as task dependencies, the availability of resources, execution time estimates, and system limits. In an effort to balance these somewhat competing goals, list scheduling keeps on updating and sorting its task list in reaction to the changing system state to achieve improved system performance and efficiency. Thus in this way it provides up the result that is required.

Moreover, list scheduling is applied in a lot of areas: real-time systems, cloud computing, high-performance computing—or parallel processing—and grid computing. It can be applied in a number of domains due to the reasons laid down for its scalability and agility, straight from mission-critical to industrial automation, turning to data analytics and scientific simulation ends. The paper tries to expose the reader to list scheduling approaches in as much detail as possible: theoretical underpinnings, algorithmic complexities, empirical assessments, and practical applications. The paper is intended to foster innovation in parallel and distributed computing paradigms.

2 List Scheduling Working Procedure

2.1 Tasks and Processors

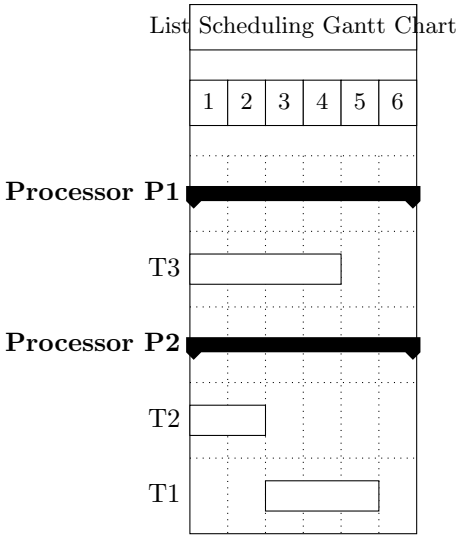
- Tasks: $T = \{T_1, T_2, T_3\}$
- Processors: $P = \{P_1, P_2\}$
- Durations: $d_1 = 3$, $d_2 = 2$, $d_3 = 4$
- Dependencies: dependencies$(T_3) = \{T_1, T_2\}$, dependencies$(T_1) = \emptyset$, dependencies$(T_2) = \emptyset$
- Priorities: priority$(T_1) = 1$, priority$(T_2) = 2$, priority$(T_3) = 3$

2.2 Scheduling Steps

1. **Initialization**:
   - Priority list $L = \{T_3, T_2, T_1\}$
   - available_time$(P_1) = 0$
   - available_time$(P_2) = 0$
2. **Schedule Tasks**:
   - **Task** T_3:
     - Calculate earliest start time:
       $$\text{earliest\_start}(T_3) = \max(\text{finish}(T_1), \text{finish}(T_2)) = 0$$
     - Assign T_3 to P_1:
       $$\text{start}(T_3) = 0, \quad \text{finish}(T_3) = 0 + 4 = 4$$
     - Update available_time$(P_1) = 4$
   - **Task** T_2:
     - Calculate earliest start time:
       $$\text{earliest\_start}(T_2) = 0$$
     - Assign T_2 to P_2:
       $$\text{start}(T_2) = 0, \quad \text{finish}(T_2) = 0 + 2 = 2$$
     - Update available_time$(P_2) = 2$
   - **Task** T_1:
     - Calculate earliest start time:
       $$\text{earliest\_start}(T_1) = 0$$
     - Assign T_1 to P_2:
       $$\text{start}(T_1) = 2, \quad \text{finish}(T_1) = 2 + 3 = 5$$
     - Update available_time$(P_2) = 5$
3. **Output Schedule**:

$$S = \{(T_3, P_1, 0, 4), (T_2, P_2, 0, 2), (T_1, P_2, 2, 5)\}$$

2.3 Gantt Chart Visualization



This Gantt chart visually represents the schedule:

- **Processor P1**: Task T3 runs from time 0 to 4.
- **Processor P2**: Task T2 runs from time 0 to 2, and Task T1 runs from time 2 to 5.

So this is how basically the list scheduling algorithms works by organizing the tasks and deadlines and everything in the perfect manner and also it shows how its necessary for the other algorithms to work. In the later part we will get to know how the algorithms work and other parts and scopes of list scheduling.

