Schedulability analysis of limited-preemptive moldable gang tasks

Joan Marcè i Igual
TU Delft

Daily Supervisor
Geoffrey Nelissen
Co-supervisor
Mitra Nasri

27th of August, 2020
Real-time systems

- Systems of which correctness does depend not only on **logical** results but also on **timing constraints**
- Higher demand for computation resources
  - Multicore systems
  - GPUs
  - FPGAs
What is gang?

Parallel threads together as a “gang”

Execution does not start until there are enough cores

Global scheduling

Gang Scheduling

Variability
Why gang?

- More efficient synchronization
- Reduces variability in the execution
- Avoids overhead when loading initial data
- Shows its full potential when executed non-preemptively
Types of gang

- **Rigid**: number of cores set by programmer
- **Moldable**: number of cores assigned when job is dispatched
- **Malleable**: number of cores can change during runtime

![Diagram showing the calculation of 2 * 10 + 1 * 10 = 30]
Bundled scheduling\cite{1} vs limited-preemptive
Bundled scheduling\textsuperscript{[1]} vs limited-preemptive

- Rigid gang reserves the whole block
- Bundled creates \textit{rigid blocks} with dependencies
- Limited-Preemptive creates \textit{moldable blocks} with dependencies
Job-level fixed-priority scheduling (JLFP) for gang

- Based on global JLFP scheduler
- Assigns maximum possible cores to a job
- We will solve this issue

<table>
<thead>
<tr>
<th></th>
<th>Priority</th>
<th>Min cores</th>
<th>Max cores</th>
<th>Deadline</th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_0$</td>
<td>High</td>
<td>1</td>
<td>3</td>
<td>20</td>
<td>30, 15, 10</td>
</tr>
<tr>
<td>$J_1$</td>
<td>Low</td>
<td>2</td>
<td>2</td>
<td>20</td>
<td>15</td>
</tr>
</tbody>
</table>
Previous work

Introduced in high-performance computing in 1982

Preemptive solutions

Schedulability tests
- Job-level fixed-priority
- Earliest deadline first

Schedulers
- Optimal for rigid gang (DP-Fair)
- Moldable gang

Non-preemptive solutions

Schedulability tests
- Earliest deadline first for rigid gang

References:
[2] Goossens et al., 2010
[5] Berten et al., 2011
Our work
Project goals

1. Design an accurate schedulability analysis for limited-preemptive moldable gang tasks

2. Evaluate the impact of the level of parallelism assigned to a job

3. Propose a new scheduling algorithm to improve the schedulability of limited-preemptive moldable gang tasks
Agenda

• Gang schedulability analysis
• New scheduling policy
Schedule abstraction graph

- It is a technique that allows:
  - Search for all possible schedules
  - Aggregate “similar” schedules

Every edge is a different scheduler decision

Every node is a different system state
Using SAG for gang scheduling

• Introduce new system state representation

• Introduce new expansion rules

• Introduce new merge rules
New system state representation

• Availabilities of cores → Know how many cores we have at time $t$
• Simultaneous groups of cores → Multiple cores released simultaneously
• Set of certainly running jobs → Jobs waiting for their predecessors

Time at which 3 cores are **possibly** available

<table>
<thead>
<tr>
<th>Cores available</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time at which 3 cores are <strong>certainly</strong> available</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Releases all cores simultaneously

$J$
New SAG expansion rules

- Previously a state was created for every dispatchable job
- Now a state is created for every job and possible number of cores
New SAG expansion rules

• Exploring more states is
  👍 Sound (does not accept unschedulable task sets)
  👎 Slower and more pessimistic

• Purge states by checking that
  ✔ • Enough cores available
  ✔ • More cores not available
  ✔ • Precedence constraints are respected

Try to minimize number of states that represent an impossible scenario

J₀ is predecessor of J₁
J₁ has higher priority

J₀ has higher priority
J₁ has higher priority
New SAG merge rules

- We have to merge
  - Availability times  ➔ Extend the intervals
  - Groups of cores  ➔ Break the groups into same size
  - Set of certainly running jobs  ➔ Keep only jobs running in both states

Reachable states from state A and B should also be reachable from the merged state
Randomly generated task sets
- System cores: 8
- System tasks: 20 rigid tasks
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic
Our analysis results

Randomly generated task sets
- System cores: 8
- System tasks: 20 moldable tasks
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic
Our analysis results

Randomly generated task sets
- System cores: 4
- System tasks:
  - 4 moldable tasks
  - 5 segments per task
- Execution time variation: 50%
- Implicit deadlines
- Rate monotonic
Our analysis results

Randomly generated task sets

• System cores: 4
• System tasks: 4 moldable tasks
• Execution time variation: 50%
• System utilization: 70%
Agenda

✓ Gang schedulability analysis

• New scheduling policy
Core assignment

- JLFP assigns the maximum possible cores to a job
- Assign minimum number of cores that meet deadline

<table>
<thead>
<tr>
<th></th>
<th>Priority</th>
<th>Min cores</th>
<th>Max cores</th>
<th>Deadline</th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_0$</td>
<td>High</td>
<td>1</td>
<td>3</td>
<td>20</td>
<td>30, 15, 10</td>
</tr>
<tr>
<td>$J_1$</td>
<td>Low</td>
<td>2</td>
<td>2</td>
<td>20</td>
<td>15</td>
</tr>
</tbody>
</table>
JLFP limitations with moldable gang

<table>
<thead>
<tr>
<th></th>
<th>Priority</th>
<th>Cores</th>
<th>Deadline</th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_0$</td>
<td>High</td>
<td>2</td>
<td>$\infty$</td>
<td>10</td>
</tr>
<tr>
<td>$J_1$</td>
<td>Mid-high</td>
<td>3</td>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>$J_2$</td>
<td>Mid-low</td>
<td>1</td>
<td>$\infty$</td>
<td>20</td>
</tr>
<tr>
<td>$J_3$</td>
<td>Low</td>
<td>1</td>
<td>$\infty$</td>
<td>20</td>
</tr>
</tbody>
</table>
Reservation-based gang scheduler

- Reservation-based
- Reserve cores of higher-priority tasks and distribute the remaining ones among lower priority tasks
- Non-work conserving scheduler

<table>
<thead>
<tr>
<th>Task</th>
<th>Priority</th>
<th>Cores</th>
<th>Deadline</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_0$</td>
<td>High</td>
<td>2</td>
<td>$\infty$</td>
<td>10</td>
</tr>
<tr>
<td>$J_1$</td>
<td>Mid-high</td>
<td>3</td>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>$J_2$</td>
<td>Mid-low</td>
<td>1</td>
<td>$\infty$</td>
<td>20</td>
</tr>
<tr>
<td>$J_3$</td>
<td>Low</td>
<td>1</td>
<td>$\infty$</td>
<td>20</td>
</tr>
</tbody>
</table>
Results JLFP vs ResG in simulator

- Evaluated in simulator
- Randomly generated task sets
  - System cores: 8
  - 20 moldable tasks
  - Implicit deadlines
  - Rate monotonic
Conclusions

• The maximum parallelism of a gang task has a big role in the schedulability

• With a better scheduling policy one can improve the schedulability of moldable gang tasks
Future work

• Further reduce sources of pessimism with precedence constraints

• Provide an SAG analysis for the new scheduling policy
Contributions

- New schedulability analysis
- Moldable gang tasks evaluation
- New scheduling policy

Paper submitted to RTSS 2020 for the non-preemptive analysis

Presented in CAPITAL workshop

Thank you!