τ

Dynamic Tool Support in HPC++

Sameer S. Shende
Department of CIS,
University of Oregon

This represents joint work by
Aida Chouchane, Kathleen Lindlan, Sameer Shende and Kurt Windisch

τ

Dynamic Tool Support in HPC++

- Performance Profiling
- Function Profiling
- Aggregate Profiling
- Extensions to HPC++lib
- pprof/Racy
- Profiling with Tracing and Replay
- Current Status of Implementation
- Future Work
- Conclusions

Performance Profiling

- What is Performance Profiling?

  Problem : Identify bottlenecks,
  Execution profile,
  Aggregate profiling

- What does it mean to profile HPC++ programs?

  Multithreaded, multicontext programs
  Different runtime systems

TAU Profiling Model

- Function Profiling
  - Output : Time spent inclusive and exclusive of child functions
  - Number of calls
  - Node information
- How do we do it?
  - Profiler objects inserted at function entry
  - Constructors and destructors - stop watch!
  - Collecting per thread function timing info

```
30s   foo()
      {
      .
      .
      .
        bar1()    ......... time spent = 5s
      .
      .
      .
        bar2()    ......... time spent = 10s
      .
      .
      .}
```

inclusive of bar1 and bar2 = 30s
exclusive time spent in foo = 15 s

Function Profiling

- What data is available?

  Percentage of time/secs spent in different functions for each <node, thread, context>

  mean

  min

  max

- Two modes :

  Percentage wrt other functions on a node

  Value (actual seconds,usec)

- For each function, for each

  <node, context, thread>

  % time

  Exclusive time spent in function

  Time spent inclusive of others

  No of calls

  time/call

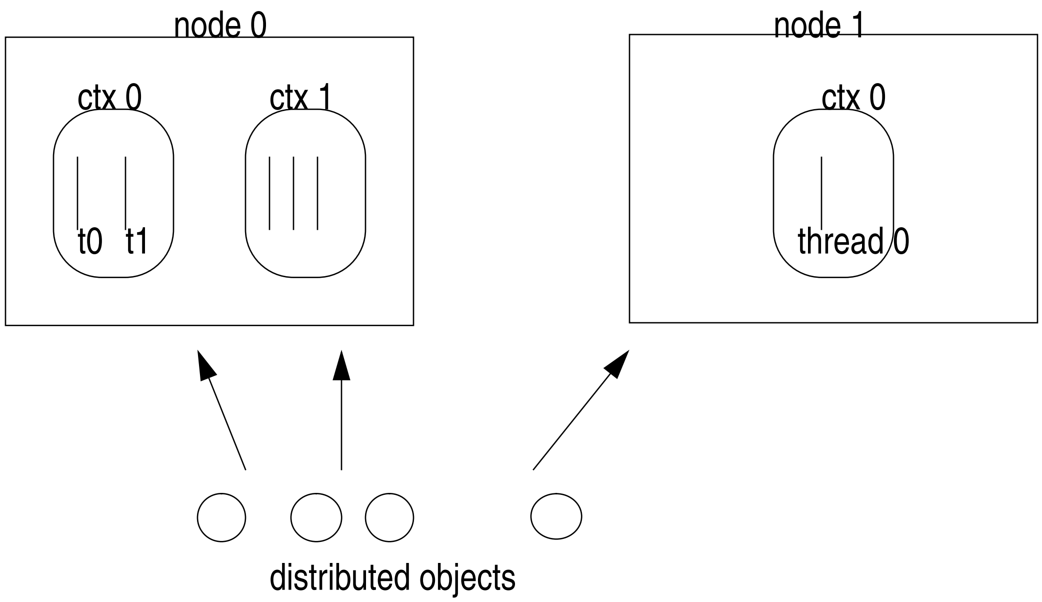
τ

Aggregate Profiling in HPC++

Aggregates are statically defined distributed objects visible over one or more threads, spanning one or more contexts

τ

Aggregate Profiling



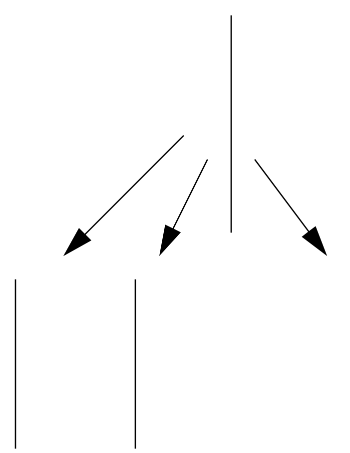

```
template <class T>
class distributed_vector : public DistributedContainer {
...
pariterator parbegin() {...}
HPCxx_GlobalReference operator[](size_type i) {...}
HPCxx_GlobalReference front() {...}
}

distributed_vector<Row> F(n);
distributed_vector<Row> U(n);
```

Extension to HPCxx_Thread Class

- Profiler object insertion in functions
- Constructors, destructors for timing
- Dumping profiling data - Destructor of top level function

- Extend Thread Class
- Wait for child threads to exit before exiting



τ

Aggregate Profiling

- Significant events in member functions of a class
- Observed over each distributed object (aggregate)
- “Pre-instrumented” versions of PSTL container classes
- User Instrumentation

```
Class Array {
public :
void Sort () {
 for (i=0; i< N; i++)
 {
  if (flag == TRUE)
  hpcxx_profileIncrementCounter(USER_MERGE);
 }
...
}


Array L, U, Diag;
```

- On which aggregate did the event USER_MERGE take place?
- How often?
- In which <node, context, thread>?