Algorithm 1 ListScheduleUsingOTs(V)

```
1: U = V − v0; F = ∅; S = {v0}
   /* initialize */
2: for each v ∈ V do
3:     schedTime[v] = 0
4: end for
   /* list schedule */
5: while U ≠ ∅ do
6:     F = {v | v ∈ U, parents(v) ⊆ S}
7:     F.sort()                    ▷ some priority function
8:     v = F.pop()
9:     t = max({schedTime[p] | p ∈ parents(v)})
10:    while DetectHazard(machineState, v, OT, t)
       do
11:        t = t + 1
12:    end while
13:    AddOperation(machineState, v, OT, t)
14:    schedTime[v] = t
15: end while
```

List Scheduling Algorithm Using OT [8]

Initialization (Lines 01-04)

- **Line 01:**
  - $U = V - v_0$: Initialize U as the set of all vertices excluding the start vertex v_0.
  - $F = \emptyset$: Initialize F as an empty set.
  - $S = \{v_0\}$: Initialize S with the start vertex v_0.
- **Lines 02-04:** Initialize the scheduled time ($schedTime$) for each task v in V to 0.

List Scheduling Loop (Lines 05-15)

- **Line 05:** While there are unscheduled tasks in U:
  - **Line 06:** Populate F with tasks from U whose dependencies (parent tasks) are already scheduled (i.e., in S).
  - **Line 07:** Sort F based on a priority function.

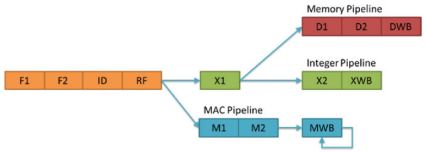


Figure 1: The Pipeline in XScale [8]

  - **Line 08:** Select and remove the highest priority task v from F.
  - **Line 09:** Set t to the maximum scheduled time of v's parent tasks, ensuring v starts after all dependencies are completed.
  - **Lines 10-12:** Check for hazards (e.g., resource conflicts) at time t and increment t until no hazards are detected.
  - **Line 13:** Schedule the operation $v.OT$ at time t.
  - **Line 14:** Record the scheduled start time t for task v.
- **Line 15:** End the scheduling loop once all tasks have been scheduled.

This algorithm ensures that tasks are scheduled in a way that respects dependencies, avoids resource conflicts, and optimizes the start times based on a defined priority function.

The different operations of the XScale processor have a complex pipeline structure, as shown in Fig. 1. The architecture introduces a number of stages for the efficient flow of instructions with a minimum number of hazards—a very important issue for list-based scheduling. Instructions are fetched in two phases: F1 and F2, decoded, ID, with the fetching of corresponding operands from the register file, RF. At this point, the pipeline then diverges into three paths: Memory Pipeline, Integer Pipeline, and MAC Pipeline. The Memory Pipeline, D1, D2, DWB, takes charge of memory access operations.The integer pipeline X1, X2, XWB deals with the handling of integer arithmetic operations. MAC Pipeline: The multiply-accumulate operations, very frequent in digital signal processing, are supported with a special

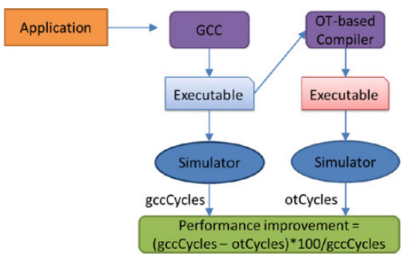


Figure 2: Experimentl Setup [8]

MAC pipeline—M1, M2, MWB. List-based scheduling algorithms identify the best ordering of instructions and time for their execution to achieve efficient processor operation by minimizing stalls and hazards

Fig. 2: Experimental setup comparing the performance of standard GNU Compiler Collection with that of OT-based compiler. This setup evaluates the impact of advanced, list-based scheduling techniques used by the OT-based compiler on general performance.

This is followed by an application, which was compiled by both GCC and the OT-based compiler separately. The compiled binaries are then run through a simulator, which measures the performance in terms of execution cycles: `gccCycles` for GCC and `otCycles` for the OT-based compiler.

