Scheduling Parallel Task Graphs

Jinzhen Wang
jwang96@charlotte.edu

Department of Computer Science
UNC Charlotte

Learning Outcomes

Lecture

At the end of this session you will know how to:

- Represent a schedule of a Parallel Task Graphs on some number of processors.
- Apply an algorithm to build a schedule.
- Interpret metrics of dependency graphs in term of parallel execution in most common scenarios.

Topics of discussion

1. Introduction to Scheduling
2. List Scheduling
3. Round-Robin Scheduling
4. Further

A FUN example of scaling

https://www.youtube.com/shorts/Rf3tFS1vNTg

Scheduling

1. Problem setup
   - **Tasks** Individual computational jobs or units of work, often represented as nodes in a task graph.
   - **Processors (Machines)** Computational units (Cores, threads, or nodes) that execute tasks in parallel.
   - **Dependencies** We discussed this previously.
   - **Objective** Minimize makespan, maximize resource utilization, or reduce communication overhead...

Scheduling

1. Problem setup
   - **Tasks** Individual computational jobs or units of work, often represented as nodes in a task graph.
   - **Processors (Machines)** Computational units (Cores, threads, or nodes) that execute tasks in parallel.
   - **Dependencies** We discussed this previously.
   - **Objective** Minimize makespan, maximize resource utilization, or reduce communication overhead...
2. Types of task scheduling problems
   - **Independent Tasks** No dependencies between tasks. Load balancing.
   - **Dependent Tasks (DAG Scheduling)** Dependencies that must be respected. Load balancing, avoid delay.
   - **Dynamic vs. Static Scheduling**

Scheduling

1. Problem setup
   - **Tasks** Individual computational jobs or units of work, often represented as nodes in a task graph.
   - **Processors (Machines)** Computational units (Cores, threads, or nodes) that execute tasks in parallel.
   - **Dependencies** We discussed this previously.
   - **Objective** Minimize makespan, maximize resource utilization, or reduce communication overhead...
2. Types of task scheduling problems
   - **Independent Tasks** No dependencies between tasks. Load balancing.
   - **Dependent Tasks (DAG Scheduling)** Dependencies that must be respected. Load balancing, avoid delay.
   - **Dynamic vs. Static Scheduling**
3. General strategies for task scheduling
   - **List Scheduling**
   - **Round-Robin Scheduling**
   - Work Stealing
   - Greedy Heuristics
   - Load Balancing Algorithms
   - Priority-Based Scheduling
   - Genetic Algorithms and Metaheuristics
   - Task Duplication Scheduling

Scheduling

Problem

- A set of tasks
- Processing times p_i
- A number of processors

A schedule

Two functions mapping

- task t to processor $\pi(t)$
- task t to a time interval $[\sigma(t); C(t)[$.
  $C(t) = \sigma(t) + p_t$
- no two tasks execute on the same processor simultaneously.
- dependencies are respected: if $x \rightarrow y$ then $\sigma(y) \geq C(x)$

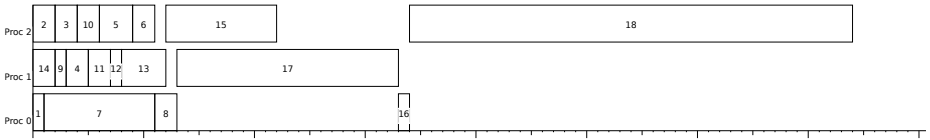
Goal

Minimize the makespan: $C_{max} = \max C(i)$

Representing Schedules

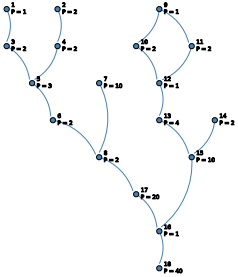
Gantt Chart

- A 2D depiction with time and processors as axes.
- makes it easy to see the makespan
- and idle times

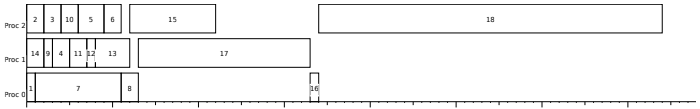


Representing Schedules

From a Parallel Task Graph and some processors



To a Gantt Chart



Scheduling optimally is hard!

