# Including Parallelism in Geant4

Gene Cooperman

Northeastern University
Boston, MA, USA

(Geant4 Workshop, SLAC)

# Geant4 Class Hierarchy

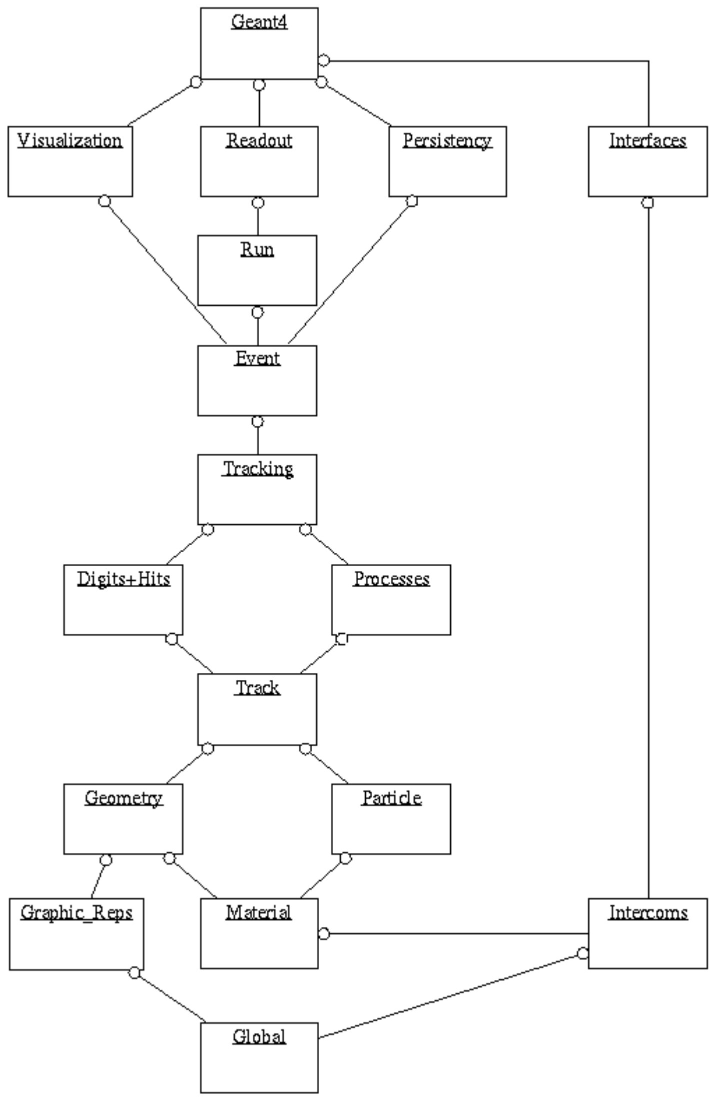



Figure 1: Geant4 classes

# Basic Parallel Approach

**Master-Worker Paradigm:** Master sends out tasks for workers to do. Workers report back to master with results. Communication is in form of star topology with master at center.

**SPMD (Single Program Multiple Data):** Master and workers each execute a single program from beginning with same initial data. Statements of form `if (is_master())` and `if (is_worker())` are used to enable them to process different data.

**Distributed Memory:** Master and workers operate in distinct processes with distinct memory. Communication is by messages sent among processes. Examples: MPI (Message Passing Interface), PVM (Parallel Virtual Machine), Corba, RMI, ...

**Binary compatible:** No need to re-compile Geant4 libraries. The original sequential Geant4 design is also flexible enough to accomodate parallelism *without modification or re-compilation*.

