Multiprocessor Scheduling IV: (A Note on) Parallelizations

Jian-Jia Chen

TU Dortmund

06, July, 2015

Parallelizations with DAG

Outline

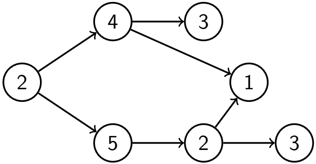
Parallelizations with DAG

Needs for Parallelizations

- To fully utilize the multiprocessor systems, a task should be able to be executed on more than one processor
- We have up to now only consider *sequential executions* of a task
- If we allow parallelizations, how should the model be looked like?

Represented by Directed Acyclic Graphs (DAG)

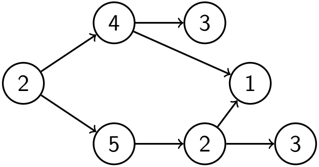
- Each task τ_i is a sporadic task:
  - period T_i
  - relative deadline D_i
- Each task is characterized by a directed acyclic graph (DAG)
  - Each task has multiple subtasks (represented by vertices here)
  - The number in each node is the worst-case execution time
  - The precedence constraints (the directed edged) represent the dependency of the subtasks
  - The acyclic property ensures that there is no cycle in the graph



Essentials Based on DAG structures

Based on the DAG structure of a task τ_i

- C_i: the overall worst-case execution time (20 in this example)
- Ψ_i: the critical-path (one of the longest paths) worst-case execution time (12 in this example)
- U_i: the utilization, defined as $\frac{C_i}{T_i}$



Scheduling Theory about This

- If the system has only one task, represented by a DAG, Graham studied this problem in 1966 under this notation $P|prec|Cmax$
- The algorithm is called *list scheduling*
  - If one of the M processors is idle, schedule one of the ready subtasks to the idle processor.
- The algorithm is widely used for many applications.
  - The order of the subtasks can be tuned
  - Graham showed that list scheduling has an approximation factor $2 - \frac{1}{M}$ with respect to minimizing the makespan.

An Informal Proof of List Scheduling

- Let ℓ be the subtask that finishes the last. Let $\ell - 1$ be the last-finished predecessor of ℓ
- We construct a series of subtasks preceding each other, starting at 1 (which has no predecessor)
- Let's now call this path Π. Clearly the length of Π is $\leq \Psi$.
- Let the starting time of the i-th subtask in Π be t_i.
- In list scheduling, the finishing time of i-th subtask in Π is then $f_i = t_i + c_i$
  - c_i is the (worst-case) execution time of the i-th subtask in Π.
- *Important observation*: between f_i and t_{i+1}, all the M processors must be busy for executing other subtasks
  - otherwise, the $(i + 1)$-th subtask in Π should have been executed earlier than t_{i+1}.
- Therefore, we know that the finishing time is at most $2 - \frac{1}{M}$ times the optimal makespan (denoted by $\mathrm{C_{max}}^{\mathrm{opt}}$)

$$\Psi + \frac{C - \Psi}{M} \leq (2 - \frac{1}{M})C_{max}^{opt}.$$

Implicit-Deadline Tasks with Global RM Scheduling

For all $0 < t \leq T_k$

$$W_k(t) = \sum_{i=1}^{k-1} (\left\lceil \frac{t}{T_i} \right\rceil - 1) C_i + 2C_i.$$

This implies that we just greedily take a head job immediately.
Clearly, lower-priority jobs have no effect for the unschedulability or schedulability.

Theorem

A system $\mathcal{T}$ of implicit-deadline periodic, independent, preemptable DAG tasks is schedulable by Global-RM on M processors if

$$\forall \tau_k \in \mathcal{T} \; \exists t \text{ with } 0 < t \leq T_k \;\; and \;\; \Psi_k + \frac{C_k - \Psi_k}{M} + \frac{W_k(t)}{M} \leq t$$

holds.

