Multiprocessor Scheduling IV: (A Note on) Parallelizations

Jian-Jia Chen

TU Dortmund

11, July, 2016
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 $\tau_i$ 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 edges) represent the dependency of the subtasks
  - The acyclic property ensures that there is no cycle in the graph
Based on the DAG structure of a task $\tau_i$

- $C_i$: the overall worst-case execution time (20 in this example)
- $\Psi_i$: 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 $P|prec|C_{max}$

- 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 $\ell$ be the subtask that finishes the last. Let $\ell - 1$ be the last-finished predecessor of $\ell$
- We construct a series of subtasks preceding each other, starting at 1 (which has no predecessor)
- Let's now call this path $\Pi$. Clearly the length of $\Pi$ is $\leq \Psi$.
- Let the starting time of the $i$-th subtask in $\Pi$ be $t_i$.
- In list scheduling, the finishing time of $i$-th subtask in $\Pi$ is then $f_i = t_i + c_i$
  - $c_i$ is the (worst-case) execution time of the $i$-th subtask in $\Pi$.
- **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 $\Pi$ 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 $C_{\text{max}}^{\text{opt}}$)

$$
\Psi + \frac{C - \Psi}{M} \leq (2 - \frac{1}{M})C_{\text{max}}^{\text{opt}}.
$$
Implicit-Deadline Tasks with Global RM Scheduling

For all $0 < t \leq T_k$

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

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

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

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:

1. Utilization does not exceed total cores, \[ \sum_{\tau_i \in \mathcal{T}} U_i \leq M \] (1)

2. For each task $\tau_i \in \mathcal{T}$, the critical path utilization $\frac{\Psi_i}{T_i} \leq 1$ (2)
Recall: Capacity Augmentation Bound

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

Utilization does not exceed total cores,
\[
\sum_{\tau_i \in T} U_i \leq M \quad (1)
\]

For each task \( \tau_i \in T \), the critical path utilization
\[
\frac{\Psi_i}{T_i} \leq 1 \quad (2)
\]

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

Utilization does not exceed total cores,
\[
\sum_{\tau_i \in T} U_i \leq \frac{M}{b} \quad (3)
\]

For each task \( \tau_i \in T \), the critical path utilization
\[
\frac{\Psi_i}{T_i} \leq \frac{1}{b} \quad (4)
\]
The task set is schedulable under Global RM if

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

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

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

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

<table>
<thead>
<tr>
<th></th>
<th>implicit deadlines</th>
<th>constrained deadlines</th>
<th>arbitrary deadlines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global EDF</td>
<td>$2 - \frac{1}{M}$ (Bonifaci et al. ECRTS 2013)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Global DM</td>
<td>$3 - \frac{1}{M}$ (Bonifaci et al. ECRTS 2013)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Capacity augmentation factors

<table>
<thead>
<tr>
<th></th>
<th>implicit deadlines</th>
<th>constrained deadlines</th>
<th>arbitrary deadlines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global EDF</td>
<td>$\frac{2+\sqrt{5}}{2} \approx 2.6181$ (Li, Chen et al. 2014)</td>
<td>unknown</td>
<td>unknown</td>
</tr>
<tr>
<td>Global DM</td>
<td>3.6215 (Chen et al. 2015)</td>
<td>unknown</td>
<td>unknown</td>
</tr>
</tbody>
</table>
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 \( \tau_i \)
  - 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.

A simple partitioned strategy can work as well

- If a task $\tau_i$ is with $\frac{C_i}{T_i} \geq 1$, we use list scheduling by dedicating some processors to this task $\tau_i$. Such a task is a heavy task.
- If a task $\tau_i$ 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.

- $M_{\text{light}}$: the number of processors used for the light tasks
- $M_{\text{heavy}}$: 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 $M_{\text{light}} + M_{\text{heavy}} \leq 2 \sum \tau_i U_i$ under the assumption $\frac{\psi_i}{T_i} \leq 0.5$ for every task $\tau_i$. 
### Speedup factors

<table>
<thead>
<tr>
<th></th>
<th>implicit deadlines</th>
<th>constrained deadlines</th>
<th>arbitrary deadlines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partitioned EDF</td>
<td>2 (Li et al. ECRTS 2014)</td>
<td>??</td>
<td>??</td>
</tr>
<tr>
<td>Partitioned DM</td>
<td>2 (Conjecture by JJ)</td>
<td>??</td>
<td>??</td>
</tr>
</tbody>
</table>
Federated Scheduling: Limits

- Baruah extended federated scheduling for constrained- and arbitrary-deadline systems
  - S. Baruah. The federated scheduling of constrained-deadline sporadic DAG task systems. In DATE 2015.

The following example shows that federated scheduling can be very bad

Table: 10 tasks on \( M = 10 \) processors.

Federated Scheduling: Limits

• Baruah extended federated scheduling for constrained- and arbitrary-deadline systems
  
  • S. Baruah. The federated scheduling of constrained-deadline sporadic DAG task systems. In DATE 2015.
  • S. Baruah. The federated scheduling of systems of conditional sporadic DAG tasks. In EMSOFT, 2015.

• The following example shows that federated scheduling can be very bad

<table>
<thead>
<tr>
<th></th>
<th>$\tau_1$</th>
<th>$\tau_2$</th>
<th>$\tau_3$</th>
<th>$\tau_4$</th>
<th>$\tau_5$</th>
<th>$\tau_6$</th>
<th>$\tau_7$</th>
<th>$\tau_9$</th>
<th>$\tau_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_i$</td>
<td>10</td>
<td>10</td>
<td>20</td>
<td>40</td>
<td>80</td>
<td>160</td>
<td>320</td>
<td>640</td>
<td>1280</td>
</tr>
<tr>
<td>$D_i$</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
<td>128</td>
<td>256</td>
</tr>
<tr>
<td>$T_i$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$\infty$</td>
</tr>
</tbody>
</table>

**Table**: 10 tasks on $M = 10$ processors.
Federated Scheduling: Limits

- Baruah extended federated scheduling for constrained- and arbitrary-deadline systems
  - S. Baruah. The federated scheduling of constrained-deadline sporadic DAG task systems. In DATE 2015.
- The following example shows that federated scheduling can be very bad

<table>
<thead>
<tr>
<th></th>
<th>$\tau_1$</th>
<th>$\tau_2$</th>
<th>$\tau_3$</th>
<th>$\tau_4$</th>
<th>$\tau_5$</th>
<th>$\tau_6$</th>
<th>$\tau_7$</th>
<th>$\tau_9$</th>
<th>$\tau_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_i$</td>
<td>10</td>
<td>10</td>
<td>20</td>
<td>40</td>
<td>80</td>
<td>160</td>
<td>320</td>
<td>640</td>
<td>1280</td>
</tr>
<tr>
<td>$D_i$</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
<td>128</td>
<td>256</td>
</tr>
<tr>
<td>$T_i$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$\infty$</td>
</tr>
</tbody>
</table>

Table: 10 tasks on $M = 10$ processors.