# Event Parallelism vs. Track Parallelism

**Two common approaches to parallelism in Geant4:**

**Event Parallelism:** Master sends event to a worker. Worker processes it and returns hits.

**Track Parallelism:** Master sends primary track to a worker. Worker processes it and returns its secondary tracks.

## Calling from main() *(Event Parallelism)*

```
int main(int argc, char** argv) {
```

  *[ Initialize parallel package ]*

```
  G4RunManager * runManager = new G4RunManager;
```

  *[ ParRunManager * runManager = new G4RunManager;]*

```
  //Initialize G4 kernel
  runManager->Initialize();
  delete runManager;
  if (argc==1)
  // Define (G)UI terminal for interactive mode
  { // G4UIterminal is a (dumb) terminal.
    G4UIsession * session = 0;
    session = new G4UIterminal();
    ...
    delete session; }
  // Batch mode
  { G4String command = "/control/execute ";
    G4String fileName = argv[1];
    UI->ApplyCommand(command+fileName); }
```

  *[ Finalize parallel package (cleanup) ]*

```
  return 0;
}
```

# Parallel Run Manager

```
class ParRunManager : public G4RunManager
{
  protected: // with description
```

*[ We shadow original DoEventLoop with our parallel version]*

```
    virtual void DoEventLoop(...);
```

*[ Add member functions for actions by master and workers ]*

```
}
```

User command `BeamOn()` will eventually call
`RunInitialization()`
and
`DoEventLoop()`.

Since
`RunManager::DoEventLoop()`
is virtual, we have shadowed it with our own version,
`ParRunManager::DoEventLoop()`

## Sequential DoEventLoop()

```
void RunManager::DoEventLoop(G4int n_event, ...)
{ ...
  for( i_event=0; i_event<n_event; i_event++ )
  {
    stateManager->SetNewState(EventProc);

    currentEvent = GenerateEvent(i_event);

    eventManager->ProcessOneEvent(currentEvent);

    AnalyzeEvent(currentEvent);

    if(i_event<n_select)
      G4UImanager::GetUIpointer()->ApplyCommand(msg);
    stateManager->SetNewState(GeomClosed);
    StackPreviousEvent(currentEvent);
    currentEvent = 0;
    if(runAborted) break;
  }
  ...
}
```

## Parallel DoEventLoop()

```
void ParRunManager::DoEventLoop(G4int n_event, ...)
{
  If I'm the master:

   for( i_event=0; i_event<n_event; i_event++ ) {
     stateManager->SetNewState(EventProc);
     currentEvent = GenerateEvent(i_event);

     Send event to worker, and receive hits

     AnalyzeEvent(currentEvent);
     if(i_event<n_select)
       G4UImanager::GetUIpointer()->ApplyCommand(msg);
     stateManager->SetNewState(GeomClosed);
     StackPreviousEvent(currentEvent);
     currentEvent = 0;
     if(runAborted) break;
   }
   Tell all workers we're done

  If I'm a worker:
    Receive event

     eventManager->ProcessOneEvent(currentEvent);

     Send hits back to master
  Until master says we're done

}
```

# Issues

**Marshalling:** How to copy hits, primary events, secondary events across network.

**Histograms:** After computation of data, merge results from all workers.

**Interactivity:** User issues commands only on master. In SPMD style, Master must pass user commands to all workers for common initialization.

**Random Numbers:** (see next slide)

**Track Level Parallelism:** (see following slide)

**Uniform Format for Tracker Hits and Calorimeter Hits:** Highly desired. Currently, end user must write his or her own marshaling for hits not in uniform format.

**Uniform Format for Histograms ??**

# Random Numbers

**Issues:**

1. Each process must draw distinct random numbers.
2. Repeatability between parallel runs with same seed.
3. Repeatability between parallel runs with n workers and a parallel run with one worker.
4. Repeatability between parallel versin and standard sequential version.

**Solution to 1, 2 and 3:** Send new seed or random state to each worker, along with each task. For example, generate a random seed or state based on event id.

# Track Level Parallelism

Track level parallelism is desirable if there is only one event or very few events.

Event Manager in charge of processing primary tracks, and collecting secondary tracks produced thereby. It controls the work dispatcher for tracks.

Use `G4EventManager::GetEventManager()` to get pointer to Event Manager. This works because Event Manager is a *singleton class* (at most one object in class).

The class `G4PrimaryTransformer` is used exclusively by `G4EventManager`. It produces primary tracks from an event. The logic can be modified to produce an event from a track.

