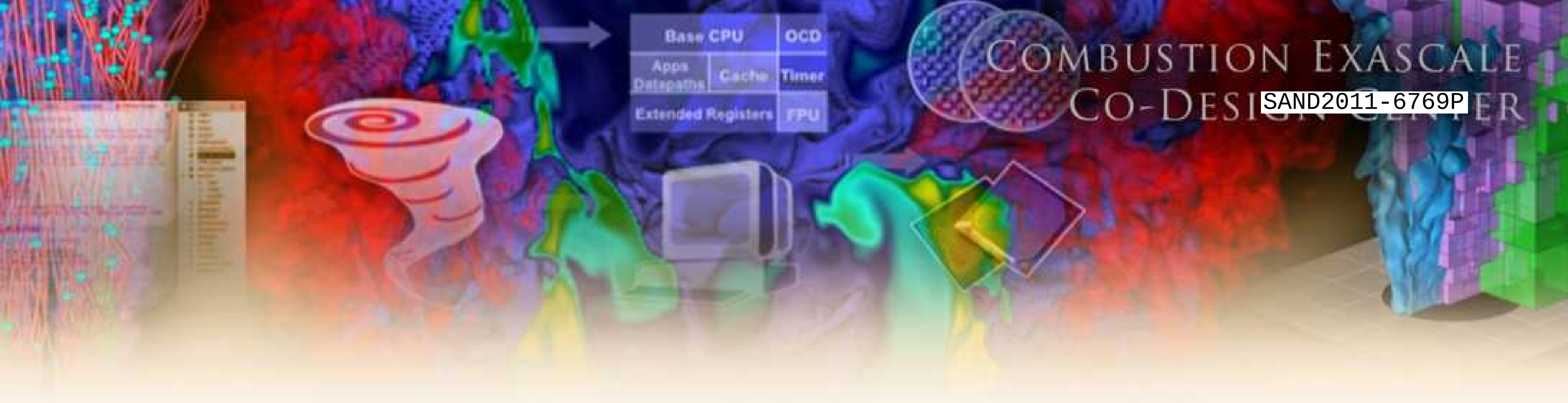


SAND2011-6769P

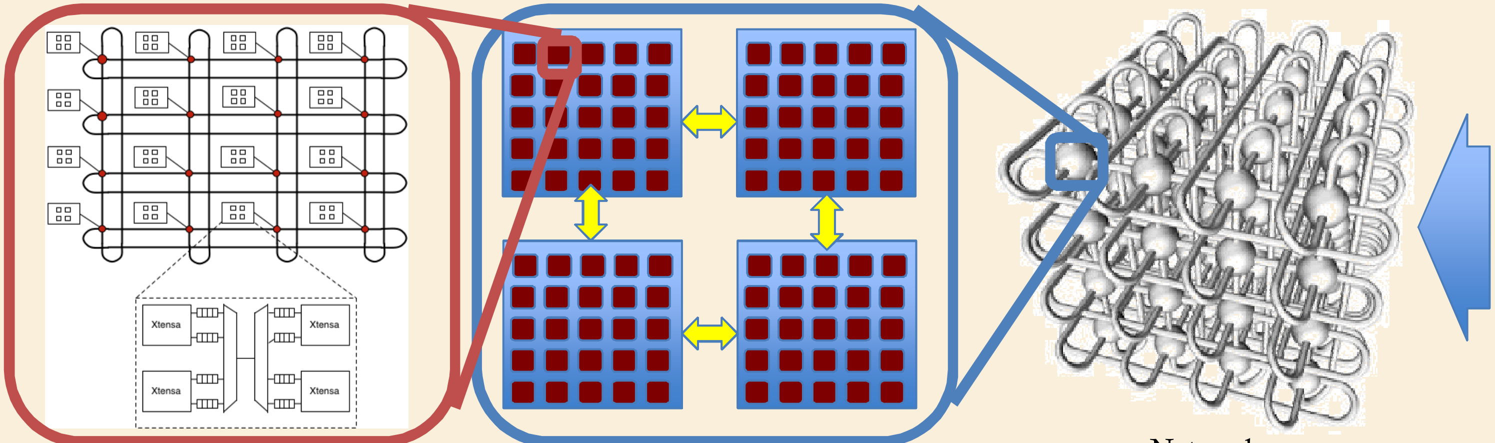
SST/macro

Coarse-grained Hardware/Software Architecture Simulation

SST/macro Team Members

- Joe Kenny – Application liaison and application models, DUMPI trace library
- Gilbert Hendry – Simulator S/W and machine models
- Khachik Sargsyan – Uncertainty quantification of architecture simulation results
- Curtis Janssen – speaker

