First Results in a Thread-Parallel Geant4

Gene Cooperman and Xin Dong
College of Computer and Information Science
Northeastern University
360 Huntington Avenue
Boston, Massachusetts 02115
USA
{gene,xindong}@ccs.neu.edu

Joint with High Performance Computing Lab
at Northeastern University
Kapil Arya
Daniel Kunkle
Eric Rubinson
Vlad Slavici
Ana Maria Visan

Organization of Talk

1. ParGeant4 (2000 – 2005: Distributed Geant4 on a Cluster)
2. Motivation for Thread-Parallel Geant4
3. Review: Memory Layout in Single-threaded and Multi-threaded Processes
4. Thread-Parallel Geant4: from case analysis to roadmap for the future

5. DMTCP: Transparent Checkpointing for Thread-Parallel Geant4
   - Works for general Linux applications: also handles distributed processes, forked processes, other examples

Typical Usage of Geant4

G4RunManager: start a new run
  G4EventManager: simulate event
    G4TrackingManager: simulate track of event
      G4SteppingManager: simulate one step of track

At the end of each step, Geant4:

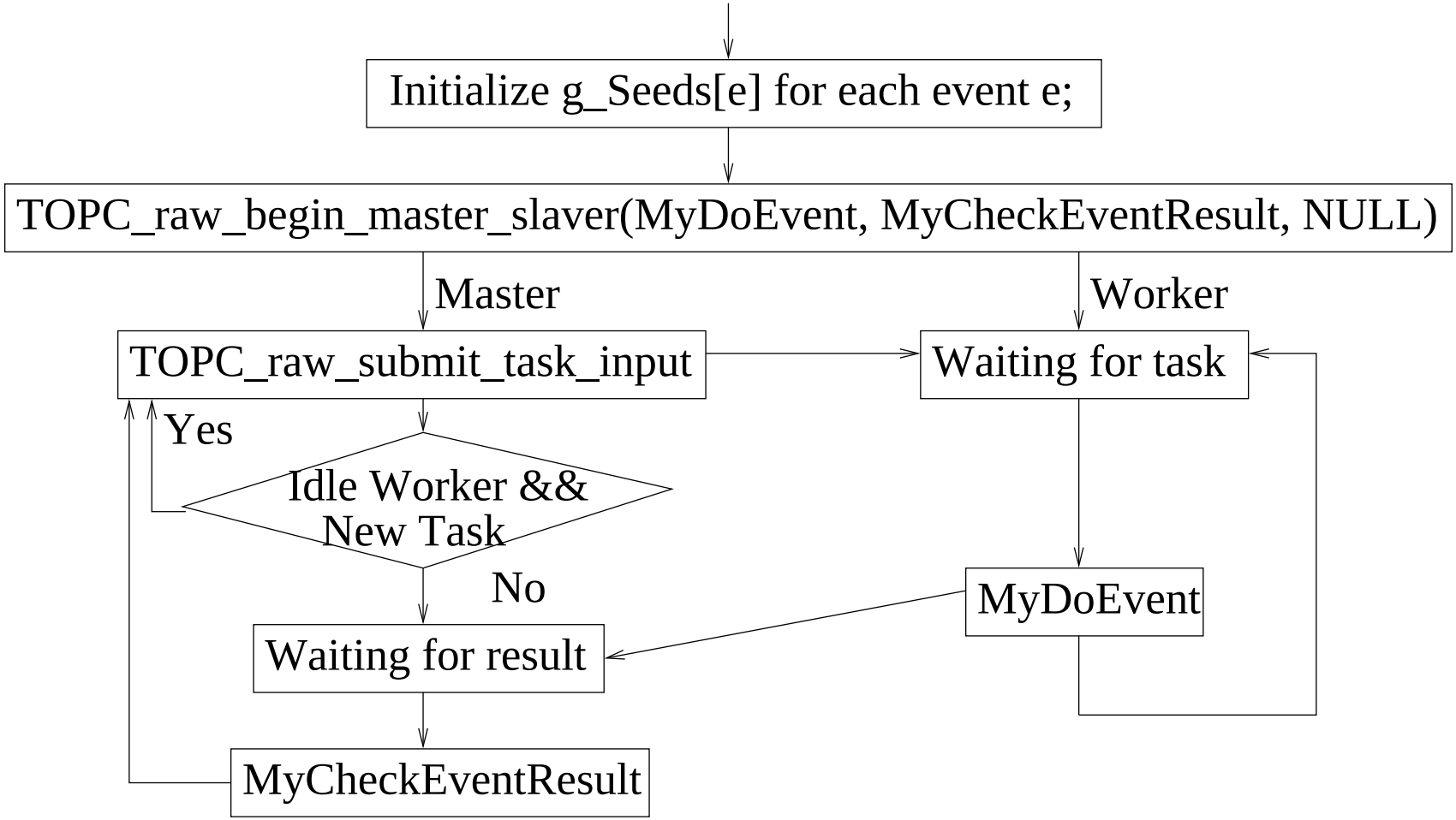
1. invokes the G4Navigator for physical coordinates