Hence, each event can be run locally on master, until many secondary tracks are produced. The secondary tracks can then be converted into events. After that, one proceeds exactly as in event level parallelism.

# ParGeant4: TOP-C parallelization of Geant4

ParGeant4 implements ideas of previous slides. The code will be in `examples/extended/parallel` in Geant4 4.1. You can preview it now from the ParGeant4 Home Page:
`http://www.ccs.neu.edu/pargeant4.html`
It is based on TOP-C (Task Oriented Parallel C/C++). It is free, open source software. It is available at
`http://www.ccs.neu.edu/home/gene/topc.html`

## Features of ParGeant4:

1. *Pre-compiled Geant4 kernel libraries used.* No need to re-compile to install ParGeant4.
2. *Event level parallelism* implemented
3. *Random number generators* parallelized using prop's 1, 2, 3
4. *Histograms* handled (providing AnalysisManager::Step() is trivial; single histogram produced for all events; no need for merging histograms from independent runs)
5. *Marshaling* support for user-defined hits (required for parallelizing Geant4 app; hits must be reported from worker to master)
6. *Examples* written parallelizing ParN02 and ParN04

## TODO List for ParGeant4:

1. *Interactive input.* (Input currently from script/macrofile.)
2. *Track level parallelism.*
3. *Other:* TrajectoryStore, PostponeOneEvent(), ...

## TOP-C approach

```
void ParRunManager::DoEventLoop(G4int n_event, ...) {
  TOPC_OPT_trace_input = trace_event_input;
  G4StateManager* stateManager
    = G4StateManager::GetStateManager();

  // Make variables accessible to TOP-C callback functions
  ImportDoEventLoopLocals( stateManager, n_event, n_select,
                           msg );

  // This is where all the parallelism occurs
  TOPC_master_slave(MyGenerateEventInput, MyDoEvent,
                    MyCheckEventResult, NULL);
}

MyGenerateEventInput:
    for( i_event=0; i_event<n_event; i_event++ ) {...}
MyDoEvent:
    eventManager->ProcessOneEvent(currentEvent);
MyCheckEventResult:
    AnalyzeEvent(currentEvent);
    if(i_event<n_select)
      G4UImanager::GetUIpointer()->ApplyCommand(msg);
    stateManager->SetNewState(GeomClosed);
    StackPreviousEvent(currentEvent);
    currentEvent = 0;
    if(runAborted) break;
```

# Advantages of TOP-C

1. Modular: different communication layers for different architectures
2. Higher level interface than traditional message passing. Only need to declare "callback functions" (MyGenerateEventInput, MyDoEvent, MyCheckEventResult) TOP-C automatically uses message passing library, such as MPI, to generate messages.
3. Faster development time by using a higher level abstraction:

   (a) A crude version (parallelizing tracks only, not hits) was implemented in about one week.

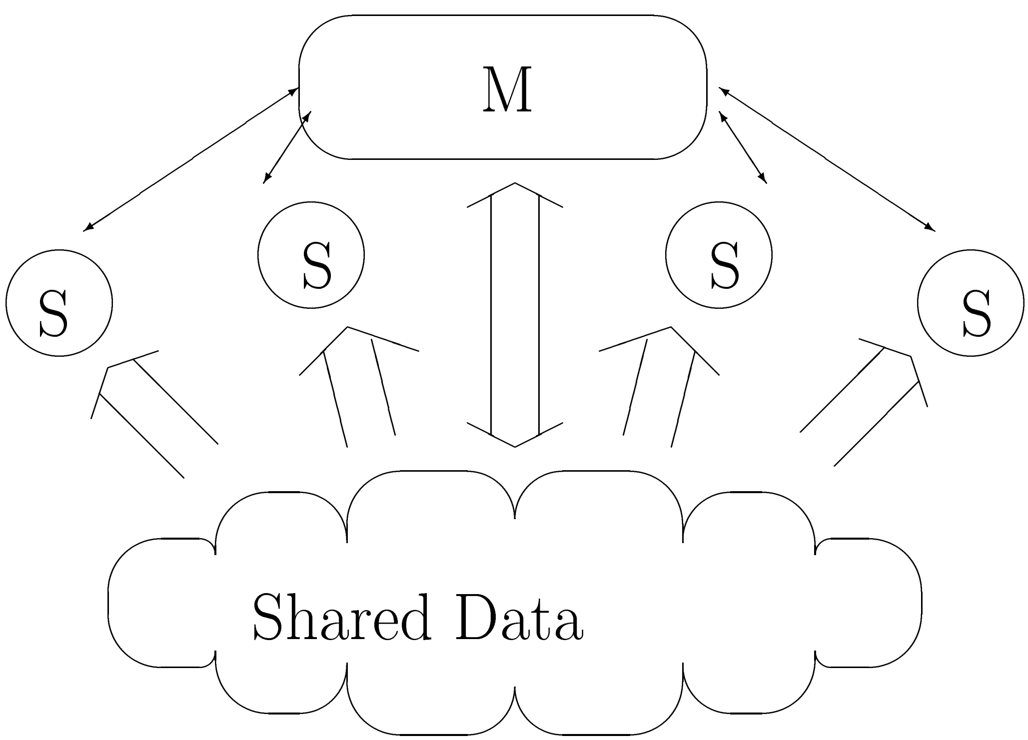
   (b) The current version includes only 450 lines of new code and the TOP-C library.