Multi-scale machine and application model




```
for (int i=0; i<nblocks-1; i++) {
  std::vector<sstmac::mpiapi::mpirequest_t> reqs;
  // Begin non-blocking left shift of A blocks
  sstmac::mpiapi::mpirequest_t req;
  mpi()->isend(blocksize, datatype, myleft,
           tag, world, req);
  reqs.push_back(req);
  mpi()->irecv(blocksize, datatype, myright,
           tag, world, req);
  reqs.push_back(req);
  // Likewise for B shifting down ...
  // Simulate computation with current blocks
  compute_api()->compute(instructions);
  mpi()->waitall(reqs, statuses);}
// Finish last block
compute_api()->compute(instructions);
```

Tighly Coupled Cores — Nodes — Network — Application

Relevance/impact of CECDC coarse-grained simulation efforts

Correctly identify causal relationships

- Network topology
- Node configuration
- Noise/imbalance
- Bandwidth
- Latency
- Resource contention

Test changes to application, middleware, or resource management

- Reordering code blocks, scheduling effects, etc.

Play "what if" games

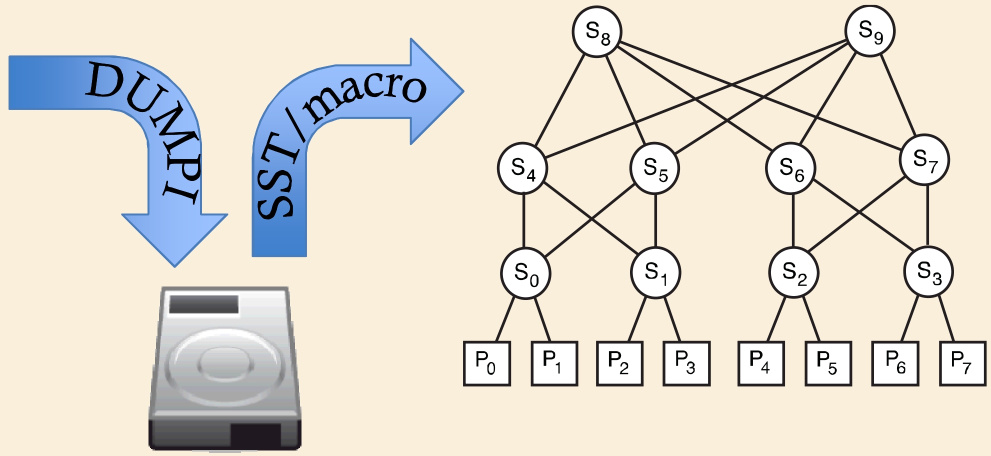
- Implementation effects for communication routines
- Infinite performance in some components to stress others.

Test novel programming models

- Fault-tolerant or fault-oblivious execution models
- Alternatives to MPI, parallel runtime designs
- Mixed programming models

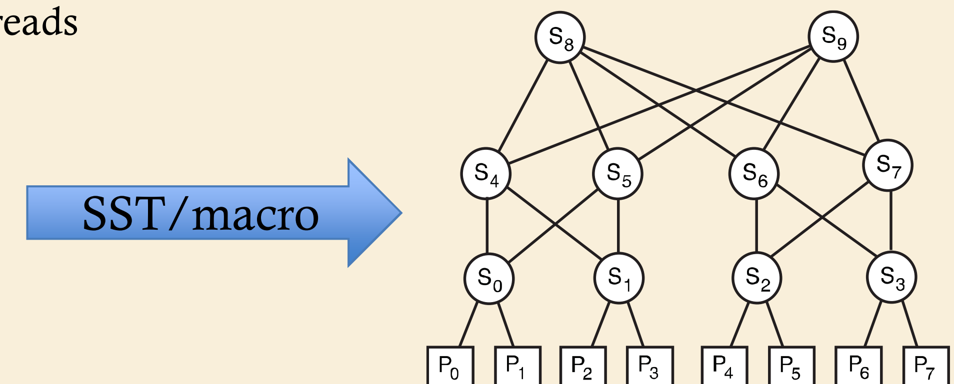
SST/macro is driven with trace files or a skeleton application