NP-Completeness

- Finding the best schedule is an NP-hard problem.
- (That means, you are unlikely to find the optimal in a short amount of time)

Scheduling optimally is hard!

NP-Completeness

- Finding the best schedule is an NP-hard problem.
- (That means, you are unlikely to find the optimal in a short amount of time)

Independent tasks variants

If all tasks are independent

- still NP-Hard!

If all tasks are independent and the number of processors is fixed

- still NP-Hard!

Scheduling optimally is hard!

NP-Completeness

- Finding the best schedule is an NP-hard problem.
- (That means, you are unlikely to find the optimal in a short amount of time)

Independent tasks variants

If all tasks are independent

- still NP-Hard!

If all tasks are independent and the number of processors is fixed

- still NP-Hard!

Variants

If all tasks have $p_i = 1$ (sometimes called UET)

- still NP-Hard!

If all tasks have $p_i = 1$, and makespan is 3

- still NP-Hard!

If all tasks have $p_i = 1$, and makespan is 3, and the graph is bipartite

- still NP-Hard!

Scheduling optimally is hard!

NP-Completeness

- Finding the best schedule is an NP-hard problem.
- (That means, you are unlikely to find the optimal in a short amount of time)

Independent tasks variants

If all tasks are independent

- still NP-Hard!

If all tasks are independent and the number of processors is fixed

- still NP-Hard!

Variants

If all tasks have $p_i = 1$ (sometimes called UET)

- still NP-Hard!

If all tasks have $p_i = 1$, and makespan is 3

- still NP-Hard!

If all tasks have $p_i = 1$, and makespan is 3, and the graph is bipartite

- still NP-Hard!

Scheduling is HARD!

List Scheduling

List scheduling is a simple and common heuristic, particularly for independent tasks and DAGs.

- Maintain a list of tasks that are ready to be scheduled (either no dependencies or predecessors have completed).
- Assign tasks to the least-loaded processor based on some priority, such as task size or the critical path.
- Example algorithms:
  - Longest Processing Time First (LPT): Tasks are scheduled in order of decreasing task length to balance the load across processors.
  - Critical Path Scheduling (CP): Tasks along the critical path are prioritized to minimize the makespan.

List Scheduling for independent tasks is a 2-approximation algorithm

2-approximate algorithm

guaranteeing the solution it produces will be at most twice as bad as the optimal solution
Key Definitions:

- Makespan: C_{LS}
- Optimal Makespan: C_{OPT}

An algorithm is a 2-approximation if:

- for minimization problem: $C_{LS} \leq 2 \times C_{OPT}$
- for maximization problem: $C_{LS} \geq \frac{1}{2} \times C_{OPT}$

List Scheduling for independent tasks is a 2-approximation algorithm

2-approximate algorithm

guaranteeing the solution it produces will be at most twice as bad as the optimal solution
Key Definitions:

- Makespan: C_{LS}
- Optimal Makespan: C_{OPT}

An algorithm is a 2-approximation if:

- for minimization problem: $C_{LS} \leq 2 \times C_{OPT}$
- for maximization problem: $C_{LS} \geq \frac{1}{2} \times C_{OPT}$

List scheduling as a greedy algorithm

- Consider the tasks in any (unspecified) order.
- Pick a task and schedule it as early as possible on the first available processor.