   (c) 250 of the 450 lines was for marshalling. A new package to *easily* generate marshalling code is being developed.
4. A Grid communication layer exists with Geant4 over TOP-C over Grid/Ampic:
   Casanova, Cooperman, Hayes and Witzel, "Using TOP-C and Ampic to Port Large Parallel Applications to the Computational Grid", *Proc. of $2^{nd}$ IEEC/ACM Interantional Symposium on Cluster Computing and the Grid (CCGrid)*, 2002, to appear
5. Command line interface for parallel binary. It allows one to adjust number of workers, tracing of messages, merging messages for better network efficiency, etc.
   For example: `./a.out --TOPC-num-slaves=5 --TOPC-trace=0`

## Approach Using TOP-C

TOP-C = Task Oriented Parallel C/C++
`http://www.ccs.neu.edu/home/gene/topc.html`

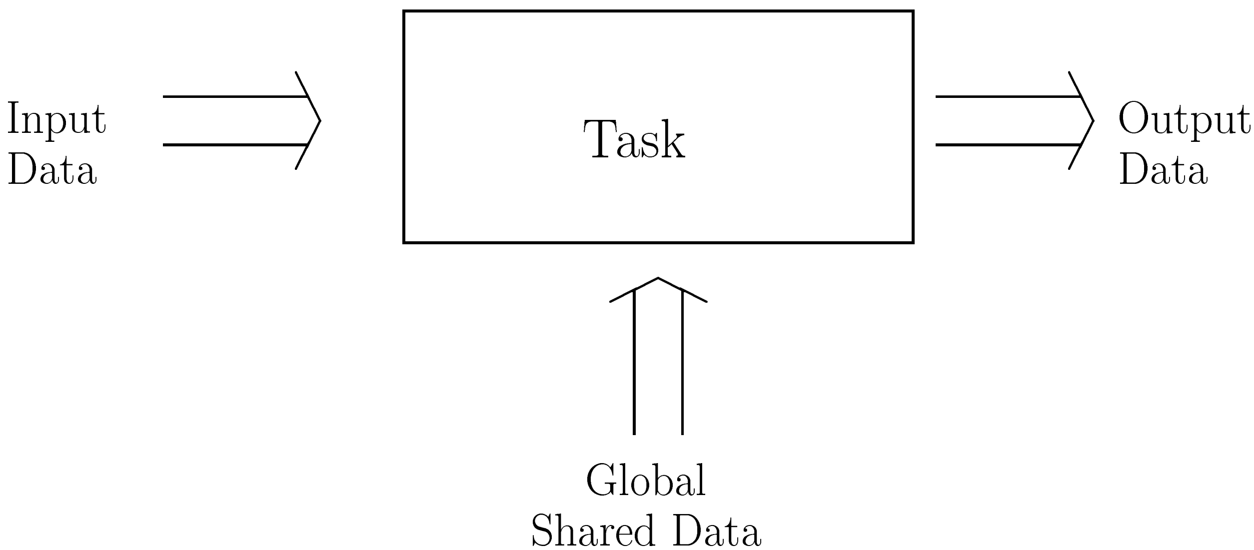
Free open source software (GNU LGPL license)
"Footprint" of approximately 40 KB as stripped binary



