Scheduling and Analys of Limited-Preemptive Modable Gang Tasks

Joan Marcè i Igual Geoffrey Nelissen Mitra Nasri Paris Panagiotou

24th of February, 2020

What is gang?

- Parallel threads executed together as a “gang”
- Execution does not start until there are enough free cores

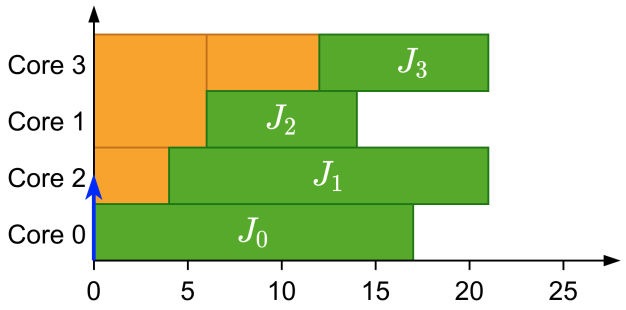
First of all, let’s explain what gang scheduling is.

It’s the execution of multiple parallel threads together as a “gang”. In these threads the execution does not start until there are enough cores to execute them together.
Let’s see an example:<click>

1. We have these jobs assigned to different cores.
2. If we release them as a gang job then we should pack them like a single task. Obtaining the following group. <click>
3. Then,<click> we have obtained the gang task result of merging the jobs

What is gang?

- Parallel threads executed together as a “gang”
- Execution does not start until there are enough free cores



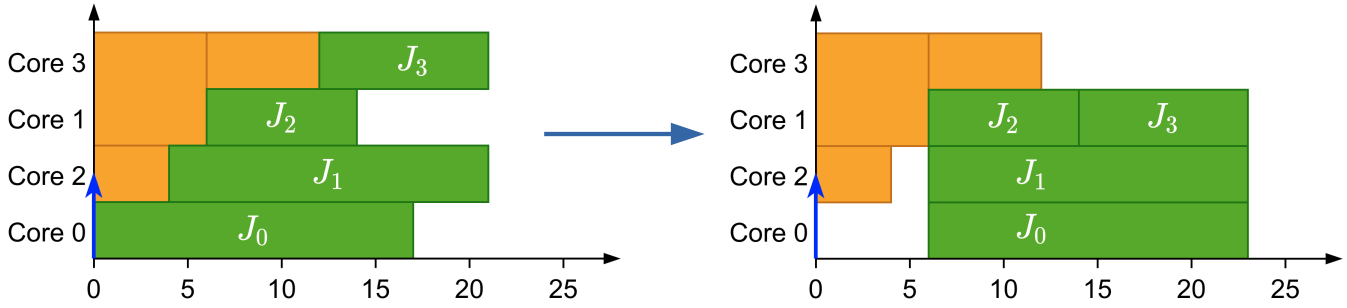
First of all, let’s explain what gang scheduling is.

It’s the execution of multiple parallel threads together as a “gang”. In these threads the execution does not start until there are enough cores to execute them together.
Let’s see an example:<click>

1.We have these jobs assigned to different cores.
2.If we release them as a gang job then we should pack them like a single task. Obtaining the following group. <click>
3.Then,<click> we have obtained the gang task result of merging the jobs

What is gang?

- Parallel threads executed together as a “gang”
- Execution does not start until there are enough free cores



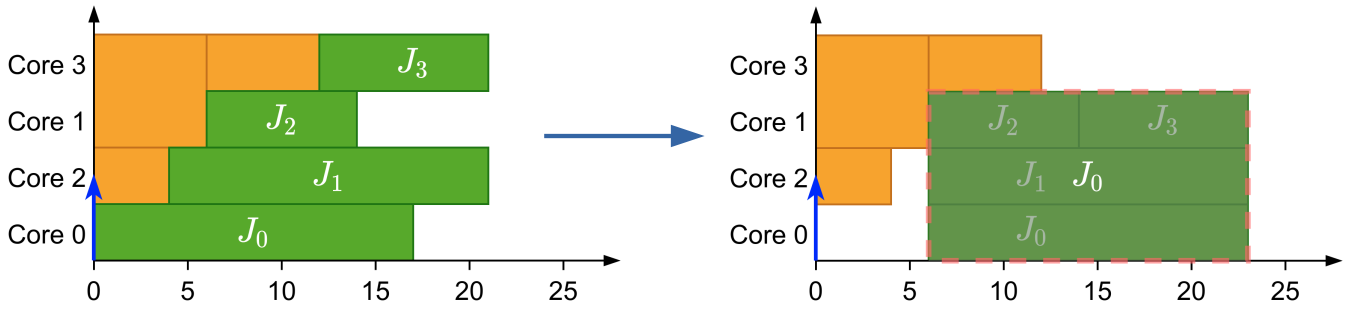
First of all, let’s explain what gang scheduling is.

