Implementing Concurrent Non-Event Transitions in CMS

In partnership with:
Context

CMS uses a multi-threaded framework
   Used in production since 2016
   Built using Intel’s Thread Building Block (TBB) task library

Initially only supported
   concurrent processing of events and
   concurrent processing of modules within an event

Goal: Allow all framework transitions to be processed concurrently
CMS Data Hierarchy

Run 1

Lumi 1  Lumi 2

Event 1  Event 2  Event 3  Event 4
CMS Data Processing Transitions

Run 1

Lumi 1

Event 1  Event 2

Lumi 2

Event 3  Event 4
CMS Data Processing Transitions

beginRun

Run 1

Lumi 1 | Lumi 2

Event 1 | Event 2 | Event 3 | Event 4
CMS Data Processing Transitions

- **beginRun**
  - Run 1

- **beginLumi**
  - Lumi 1
  - Lumi 2

  - Event 1
  - Event 2
  - Event 3
  - Event 4
CMS Data Processing Transitions

- **beginRun**
  - Run 1

- **beginLumi**
  - Lumi 1
  - Lumi 2

- **event**
  - Event 1
  - Event 2
  - Event 3
  - Event 4
CMS Data Processing Transitions

beginRun

Run 1

beginLumi

Lumi 1

Lumi 2

event

event

event

event
CMS Data Processing Transitions

beginRun

Run 1

beginLumi  

Lumi 1

endLumi  

Lumi 2

event  

Event 1

event  

Event 2

event  

Event 3

event  

Event 4
CMS Data Processing Transitions

- **beginRun**
  - **Run 1**
  - **beginLumi**
  - **Lumi 1**
  - **endLumi**
  - **beginLumi**
  - **Lumi 2**
  - **Event 1**
  - **Event 2**
  - **Event 3**
  - **Event 4**
CMS Data Processing Transitions

- **beginRun**: Start of a run
- **beginLumi**: Start of a lumi block
- **endLumi**: End of a lumi block
- **event**: Data processing event

### Run 1
- **Lumi 1**
  - Event 1
  - Event 2
- **Lumi 2**
  - Event 3
  - Event 4
CMS Data Processing Transitions

beginRun

Run 1

beginLumi       endLumi       beginLumi

Lumi 1    Lumi 2

event       event

Event 1    Event 2    Event 3    Event 4
CMS Data Processing Transitions
CMS Data Processing Transitions

- **beginRun**
- **endRun**
- **beginLumi**
- **endLumi**

Run 1

Lumi 1

Lumi 2

Event 1

Event 2

Event 3

Event 4

- event
- event
- event
- event
Original Concurrent Transitions

Run 1

Lumi 1

Event 1

Event 2

Lumi 2

Event 3

Event 4
Fully Concurrent Transitions

- Run 1
  - Lumi 1
    - Event 1
    - Event 2
  - Lumi 2
    - Event 3
    - Event 4
Constraining Memory

CMS’ driving force for multi-threading is to reduce memory usage
   Allows average memory per core to be decreased

Configuration used to set limits
   Independently control number of allowed concurrent events, lumis and runs
Shared Resources and Task Queues

All work in the framework is done via TBB tasks

Tasks needing the same resource are placed in a queue
  Each unique resource gets its own queue
    E.g. writing to a particular TFile
    E.g. processing Lumis

When a resource is available, the task queue starts a waiting task
  E.g. when a task using a resources finishes, the queue starts the next task

Chains of tasks needing a resource are handled by pausing the queue
  When the last task in a chain finishes, the queue is resumed
Lumi Limited Task Queue

Limited Task Queue

- Has multiple independent *lanes* where each lane runs its own task
- All lanes pull tasks from the same waiting task list
- Each lane can be paused/restarted independently
- If all lanes are paused, no new tasks will be started from the queue

Number of concurrent Lumis controlled via a queue

- How many concurrent Lumis is set in the configuration to constrain memory use
Lumi Processing with Queue

Source: Lumi, Event, Event, Lumi

Lumi Queue

Read Lumi

Is First?

Lumi?

Is Last?

Peek Event

Do Event

Pause

Restart

End Lumi

Lumi?
Lumi Processing with Queue

Source Lumi Event Event Lumi

Lumi Queue

Peek Lumi? Read Lumi Begin Lumi

Lumi? Is First?

Pause

Is Last?

End Lumi

Restart

Event? Do Event

Do Event

Event?
Lumi Processing with Queue

Source | Lumi | Event | Event

Lumi Queue

Is First?

Is Last?

Lumi?

Pause

Restart

Event?

Event?

Do Event

Do Event
Lumi Processing with Queue
Lumi Processing with Queue

Source Lumi

Lumi Queue

Read Lumi → Peek Lumi

End Lumi

Is First?

Lumi?

Is Last?

Pause

Peek

Event?

Do Event

Do Event

Restart

Read Lumi

Begin Lumi
Measurements

Input file
  1 Run
  8 Lumis
  200 events per Lumi

Standard CMS reconstruction job

KNL Hardware
  Use 64 threads

Measurement variations
  Only one Lumi at a time
  8 concurrent Lumis
Reading Concurrency Plots

Total number of concurrent modules

Perfect efficiency when
number of modules == number of threads
Reading Concurrency Plots

Total number of concurrent modules
Perfect efficiency when number of modules == number of threads

Dark Green
Number of concurrent events with modules actually running
Reading Concurrency Plots

Total number of concurrent modules
Perfect efficiency when number of modules == number of threads

Dark Green
Number of concurrent events with modules actually running

Light Green
Number of concurrent modules processing Lumis or Runs
Measurement Results

Single Lumi
Synchronizing on Lumi Boundaries
Thread utilization is poor
Results

Single Lumi
Synchronizing on Lumi Boundaries
Thread utilization is poor

8 Concurrent Lumis
Synchronizations are gone
Excellent thread utilization
Job finishes faster (~15%)
Complication

CMS supports modules which can only handle one thread at a time
   The framework serializes access to those modules

Serial module can *opt in* to see Lumi and/or Run transitions
   Module will not see next Lumis beginLumi until it has seen last Lumis endLumi
Serial Modules and Lumis

Event in Lumi 1

Serial Module

Module

end Lumi 1

Serial Module

Module

begin Lumi 2

Serial Module

Module

Event in Lumi 2

Serial Module

Module

Time
## Serial Modules and Lumis

<table>
<thead>
<tr>
<th>Transitions</th>
<th>Event in Lumi 1</th>
<th>begin Lumi 2</th>
<th>end Lumi 1</th>
<th>Event in Lumi 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial Module</td>
<td>Module</td>
<td>Serial Module</td>
<td>Module</td>
<td>Serial Module</td>
</tr>
</tbody>
</table>

**Time**
Serial Modules and Lumis

Event in Lumi 1: Serial Module
end Lumi 1: Serial Module
begin Lumi 2: Serial Module
Event in Lumi 2: Serial Module

Time
Serial Modules and Lumis

Event in Lumi 1

Serial Module

Module

end Lumi 1

Serial Module

Module

begin Lumi 2

Serial Module

Module

Event in Lumi 2

Serial Module

Module

Time
Serial Module and Concurrent Lumis

Just 1 Serial Module in the job
  opted in for Lumi transitions

Synchronizing on Lumi boundaries again
  Events from new Lumi wait until module completes old Lumi
Conclusion

CMS can concurrently process events across Lumi boundaries
  Increases Event throughput
  Allows more efficient processing of files with few Events per Lumi

Task queues are helpful to manage shared resources

Full utilization is hampered by serial modules which watch Lumis