Intra-Module Parallelism

T. Hauth, V. Innocente, D. Piparo

Annual Concurrency Forum Meeting

Contents

- What is intra-module parallelism
- Why intra-module matters
- How to achieve it
- An example from CMS: triplet seeding
- Lessons learned and conclusions

Note:

The fundamental data processing unit (usually implemented as a C++ Class) will be referred as to *module* according to the **CMS nomenclature**

Parallelism at Multiple Levels

GOAL of a parallel framework:

- Achieve maximum rate of event processing

Take into account different type of parallelism, for example:

1) Concurrent execution of modules:

- Provided by the framework. Conditions: no simultaneous usage of thread unsafe resources

2) Parallelism within single modules (*intra-module* parallelism):

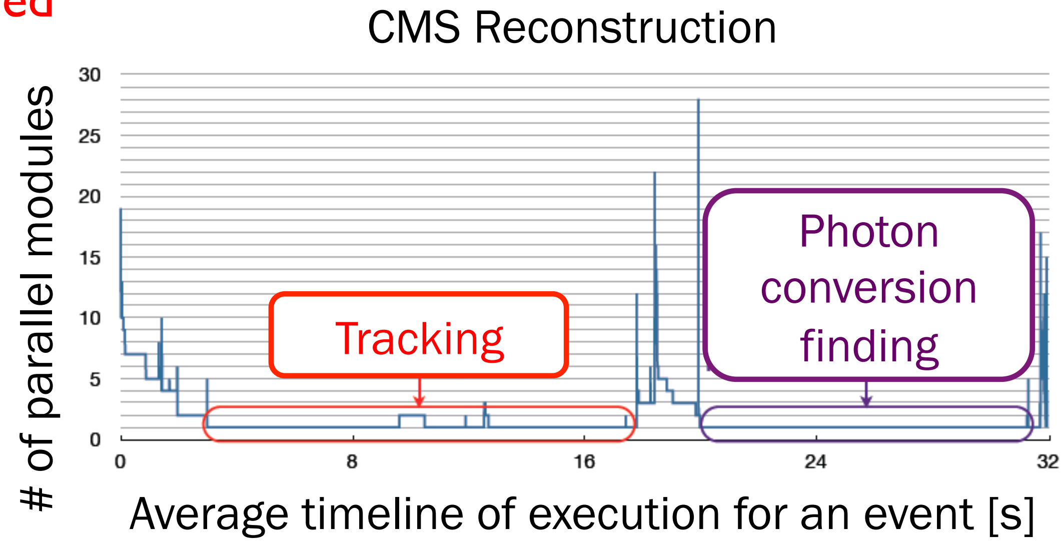
- Feature of data processing algorithm's implementation
- Provided by the developer(s) of the module itself

Why Intra-module Parallelism?

Bare simultaneous execution of serial modules has costs:

- Present data processing workflows (e.g. CMS reconstruction)
  - Few modules can be parallel for a given event
  - Long running modules* that may only execute w/o anything else simultaneously
- Increase probability to schedule a module: process several events simultaneously
- More events in flight mean:
  - Potential increase of event backlog (difference in DAQ timestamp between newest and oldest event in flight – e.g. repercussions on detector conditions management)
  - More memory needed

Intra-module parallelism: a "memory reduction technique"?



Plot: C. Jones

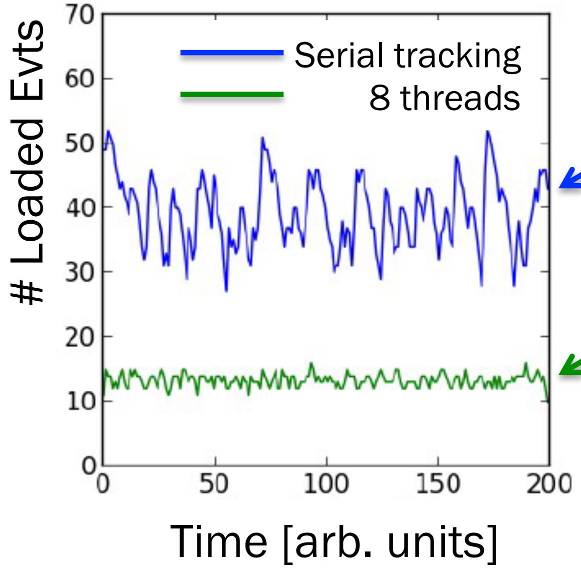
* Or "sequences" of modules

Why Intra-module Parallelism?