- Replay application traces in SST/macro



- Provide a skeleton application to the simulator
  - Implemented with lightweight threads

```
void sampleapp::run() {
  sstmac::mpicomm world = mpi()->comm_world();
  sstmac::timestamp start = mpi()->init();
  const mpiid root(0);
  mpi()->bcast(1, sstmac::mpitype::mpi_double,
          root, world);
  sstmac::timestamp done = mpi()->finalize();}
```



  - Allows extreme scale/application concept exploration

DUMPI: The MPI tracer

- PMPI link-time library for trace file generation
- Full fingerprints for all MPI-2 functions
- Can add annotations
- Can selectively control profiling
  - Globally/statically with configuration file
  - Locally/dynamically with function calls
- Writes a (reasonably compact) binary trace file
- Negligible runtime overhead
- Reasonably portable C code

libdumpi	common	libundumpi
PMPI bindings Type mapping Call tree tracing (gcc/icc)	MPI type identifiers MPI function identifiers Trace file IO Timers Performance counters	Parsing of trace files

Example of data output by DUMPI

(converted using dumpi2ascii)

```
MPI_Allgatherv entering at walltime 1274314439.744512000,        \
cputime 0.201756000 seconds in thread 0.
int commsize=16
int sendcount=1024
MPI_Datatype sendtype=14 (MPI_DOUBLE)
int recvcounts[16]=[1024, 1024, 1024, 1024, 1024, 1024, 1024,    \
1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024]
int displs[16]=[0, 1024, 2048, 3072, 4096, 5120, 6144, 7168,     \
8192, 9216, 10240, 11264, 12288, 13312, 14336, 15360]
MPI_Datatype recvtype=14 (MPI_DOUBLE)
MPI_Comm comm=2 (MPI_COMM_WORLD)
MPI_Allgatherv returning at walltime 1274314439.749554000,       \
cputime 0.202159000 seconds in thread 0.
```

Skeleton Apps in Fortan/C/C++

- Native messaging library interfaces
- Minor modification to run under SST/Macro
  - Replace native header
  - Rename main()
- Compute blocks and memory allocation abstracted out by hand or using ROSE compiler
- Use preprocessor to maintain single source

Unified skeleton and mini-application

Rename main

```
program main
```

```
#ifdef SSTMAC_SKELETON
    subroutine skeleton_main()
#else
    program main
#endif
```

Include SST/macro header

```
#include <mpi.h>
```

```
#ifdef SSTMAC_SKELETON
#  include <sstmac/mpif.h>
#  include <sstmac/processor.h>
#else
#  include <mpif.h>
#endif
```

Abstract out computation

```
call do_computation(data)
```

```
#ifdef SSTMAC_SKELETON
  sstmac_compute(param);
#else
   call do_computation(data)
#endif
```

Avoid large data allocations

```
array = new double[ndata];
```

```
#ifdef SSTMAC_SKELETON
  array = 0;
#else
   array = new double[ndata];
endif
```

ASCR Execution Models Projects

- Goal: demonstrate ability to quantify impact of execution model choice on performance, power, etc. Develop methodology for execution model co-design.
- Three projects:
  - Study limitations of current execution models (ISI/LBNL)
  - "Top-down" study of execution model co-design: Develop definitions and formalism for execution models. Study full applications and model performance. (PNNL/IU)
  - "Bottom-up" study of execution model co-design: Use simulation to evaluate execution models and design applications (SNL/LBNL/IU)
- AMR will be initial app for all projects
  - Challenging problem for exascale
  - Directly relevant to Combustion ECDC
  - Hope to heavily leverage the CECDC work