SAND2022-5716C

Sandia National Laboratories

Supercomputer in a laptop: Distributed application and runtime development via architecture simulation

Samuel Knight, Joseph Kenny and Jeremiah Wilke

Sandia National Laboratories, Livermore, CA

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Outline

- Introduction
- MPI simulator design
- SST Macro Simulator design
- Architectural simulation as a communication library development tool
- Example

Outline

- Introduction
- MPI simulator design
- SST Macro Simulator design
- Architectural simulation as a communication library development tool
- Example

Introduction

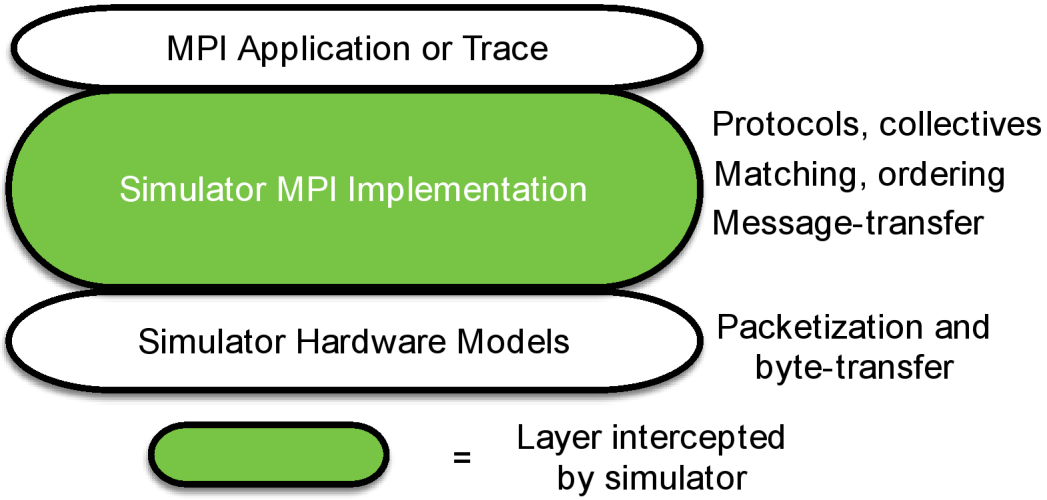
Developing a network runtime is hard

- Hardware is non-deterministic
  - Performance or correctness bugs might only occur with certain message ordering
- Testing performance on system designs infeasible
  - Can't purchase a "system scale" testbed
  - Possible configurations are limited
- Distributed systems are difficult to debug
  - Performance or correctness bugs might only emerge at certain scales
  - Debugging distributed memory can be difficult - can't just GDB/Valgrind on your laptop

Outline

- Introduction
- MPI simulator design
- SST Macro Simulator design
- Architectural simulation as a communication library development tool
- Example

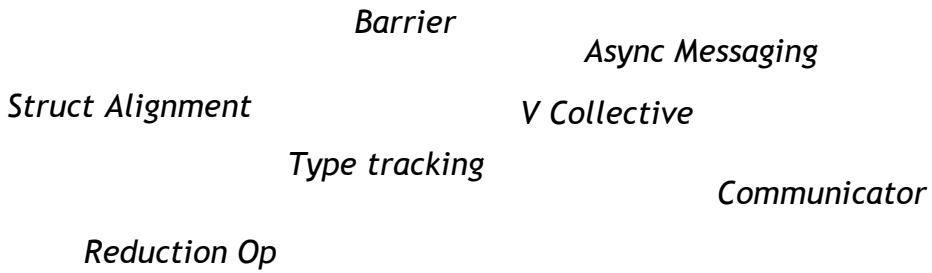
Common MPI simulator Implementations



- MPI Simulations intercept MPI calls using one of two common methods
  - Read post-mortem traces from a previous MPI execution
  - Capture MPI calls from an active MPI application (or representation of an application)
- Intercepted MPI calls handled by simulator MPI implementation, rather than system MPI (e.g. OpenMPI)

Common MPI simulator implementation challenges

- MPI operations can be complex



- MPI Simulators must have an accurate network model **and MPI runtime**

- The simulator's MPI implementation is yet another component to:
  - **Validate:** Correctly implemented API and semantics can have different implementations.
  - **Maintain:** MPI standard change, new features and algorithms