List Scheduling for independent tasks is a 2-approximation algorithm

2-approximate algorithm

guaranteeing the solution it produces will be at most twice as bad as the optimal solution
Key Definitions:

- Makespan: C_{LS}
- Optimal Makespan: C_{OPT}

An algorithm is a 2-approximation if:

- for minimization problem: $C_{LS} \leq 2 \times C_{OPT}$
- for maximization problem: $C_{LS} \geq \frac{1}{2} \times C_{OPT}$

List scheduling as a greedy algorithm

- Consider the tasks in any (unspecified) order.
- Pick a task and schedule it as early as possible on the first available processor.

Lower Bound for the Optimal Makespan

- $L = \sum_{i=1}^{n} p_i$
- $C_{OPT} \geq \frac{L}{m}$
- $C_{OPT} \geq max(p_i)$
- $C_{OPT} \geq max(\frac{L}{m}, max(p_i))$

List Scheduling for independent tasks is a 2-approximation algorithm

2-approximate algorithm

guaranteeing the solution it produces will be at most twice as bad as the optimal solution
Key Definitions:

- Makespan: C_{LS}
- Optimal Makespan: C_{OPT}

An algorithm is a 2-approximation if:

- for minimization problem: $C_{LS} \leq 2 \times C_{OPT}$
- for maximization problem: $C_{LS} \geq \frac{1}{2} \times C_{OPT}$

List scheduling as a greedy algorithm

- Consider the tasks in any (unspecified) order.
- Pick a task and schedule it as early as possible on the first available processor.

Lower Bound for the Optimal Makespan

- $L = \sum_{i=1}^{n} p_i$
- $C_{OPT} \geq \frac{L}{m}$
- $C_{OPT} \geq max(p_i)$
- $C_{OPT} \geq max(\frac{L}{m}, max(p_i))$

Worst-Case Behavior of List Scheduling

- $C_{LS} \leq max(p_i)$
- $C_{LS} \leq \frac{L}{m} + max(p_i)$

List Scheduling for independent tasks is a 2-approximation algorithm

2-approximate algorithm

guaranteeing the solution it produces will be at most twice as bad as the optimal solution
Key Definitions:

- Makespan: C_{LS}
- Optimal Makespan: C_{OPT}

An algorithm is a 2-approximation if:

- for minimization problem: $C_{LS} \leq 2 \times C_{OPT}$
- for maximization problem: $C_{LS} \geq \frac{1}{2} \times C_{OPT}$

List scheduling as a greedy algorithm

- Consider the tasks in any (unspecified) order.
- Pick a task and schedule it as early as possible on the first available processor.

Lower Bound for the Optimal Makespan

- $L = \sum_{i=1}^{n} p_i$
- $C_{OPT} \geq \frac{L}{m}$
- $C_{OPT} \geq max(p_i)$
- $C_{OPT} \geq max(\frac{L}{m}, max(p_i))$

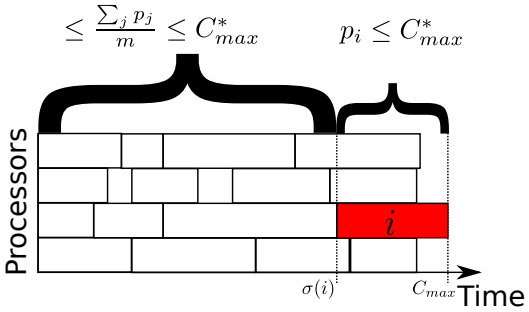
Worst-Case Behavior of List Scheduling

- $C_{LS} \leq max(p_i)$
- $C_{LS} \leq \frac{L}{m} + max(p_i)$

Combining the Bounds

- $C_{LS} \leq 2 \times C_{OPT}$

List Scheduling for independent tasks is a 2-approximation algorithm

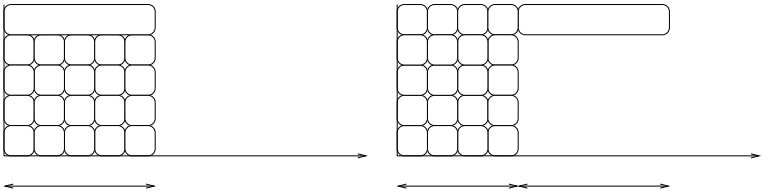


How bad can List Scheduling really be ?

