Shared Memory Prototype for LHC

Status and Plans

CERN Multi-core R&D Effort Meeting
3rd July 2008

Marc Magrans de Abril
Vincenzo Innocente (supervisor)



Outline

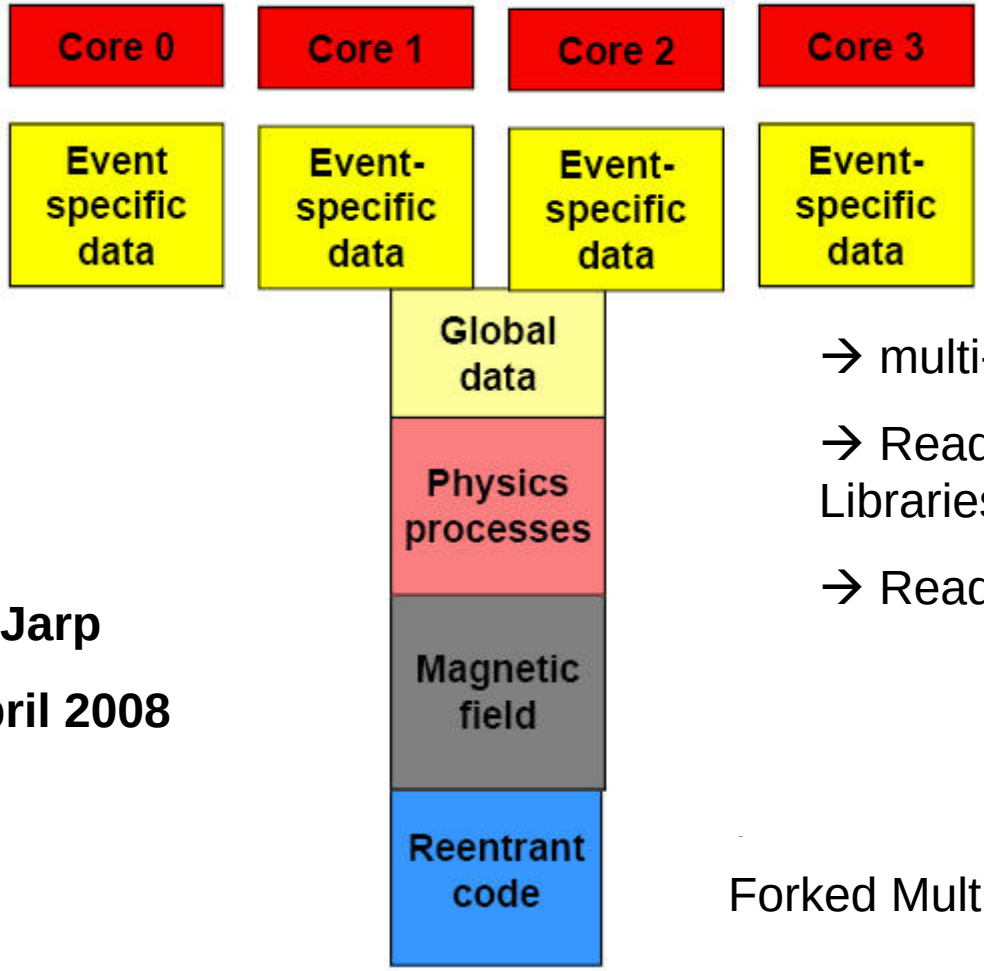
1. **Problem**
2. C++ Objects in Shared Memory
3. STL Containers in Shared Memory
4. Tools
5. Lessons Learned
6. ToDo

References

1 Problem

PROBLEM: CMS Reconstruction Footprint shows large condition data

How to share common data between different process?



S. Jarp

April 2008

→ multi-process vs multi-threaded

→ Read-only: Copy-on-write, Shared Libraries

→ Read-write: Shared Memory or files

Forked Multi-process & Shared Memory

Outline

1. Problem
2. **C++ Objects in Shared Memory**
3. STL Containers in Shared Memory
4. Tools
5. Lessons Learned
6. ToDo

References

2 C++ Objects in Shared Memory

Details: https://twiki.cern.ch/twiki/bin/view/LCG/SharedMemoryTests

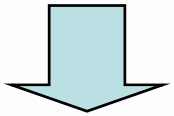
Problem: Study the limitations of C++ objects in Shared Memory.