2. invokes *physics processes* from the *physics list* to modify particle characteristics (energy, momentum, ...), create secondary particles and tracks, etc.

ParGeant4 (Previous Work: 2000 – 2005

- A distributed parallel version of Geant4 (for computer clusters)
  - `examples/extended/parallel/ParN02` (and ParN04) at `http://geant4.web.cern.ch/geant4/G4UsersDocuments/UsersGuides/ForApplicationDeveloper/html/Examples/parallel.html`
  - Master/Worker paradigm
- Utilize TOP-C "Raw Interface": `http://www.ccs.neu.edu/home/gene/topc.ht`
- Event-level parallelism to simulate separate events on remote processes
- *Reproducibility:* Given same initial random seed, ParGeant4 produces same result.
  - *Current implementation of reproducibility:*
    - For each event, there is a corresponding seed for CLHEP random number generator
    - Seeds come from a sequence of random numbers on master
- ParGeant4 (like Geant4) summarizes hits in histogram for later analysis.

Parallelized DoEventLoop



Features of ParGeant4

- The master and the workers are processes using distributed-memory model
- No need to recompile Geant4 kernel
- Aggregated-tasks: `--TOPC-aggregated-tasks=5` means send 5 events per worker task
- Master summarizes hits as in sequential Geant4 for later analysis
- Nearly linear speed-up (depending on Geant4 application)
- Results are reproducible

Motivation for Thread-Parallel Geant4

By using a quad-core processor (or 8- or 16-cores), one can run four processes (or 8 or 16) simultaneously at full speed. But such usage may require four times (or 8 or 16) as much RAM. It also stresses the bus between CPU and RAM with four (or 8 or 16) times the traffic.

Three methods of using a many-core processor:

Many Geant4 Processes: Memory footprint/traffic to RAM grows with number of cores.

Forked Child Geant4 Processes: Coarse-grained data sharing; UNIX *copy-on-write*:

- A child process initially shares the parent's address space. If a process (parent or child) writes to a memory page, then a private copy of the memory page is created. Unfortunately, if one writes to even a single byte on a memory page (typically 4 KB), then a copy of the entire entire page is created.

Thread-Parallel Geant4: Fine-grained data sharing; many threads:

- Default: local variables (on stack) are also thread-local.
- Default: non-local variables are shared among threads.
- However, some non-local variables (e.g. `theStateManager`) must be thread-local (different for each thread), *instead* of the default (thread-shared). See following slides for a solution.

Running times and Image Size

NO2/run2.mac (Last run changed to 5000 events, instead of 1 event)

Four processes/threads implies one master and three workers: *one should expect a three-fold speedup compared to the single worker in the sequential case.*