Recall: Capacity Augmentation Bound

Given a task set $\mathcal{T}$ with total utilization of $U_{\sum}$, a scheduling algorithm $\mathcal{A}$ with capacity augmentation bound b can always schedule this task set on M processors of speed b as long as $\mathcal{T}$ satisfies the following conditions:

$$\text{Utilization does not exceed total cores, } \sum_{\tau_i \in \mathcal{T}} U_i \leq M \qquad (1)$$

$$\text{For each task } \tau_i \in \mathcal{T}, \text{the critical path utilization } \frac{\Psi_i}{T_i} \leq 1 \qquad (2)$$

Recall: Capacity Augmentation Bound

Given a task set $\mathcal{T}$ with total utilization of $U_{\sum}$, a scheduling algorithm $\mathcal{A}$ with capacity augmentation bound b can always schedule this task set on M processors of speed b as long as $\mathcal{T}$ satisfies the following conditions:

$$\text{Utilization does not exceed total cores, } \sum_{\tau_i \in \mathcal{T}} U_i \leq M \tag{1}$$

$$\text{For each task } \tau_i \in \mathcal{T}, \text{the critical path utilization } \frac{\Psi_i}{T_i} \leq 1 \tag{2}$$

This means that the algorithm guarantees the schedulability if the following conditions are satisfied:

$$\text{Utilization does not exceed total cores, } \sum_{\tau_i \in \mathcal{T}} U_i \leq \frac{M}{b} \tag{3}$$

$$\text{For each task } \tau_i \in \mathcal{T}, \text{the critical path utilization } \frac{\Psi_i}{T_i} \leq \frac{1}{b} \tag{4}$$

Capacity Augmentation Bound of Global RM

The task set is schedulable under Global RM if

$$\forall k, \left(2 + \frac{\Psi_k}{T_k} + \frac{C_k - \Psi_k}{MT_k}\right) \Pi_{i=1}^{k-1}(U_i/M + 1) \leq 3. \tag{5}$$

$$\Rightarrow \left(2 + \frac{\Psi_k}{T_k}\right) \Pi_{i=1}^{k}(U_i/M + 1) \leq 3. \tag{6}$$

$$\Rightarrow \left(2 + \frac{1}{b}\right) \left(\frac{1}{(k-1)b} + 1\right)^{k-1} \leq 3. \tag{7}$$

$$\Rightarrow \left(2 + \frac{1}{b}\right) e^{1/b} \leq 3. \tag{8}$$

Again, we use the worst cases by setting all the tasks with the same utilization as we did in the analysis for uniprocessor systems. This concludes that $b \geq 3.6215$ enforces the above inequality.

A Short Summary about Global DAG Scheduling

Speedup factors

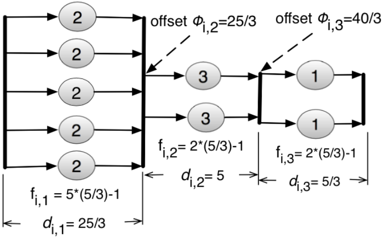
	implicit deadlines	constrained deadlines	arbitrary deadlines
Global EDF	$2 - \frac{1}{M}$ (Bonifaci et al. ECRTS 2013)		
Global DM	$3 - \frac{1}{M}$ (Bonifaci et al. ECRTS 2013)		

Capacity augmentation factors

	implicit deadlines	constrained deadlines	arbitrary deadlines
Global EDF	$\frac{2+\sqrt{5}}{2} \approx 2.6181$ (Li, Chen et al. 2014)	unknown	unknown
Global DM	3.6215 (Chen et al. 2015)	unknown	unknown

How about Partitioned Scheduling?

- Each subtask should be assigned on one processor
- Different subtasks can be assigned on different processors
- For each subtask of task τ_i
  - specify its offset to start with
  - specify its relative deadline after the offset
  - perform timing control



Saifullah et al.: With a proper assignment of relative deadlines and offsets, speedup factor 5 can be achieved by using partitioned EDF.

Abusayeed Saifullah et al. “Multi-core Real-Time Scheduling for Generalized Parallel Task Models”. RTSS 2011

Can We Improve It?

- A simple partitioned strategy can work as well
  - If a task τ_i is with $\frac{C_i}{T_i} \geq 1$, we use list scheduling by *dedicating some processors* to this task τ_i. Such a task is a *heavy* task.
  - If a task τ_i is with $\frac{C_i}{T_i} < 1$, we do not consider to run this task on more than one processor. Such a task is a *light* task.
- Let's use List Scheduling to schedule the heavy tasks.
- Let's use LUF^+ (largest utilization first for bin packing) to pack these light tasks on the remaining processors based on partitioned EDF.
- M_{light}: the number of processors used for the light tasks
- M_{heavy}: the number of processors used for the heavy tasks
- If there is no heavy task, this is identical to partition the given periodic tasks without any intra-task parallelization
- If there is a heavy task, it is easy to argue $M_{light} + M_{heavy} \leq 2\sum_{\tau_i} U_i$ under the assumption $\frac{\Psi_i}{T_i} \leq 0.5$ for every task τ_i