Performance improvement(P.I.) is calculated using:

$$\text{P.I.} = \left(\frac{\text{gccCycles} - \text{otCycles}}{\text{gccCycles}} \right) \times 100$$

[8] This formula quantifies the reduction in cycle count, highlighting the efficiency gains from list-based scheduling. The OT-based compiler optimizes instruction scheduling, leading to fewer execution cycles and improved processor performance.

The pseudocode represents a list-based scheduling algorithm likely used by the OT-based compiler. These algorithms optimize the order and timing of operations in a pipeline by:

- Identifying operations and mapping dependencies.
- Scheduling ready operations (with resolved dependencies).
- Using a priority function for scheduling order.
- Detecting and handling hazards to ensure efficient execution.

It is the advanced scheduling techniques which make OT-based compiler basics that turn out to be more efficient executables, shown by the reduced cycle counts with respect to GCC.

These figures add weight for showing the relevance of list-based scheduling in optimizing instruction scheduling within pipelined processor architectures. Such techniques go very well with the XScale pipeline, rendering significant performance improvements with the OT-based compiler and demonstrating just that in real-world scenarios, advanced scheduling has enormous potential to strengthen modern processor efficiency and performance.

3 Promoting Innovation in Parallel and Distributed Computing

Innovation in parallel and distributed computing involves new techniques and methodologies for performance, scalability, and efficiency. The main areas in which major improvements and significant advancements can be reached is by task scheduling. A high degree of improvement in efficiency and capability in computing can be achieved by efficiently scheduling and executing tasks on a number of processors or machines. [6]

Parallel Computing Innovations

In parallel computing, a problem is solved by executing concurrently by several processors in a single machine. Resource use can be maximized while execution time can be reduced through proper task scheduling.

- **Advanced Load Balancing**: Load balancing innovations guarantee superb division of tasks on processors to ensure that not all processors are overwhelmed in work while others sit idle. This will ensure not only the maximum usage of resources but also decrease general execution time.

- **Optimized Task Scheduling**: It should ensure task scheduling using advanced list scheduling algorithms so that the dependencies and execution time for the tasks are usually considered without the occurrence of any idle time. This optimization makes sure that processors are always working on the most critical tasks, which helps in fastening the computation process.

- **Scalability Enhancements**: Scalable scheduling algorithms, in case of a system with an increased count of processors, guarantee that the additional workload of the system is treated efficiently and that performance does not degrade. This scalability is essential for a lot of applications that use high-performance computing, such as modeling climate, genomics, and real-time data analytics. [7]

Distributed Computing Innovations

- **Effective Management of Resources**: Advanced scheduling algorithms will dynamically schedule tasks to machines based upon prevailing resource availability and network conditions, all to ensure that every task is executed promptly and with the best use of resources..

- **Enhanced Fault Tolerance**: In distributed systems, it is quite common for some nodes to develop faults and also experience network problems. Advanced scheduling techniques make up fault detection and recovery mechanisms that allow the tasks redistribution from a failed node to active nodes, thereby maintaining system continuous operations and reliability.

- **Energy-Efficient Scheduling**: Due to the ever-growing need for green computing, green scheduling algorithms are being designed. Such algorithms schedule the tasks in a way that reduces energy consumption by running tasks together in periods of low demand and turning off idle machines. [2]

4 Empirical Assessments of List Scheduling Algorithms

Assessment at an empirical level is important in establishing the performance and efficiency of developed list scheduling algorithms. This includes experimentation and analysis of real-world data to couple theoretical models into practical applications.

Experimental Setup

It must have a very clear, controlled experimental setup; this includes the definition of computing environment hardware specifications, operating systems used, and configurations of software. The need to pick or generate adequate workloads that will realistically present the normal tasks. These workloads are usually in the form of synthetic benchmarks or real application traces. Some of the key performance metrics that may be targeted include execution time, resource utilization, throughput, latency, and energy consumption.

Data Collection

Data collection is done using a set of tools from within the project or as part of the operating system, including the system monitors, logging frameworks, or even simple custom scripts. Timing against the existing or alternative scheduling algorithms, such as FCFS or Round Robin, will have to be provided for reference or baseline comparison in the testing process. [9]

Performance Evaluation

- **Execution Time**: It basically measures the total time to complete all the tasks that also includes scheduling overhead.