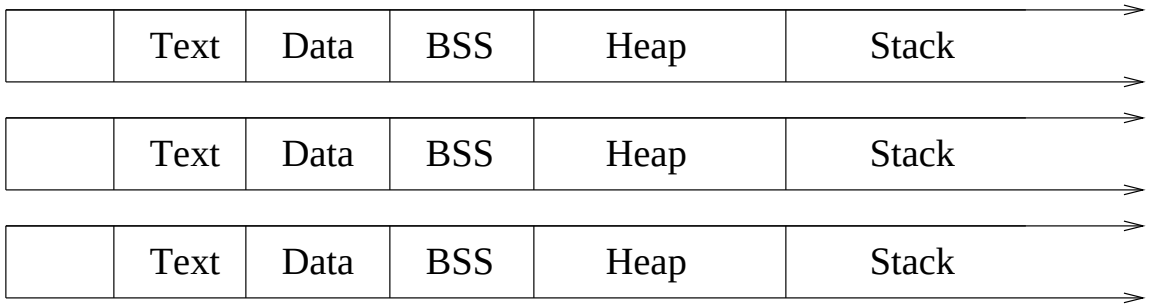
Model	Total Image Size	Time	checkpoint time	Restart time
sequential	26.5MB	71.8s	0.7s	0.3s
sequential compressed	6.3MB		0.8s	0.3s
4 threads	50MB	24.5s	0.2s	0.1s
4 threads compressed	12.6MB		1.4s	0.5s
4 processes MPICH-2	72.0MB	28.8s	1.0s	0.5s
4 processes MPICH-2 compressed	15.3MB		3.0s	1.0s

Table 1: Experimental results for checkpointed ParN02

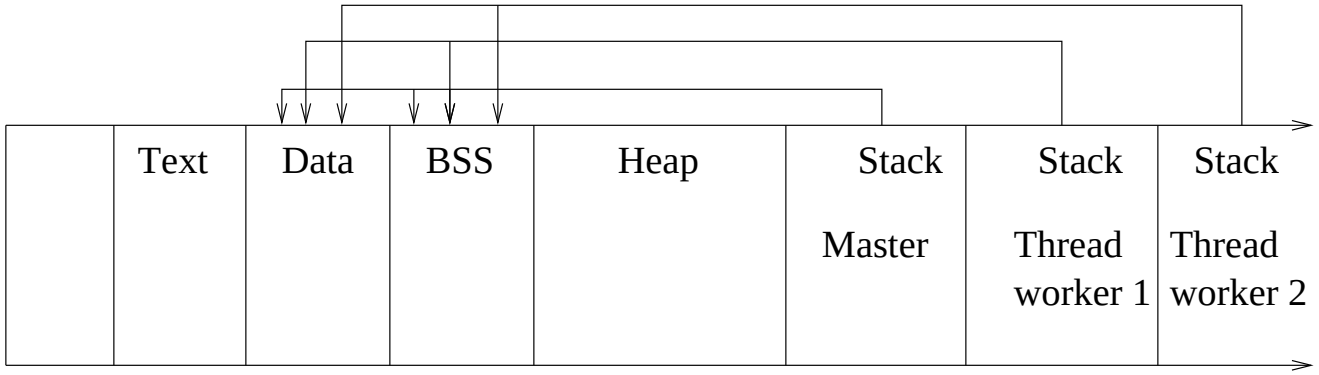
Review: Process Image Layout

- Text: This segment, also known as the code segment, holds the executable instructions of a program
- Data: This section holds all initialized data. Initialized data includes statically allocated and global data that are initialized
- BSS: This section holds uninitialized data
- Heap: This is used to grow the linear address space of a process
- Stack: This contains all the local variables that get allocated

Process master and workers:



Review: Thread master and workers



- With multi-core CPUs, worker threads share data that can be shared, thus using much less memory; fewer bottlenecks to RAM;