- New MPI standards create a moving development target

Outline

- Introduction
- MPI simulator design
- SST Macro Simulator design
- Architectural simulation as a communication library development tool
- Example

SST Macro implementation challenges

Several design challenges influenced SST Macro's implementation

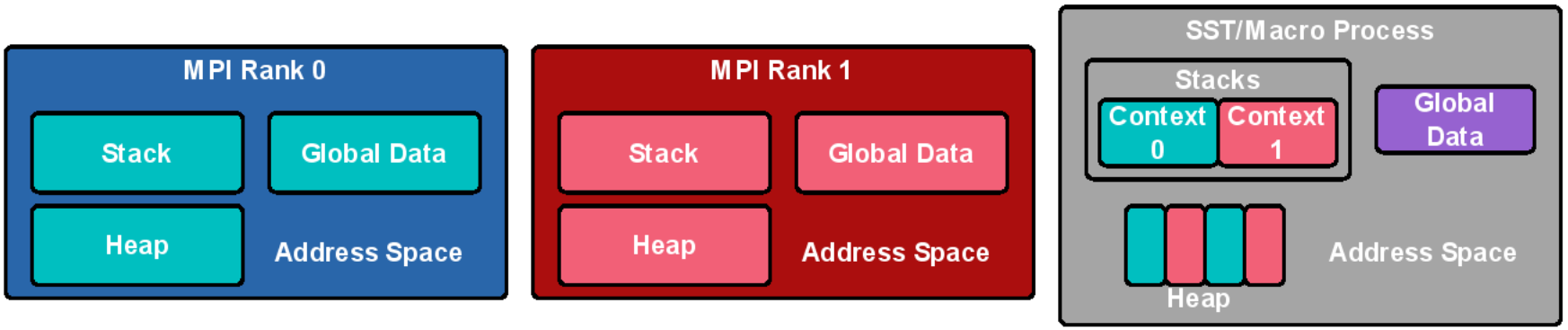
- **Encapsulation** – The emulation of thousands of concurrent virtual processes in a single simulator process
- **Interception** - The transfer of control between the user's software stack and the simulator's network model
- **Skeletonization** - Strategies for reducing a full application to a communication skeleton that estimates delays of compute intensive and does not allocate large blocks of memory

Memory space separation (Encapsulation)

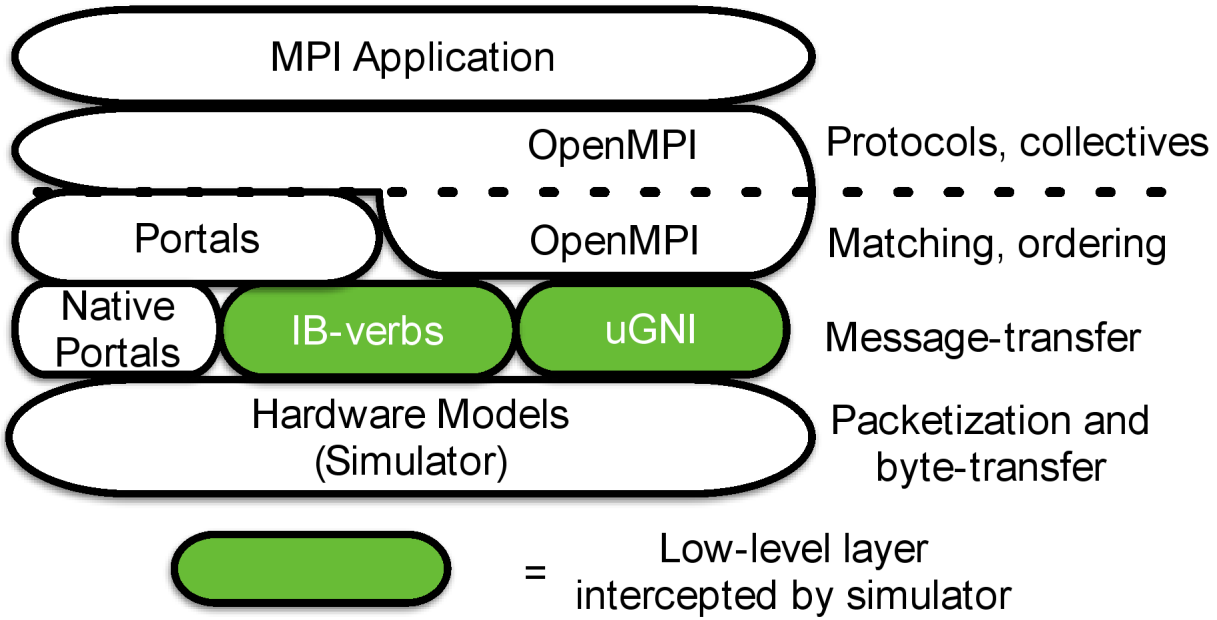
- Simulator runtime must mimic memory separation of a distributed system

