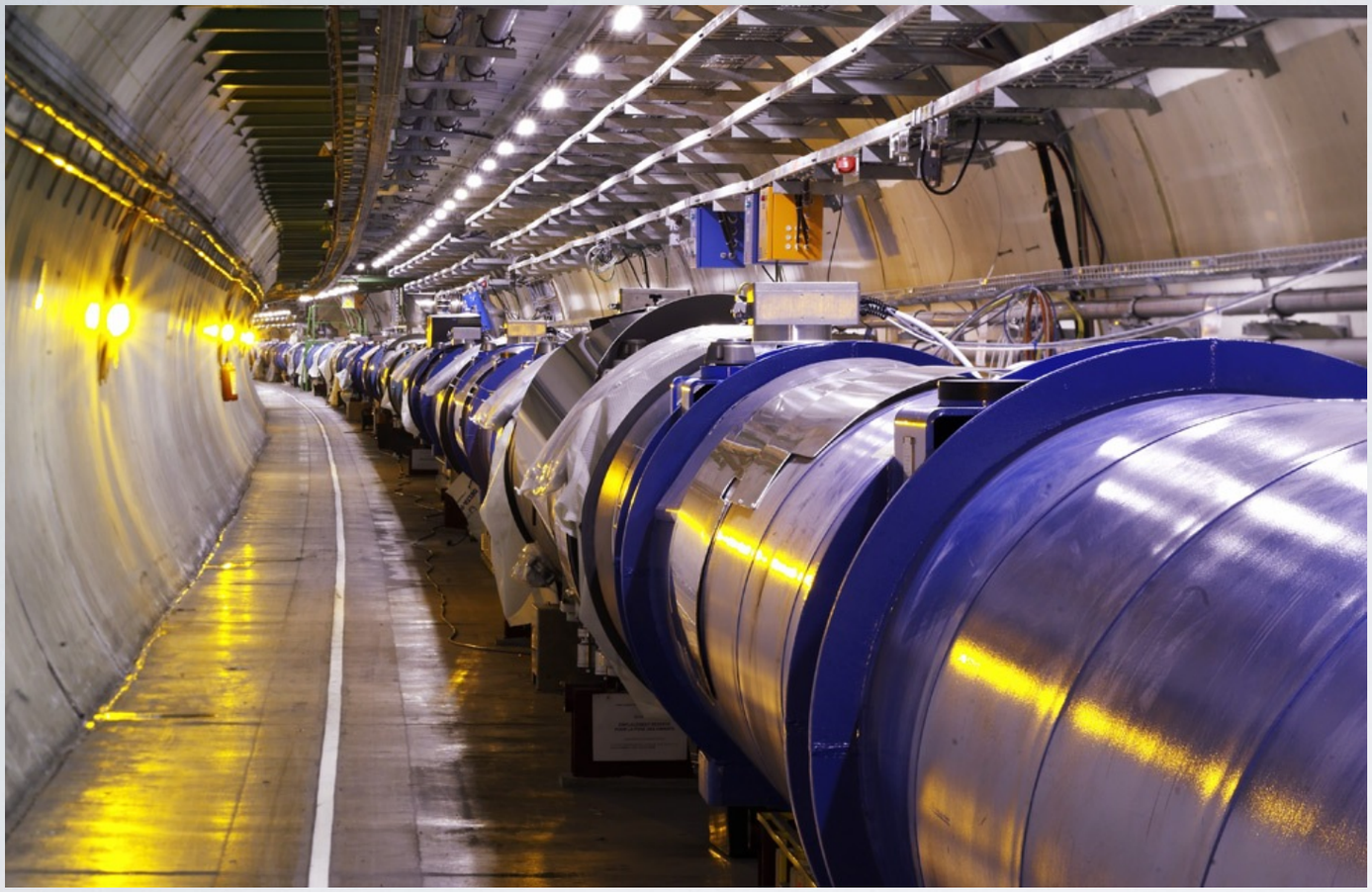


Peter Elmer
Princeton University

Gene Cooperman, Kapil Arya
Northeastern University

Andrea Dotti
SLAC

USE OF CHECKPOINT-RESTART FOR COMPLEX HEP SOFTWARE ON TRADITIONAL ARCHITECTURES AND INTEL MIC

CHECKPOINT-RESTART

- It is desirable in certain circumstances to "checkpoint" the state of a unix process, or set of processes, to disk with the possibility of restarting it at a later time.