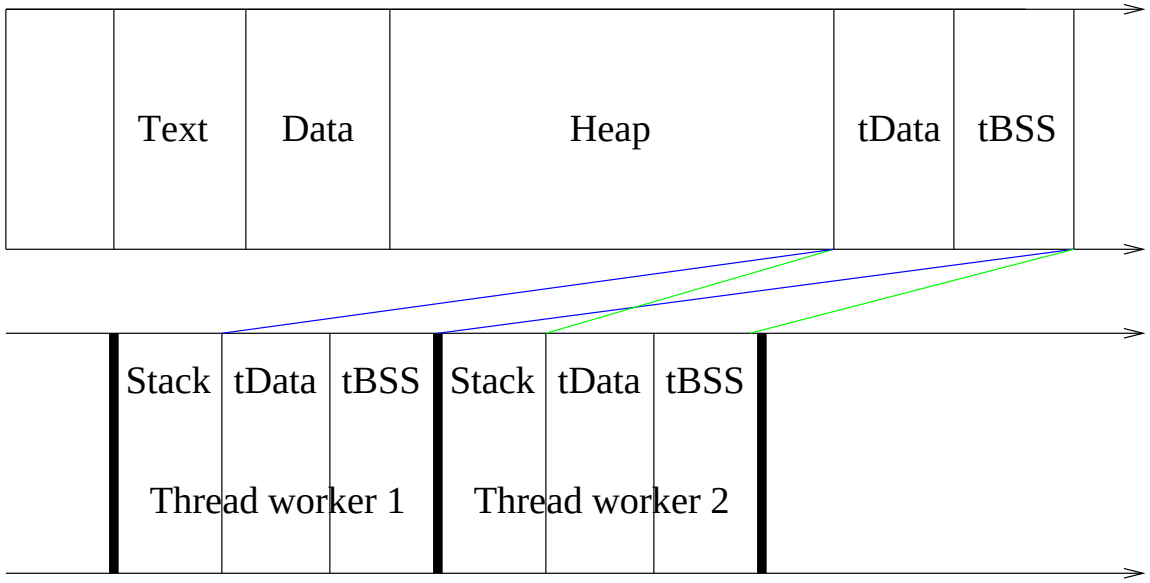
Review: Thread local storage (TLS): An example


```
#include <stdio.h>
#include <pthread.h>
__thread int gvar = 0; //int gvar = 0;
void * increase(void *)
{   gvar++;
    printf("Value in child thread: %d\n", gvar);
}
int main(int argc, char * argv[])
{  pthread_t tid;
    printf("Value in main thread: %d\n", gvar);
    pthread_create( & tid, NULL, increase, NULL );
    pthread_join(tid, NULL);
    printf("Value in main thread: %d\n", gvar);
    return 0;
}
```

```
Value in main thread:  0
Value in child thread:  1
Value in main thread:  0
```

Review: The usage of thread local storage (TLS)

- tData and tBSS segments



- statically initialized thread data does not support dynamic initialization
- "`static __thread int i = j;`" is not correct when j is a variable

Thread-local Geant4: 3 cases to handle

1. *static class members:*

```
class G4StateManager
{ ···
   static G4StateManager * theStateManager;
   ··· }
```

2. *static local variables:*

```
G4double G4Navigator::ComputeStep(···) {
   static G4int sNavCScalls=0;
   ··· }
```

3. *global variables:*

```
G4Allocator< G4NavigationLevel> aNavigationLevelAllocator;
G4Allocator< G4NavigationLevelRep> aNavigLevelRepAllocator;
```

. . . and a very few special cases

How to initialize a TLS variable dynamically

Fun(int j)
{
static int i = j;
 	
i++;
return i;
}
	Fun(int j)
	{
	static __thread int * i_NEW_PTR_ = 0;
	if (! i_NEW_PTR_) {
	i_NEW_PTR_ = new int;
	* i_NEW_PTR_ = j;
	}
	int & i = * i_NEW_PTR_;
	i++;
	return i;
	}

The rule to add "__thread"

- Use a pointer whose name is new
- The initial value of the pointer is NULL
- Allocate space for the pointer only when the value of the pointer is NULL. Then assign the value with the original right side
- Refer to the value of the pointer using the original name
- This guarantees each variable is initialized once and only once

Automatically find what is not thread-safe

- Patch parser.c of gcc to output static and global declarations in Geant4 source code; recompile and reinstall gcc
- Build Geant4 and collect output of parser.c (similar to UNIX grep)
  1. static local variables in each function or method
  2. static class members
  3. global variables and if they exist, all corresponding “extern” declarations

Afterwards, find what is not thread-safe

- "change list" produced by modified gcc
  1. G4WeightWindowProcess.cc: 306 : static G4FieldTrack endTrack('0');
  2. G4NuclearLevelStore.hh: 54 : static G4String dirName;
  3. G4NuclearLevelStore.cc: 32 : G4String G4NuclearLevelStore:: dirName;
  4. G4AttDefStore.cc: 36 : std:: map< G4String, std:: map< G4String, G4AttDef>*> m_defsmaps;
  5. G4AttDefStore.hh: 53 : extern std:: map< G4String, std:: map< G4String, G4AttDef>*> m_defsmaps;

Automatically transform Geant4

- Follow the “change list”
- Transform the original Geant4 to be thread-safe
- Example: static local variable in a function/method

BEFORE:

```
static G4FieldTrack endTrack( ’0’);
```

AFTER:

```
static __thread G4FieldTrack * endTrack_NEW_PTR_ = 0 ;
if ( ! endTrack_NEW_PTR_ )
  endTrack_NEW_PTR_ = new G4FieldTrack (’0’) ;
G4FieldTrack &endTrack = * endTrack_NEW_PTR_;
```

Automatically transform Geant4 (cont.)

