Including Parallelism in Geant4

Gene Cooperman

Northeastern University
Boston, MA, USA

(Geant4 Workshop, SLAC)
Geant4 Class Hierarchy
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_slave())` 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 accommodate 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
        { // G4UItterminal is a (dumb) terminal.
            G4UIsession * session = 0;
            session = new G4UItterminal();
            ...
            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.

Track Level Parallelism: (see next slide)
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.
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
    ExportDoEventLoopLocals( stateManager, n_event, n_select, m_data);

    // 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(GemClosed);
      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:

5. Command line interface for parallel binary. It allows one to adjust number of slaves, 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)
Track-Level Parallelism

- Each slave 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 250 new lines of TOP-C related Geant4 code
  3. about 550 lines for marshalling Geant4 classes:
     (a) G4ElectronOccupancy
     (b) G4DynamicParticle
     (c) G4Track
- 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.)
Issues in Parallelizing Geant4

• **minor issue:** a few missing Set/Get functions for certain members

• **pointer members:** (difficult to marshal)
  1. **G4DynamicParticle:** theParticleDefinition, thePreAssignedDecayProducts
  2. **G4Track:** fpTouchable, fpNextTouchable, fpStep, fpLVAtVertex, fpCreatorProcess, fpUserInformation

• parallelization of user function **ProcessHits()**
  
  **– PROBLEM:**
  1. Two tracks are processed on two different slaves
  2. Each track generates a HitsCollection on that slave
  3. Each slave passes back to the master its HitsCollection
  4. **PROBLEM:** The master needs to combine the two HitsCollections

  **– EXAMPLE from novice/N04:**
  1. detector is calorimeter
  2. if two tracks strike the same cell, they have the same HCID (Hits Collection ID),
  3. in sequential code, the user function ProcessHits() sums the two energies and store only a single G4VHit;
  4. But on the master, information from aStep is no longer available, and so ProcessHits() can’t be used.