Dynamic Tool Support in HPC++

- Performance Profiling
- Function Profiling
- Aggregate Profiling
- Extensions to HPC++lib
- pprof/Racy
- Profiling with Tracing and Replay
- Current Status of Implementation
- Future Work
- Conclusions

τ

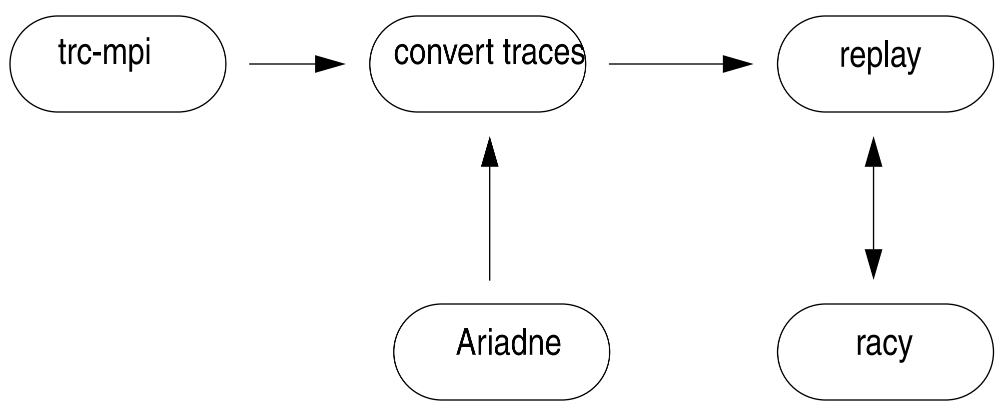
pprof

- Similar to prof (UNIX)
- Multithreaded multicontext profile data
- Aggregates in HPC++
- Collections in pC++

```
% pprof
NODE 0;CONTEXT 0;THREAD 0:
-------------------------------------------------------------
%time         msec   total msec    #call  usec/call name
-------------------------------------------------------------
100.0        6.090     3:30.795        1  210795000 test
 96.7    2:29.746     3:23.858       15   13590533 cgsol
 13.2       27.763       27.763     1125      24679 DistVector::saxpy
 11.9       18.284       25.044      765      32738 DistVector::dotprod
  3.2        6.759        6.759   357255         19 dot::operator()
  0.3          658          658       30      21957 DistVector::sub
  0.3          301          555       15      37003 DistVector::specialmax
  0.2          340          340       30      11362 DistVector::asgn
  0.1          172          304       15      20318 DistVector::gnorm
  0.1          171          171       15      11424 DistVector::asgnm
  0.1          132          132     7005         19 nrm::operator()
  0.1          126          126     7035         18 Max::operator()
  0.1          126          126     7005         18 abst::operator()
  0.1           52          119     1400         85 DistVector::setValue
  0.0           67           67     1400         48 DistVector::islocal
aggregates DistSparseMatrix <NULL> S
Event id 7 name "begin" : 61822 100.00 percent
aggregates DistVector <NULL> x
Event id 5 name "parbegin"      : 75 11.89 percent
Event id 6 name "parend"        : 30 4.75 percent
Event id 7 name "begin" : 466 73.85 percent
Event id 26 name "begin_local"  : 15 2.38 percent
Event id 27 name "end_local"    : 15 2.38 percent
Event id 28 name "par_begin_local"      : 15 2.38 percent
Event id 30 name "begin_relative"       : 15 2.38 percent
aggregates DistVector <NULL> z
Event id 5 name "parbegin"      : 2025 38.90 percent
Event id 6 name "parend"        : 810 15.56 percent
Event id 26 name "begin_local"  : 405 7.78 percent
```

Tracing and Replay in TAU

- PSTL, Tulip RTS, MPI programs
- Trace Generation
- Trace Processing
- Deterministic Replay
- Tracing, Replay with Profiling



Current Status of Implementation

- Profiling Library ready for PSTL and HPC++lib
- pprof extensions and Racy extensions
- Extensions to HPC++lib Thread class to support profiling
- Hand Instrumented source code for profiling (functions and aggregates)
- Multiple source files supported for Aggregate profiling

Future Work

- Automatic Instrumentation of source code (Sage 2 dependency)
- Multiple source file support planned for function profiling

Conclusions and Contributions

- TAU profiling model extended for HPC++
- Aggregate based profiling of Distributed objects
- Racy and τ
- Tracing and Deterministic Replay of PSTL programs