- Example: static class member

  BEFORE:

  static G4String dirName;

  AFTER:

  static __thread G4String dirName_NEW_PTR_;

  BEFORE:

  G4String G4NuclearLevelStore::dirName("");

  AFTER:

  __thread G4String G4NuclearLevelStore::dirName_NEW_PTR_ = 0;

  G4NuclearLevelStore * G4NuclearLevelStore::GetInstance()

  {if (! dirName_NEW_PTR_)

  dirName_NEW_PTR_ = new G4String("") ;

  G4String &dirName = * dirName_NEW_PTR_;

  . . . }

Special case: non-automated transformations

In a few cases, Geant4 uses additional variables that should be declared thread-local, but will not be transformed automatically. A common case is when information is cached in a non-static class member of an object, and read back later.

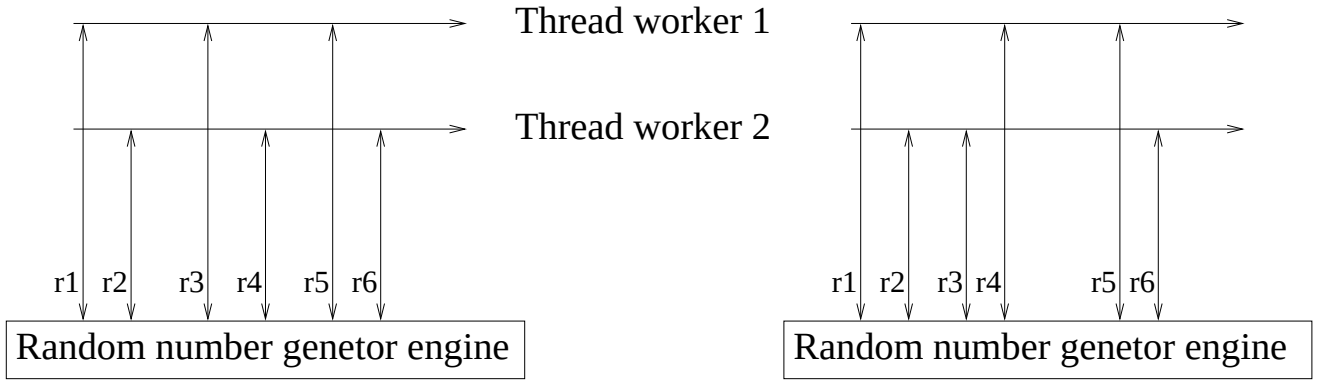
In these cases, we anticipate adding a stylized comment to the Geant4 source code, specifying to the automated tool that the variable is thread-local.

Roadmap to Use of Thread-Local Geant4

1. Source code for Geant4 is automatically transformed to thread-local code.
   - *Currently, about 10,000 lines of Geant4 code are automatically transformed to thread-local code.*
2. Binaries of the thread-local library are made available to end users similarly to current Geant4 libraries
3. End user code that defines hits, physics processes, and some other constructs must be automatically transformed. We will provide the tool to automaically transform the code.
4. *Non-automated case:* If the end-user code includes a non-automated special case, the end user must add a stylized "thread-local" comment. The automated tool will detect such comments and transform the corresponding code. A guide for finding such exceptional cases will be written.
   - *Note that for the simple examples in N02 and N04 no additional "thread-local" comments are needed.*

CLHEP: Reproducibility and Random Number Generators

- For reproducibility, each Geant4 thread requires an independent random number generator engine



- CLHEP interfaces to random engines
  1. *Static interface* through the class HepRandom (current Geant4 usage) (If it is really needed, apply the above method to make this interface thread-safe.)
  2. *Non-static interface* by defining different generator objects by means of the various engines available (alternative strategy requiring non-automatic changes to Geant4)