Intra-module parallelism allows to use resources w/o increasing memory

Toy model of a reco job in a parallel framework:

- 64 cores machine
- Tracking: 60% of runtime
  - Serial (1 thread) OR
  - 8 threads
- Other modules: 40% of runtime
  - 4 threads



40 on avg

12 on avg

Event size in memory: conservative rough estimate for CMS: ~150MB / evt

The interplay of module and intra-module parallelism is the target to aim at.

- Intra-module parallelism alone is not enough to efficiently use all resources.

Is This a Silver Bullet?

No. Intra-module parallelism has drawbacks

A handful of modules could benefit from it

  - Overhead: not profitable if module runtime too short

- Module developers need important skills
  - Code must be correct (not a trivial requirement)
  - Increase of code complexity
- Noticeable validation effort involved:
  - Identical results wrt serial version may not be achieved
  - Deep understanding of the physics involved to declare results correct (or correct enough or compatible with the previous one)

Achieving Intra-Module Parallelism

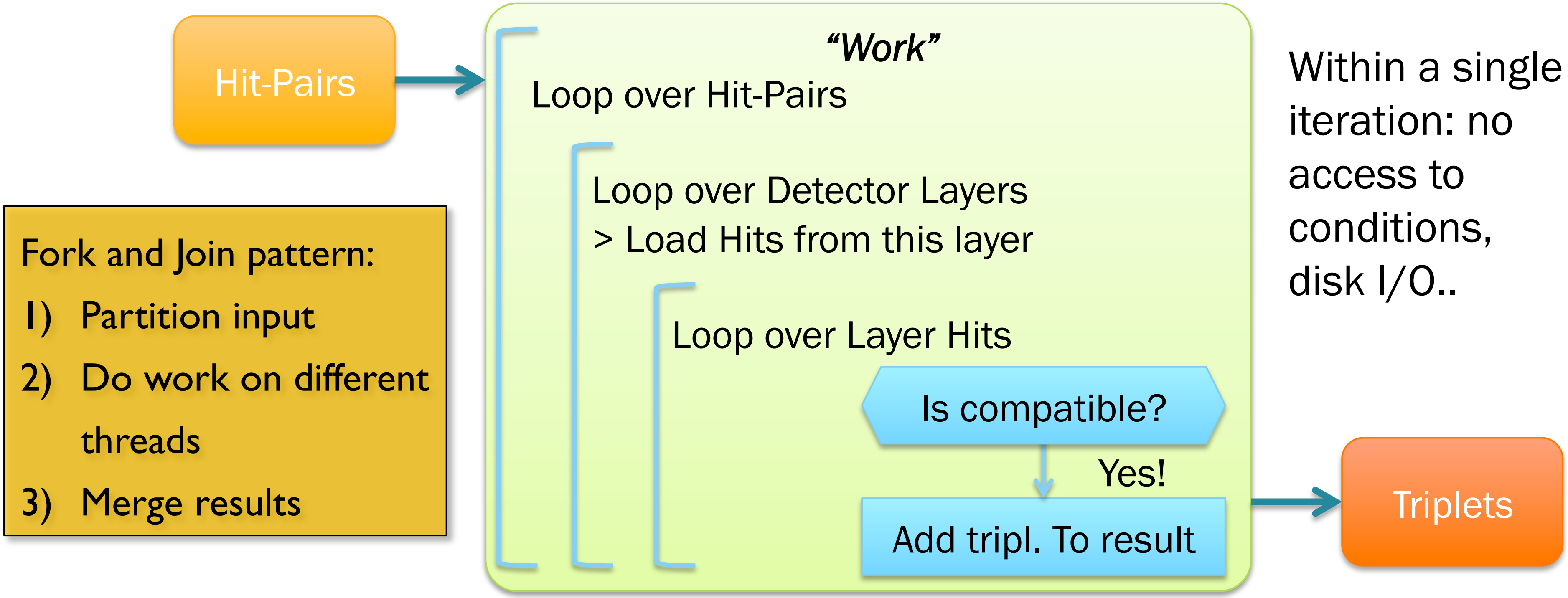
- Technology adopted: should be the same used by the framework for module parallelism
- Several sub-frameworks, developed separately maybe relying on different technologies: NO GO

TBB is a technology suited in this case. It supports:

- Based on a task based programming model
- Handy tools like *parallel_for* construct
- TBB scheduler handles tasks holding a module and tasks spawned by *parallel_for*. Tasks spawned within task get proper priority.