- Each virtual process needs a private:
  - **Stack** – User space-threads for scalable stack separation
  - **Heap** – Each individual heap allocation already "private"
  - **Globals** – Skeletonizer renames global variables to be accessible in a thread-local context

- Resulting simulation emulates concurrent execution of many *virtual* processes in one *physical* simulator processes (or a few simulator processes for parallel discrete event simulation – PDES)



Low-level Interception



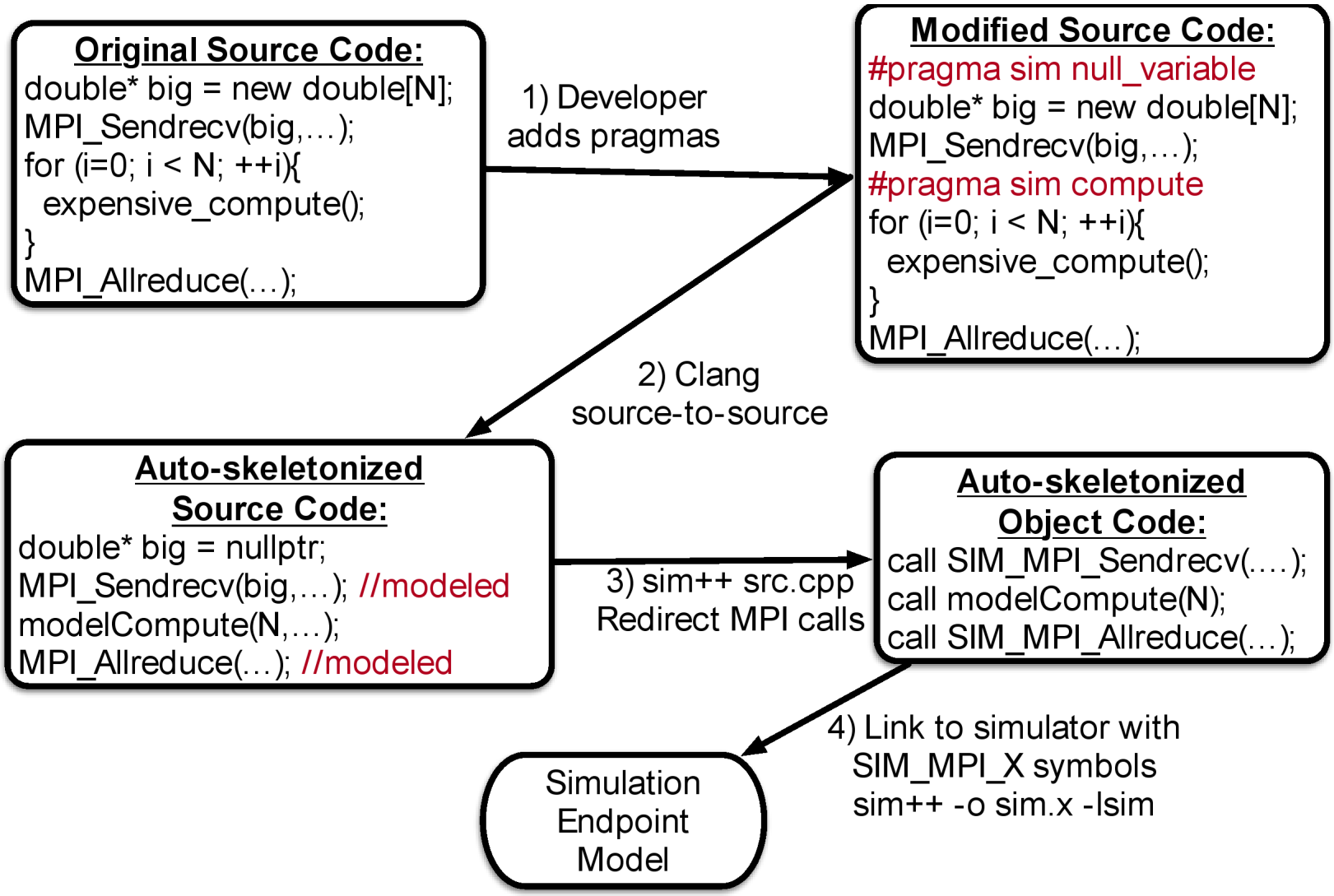
- A lot of code complexity above the hardware model
- Requiring simulator to implement stand-alone MPI adds significant complexity to the simulator
- Easier to maintain a simulator-specific API implementation for uGNI or verbs