Design Decisions:

- Boost Interprocess Library

Conclusions:

- POD can be stored in shared memory
- Objects with virtual methods can be stored in shared memory if (AND):
  a) Process are created using fork (COW)
  b) Libraries are loaded before fork
- Pointers are not shared (instead Boost smart pointer)
- Static members are not shared
- STL (e.g. std::string) are not POD



Shared memory should be used in forked process

STL needs shared allocator

Smart pointers needed

Transparency is not possible

2.1 Example (1/3)

```
#include <unistd.h>
#include "VirtualClass.h"
#include <boost/interprocess/managed_shared_memory.hpp>
#include <boost/interprocess/sync/interprocess_condition.hpp>

namespace ip = boost::interprocess;

int main( int argc, char* argv[] ) {
  pid_t pid = fork();
  if (pid >0)  {
    ip::managed_shared_memory segment(ip::create_only, "MySharedMemory",65536);

    VirtualClass *virt = segment.construct<VirtualClass>("VirtualObject")();
    std::cout << "PARENT: virt.mystr() = '" << virt->mystr() << "'" << std::endl;
    std::cout << "PARENT: virt.myint() = '" << virt->myint() << "'" << std::endl;

    std::cout << "---PARENT: Press a key to read again the shared objects..." << std::endl;
    char c(getchar());

    ip::shared_memory_object::remove("MySharedMemory");

  } else if (pid ==0) {
    std::cout << "---CHILD: Press a key to read again the shared objects..." << std::endl;
    char c(getchar());

    ip::managed_shared_memory segment(ip::open_only, "MySharedMemory");

    std::pair<VirtualClass*, std::size_t> virt_res;
    virt_res = segment.find<VirtualClass> ("VirtualObject");
    std::cout << "---CHILD: VirtualObject::mystr() = '" << virt_res.first->mystr() << "'" << std::endl;
    std::cout << "---CHILD: VirtualObject::myint() = '" << virt_res.first->myint() << "'" << std::endl;
  }
}
```

1) fork

2) Create sh. mem

3) Create sh. virt. obj

6) Destroy sh. mem.

4) Open sh. seg.

5) Retrieve sh. virt. obj.

2.1 Example (2/3)

```
class VirtualParent {
 public:
  Parent():mystr ("Parent"), myint (0) {};
  virtual std::string mystr() const {
    return mystr_;
  }
  virtual int myint() const {
    return myint_;
  }

 protected:
  std::string mystr_;
  int myint_;
};

class VirtualClass: public VirtualParent {
 public:
  std::string mystr() const {
    return "Child "+ mystr_ ;
  }
  int myint() const {
    return myint_ + 1000;
  }
};
```

virtual methods: mystr and myint

Expected return value for VirtualClass methods:

v.mystr() → “Child “ + “Parent”

v.myint() → 0 + 1000

2.1 Example (3/3)

Expected:

```
$ ./program2.o
---CHILD: Press a key to read again the shared objects...
PARENT: virt.mystr() = 'Child Parent'
PARENT: virt.myint() = '1000'
---PARENT: Press a key to read again the shared objects...

---CHILD: VirtualObject::mystr() = 'Child Parent'
---CHILD: VirtualObject::myint() = '1000'
```

Obtained:

```
$ ./program2.o
---CHILD: Press a key to read again the shared objects...
PARENT: virt.mystr() = 'Child Parent'
PARENT: virt.myint() = '1000'
---PARENT: Press a key to read again the shared objects...

---CHILD: VirtualObject::mystr() = 'Child '
---CHILD: VirtualObject::myint() = '1000'
```

Remarks:

- No segmentation faults
- Virtual methods mystr() and myint() executed

→ VTable is resolved

- std::string inside the class is not shared → basic_string<char> is STL!!

Outline

1. Problem
2. C++ Objects in Shared Memory
3. **STL Containers in Shared Memory**
4. Tools
5. Lessons Learned
6. ToDo

References

3 STL Containers in Shard Memory

Details: https://twiki.cern.ch/twiki/bin/view/LCG/SharedMemoryAllocatorTests

Problems:

a) How to store STL containers in shared memory → Use Boost
b) How to simplify client code syntax
c) How to simplify client code “semantics”

Design Decisions:

- Boost Interprocess Library
- “template typedef” (section 3.1)
- Factory template pattern (section 3.2)
- RAII (section 3.3)

Conclusions:

- Shared memory segment can not grow
- Use of shared memory implies modifications in the client code
  → FACTORIZATION COST
- Production code should be based on templates
  → CODE & DEBUG COST

3.1 “template typedef”

Problem: Simplify the declaration of shared containers

Solution: “template typedef” tricks

Example: Creation of a PODClass vector

a) Without “template typedef”:

```
typedef ip::allocator<PODClass, ip::managed_shared_memory::segment_manager> PODAllocator;
typedef ip::vector<PODClass, PODAllocator> PODVector;
PODAllocator alloc(getSegment().get_segment_manager());
PODVector* v= getSegment().construct<PODVector>(“myvector”)(alloc);
```

