Geant4MT: Maintenance for Both Geant4 and Geant4MT

Xin Dong[1], Gene Cooperman[2], and John Apostolakis[3]

[1] College of Computer and Information Science
Northeastern University, Boston, MA 02115, USA
xindong@ccs.neu.edu

[2] College of Computer and Information Science
Northeastern University, Boston, MA 02115, USA
gene@ccs.neu.edu

[3] PH/SFT, CERN, CH-1211, Geneva 23, Switzerland
John.Apostolakis@cern.ch

October 5th, 2010

Changes from Geant4 to Geant4MT

Given a computer with *k* cores, use the many-core machine in a memory-efficient scalable manner

- Event-level parallelism as the prior distributed memory parallelization
- Replace *k* independent copies of the Geant4 process with an equivalent single process with *k* threads
- Modify the source code for both the Geant4 kernel and Geant4 applications
  1. the code modification for thread safety
  2. the code modification for memory footprint reduction
  3. the code for the worker thread initialization
  4. the thread safe CLHEP interface
  5. the code for the application parallelization
  6. the thread-private malloc library

Geant4 Modification for Thread Safety

Apply the thread local storage keyword *__thread*. Currently, our method

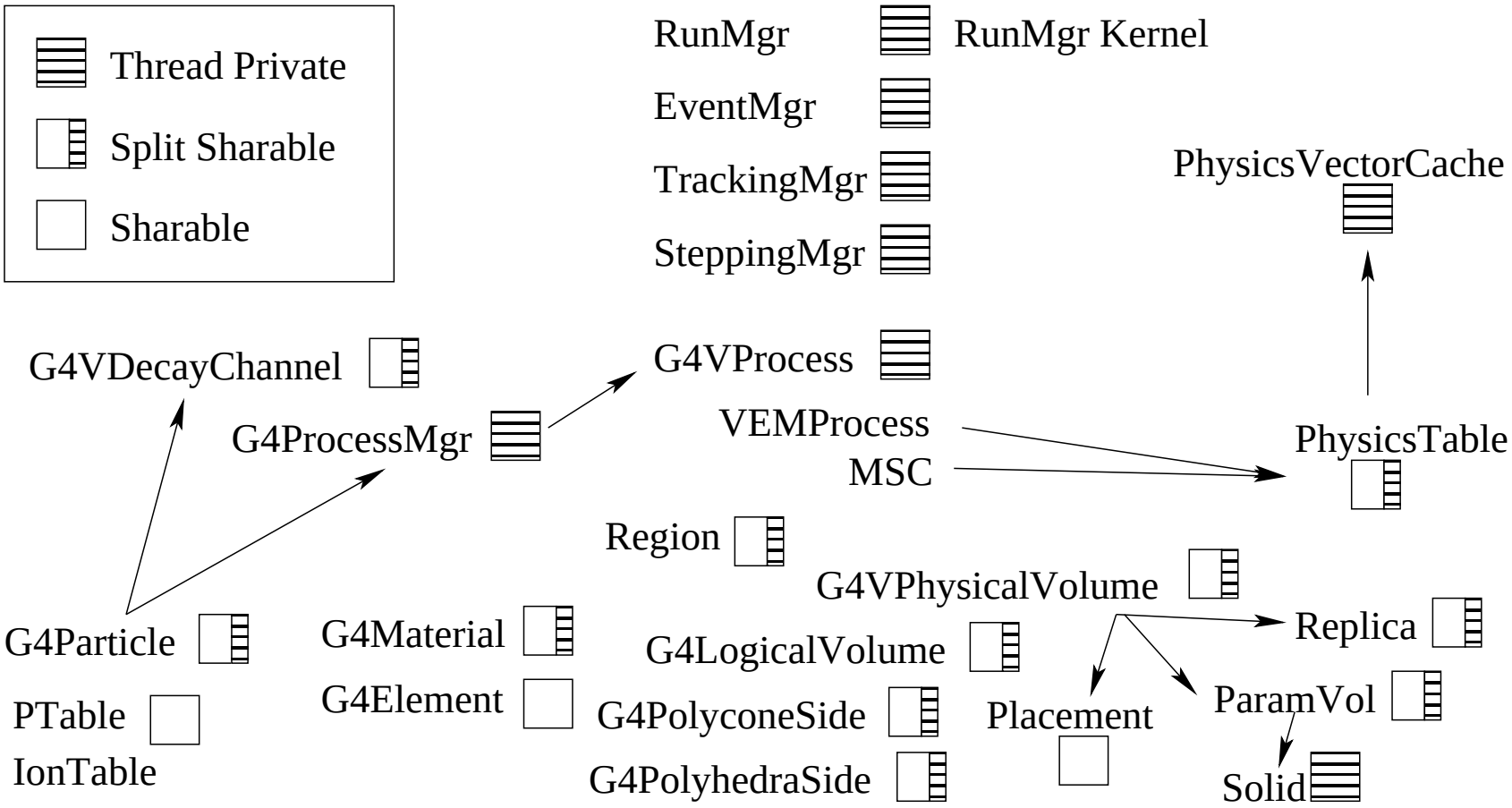
- Patch parser.c of gcc, recompile and reinstall gcc. Build Geant4 and collect: (i) static variables; (ii) static class members; and (iii) global variables and corresponding "extern" declarations
- Transform the Geant4 source code using the "__thread" keyword