The idea

List Scheduling says "Given any order of the task". Let us use it to make it the worse possible.

Counter example



List Scheduling is a $(2 - \frac{1}{m})$-approximation algorithm and a counter example reaching this bound exists. The bound is said to be tight.

Is this the best algorithm ?

Is there a fast algorithm with guaranteed performance better than 2?

Largest Processing Time first

The idea

List Scheduling reaches a ratio of 2 because the last task is the largest one.

Algorithm

Sort task by non-increasing order of processing times. Use List Scheduling. Complexity $O(n \log n + nm)$.

Approximation ratio of LPT

- If the most loaded machine has one task : $C_{max} = p_{max}$ is optimal.
- If the most loaded machine has two tasks : one can show the mapping is optimal
- If the most loaded machine has k tasks : the imbalance is less than $\frac{C_{max}}{k}$ and the ratio is $\frac{k+1}{k}$.

Theorem

LPT is a $(\frac{4}{3} - \frac{1}{3m})$-approximation algorithm and the bound is tight.

List Scheduling for DAG is a 2-approximation algorithm

Algorithm

For DAGs, list scheduling is similar to the one for independent tasks.

- Pick the earliest available task.
- Schedule it on a free processor as early as possible.

Key Definitions:

- Makespan: C_{LS}
- Optimal Makespan: C_{OPT}
- Critical Path: C_{CP}

List Scheduling for DAG is a 2-approximation algorithm

Algorithm

For DAGs, list scheduling is similar to the one for independent tasks.

- Pick the earliest available task.
- Schedule it on a free processor as early as possible.

Key Definitions:

- Makespan: C_{LS}
- Optimal Makespan: C_{OPT}
- Critical Path: C_{CP}

List scheduling for DAG

- Only tasks whose dependencies are fully satisfied can be scheduled.
- Among those ready tasks, the next available task is assigned to the least-loaded processor, just like with independent tasks.

Lower Bound for the Optimal Makespan

- $C_{OPT} \geq max(\frac{L}{m}, C_{CP})$

Worst-Case Behavior of List Scheduling

- $C_{LS} \geq C_{CP}$
- $C_{LS} \leq C_{CP} + \frac{L}{m}$

Combining the Bounds

- $C_{LS} \leq C_{CP} + \frac{L}{m}$
- $C_{OPT} \geq max(\frac{L}{m}, C_{CP})$
- $C_{LS} \leq 2 \times C_{OPT}$

Scheduling: Summary

Scheduling

- Finding for each task a processor for it to run on.
- Finding for each task a time interval for it to run on.

Finding the best schedule is hard! (even in sub-cases.)

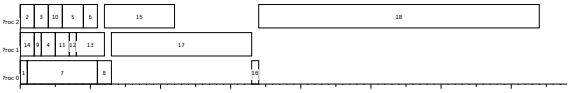
Scheduling: Summary

Scheduling

- Finding for each task a processor for it to run on.
- Finding for each task a time interval for it to run on.

Finding the best schedule is hard! (even in sub-cases.)

Gantt Chart



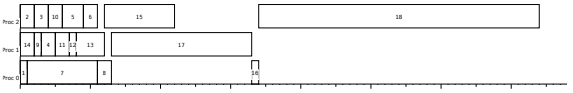
Scheduling: Summary

Scheduling

- Finding for each task a processor for it to run on.
- Finding for each task a time interval for it to run on.

Finding the best schedule is hard! (even in sub-cases.)