An example from CMS: Track Seeding

- About 10% of the overall runtime of the reconstruction
- Match pairs of tracker hits with a third one
- Parallelise loop on pairs using TBB *parallel_for* within CMSSW
- Compatible with parallel CMSSW design!



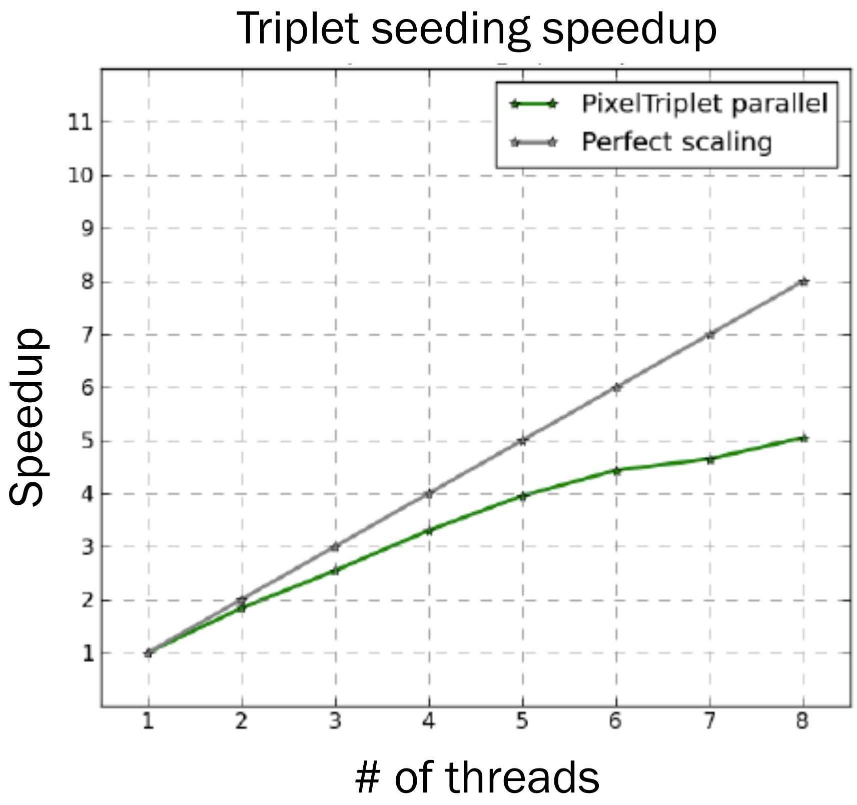
Fork and Join pattern:

1) Partition input
2) Do work on different threads
3) Merge results

Within a single iteration: no access to conditions, disk I/O..

An example from CMS: Track Seeding

- Good scaling up to 5 threads (40 PileUp events, probably better with higher occupancies)
- Memory overhead verified to be negligible:
  - Additional RSS: ~2MB/thread
- Validation accomplished: results identical in serial and parallel case
  - Order of processed pairs could be preserved
- Ready for production