Skeletonization

- Execution of encapsulated HPC jobs won't fit on a laptop; compute and memory resources are too scarce
- Many compute and data intensive operations are not really needed to generate a network model
- SST Macro's compiler wrapper supports Clang-based source-to-source transformation
- Uses preprocessor hints from app developer to
  - Remove large memory allocations
  - Substitute compute regions with simulated delay
  - Simulate movement large memory allocations across network

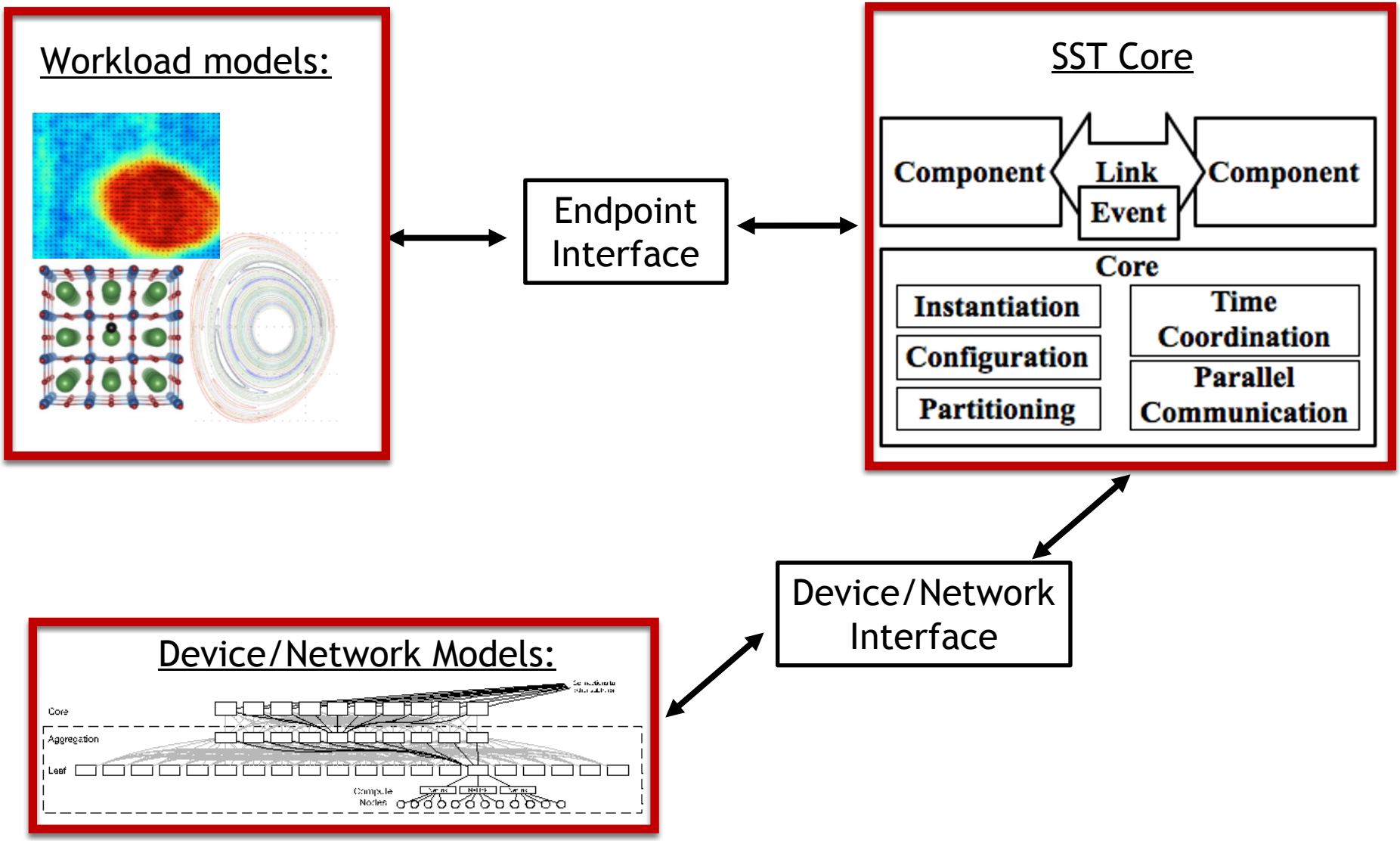
* Communication libraries above the low-level intercept must safely handle null buffers