- This can be done in an application-specific custom fashion, but it requires the addition and maintenance of dedicated code.

- A generalized technology capable of checkpointing all types of applications is thus desirable. In fact such technologies have been in use in High Performance Computing (HPC) and batch systems since more than 20 years.

INTERESTING USE-CASES

- Avoiding CPU-intensive initialization steps in frequently run applications
- Reproducibility of problems in long running jobs for debugging
  - The application can be "replayed" from a point just before the error or crash, rather than from the beginning
- In situations where resources are being used opportunistically, it can be used to efficiently give access back to the "owner" and then later restart when resources are free again
- In interactive applications, the current state can be saved ("workspace")
- For long-running parallel applications sensitive to hardware failure, the state of calculations can be saved periodically to allow restart.

DMTCP

- This presentation will present some information about tests done with the Distributed MultiThreaded CheckPointing package (DMTCP), developed at Northeastern University

- http://dmtcp.sourceforge.net

DMTCP

Key Features

Userspace checkpointing, no kernel-level access required

Minimum runtime overhead

Checkpoints multithreaded applications

Optional compression of checkpoint images

Checkpoints distributed applications

Open source

Can handle fork, exec, ssh, open file descriptors, TCP/IP sockets, etc.

Works on linux and supports a wide range of kernels

BASIC IDEA

- No code changes or recompilation are required, the application can simply be run as:
  - dmtcp_checkpoint cmsRun MinBias_cfi_GEN_SIM.py
- and an external checkpointing "coordinator" is used to trigger the checkpoint.
- Alternatively a dedicated DMTCP API can be used from within the application to place control of checkpointing into the application itself, e.g. to allow checkpointing at predefined specific points during the execution such as "every N events"

CMS EXAMPLE

- Quick test with CMS Framework-based generation/simulation application, memory footprint ~750MB RSS
- ~10s required to create compressed checkpoint image, 220MB
- 1-2s for uncompressed checkpoint

```
Begin processing the 1st record. Run 1, Event 1, LumiSection 1 at 18-May-2013 05
:10:01.557 CEST
Begin processing the 2nd record. Run 1, Event 2, LumiSection 1 at 18-May-2013 05
:10:01.584 CEST
WARNING: G4QPDGToG4Particle is deprecated and will be removed in GEANT4 version
10.0.
Begin processing the 3rd record. Run 1, Event 3, LumiSection 1 at 18-May-2013 05
:10:10.014 CEST
Begin processing the 4th record. Run 1, Event 4, LumiSection 1 at 18-May-2013 05
:10:14.434 CEST
Begin processing the 5th record. Run 1, Event 5, LumiSection 1 at 18-May-2013 05
:10:18.362 CEST
```

Trigger checkpoint externally while processing event #5

CMS EXAMPLE

- And restart:

```
vocms101> ./dmtcp_restart_script_a9bf217912ea647-48000-5196f087.sh
dmtcp_restart (DMTCP + MTCP) 2.0
Copyright (C) 2006-2011  Jason Ansel, Michael Rieker, Kapil Arya, and
                                                        Gene Cooperman
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions; see COPYING file for details.
(Use flag "-q" to hide this message.)

Begin processing the 6th record. Run 1, Event 6, LumiSection 1 at 18-May-2013 05
:15:53.486 CEST
Begin processing the 7th record. Run 1, Event 7, LumiSection 1 at 18-May-2013 05
:15:56.754 CEST
G4Fragment::CalculateExcitationEnergy(): WARNING
Fragment: A =  26, Z =  12, U = -3.520e-01 MeV  IsStable= 1
          P = (-9.200e+01,1.075e+02,-2.607e-01) MeV   E = 2.420e+04 MeV
```

- This was on x86-64, ARM also supported.