It’s the execution of multiple parallel threads together as a “gang”. In these threads the execution does not start until there are enough cores to execute them together.
Let’s see an example:<click>

1.We have these jobs assigned to different cores.
2.If we release them as a gang job then we should pack them like a single task. Obtaining the following group. <click>
3.Then,<click> we have obtained the gang task result of merging the jobs

What is gang?

- Parallel threads executed together as a “gang”
- Execution does not start until there are enough free cores



First of all, let’s explain what gang scheduling is.

It’s the execution of multiple parallel threads together as a “gang”. In these threads the execution does not start until there are enough cores to execute them together.
Let’s see an example:<click>

1.We have these jobs assigned to different cores.
2.If we release them as a gang job then we should pack them like a single task. Obtaining the following group. <click>
3.Then,<click> we have obtained the gang task result of merging the jobs

Why gang?

- Avoids overhead when loading initial data

So then, why do we need gang scheduling?

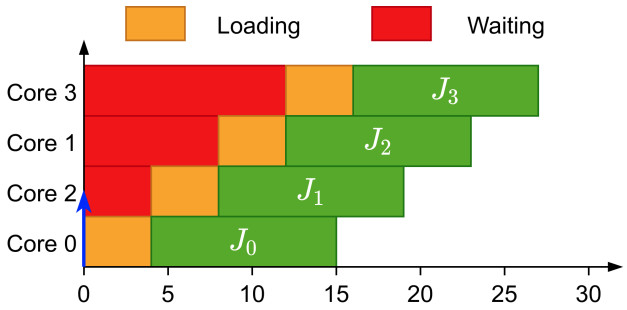
Well, it helps in reducing overhead when loading data initially.

<click>
In this example all these threads have been scheduled at the same time but have to wait until the previous one have finished loading.

On the contrary<click>, if we know that all these threads are starting at the same time we can load the data once for all the threads at the same time which is what we would do with gang scheduling.

Why gang?

- Avoids overhead when loading initial data



So then, why do we need gang scheduling?

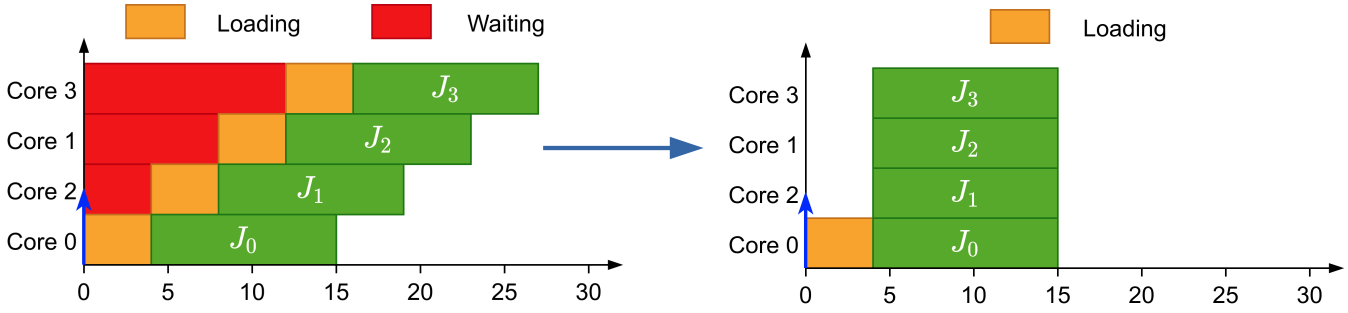
Well, it helps in reducing overhead when loading data initially.

<click>
In this example all these threads have been scheduled at the same time but have to wait until the previous one have finished loading.

On the contrary<click>, if we know that all these threads are starting at the same time we can load the data once for all the threads at the same time which is what we would do with gang scheduling.

Why gang?

- Avoids overhead when loading initial data



So then, why do we need gang scheduling?

Well, it helps in reducing overhead when loading data initially.

<click>
In this example all these threads have been scheduled at the same time but have to wait until the previous one have finished loading.

On the contrary<click>, if we know that all these threads are starting at the same time we can load the data once for all the threads at the same time which is what we would do with gang scheduling.

Why gang?

- Avoids overhead when loading initial data
- Allows synchronization

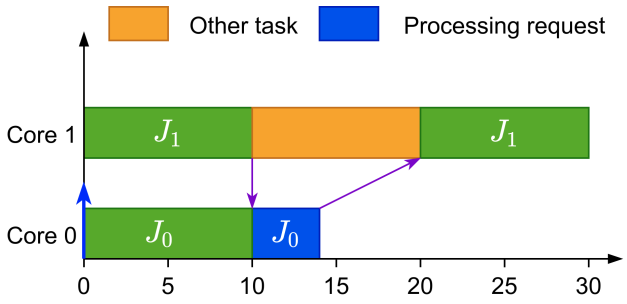
Additionally it helps in synchronization situations, like in the following example<click>

Here there are two jobs and job 1 sends some data to job 0 that is processed and sent back. In this case another task has been scheduled after job 1 sent the data and after the other task finishes then job 1 can continue its execution

On the other hand if we use gang<click> the job would wait for job 0 to finish the processing part and return the results as no other task would be allowed to execute while job 1 is waiting.