Skeletonization: Illustration with MPI – extended in this study to uGNI/verbs/GASNet



- Can be extended to intercept ibv, uGNI

Collaborative model for system design with the Structural Simulation Toolkit (SST)



Outline

- Introduction
- MPI simulator design
- SST Macro Simulator design
- Architectural simulation as a communication library development tool
- Example

Debugging support

- SST Macro's simulation runtime can scale down to a single process
- Network endpoints are user-space threads, which run in shared memory. One instance of Valgrind could globally test a runtime for memory leaks
- SST Macro's compiler is a C/C++ wrapper, and will emit GDB and LLDB compatible symbols
- Configurable runtime, change the configuration file and go.
  - Seeded pseudo-random packet and orderings and delays
  - Configurable topology/node counts/hardware specs
  - Source code pragma annotations

Revisiting the Introduction

Developing a network runtime is hard

- Hardware is non-deterministic
  - Performance or correctness bugs might only occur with certain message ordering

- Testing performance on system designs infeasible
  - Can’t purchase a “system scale” testbed
  - Possible configurations are limited

- Distributed systems are difficult to debug
  - Performance or correctness bugs might only emerge at certain scales
  - Debugging distributed memory can be difficult - can’t just GDB/Valgrind on your laptop

Revisiting the Introduction

Developing a network runtime is hard

- ~~Hardware is non-deterministic~~
  - ~~Performance or correctness bugs might only occur with certain message ordering~~

Simulated hardware is deterministic, packet ordering can change on demand

- Testing performance on system designs infeasible
  - Can't purchase a "system scale" testbed
  - Possible configurations are limited

- Distributed systems are difficult to debug
  - Performance or correctness bugs might only emerge at certain scales
  - Debugging distributed memory can be difficult - can't just GDB/Valgrind on your laptop

Revisiting the Introduction

Developing a network runtime is hard

- ~~Hardware is non-deterministic~~
  - ~~Performance or correctness bugs might only occur with certain message ordering~~

Simulated hardware is deterministic, packet ordering can change on demand

- ~~Testing performance on system designs infeasible~~
  - ~~Can't purchase a "system scale" testbed~~
  - ~~Possible configurations are limited~~

Network components are transparent; routing and topology are settings

- Distributed systems are difficult to debug
  - Performance or correctness bugs might only emerge at certain scales
  - Debugging distributed memory can be difficult - can't just GDB/Valgrind on your laptop

Revisiting the Introduction

Developing a network runtime is hard

- ~~Hardware is non-deterministic~~
  - ~~Performance or correctness bugs might only occur with certain message ordering~~

Simulated hardware is deterministic, packet ordering can change on demand

- ~~Testing performance on system designs infeasible~~
  - ~~Can't purchase a "system scale" testbed~~
  - ~~Possible configurations are limited~~