MULTI-THREADED GEANT4

- Geant4 version10 is being prepared for release in Dec2013
- A major feature is the possibility to simulate events in parallel on different threads
- The results reported here use the internal alpha version, which is still being heavily developed
- A public beta is expected at the end of June

ADDING DMTCP TO A G4 APPLICATION

- Compiled against the alpha version of Geant4.10 (May2013) with multithread (MT) support enabled
- Use Geant4 "user-hooks" to add checkpointing functionality
  - No need to modify or recompile Geant4 code itself

EXAMPLE HOOKS

MyUserWorkerInitialization::WorkerRunStart() and ::WorkerRunEnd()

```
void MyUserWorkerInitialization::WorkerRunEnd() const
{
   if ( G4RunManager::GetRunManager()->GetRunManagerType()
           == G4RunManager::workerRM ) {
    //If this is worker, decrease number of threads
    G4AutoLock l(&threadCtrMtx);
    --threadCounter;
    //Is the number of alive threads bwlow a threshold?
    if ( threadCounter <= 5 ) {
      //Avoid multiple checkpoints,
      if ( ! ckpRequest ) {
          ckpRequest = true;
          makeCheckPoint();
      }
    }
  }
}
```

Caution: from alpha release, names may change for final G4.10

TESTING

- To test the use of checkpointing with the multithreaded Geant4 two platforms were used
  - A standard Intel Xeon box (32 logical cores)
  - An Intel Xeon Phi co-processor (240 logical cores)
- The preliminary results show good linearity of scaling
- Minor issues stemming from the use of the alpha version: I/O optimization on co-processor not done yet

TESTING

- Any Geant4 application has an initialization portion where the following things are done:
  - reading in the geometry, e.g. from xml files
  - initializing the geometry
  - initializing the physics tables, e.g. cross section tables
- Random Number Generator (RNG) seeds are pre-generated for each event to guarantee reproducibility
- Threads are spawned and events are processed in parallel

HOT START

- The cost of this sequential initialization piece is quite high if many cores are idle waiting for it to finish, as on the Intel Xeon Phi coprocessor (60 physical cores, 240 logical cores)

- The cost of this initialization (in terms of idling other cores) can be significantly reduced by checkpointing

EXAMPLE WORKFLOW

Initialization
- Geometry Construction (GDML)
- Physics Models initialization (reading DB, building cross-section tables)

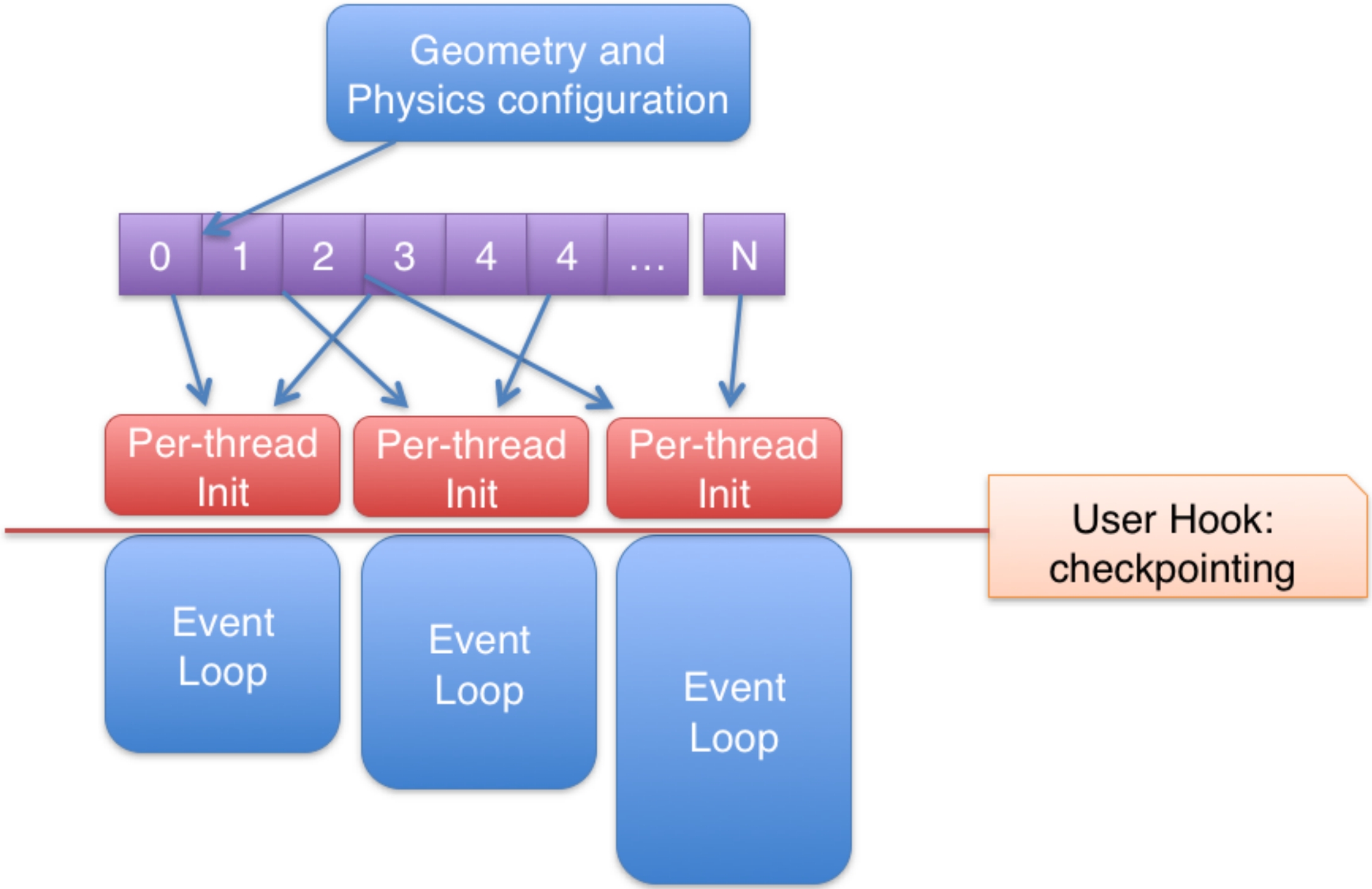
Checkpointing
- Threads are spawn
- Threads are initialized

