Shared Memory Prototype for LHC

Status and Plans

MultiCore R&D Workshop
Parallelization of the Event-processing Framework Using Multiple Processes/Threads
10th October 2008

Marc Magrans de Abril
Vincenzo Innocente (supervisor)



Outline

1. Problem
2. Objectives
3. Results
4. Tools
5. Lessons Learned

References

More details about the development process:

https://twiki.cern.ch/twiki/bin/view/LCG/SharedMemoryLhc

Outline

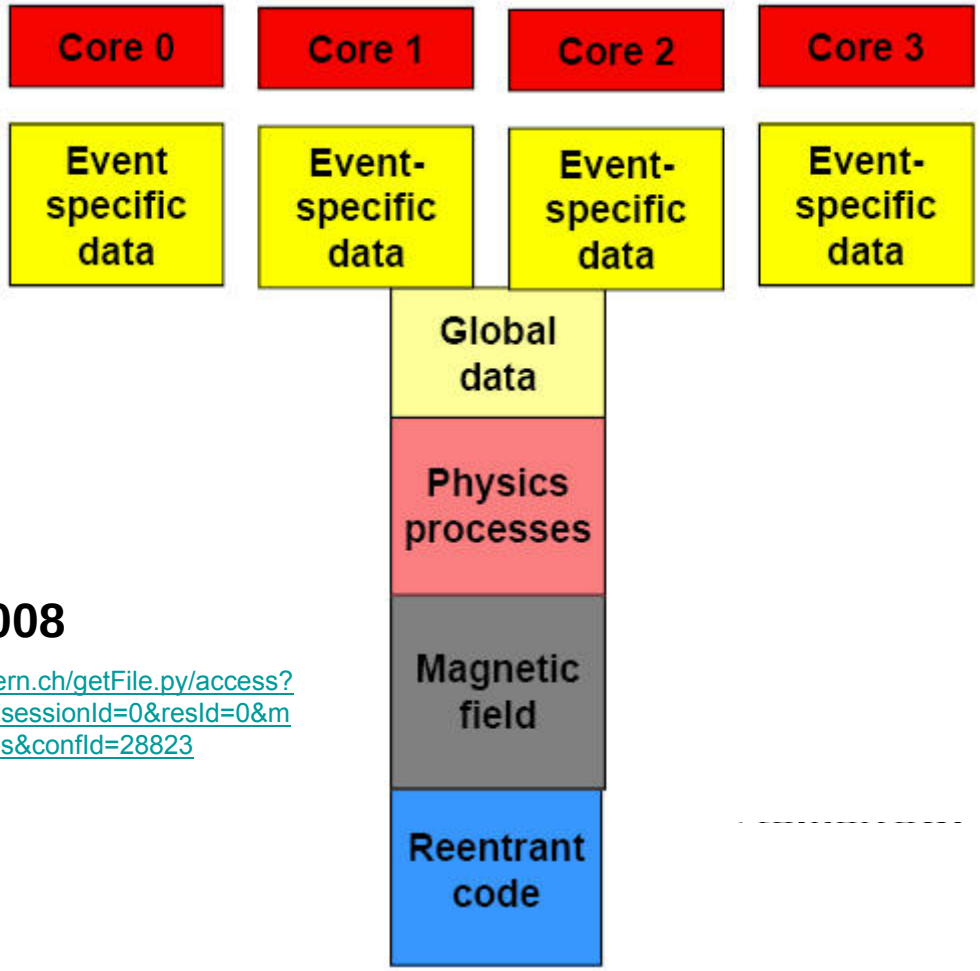
1. **Problem**
2. Objectives
3. Results
4. Tools
5. Lessons Learned

References

1 Problem (1/3)

PROBLEM: CMS Reconstruction Footprint shows large condition data

How to share common data between different process?



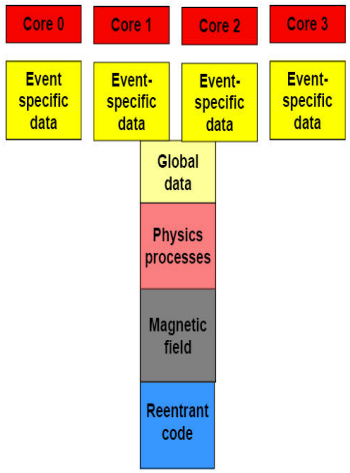
S. Jarp

April 2008

http://indico.cern.ch/getFile.py/access?contribId=23&sessionId=0&resId=0&materialId=slides&confId=28823

1 Problem (2/3)

How can we reach our objective?

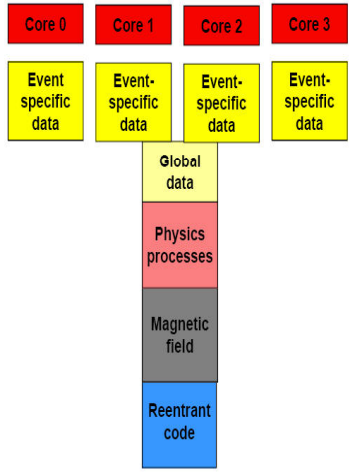


- **Shared Libraries:**
  - code-only
- **Multi-threading:**
  - Event Processor ↔ thread
  - Needs Integration of programs into threads
- **Copy-on-write (COW) or KSM-like:**
  - Event Processor ↔ process
  - Forked process: Needs Integration of programs into super-program
  - read-only
  - A single write invalidates the whole page!
- **Shared Memory:**
  - Event Processor ↔ process
  - COW still works
  - read-write
  - Needs factorization of shared objects

1 Problem (3/3)

COST vs MEMORY SAVING

What we need to answer?