Network components are transparent; routing and topology are settings

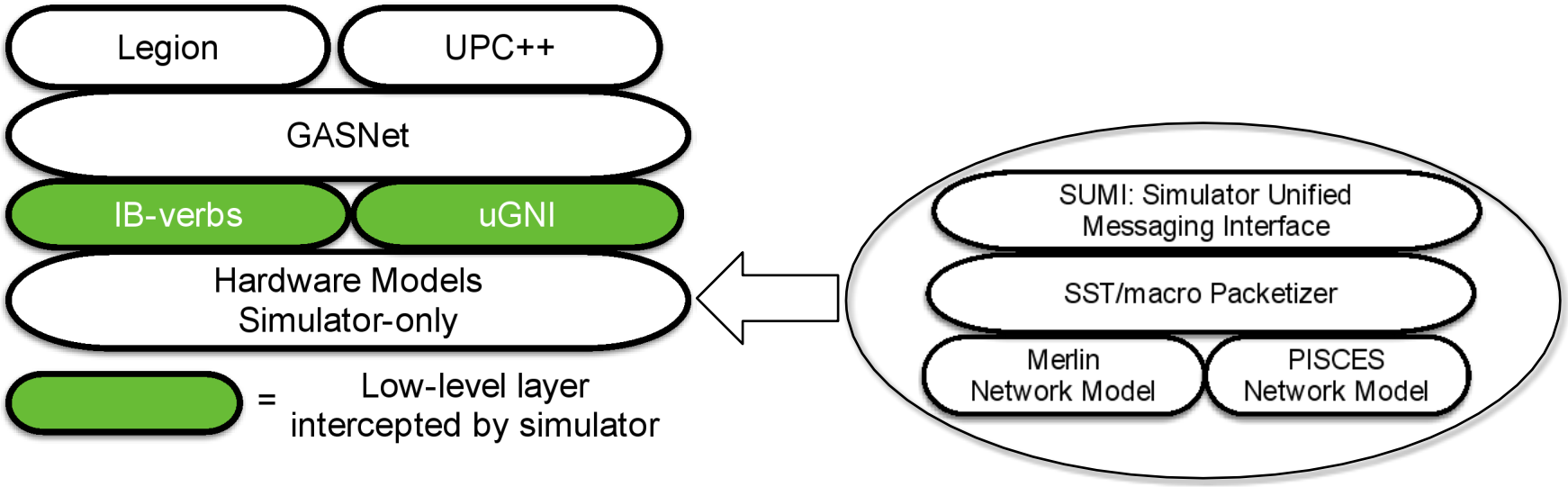
- ~~Distributed systems are difficult to debug~~
  - ~~Performance or correctness bugs might only emerge at certain scales~~
  - ~~Debugging distributed memory can be difficult - can't just GDB/Valgrind on your laptop~~

Arbitrarily defined system that can run in a processes with user-space threads, tools like GDB and Valgrind globally inspect the runtime.

Outline

- Introduction
- MPI simulator design
- SST Macro Simulator design
- Architectural simulation as a communication library development tool
- Example

GASNet on SST Macro example



- GASNet builds with uGNI or verbs conduit, and uses APIs provided by SST Macro's compiler wrapper
- At runtime, GASNet's runtime calls to uGNI are intercepted by the simulator and pass into the simulated network.

GASNet benchmark with SST Macro

Testcore2 benchmark

```
* Description: GASNet Core checksum test
* This stress tests the ability of the core to successfully send
* AM Requests/Replies with correct data delivery
* testing is run 'iters' times with Medium/Long payload sizes ranging from 1..'max_payload',
* with up to 'depth' AMs in-flight from a given node at any moment
```

Changes for Auto-Skeletonization

Allocations replaced with null pointers

```
$ grep -rn "pragma sst null_variable" tests/testcore2.c
#pragma sst null_variable replace nullptr
uint8_t *peerreqseg; /* long request landing zone */
#pragma sst null_variable replace nullptr
uint8_t *peerrepseg; /* long reply landing zone */
#pragma sst null_variable replace nullptr
uint8_t *localseg;
#pragma sst null_variable replace nullptr
…
```