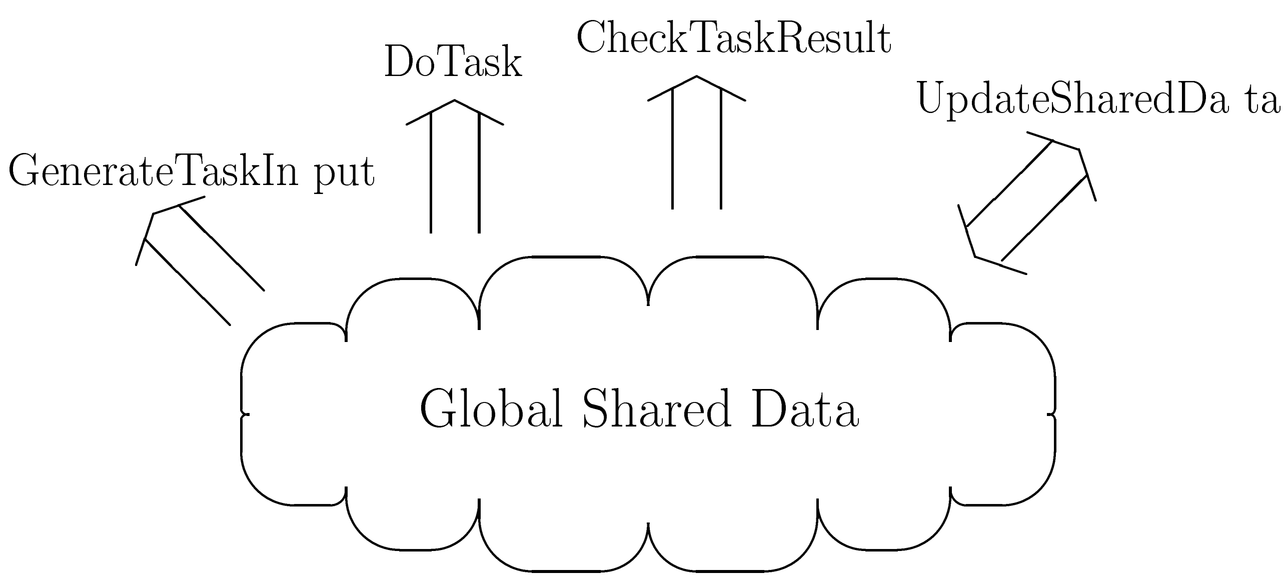
# TOP-C Model

TOP-C = Task Oriented Parallel C/C++
`http://www.ccs.neu.edu/home/gene/topc.html`