Some improvement:

- A gcc plugin to replace the first step
- The style to facilitate __thread applying
  1. Eliminate static variables, static class members and global variables: const, stateless, singleton class members, etc.
  2. Make static variables, static class members and global variables POD (plain old data structure)
  3. Initialize static pointers dynamically

Geant4 Modification to Share Data

Worker threads share read-only fields for some object instances



Worker Thread Initialization I

Sharable classes: one copy of sharable instances + thread-private volatile subinstances

Initialization for thread-private volatile subinstances

Sharable Class	Volatile Class Members	0 ?	Additional Processing
G4LogicalVolume	fSolid,fSensitiveDetector	No*	
G4VPhysicalVolume G4PVReplica	frot, ftrans fcopyNo	No*	 SlaveG4PVReplica()* If Parameterised, Solid clone*
G4PolyconeSide	fPhi	Yes	
G4PolyhedraSide	fPhi	Yes	
G4ParticleDefinition	theProcessManager	Yes	
G4VDecayChannel	parent, daughters parent_mass, daughters_mass	Yes	
G4Region	fFastSimulationManager fRegionalSteppingAction	Yes	
G4Material	fIonisation, fSandiaTable	No*	Per Instance SlaveG4Material()*
G4PhysicsVector	cache	Yes	

Worker Thread Initialization II

G4VEnergyLossProcess or G4VMultipleScattering instances: multiple copies to share physics tables respectively

Initialization for side-effect by skipping physics table accesses

G4VUserPhysicsList: enhance AddProcessManager*, BuildPhysicsTable* and PreparePhysicsTable*

To support the physics tables&vectors sharing via the dynamic cast mechanism to filter out G4VMultipleScattering or G4VEnergyLossProcess instances for additional handling.

- the master thread setup a shadow process manager pointer–firstProcess
- each thread worker call SlavePreparePhysicsTable(firstProcess)** for qualified instances
- each thread worker call SlaveBuildPhysicsTable(firstProcess)** for qualified instances

Worker Thread Initialization III

Change G4RunManager* and G4RunManagerKerne*

To direct the initialization for worker threads: in SetPhysics* method, worker threads skip ConstructParticles

- the process manager is thread private
- each worker thread initializes its private process manager
- initialization follows the same way as the master thread
- physics processes are created again by each worker thread.

Currently, we use a patch to maintain all those changes mentioned above. This is because new methods is hard to generate automatically.

We push all these changes into the repository as new methods to support Geant4MT and maintain them by all developers.

A side-effect for the parallelism based on copy-on-write: memory footprint is reduced no matter whether Geant4MT is thread-safe or not.

Thread-Safe Interface to CLHEP

The current interface to CLHEP

- CLHEP, maintained outside of the Geant4 kernel, is crucial for Geant4MT
- Geant4 uses CLHEP via a static interface, that is not thread-safe
- We have to make CLHEP thread-safe, following the same methodology as used in Geant4MT.

A thread-safe interface to CLHEP

- G4MTHepRandom, multithreaded HepRandom class
- Distributes required by Geant4, lifted from CLHEP into Geant4MT as child classes of G4MTHepRandom
- Geant4MT will be installed with original CLHEP