Checkpointing Geant4 by DMTCP

- DMTCP: Distributed MultiThreaded CheckPointing
  - GPL license: `http://sourceforge.net/projects/dmtcp`
- Checkpoint: Save state of all processes and threads to disk
- Restart: Restore state from disk
- Works for distributed and for thread-parallel Geant4 (assumes recent Linux kernel)
- User-level: No modification of O/S kernel or of (Geant4) binary
- Process migration: Can restart on different processors
- Can save on startup initialization: checkpoint after an initialization, and restart from checkpoint image in future

Checkpointing Multithreaded Geant4 (32 threads)


```
>>> Event 400
    0 trajectories stored in this event.
>>> Event 500
    0 trajectories stored in this event.
[21943] NOTE at mtcpinterface.cpp:99 in callbackPostCheckpoint; REASON='checkpoint
     dmtcp::UniquePid::checkpointFilename() = /home/xdong/geant4.9.0/examples/exte
[21943] WARNING at checkpointcoordinator.cpp:110 in postCheckpoint; REASON='JWARNI
     i->first.conId() = 99035
     i->first.conId() = 99036
>>> Event 600
    0 trajectories stored in this event.
>>> Event 700
    0 trajectories stored in this event.
```

Restarting Multithreaded Geant4 (32 threads)


```
[xdong@teracluster ParN02]$ ./dmtcp_restart_script.sh
[22782] WARNING at checkpointcoordinator.cpp:128 in postRestart; REASON='JWARNING(
Listing FDs...
0 -> socket:[91213878] inTable=0
1 -> socket:[91213878] inTable=0
2 -> socket:[91213880] inTable=0
3 -> file[3]:/home/xdong/geant4.9.0/examples/extended/parallel/ParN02/exampleN02.l
4 -> file[4]:/home/xdong/geant4.9.0/examples/extended/parallel/ParN02/exampleN02.l
5 -> file[5]:/home/xdong/geant4.9.0/examples/extended/parallel/ParN02/exampleN02.l
...
31 -> file[31]:/home/xdong/geant4.9.0/examples/extended/parallel/ParN02/exampleN02
32 -> file[32]:/home/xdong/geant4.9.0/examples/extended/parallel/ParN02/exampleN02
33 -> file[33]:/home/xdong/geant4.9.0/examples/extended/parallel/ParN02/exampleN02
34 -> file[34]:/home/xdong/geant4.9.0/examples/extended/parallel/ParN02/exampleN02
821 -> socket:[91213987] inTable=0
823 -> socket:[91213973] inTable=0
```

```
826 -> socket:[91213880] inTable=0
827 -> file[827]:/tmp/jassertlog.22782 inTable=1
     i->first.conId() = 99035
     i->first.conId() = 99036
     i->first.conId() = 99037
     ...
     i->first.conId() = 99069
Message: stale connections should be gone by now
>>> Event 600
    0 trajectories stored in this event.
>>> Event 700
    0 trajectories stored in this event.
>>> Event 800
    0 trajectories stored in this event.
>>> Event 900
    0 trajectories stored in this event.
### Run 3 start.
In DoEventLoop
### Run 3 start.
### Run 3 start.
### Run 3 start.
### Run 3 start.
### Run 3 start.
### Run 3 start.
### Run 3 start.
```

Example applications and experimental results

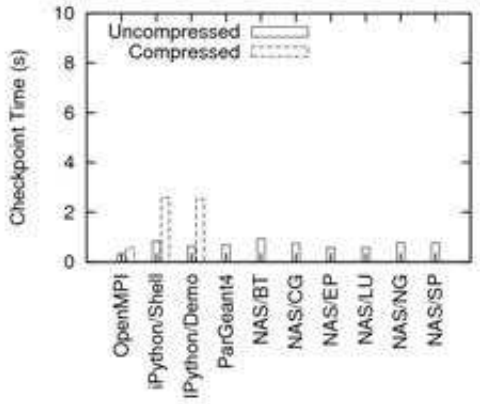

```
dmtcp_coordinator
dmtcp_checkpoint ParN02 --TOPC-num-slaves=31 --TOPC-trace=0 exampleN02.large_N.i
(or dmtcp_checkpoint MPI -n 32 ParN02 exampleN02.large_N.in)
... Machine crashes ...
dmtcp_restart_script.sh  [ Script generated by checkpoint ]
```