- **Resource Utilization**: It shows the effectiveness of CPU, memories and I/O resources used by the algorithm.
- **Throughput and Latency**: It means that throughput is a measure of the number of tasks completed in a unit of time, whereas latency is a measure of time for each task to begin and end.
- **Scalability**: With the increasing workload size the evaluation of performance is done,
- **Energy Consumption**: It consumes the power and also important for energy-sensitive applications.

Comparative Analysis

This involves comparing the performance of the list scheduling algorithm with other algorithms. Statistical methods are used to ensure results are significant. [5]

5 Practical Applications of List Scheduling Algorithms

High-Performance Computing (HPC)

List scheduling algorithms are applied in high-performance computing environments for scheduling large numbers of tasks over supercomputers or clusters. Normally, the tasks include complex computation and data processing that should run efficiently to maximize resource usage and reduce execution time. [1]

Real-Time Systems

The timing constraints associated with performing tasks are stringent in the real-time system, like in embedded systems or robotics. [3] The list scheduling algorithms becomes very important in such an environment to ensure that all high-priority tasks finish before deadlines and the systems resources are effectively utilized. Such algorithms rank tasks by urgency and resource requirements to dynamically adjust the schedule for changing conditions and timely execution.

Cloud Computing

Resources, much like virtual machines or containers, are allocated dynamically in cloud computing against fluctuating workloads. The list scheduling algorithms enable efficient management of these resources by prioritizing tasks based on their priority and needs of resources. [4] This, in turn, ensures improved resource utilization, reduced operational costs, and enhanced performance.

Parallel Processing

Parallel processing is a technique whereby one large task is broken down into smaller subtasks that can then be run concurrently over many processors. That is specifically used in tasks like graphics rendering, signal processing, and machine learning, where tasks can be parallelized for faster processing times. It is through efficient scheduling of tasks that list scheduling algorithms help in maximizing the many benefits of parallel processing architectures; hence, high performance improvement.

6 Conclusion

The list scheduling algorithm is important and integral to the computing environment at large, which encompasses high-performance computing, real-time systems, cloud computing, and parallel processing. They provide efficient management of complex computations in high-performance computing. Real-time systems enjoy satisfying strict timing constraints; this is particularly useful in applications such as autonomous vehicles and industrial automation. Dynamic resource allocation in cloud computing elevates performance and cost efficiency. Parallel processing performance is boosted by ensuring better distribution of tasks, becoming important in applications such as graphics rendering and machine learning. Promotion of innovation in parallel and distributed computing is a force that drives this discipline for

progress in advanced scheduling techniques. This brings, consequently, innovations such as load balancing, optimum task scheduling, scalability, resource management, fault tolerance, energy efficiency, and so on, increasing system performance and realizing new applications. In other words, flexibility and efficiency in the management of tasks are provided by list scheduling algorithms, a requirement previous to all progress in parallel and distributed computing. Further research and development work will increase their powers, hence encouraging innovation while improving the performance and efficiency of systems.

References

[1] David A Bader. *High-Performance Computing*. CRC Press, 2012.

[2] T. D. Burd and R. W. Brodersen. Energy efficient cmos microprocessor design. In *Proceedings of the 28th Hawaii International Conference on System Sciences*. IEEE, 1995.

[3] Giorgio C Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer, 2011.

[4] Rajkumar Buyya, Chee Shin Yeo, and Srikumar Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications*. IEEE, 2008.

[5] Dror G Feitelson. Experimental analysis of the root causes of overheads in parallel job schedulers. *ACM Transactions on Parallel Computing*, 5(1):1–26, 2018.

[6] Michael J Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, C-21(9):948–960, 1972.

[7] Ian Foster. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Addison-Wesley, 1995.

[8] Jürgen Teich. *Handbook of Hardware/Software Codesign*. Springer, 2016. Code taken from page 822.

[9] Haluk Topcuoglu, Salim Hariri, and Min-You Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, 2002.

AFFIDAVIT

I, Masrur Jamil Prochchhod, hereby confirm that I have written this paper independently and have not used any sources or aids other than those indicated. All statements taken from other sources in wording or sense are clearly marked. Furthermore, I assure that this paper has not been part of a course or examination in the same or a similar version.

Masrur Jamil Prochchhod

Lippstadt, 04.07.2024