Lesson Learned

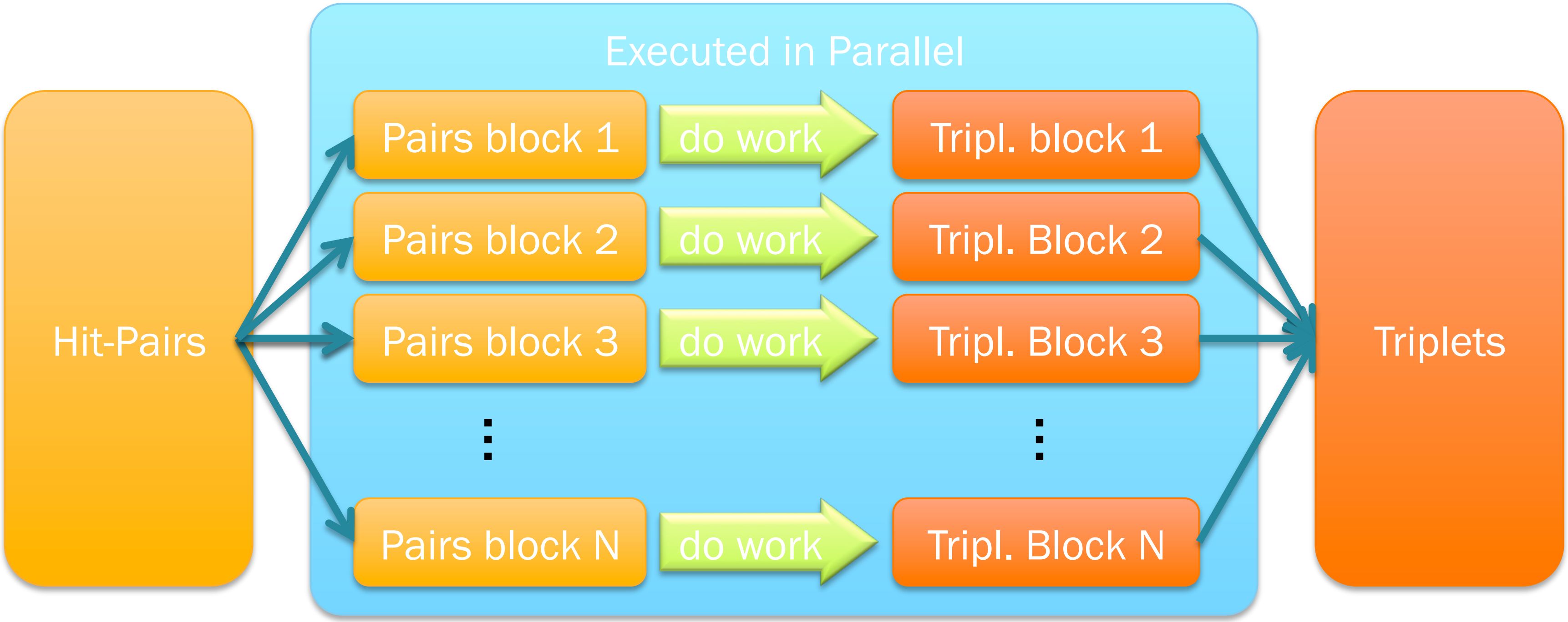
- Intra-module parallelism: increases processing speed with ~constant memory footprint
- Full potential reached only in combination with module parallelism
- Sizeable effort of developers may be needed: physics understanding, coding, validation
- CMS triplet seeding parallelised with TBB *parallel_for* construct:
  - Successful example of the fork-join pattern
- No general rule to achieve intra-module parallelism: case-by-case study

Conclusions

- Interplay of module and intra-module parallelism: key feature of forthcoming frameworks
- Technology to achieve intra-module parallelism must be provided by the framework
  - Avoid several "custom mini-frameworks"
  - TBB is a good candidate: lightweight tasks, handy high level constructs (e.g. parallel_for), smart scheduler
- We must not parallelise all our modules:
  - Focus on ideal candidates: modules or chain of modules with long runtimes which may run only w/o anything in parallel

Backup

Parallel Triplet Seeding



- Preserve the ordering of output collection
- Hit-pairs of input collection split in equally sized blocks
- A private result list is associated with every block
  - merged in the correct order into the global result list
  - No explicit sorting needed!

Hardware Threading: Food for Thought

- Many of the CPUs at our computing centres have Hyperthreading
- With a multi-threaded application we can use more (Hyperthreaded) Cores with very little memory overhead (less than 2 MB per Thread)

- Intel Core i7-3930K CPU at 3.20GHz
- 6 Physical Cores (12 Hyperthreaded)
- 16 GB RAM
- Scientific Linux 6.2
- Same 50 High-Pileup Data Events

Runtime of **6 Single-Threaded** CMSSW Applications: **14.40 min +/- 0.10 min**
Runtime of **6 Two-Threaded** CMSSW Applications: **13.79 min +/- 0.08 min**

- Hyperthreading → decrease in runtime of 4.3 %
- Very close theoretical decrease of 5% with 2 threads (10%/2).
  - Not physical but hyperthreaded cores!

A possible way to better exploit the already purchased resources?