- **Percentage of shared memory?**
  - private → We have a problem!!
  - shared (smaps, P. Mato measurements, >50%, LHCb)

- **Percentage of read-write shared memory?**
  - Conditions data modification in long runs
  - Invalidation of COW pages non-deterministic

- **Integration/factorization cost of Event Processors:**

  Shared memory > Multi-threaded > COW > Shared Libraries

WHITE BOX ⟶ **BLACK BOX**

Outline

1. Problem
2. **Objectives**
3. Results
4. Tools
5. Lessons Learned

References

2 Objectives

- Study **possibilities and limitations** of shared memory
- Provide **tested code** for developers
- boost::interprocess (**avoid** mmap, pthreads, ipc, shm, etc.)

Outline

1. Problem
2. Objectives
3. **Results**
4. Tools
5. Lessons Learned

References

3 Results (1/2)

Factory class provides:

- Creation/destruction of shared memory segments and STL allocators
- Creation/destruction of shared objects
- Creation/destruction of shared STL containers
- Thread-and-process-safe (WIP)
- Exception-safe (WIP)

OWNER

```
shName("MySharedSegment");

Factory f(shName, 10000*10000);

double& d = f.create<double>("d");
d = rand()/rand();
std::cout << "d = " << d << std::endl;

f.destroy<double>("d");
```

USER

```
shName("MySharedSegment");

Factory f(shName);

double& d = f.get<double>("d");
std::cout << "d = " << d << std::endl;
```

sources:
https://twiki.cern.ch/twiki/bin/viewfile/LCG/SharedMemoryLhc?rev=1;filename=factory20081002.zip

3 Results (2/2)

Shared memory allows:

- Shared memory IS NOT TRANSPARENT
- POD objects
- Arrays of POD objects
- STL containers with POD obejcts
- Smart Pointers to shared memory (e.g. Linked List)
- Objects with virtual methods (FORKED processes).
- Dynamic library loading of objects with virtual methods should be BEFORE forking.

Shared memory does NOT allow:

- Shared static objects
- Normal pointers to shared memory
- Object with virtual methods (NON-FORKED processes)

sources:

https://twiki.cern.ch/twiki/bin/viewfile/LCG/SharedMemoryLhc?rev=1;filename=factory20081002.zip

Outline

1. Problem
2. Objectives
3. Results
4. **Tools**
5. Lessons Learned

References

4 Tools (1/2)

GDB 6.8: Forked process debugging (GDB manual section 4.10)

set follow-fork-mode [parent/child]: Choose the process being debugged after fork
set detach-on-fork [on/off]: if *on a*ll the process will be under the control of GDB
info forks: List of forked processid
process [processid]: Debug another forked process

4 Tools (2/2)

proc/<pid>/smaps scripts: COW and shared memory usage

a) smem.pl (requires Linux::Smaps)

http://bmaurer.blogspot.com/2006/03/memory-usage-with-smaps.html

b) mem_usage.py

Resident Set Size. Physical Memory Usage

http://wingolog.org/archives/2007/11/27/reducing-the-footprint-of-python-applications

Outline

1. Problem
2. Objectives
3. Results
4. Tools
5. **Lessons Learned**

References

5 Lessons Learned

- boost::interprocess is **professional and well-documented**
- Shared memory usage **IS NOT TRANSPARENT**:
  - Synchronization
  - Creation/Destruction of memory and objects
  - Restrictions: pointers, statics, virtual classes

  → **More complexity than multi-threaded**
- **Factory pattern simplifies client code** syntax
- **gdb 6.8** allows inter-process debugging
- Memory usage can be analyzed with **smaps** scripts

Q &A

THANKS!

References →

6 References

S. Jarp, “(Some of) the Issues Facing CERN and HEP in the Many-core Computing Era”, Workshop on Virtualization and Multi-core technologies for the LHC, Apr 2008, http://indico.cern.ch/getFile.py/access?contribId=23&sessionId=0&resId=0&materialId=slides&confId=28823

M. Zanetti, “Filter Unit Shared Memory Buffer”, https://twiki.cern.ch/twiki/bin/view/CMS/FUShmBuffer

V. Innocente, “How o exploit Multi-core”, Workshop on Virtualization and Multi-core technologies for the LHC, Apr 2008, http://indico.cern.ch/getFile.py/access?contribId=19&sessionId=1&resId=0&materialId=slides&confId=28823

Boost Interprocess Library, http://www.boost.org/doc/libs/release/libs/interprocess/index.html

Workshop on Virtualization and Multi-core technologies for the LHC, Apr 2008, http://indico.cern.ch/conferenceDisplay.py?confId=28823

M. Magrans de Abril, “Shared Memory for the LHC”, https://twiki.cern.ch/twiki/bin/view/LCG/SharedMemoryLhc

Qumranet , Increasing the Virtual Memory Size with KSM, http://kvm.qumranet.com/kvmwiki/KvmForum2008?action=AttachFile&do=get&target=kdf2008_12.pdf