→ 2 “typedefs” per template class

b) With “template typedef” trick:

```
Allocator<PODClass> alloc(getSegment().get_segment_manager());
Vector<PODClass>* v = getSegment().construct<Vector<PODClass> >(“myvector”)(alloc);
```

Where “Allocator” and “Vector” are templates

3.2 Factory Template Pattern

Problem: Simplify creation/destruction/retrieval of shared objects

Solution: Create a Factory

Design Decisions:

- Boost Interprocess Library
- Factory template pattern

Example: Creation/Destruction/Retrieval code:

a) Without Factory:

```
ip::managed_shared_memory segment(ip::create_only, "MySharedMemory",65536);          //create sh. mem.
Allocator<PODClass> alloc(getSegment().get_segment_manager());                       //create vec.
Vector<PODClass>* v = getSegment().construct< Vector<PODClass> >(“myvector”)(alloc); //retrieve vec.
std::pair<Vector<PODClass>*, std::size_t> res = segment.find<Vector>(“myvector”);
Vector<PODClass>* vr = res->first;
    ip::shared_memory_object::remove("MySharedMemory");                              //destroy sh. mem.
```

b) With Factory:

```
Factory f;                                                  //create sh. mem.
f.createSegment(“MySharedMemory”, 65536);
f.createContainer<Vector<PODClass> >(“myvector”);           //create vec.
Vector<PODClass>& v = f.get<Vector<PODClass> >(“myvector”);  //retrieve vec.
f.destroySegment(“MySharedSegment”)                         //destroy sh. mem.
```

3.3 RAII (1/2)

Problem: Simplify ownership of shared memory and shared objects

Solution: Resource Allocation Is Initialization (IRAII)

Design Decision:

- Shared Memory is owned by the process that creates it
- Shared objects are owned by the process that creates it

<u>Example</u>: Creation and destruction of shared memory

a) Without RAII

PARENT

```
Factory f;
f.createSegment("MySharedSegment", 65536);
// use shared memory
f.destroySegment("MySharedSegment")
```

sync - - - - - - - - →

CHILD

```
Factory f;
f.attachSegment("MySharedMemory");
// use shared memory
```

b) With RAII

PARENT

```
Factory f("MySharedSegment", 65536);
// use shared memory
// implicit destruction of shared memory
```

sync - - - - - - - - →

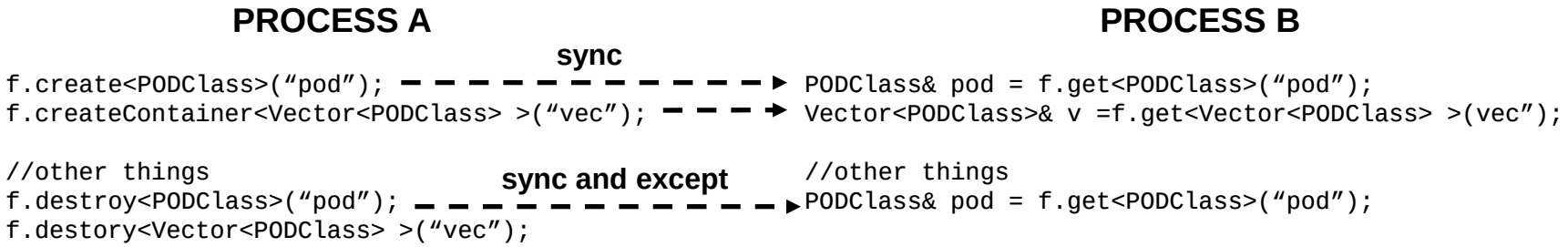
CHILD

```
Factory f("MySharedSegment");
// use shared memory
```

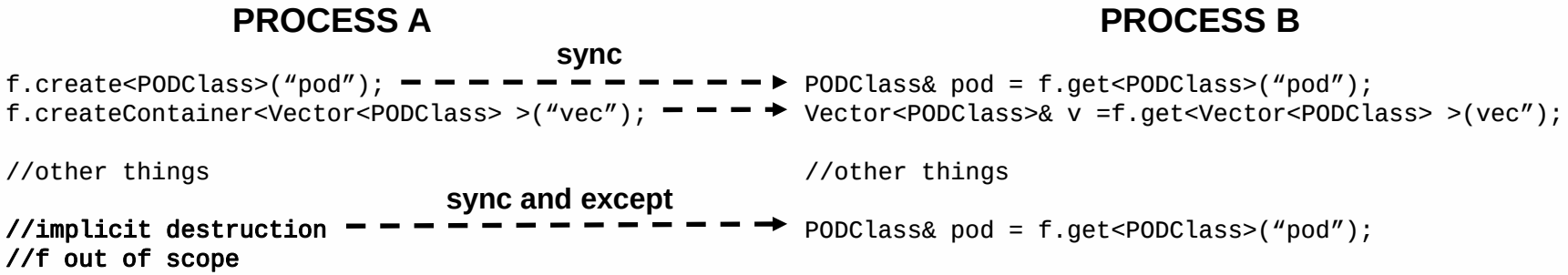
3.3 RAII (2/2)