Gantt Chart



Three heuristic algorithms

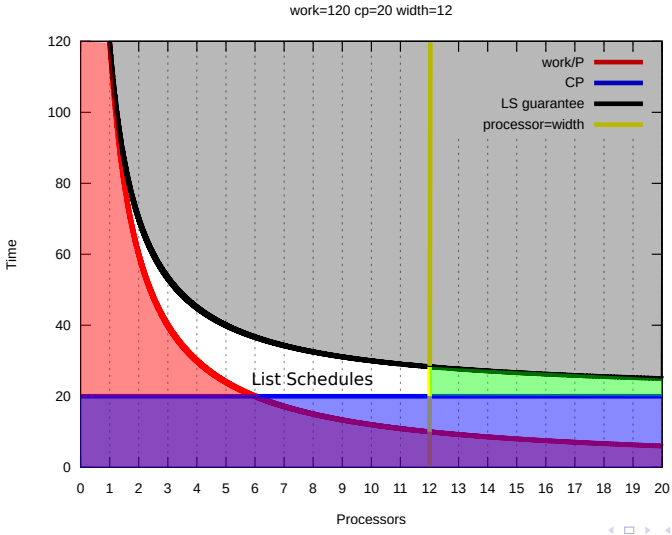
For independent tasks:

- List Scheduling
  - Pick any available task
  - Schedule it ASAP
- LPT
  - Sort tasks by decreasing processing time
  - Then List Scheduling

For DAGs:

- List Scheduling
  - Pick the earliest available task
  - Schedule it ASAP
  - Guarantees $CP + \frac{work}{m}$

Interpreting all that : aka "What will I get in practice?"



Round-Robin Scheduling

- In Round-Robin Scheduling, tasks are assigned to processors in a **cyclic manner**, without considering the specific workload of each task.
- It's a simple but effective strategy when tasks are roughly of the same size and the system has little communication overhead. For example:
  - Processor 1 gets Task 1, Processor 2 gets Task 2, and so on, until all tasks are assigned.
- Round-robin scheduling works well for homogeneous tasks but may lead to load imbalances when tasks are of varying sizes.

External

Textbook:

- Chapter 2 to 5.1 of Oliver Sinnen. Task Scheduling for Parallel Systems. John Wiley & Sons, Inc. 2007. Access it through the library: https://librarylink.uncc.edu/login?url=https://onlinelibrary.wiley.com/doi/book/10.1002/0470121173

Cilk on graphs metrics:

- The Cilkview Scalability Analyzer, SPAA 2010.http://web.mit.edu/willtor/www/res/cilkview-spaa-10.pdf. a paper describing parallel application as a DAG and metrics.

Conflict graph and coloring:

- Conflict graphs: http://math.cmu.edu/~bkell/21110-2010s/conflict-graphs.html
- A. H Gebremedhin, F. Manne, Alex Pothen. What Color Is Your Jacobian? Graph Coloring for Computing Derivatives. Siam Review 2005.
- M. Deveci, E. Boman, K. Devine, and S. Rajamanickam. Parallel Graph Coloring for Manycore Architectures. IPDPS 2016.

Scheduling:

- A taxonomy of scheduling problems: Srishti Srivastava and Ioana Banicescu. Scheduling in Parallel and Distributed Computing Systems. Chapter 11 of Prasad, Gupta, Rosenberg, Sussman, and Weems. Topics in Parallel and Distributed Computing: Enhancing the Undergraduate Curriculum: Performance, Concurrency, and Programming on Modern Platforms, Springer International Publishing, 2018. https://grid.cs.gsu.edu/~tcpp/curriculum/?q=system/files/Ch11_4.pdf
- Scheduling is NP-Hard: M. Garey and D. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman. 1979.
- LS for independent tasks: R. Graham. Bounds for certain multiprocessing anomalies. Bell System Technical Journal. 1966
- LPT and LS with precedence: R. Graham. Bounds on Multiprocessing Timing Anomalies. SIAM Journal on Applied Mathematics. 1969.
- Chapter 1 and 7 of. H. Casanova, A. Legrand, Y. Robert. Parallel Algorithms, CRC Press. 2008