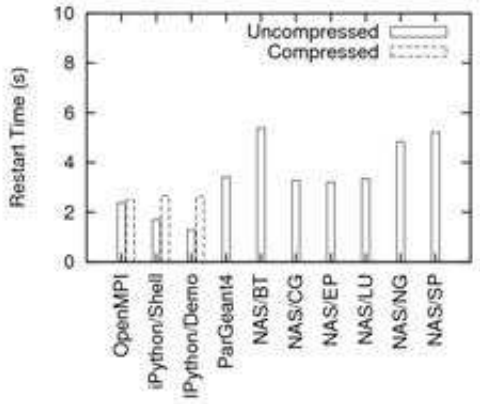
Model	File format	Total Image Size	Checkpoint time	Restart time
Sequential	Compressed	6.3MB	0.8s	0.3s
Sequential	Uncompressed	26.5MB	0.7s	0.3s
32 threads	Compressed	65.1M	5.5s	2.6s
32 threads	Uncompressed	204.5M	0.6s	0.4s
32 processes MPICH-2	Compressed	1.3G	2.2s	3.6s
32 processes MPICH-2	Uncompressed	5.6G	0.8s	3.2s

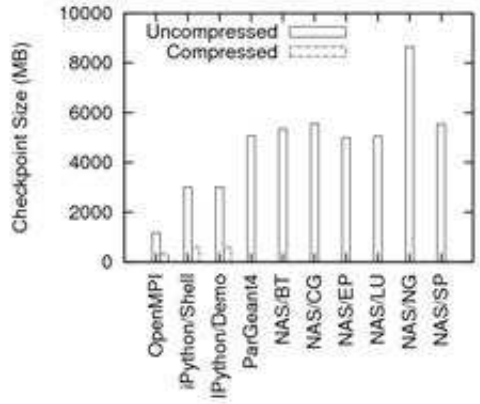
Table 2: Experimental results for checkpointed ParN02 with 32 processes/threads

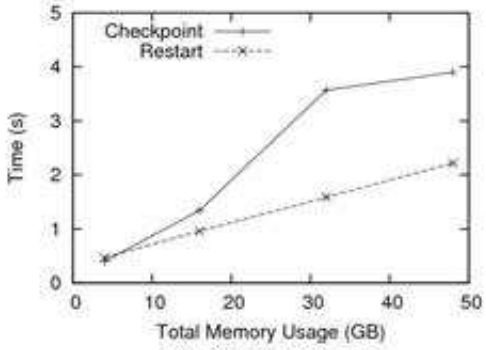
Example applications and experimental results

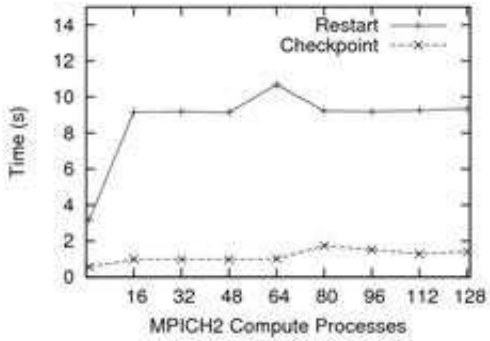
- OpenMPI, MPICH-2, IPython, ParGeant4 and NAS benchmarks











Future Extensions

- Checkpoint X server and/or applications using X server:
  - Useful for analysis.
  - Checkpoint all processes and graphic windows of analysis tool.
  - Copy checkpoint image to USB key.
  - At home, restart analysis on home Linux machine.
- Checkpoint GDB:
  - Checkpoint GDB session every two seconds.
  - At segmentation fault, diagnose cause.
  - Backup 2 seconds at a time in debugging session to discover origin of cause.
  - Move forward in GDB from any checkpoint.

Questions?