Example: Creation and destruction of shared objects

a) Without RAII



b) With RAII



Outline

1. Problem
2. C++ Objects in Shared Memory
3. STL Containers in Shared Memory
4. **Tools**
5. Lessons Learned
6. ToDo

References

4 Tools (1/3)

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 (2/3)

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

```
$ sudo perl smem.pl 19057
     VMSIZE:     101992 kb
     RSS:         14012 kb total
                   7288 kb shared
                      4 kb private clean
                   6720 kb private dirty
     PRIVATE MAPPINGS
         vmsize   rss clean   rss dirty   file
       1564 kb        0 kb      864 kb   /opt/xdaq/lib/libxdata.so
        752 kb        4 kb      676 kb
       1136 kb        0 kb      656 kb   /opt/xdaq/lib/libtoolbox.so
        944 kb        0 kb      516 kb   /opt/xdaq/lib/libxdaq.so
        708 kb        0 kb      444 kb   /opt/xdaq/lib/libwseventing.so
      11100 kb        0 kb      372 kb
        …

     SHARED MAPPINGS
         vmsize   rss clean   rss dirty   file
       3656 kb     1692 kb        0 kb   /opt/xdaq/lib/libxerces-c.so.27
       1564 kb      700 kb        0 kb   /opt/xdaq/lib/libxdata.so
       1136 kb      480 kb        0 kb   /opt/xdaq/lib/libtoolbox.so
        804 kb      436 kb        0 kb   /usr/lib/libstdc++.so.6.0.3
        944 kb      428 kb        0 kb   /opt/xdaq/lib/libxdaq.so
        …
```

Resident Set Size. Physical Memory Usage

RSS Shared

RSS private

COW!

Information per library or shared file

4 Tools (3/3)

b) mem_usage.py

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

```
$ sudo python mem_usage.py 19057
     Mapped memory:
                    Shared              Private
                Clean    Dirty    Clean    Dirty
       r-xp      7284        0        0     5204  -- Code
       rw-p         0        0        0      272  -- Data
       r--p         0        0        0       40  -- Read-only data
       r--s         4        0        0        0
      total      7288        0        0     5516
     Anonymous memory:
                    Shared              Private
                Clean    Dirty    Clean    Dirty
       r-xp         0        0        0        0
       rw-p         0        0        4     1204  -- Data (malloc, mmap)
       ---p         0        0        0        0
      total         0        0        4     1204
      ------------------------------------------
      total      7288        0        4     6720
```

COW!

mman Kernel flags:

VM_READ ,VM_WRTIE, VM_EXEC,VM_MAYSHARE

Outline

1. Problem
2. C++ Objects in Shared Memory
3. STL Containers in Shared Memory
4. Tools
5. **Lessons Learned**
6. ToDo

References

5 Lessons Learned

- Shared memory can not be used transparently
- “template typedef” tricks
- Factory simplifies client code syntax
- RAII reduces client code complexity
- Inter-process synchronization is needed
- gdb 6.8 allows inter-process debugging
- COW and memory usage can be analyzed with smaps scripts

Outline

1. Problem
2. C++ Objects in Shared Memory
3. STL Containers in Shared Memory
4. Tools
5. Lessons Learned
6. **ToDo**

References

6 ToDo

- Priorities?
- Shared Memory tests:
  - pointers in shared memory test
- Shared Memory Factory:
  - Code review needed
  - Inter-process creation/destruction synchronization
  - Thread and exception safe factorization
  - Performance Measures: STL versus shared memory STL
- Tools:
  - /proc/<pid>/smaps improvement of the parsing script
  - Further interpretation and documentation of smaps
- Other:
  - Measure the possible impact of shared memory in CMSSW
  - Create a shared memory malloc?
  - Serialize objects to shared memory?

Outline

1. Problem
2. C++ Objects in Shared Memory
3. STL Containers in Shared Memory
4. Tools
5. Lessons Learned
6. ToDo

References

7 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

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

H. Sutter, “The New C++: typedef templates”, Dr. Dobb’s, http://www.ddj.com/cpp/184403850, Dec 2001

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 Tests”, https://twiki.cern.ch/twiki/bin/view/LCG/SharedMemoryTests

M. Magrans de Abril, “Shared Memory Allocator Tests”, https://twiki.cern.ch/twiki/bin/view/LCG/SharedMemoryAllocatorTests