EventLoop
- Threads process events

Checkpoint Status

Restart from This point

Restart possible in a fraction of time required for a cold start

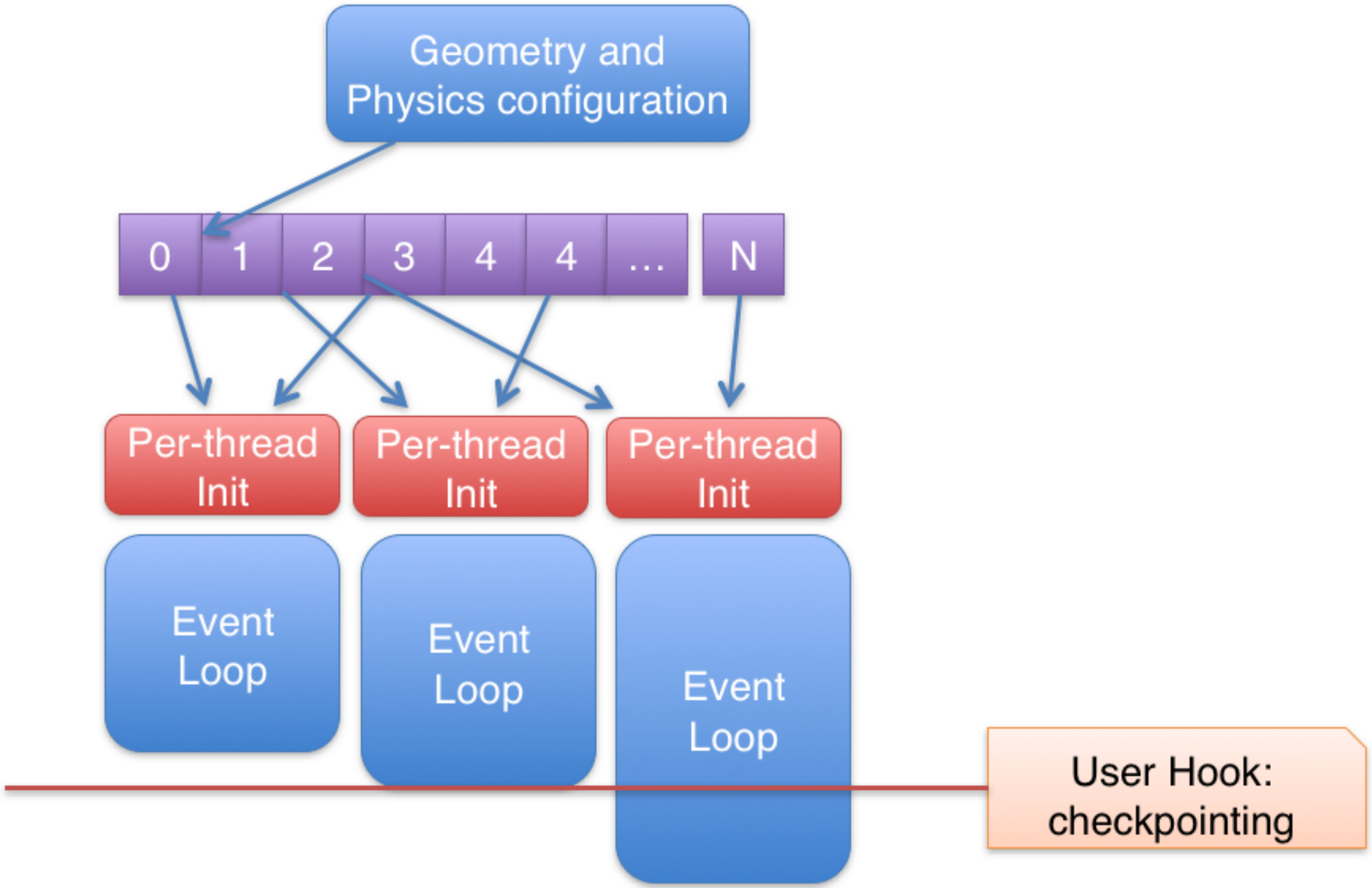

Geometry and Physics configuration
0
1
2
3
4
4
…
N
Per-thread Init
Per-thread Init
Per-thread Init
Event Loop
Event Loop
Event Loop
User Hook: checkpointing

RESULTS

- Use the "FullCMS" Geant4 benchmark application on an Intel Xeon Phi co-processor
- Checkpoint image preparation:
  - Initialization takes about 5 minutes
  - Checkpointing takes about 1 minute (I/O issue on Xeon Phi?)
  - Checkpoint image file size is 1.4GB (uncompressed)
- Restart from checkpoint image file takes less than 10 seconds
- The checkpoint image can be distributed to other nodes and the simulation process "cloned" (need to reset RNG seeds, though)

TAILS

- A simulation job will finish only when all requested events have been simulated. Some events can take more time than others.
- Some threads may finish earlier than others processing "slow" events
  - For a fraction of the job runtime, the number of active threads may be less than the number of cores assigned to the job
- Strategy:
  - checkpoint when the number of active threads drops below some threshold and kill process.
  - Start a new job and repeat
  - When enough "tails" have accumulated, start them together


Geometry and
Physics configuration
0
1
2
3
4
4
...
N
Per-thread
Init
Per-thread
Init
Per-thread
Init
Event
Loop
Event
Loop
Event
Loop
User Hook:
checkpointing

RESULTS

- Simplified test:
  - Start simplified CMS with 10 threads
  - The first 5 events process "fast" events, others "long" events
  - If active threads drop below 6, then checkpoint
  - Restart application from checkpoint and verify that all events are completed
- On real-life applications one needs to identify "fast" and "long" events, of course

G4-MT CONCLUSIONS

- Preliminary tests with an alpha version of Geant4.10 (MT-capable) show interesting results

- Checkpointing can be used to:

  - Perform "hot start" of MT applications, substantially reducing the initialization time

  - Suspend jobs with "tails" (a few long events) production systems to increase CPU utilization efficiency

- We did not need to modify Geant4 code

  - Checkpointing can be added in the application via Geant4 user-hooks, and used to trigger the checkpoint at the correct moment

SUMMARY

- We have done tests and demonstrated the the checkpointing and restart of large HEP applications (CMS, ROOT, Geant4) using the DMTCP checkpoint package

- We have demonstrated and checked basic performance of the checkpointing on x86-64, ARM and Intel MIC, including both single and multithreaded applications

- Still some work to do to bring into production use, but we expect that this technology has great potential to provide interesting new capabilities for software efficiency, debugging, etc.