Features and future extensions

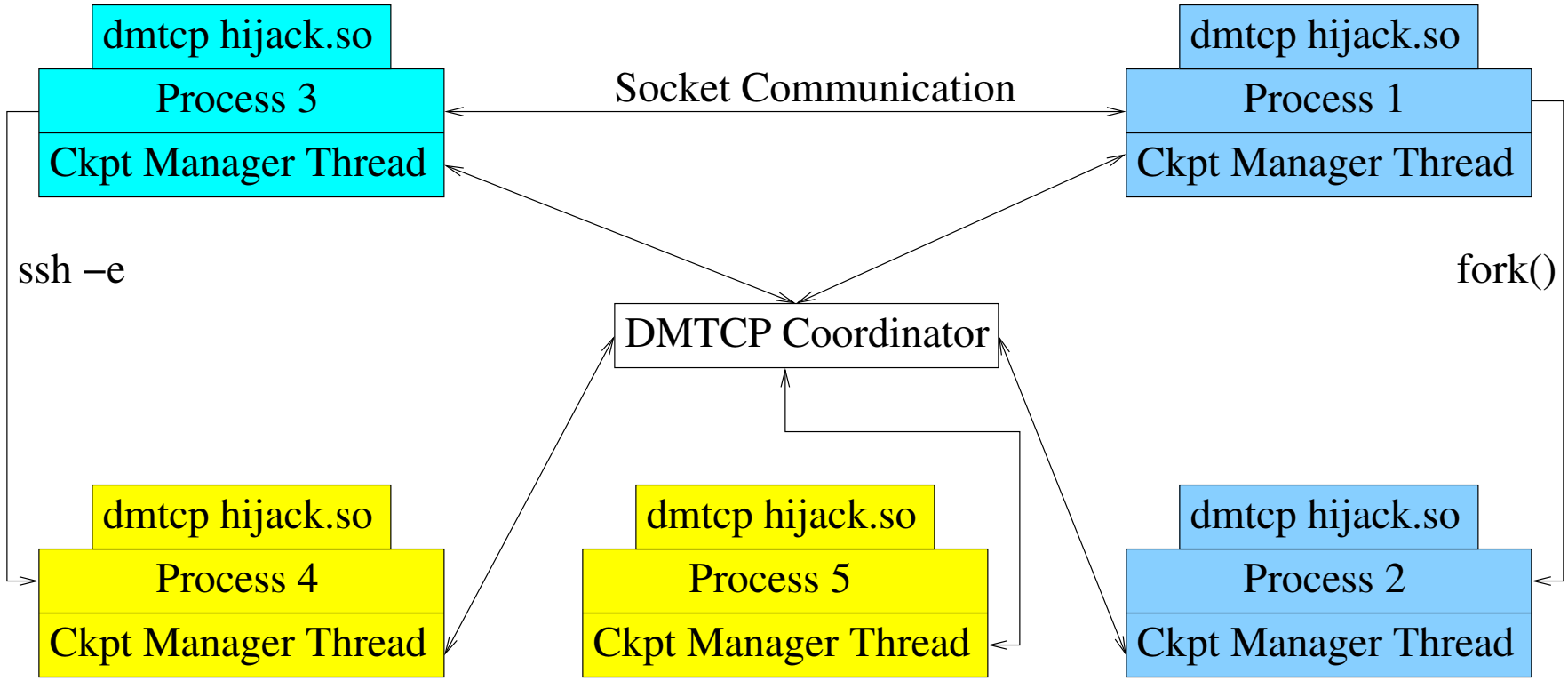
- Transparent user-level approach
- On the fly gzip compression
- Virtualization of pseudo terminals (pty)

Details: What is checkpointed

- Thread state/Process state
- Memory maps
- Open file descriptors
- Sockets
- Pseudo terminals
- Shared memory (via mmap())
- Remote/Local child processes created by fork(), exec(), ssh -e, system calls etc.

Details: DMTCP architecture

- DMTCP coordinator (stateless), DMTCP hijack library, Checkpoint manager thread and MTCP shared library



Details: How to checkpoint

- Do checkpoint: MTCP sends SUSPEND signal to all user threads
- Elect leader for file descriptors
- Drain sockets, save all open file descriptors
- MTCP writes user space memory to disk
- Restore file descriptors, refill socket buffers
- Resume application processes

Details: How to restart

- Connect to coordinator and register
- Restore file descriptors
- Broadcast expected incoming/outgoing connections
- Wait until all other nodes are restarted
- Rewire connections and refill socket buffers
- Use MTCP to restore process state, shared memory from disk
- Resume application processes