Remove a compute intensive

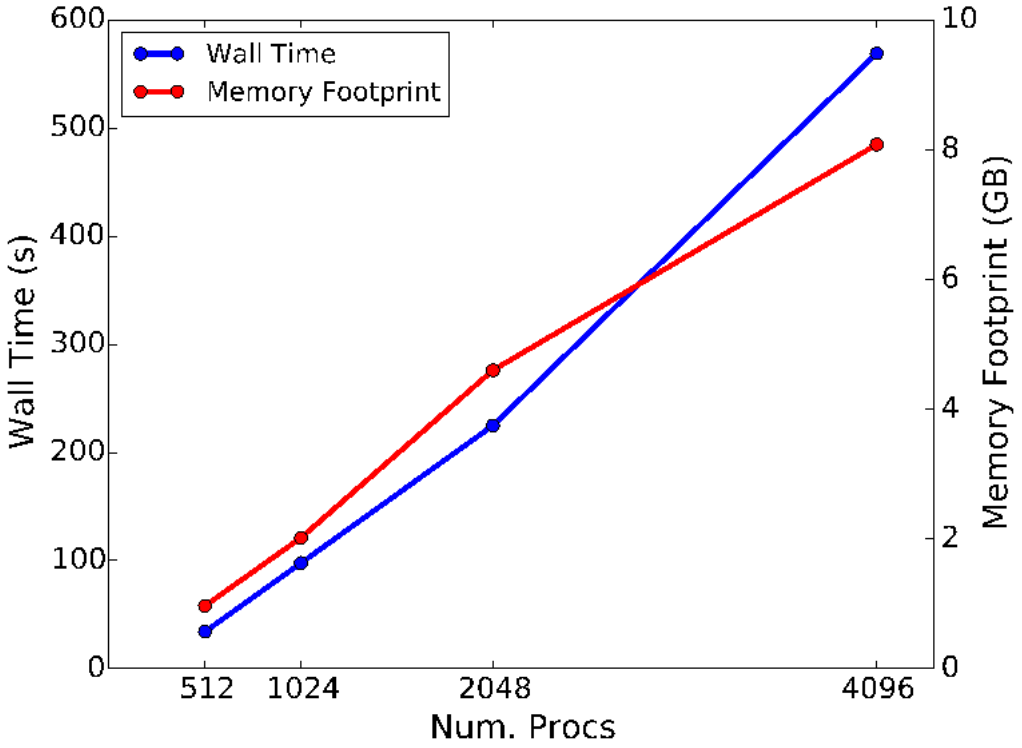
```
#pragma sst compute
for (elemidx = 0; elemidx < sz; elemidx++) {
 …
}
```

14 “#pragmas sst” substitutions

Parameter tag inserted into GASNet

```
#pragma sst overhead gni_mem_register
  status = GNI_MemRegister(nic_handle, addr, nbytes, NULL,
                           flags, -1, &pd->local_mem_hndl);
```

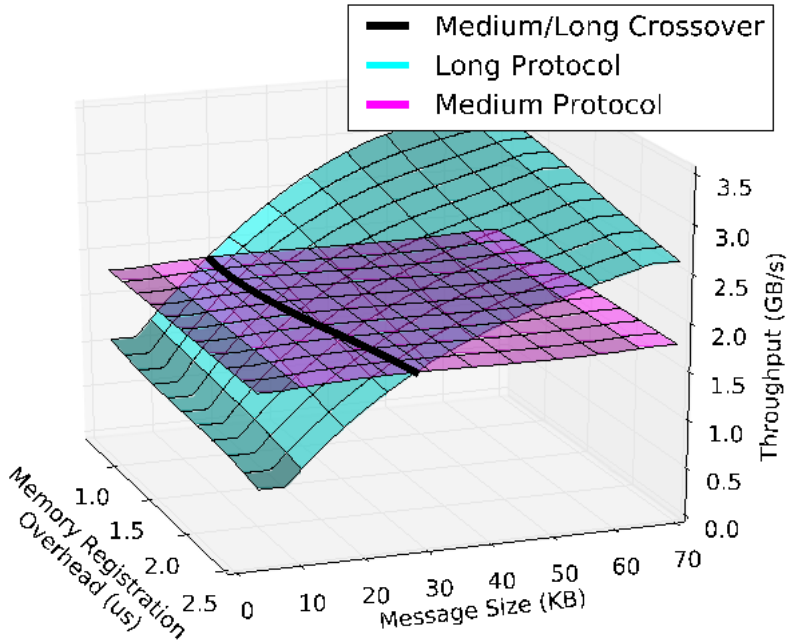
GASNet simulation scaling SST Macro example