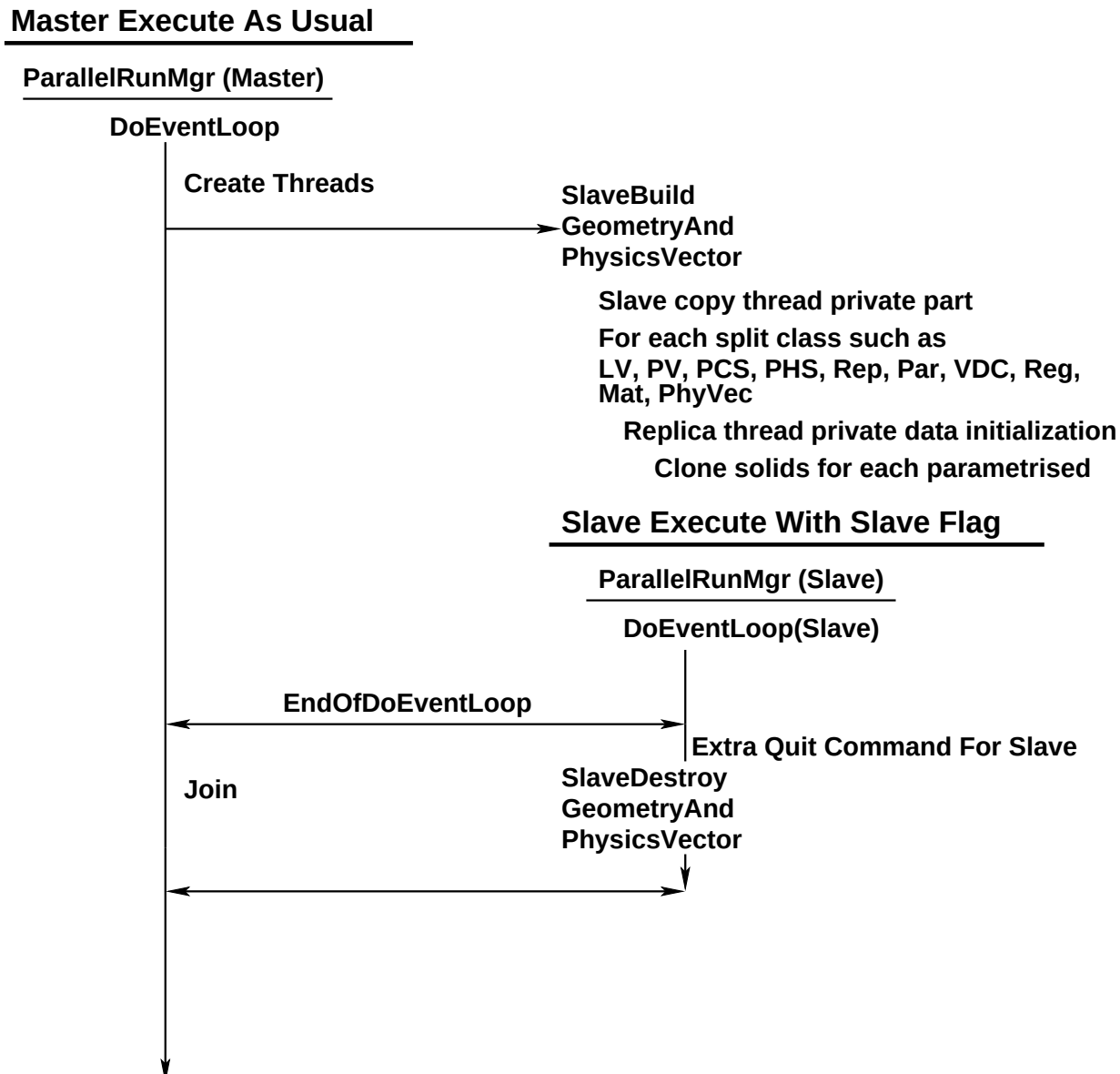
Thread-Safe Interface to CLHEP (Cont.)

Implementation for the thread-safe interface to CLHEP

- Introduce new matros to unify the name for CLHEP methods
- Copy and change some CLHEPMT code
- Redefine matros to use the thread-safe version of CLHEP methods

CLHEP Class	Geant4 Macro	Current Definition
HepRandom*	G4UniformRand()	G4MTHepRandom::getTheEngine()− >flat()
RandBit*	G4RandBit*	G4MTHepRandom::shootBit
RandExponential*	G4RandExponential*	G4MTHepRandom::shoot
RandGamma*	G4RandGamma*	G4MTHepRandom::shoot
RandFlat*	G4RandFlatArray* G4RandFlatInt* G4RandFlat*	G4MTHepRandom::shootArray G4MTHepRandom::shootInt G4MTHepRandom::shoot
RandGauss* RandGaussQ*	G4RandGauss*	G4MTHepRandom::shoot
RandGeneral*	G4RandGeneralTmp* G4RandGeneral*	G4MTHepRandom::RandGeneral G4MTHepRandom::RandGeneral

Parallelization for Applications (Large Picture)



Parallelization for Applications (Cont.)

In a fashion similar to the parallelization on distributed-memory.

- A new main function and a thread function as wrappers
- Minor change in the real application main function to coordinate two phases of initialization
- Worker threads spawned by DoEventLoop in parallel run manager
- User decides: pattern for parallel simulation and result aggregation
- Thread safety

Thread-private malloc library

- Using a separate malloc arena
- Under the assumption: thread allocates memory, then the same thread will free it
- Satisfied by task-oriented parallelism

Parallelization for Applications (Cont.)

Output demangling

- A child class for the G4coutDestination class
- Each thread creates one instance of this child class
- This instance is associated to G4coutbuf and G4cerrbuf
- Redirect the output to a private file

Debug two kind of possible error introduced by Geant4MT

- Data initialized incorrectly by work threads
  —Urdb to compare the execution from the single worker thread case with the execution from the original sequential case
- Shared data written by work threads (Data race)
  —Memory write protection for the master heap part