Why gang?

- Avoids overhead when loading initial data
- Allows synchronization



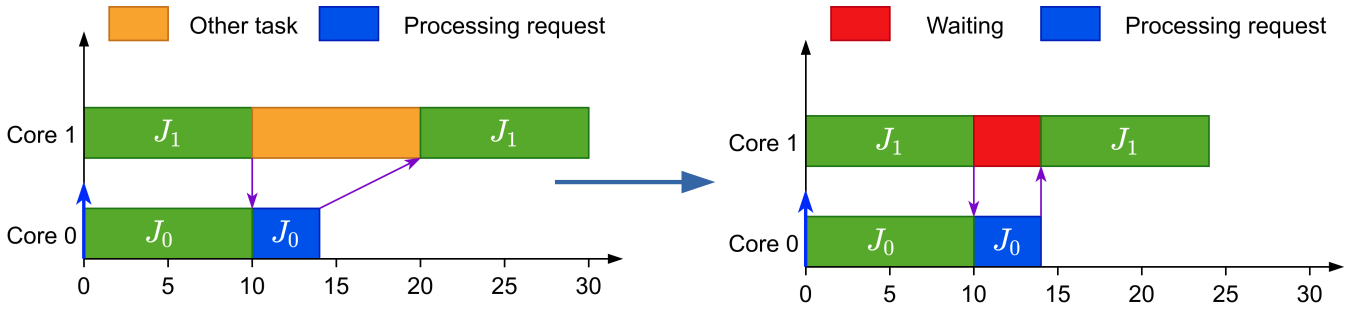
Additionally it helps in synchronization situations, like in the following example<click>

Here there are two jobs and job 1 sends some data to job 0 that is processed and sent back. In this case another task has been scheduled after job 1 sent the data and after the other task finishes then job 1 can continue its execution

On the other hand if we use gang<click> the job would wait for job 0 to finish the processing part and return the results as no other task would be allowed to execute while job 1 is waiting.

Why gang?

- Avoids overhead when loading initial data
- Allows synchronization



Additionally it helps in synchronization situations, like in the following example<click>

Here there are two jobs and job 1 sends some data to job 0 that is processed and sent back. In this case another task has been scheduled after job 1 sent the data and after the other task finishes then job 1 can continue its execution

On the other hand if we use gang<click> the job would wait for job 0 to finish the processing part and return the results as no other task would be allowed to execute while job 1 is waiting.

Types of gang

When talking about gang scheduling, we can have three types:

- <click>Rigid gang, where the number of cores is set by the programmer and fixed during execution
- <click>Moldable gang, where the number of cores can be in a range of values and the actual chosen value is decided when scheduling the task
- <click>Malleable gang, where the number of cores can vary during the execution of the job

Types of gang

- **Rigid**: number of cores set by programmer

When talking about gang scheduling, we can have three types:

- <click>Rigid gang, where the number of cores is set by the programmer and fixed during execution
- <click>Moldable gang, where the number of cores can be in a range of values and the actual chosen value is decided when scheduling the task
- <click>Malleable gang, where the number of cores can vary during the execution of the job

Types of gang

- **Rigid**: number of cores set by programmer
- **Moldable**: number of cores assigned during scheduling

When talking about gang scheduling, we can have three types:

- <click>Rigid gang, where the number of cores is set by the programmer and fixed during execution
- <click>Moldable gang, where the number of cores can be in a range of values and the actual chosen value is decided when scheduling the task
- <click>Malleable gang, where the number of cores can vary during the execution of the job

Types of gang

- **Rigid**: number of cores set by programmer
- **Moldable**: number of cores assigned during scheduling
- **Malleable**: number of cores can change during runtime

When talking about gang scheduling, we can have three types:

- <click>Rigid gang, where the number of cores is set by the programmer and fixed during execution
- <click>Moldable gang, where the number of cores can be in a range of values and the actual chosen value is decided when scheduling the task
- <click>Malleable gang, where the number of cores can vary during the execution of the job

Previous work

- Introduced in the context of high-performance computing[1]
- In real-time:
  - For rigid tasks:
    - Job-Level Fixed Priority is not predictable[2]
    - An optimal scheduler (DP-Fair) exists for preemptive tasks[3]
  - For moldable tasks
    - Global EDF has been adapted[4]
    - Scheduler that chooses cores to meet the deadline[5]

[1]Ousterhout, 1982
[2]Goossens et al., 2010
[3]Goossens et al., 2016
[4]Kato et al.,2009
[5]Berten et al., 2011

Previous work

- In real-time:
  - For malleable tasks:
    - An optimal algorithm, in terms of processors, has been proposed[6]
  - Bundled task-model[7]:

[6]Collette et al., 2008

[7]Wasly et al., 2017

Schedulability analysis

Our work

LPMRGS