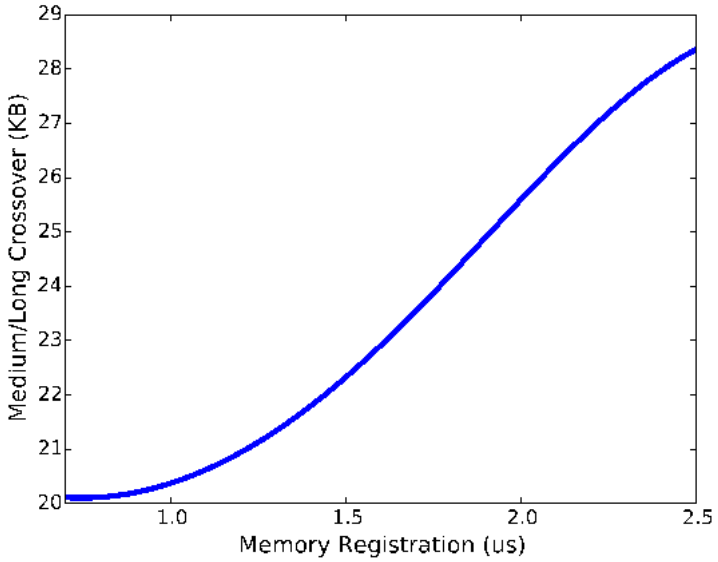


- Simulations with up to 4K skeletonized GASNet ranks from testcore2 test fit on a single machine
- “non-skeletonized” version minimally needs 128GB
  - 32 1MB messages in flight per process
  - Probably more due to extra buffers in the GASNet runtime

GASNet benchmark with SST Macro

Benchmarking performance crossover between GASNet's medium and log protocols. Memory registration overhead was varied by an input file parameter.





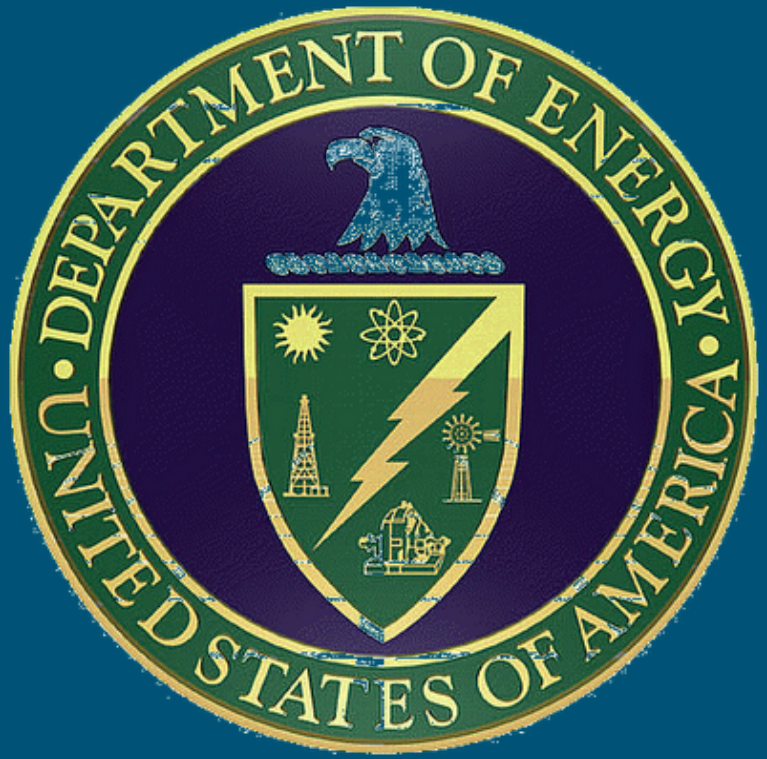
(Left) Visualization of GASNet's Medium and Long protocol throughput and latency, varied by message size and registration overhead.
(Right) Visualization of the performance crossover (intersection of the two surfaces on the left) where Long Protocol has more throughput than Medium

Conclusion

- Simulators often implement MPI libraries
  - May be difficult to maintain
  - Contribute to model inaccuracy
- Low level interception reduces simulator runtime complexity, increases application variety
- Encapsulation via user-space threading and global variable skeletonization puts a distributed runtime into a single simulator process
- Common development tools, e.g. Valgrind and GDB, can operate on the entire simulated memory space at once
- Deterministic and transparent simulated hardware
- Rich options for unit testing and parameter sweeps
  - Seeded pseudo-random message ordering, deterministic replay
  - Network topology, routing, and bandwidth
  - In-code annotations for simulated delays (e.g. test effectiveness of hardware support or future optimizations)

Acknowledgments

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.



National Nuclear Security Administration

Sandia National Laboratories