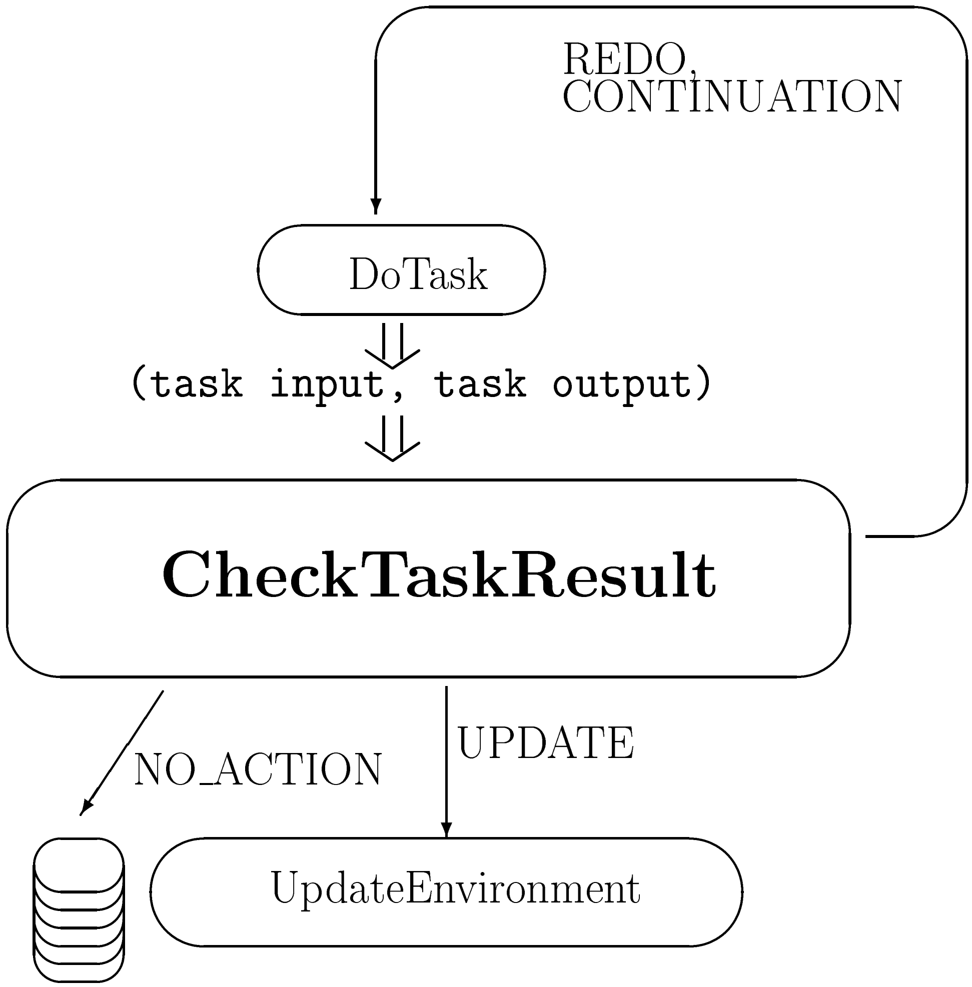
**Concept 1: Task**




**Concept 2: Shared Data**




**Concept 3: Action (implements parallel strategy)**


REDO,
CONTINUATION
DoTask
(task input, task output)
CheckTaskResult
NO_ACTION
UPDATE
UpdateEnvironment

## Track-Level Parallelism

- Each worker executes a task
- The input to the task is a primary track
- The output of the task is all secondary tracks (and all detector hits)
- The global shared data includes all ParticleDefinition's, PreAssignedDecayProduct's, Touchable's, CreatorProcess, UserInformation, etc.
  *NOTE: TOP-C does not need to know where global shared data is stored*

## Lessons Learned:

- Geant4 can be parallelized primarily by modifying G4EventManager.cc
- The same principles could be applied to implement event-level parallelism
- By using the TOP-C library, relatively little effort is required to parallelize Geant4. A crude version (parallelizing tracks only, not hits) was implemented in about one week.
- The main difficulty (for an outsider) is to understand and take advantage of the Geant4 design decisions and coding style.

## Features of TOP-C model

- Easy to Use (small number of primitives, primarily to define the task)
- Same Geant4/TOP-C source code works with three TOP-C communication layers:
  1. Sequential (emulation by single process, easy to debug)
  2. Distributed Memory (over sockets, or over MPI)
  3. Shared Memory (over POSIX sockets)
- Easy to parallelize sequential code
  1. No requirement to declare to TOP-C the global shared data
  2. about 200 new lines of TOP-C related Geant4 code
  3. about 250 – 550 lines for marshalling Geant4 classes:
- Simple Calling Parameters

  (Assume `a.out` compiled and linked with TOP-C libraries)

  ```
  ./a.out --TOPC-help
  ./a.out --TOPC-num-slaves=5 --TOPC-verbose
          --TOPC-trace=0 --TOPC-safety=10 ...
  ```
- Easy To Install (./configure; make)
- 40-page manual (on-line and postscript; examples, advanced features, etc.)