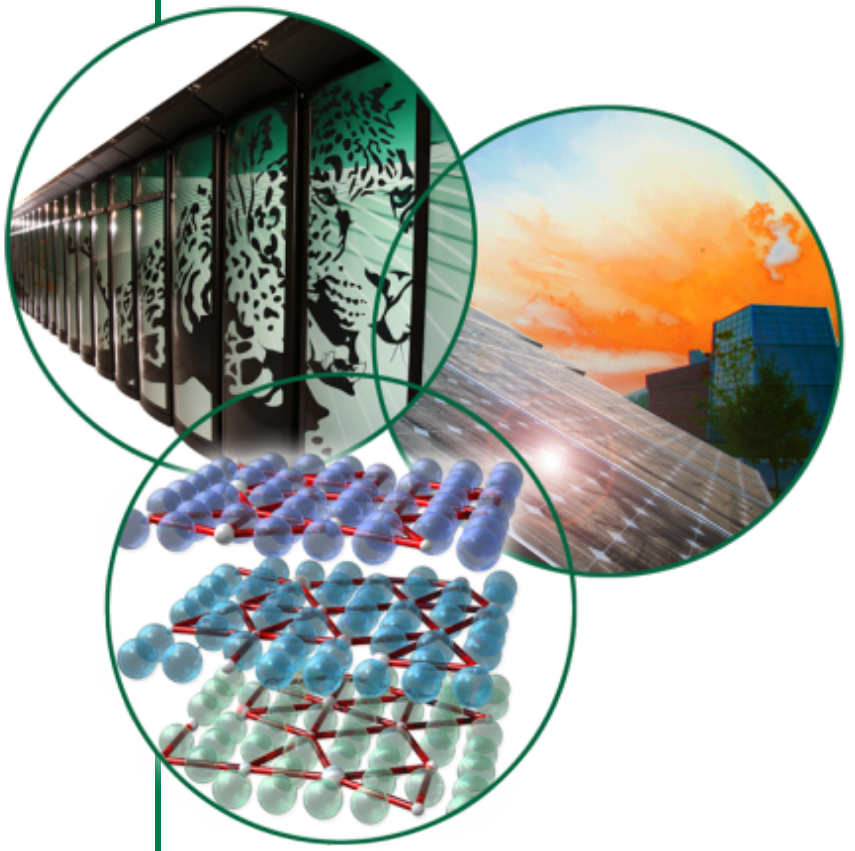
Tools for Simulation and Benchmark Generation at Exascale



Christian Engelmann
Oak Ridge National Laboratory

Frank Mueller and Mahesh Lagadapati
North Carolina State University

Parallel Tools Workshop 2013

U.S. DEPARTMENT OF ENERGY

OAK RIDGE NATIONAL LABORATORY
MANAGED BY UT-BATTELLE FOR THE DEPARTMENT OF ENERGY

The Road to Exascale

- Current top systems are at ~1-15 PFlops:
  - #1: NUDT, NUDT, 3,120,000 cores, 33,9 PFlops, 62% Eff.
  - #2: ORNL, Cray XK7, 560,640 cores, 17.6 PFlops, 65% Eff.
  - #3: LLNL, IBM BG/Q, 1,572,864 cores, 16.3 PFlops, 81% Eff.
- Need 30-60 times performance increase in the next 9 years
- Major challenges:
  - *Power consumption*: Envelope of ~20MW (drives everything else)
  - *Programmability*: Accelerators and PIM-like architectures
  - *Performance*: Extreme-scale parallelism (up to 1B)
  - *Data movement*: Complex memory hierarchy, locality
  - *Data management*: Too much data to track and store
  - *Resilience*: Faults will occur continuously

Discussed Exascale Road Map (2011)

Many design factors are driven by the power ceiling of 20MW

Systems	*2009*	*2012*	*2016*	*2020*
System peak	2 Peta	20 Peta	100-200 Peta	1 Exa
System memory	0.3 PB	1.6 PB	5 PB	10 PB
Node performance	125 GF	200GF	200-400 GF	1-10TF
Node memory BW	25 GB/s	40 GB/s	100 GB/s	200-400 GB/s
Node concurrency	12	32	O(100)	O(1000)
Interconnect BW	1.5 GB/s	22 GB/s	25 GB/s	50 GB/s
System size (nodes)	18,700	100,000	500,000	O(million)
Total concurrency	225,000	3,200,000	O(50,000,000)	O(billion)
Storage	15 PB	30 PB	150 PB	300 PB
IO	0.2 TB/s	2 TB/s	10 TB/s	20 TB/s
MTTI	1-4 days	5-19 hours	50-230 min	22-120 min
Power	6 MW	~10MW	~10 MW	~20 MW

HPC Hardware/Software Co-Design

- Aims at closing the gap between the peak capabilities of the hardware and the performance realized by applications (application-architecture performance gap, system efficiency)
- Relies on hardware prototypes of future HPC architectures at small scale for performance profiling (typically node level)
- Utilizes simulation of future HPC architectures at small and large scale for performance profiling to reduce costs for prototyping
- Simulation approaches investigate the impact of different architectural parameters on parallel application performance
- Parallel discrete event simulation (PDES) is often employed with cycle accuracy at small scale and less accuracy at large scale

Traditional Approaches

- Application execution
  1. Run applications in a simulated HPC system
  - Highly accurate only at small scale
  - Less accurate and time consuming at large scale
- Trace replay
  1. Run applications on existing HPC systems to extract MPI traces
  2. Replay the extracted MPI traces in a simulated HPC system
  - Similar accuracy at large scale and less time consuming
  - Simple trace replay is data intensive and not scalable
  - Simple trace replay is limited to the extracted scales

Proposed Approach

1. Run applications on existing HPC systems to extract MPI traces
2. Generate simplified benchmarks from the extracted MPI traces
3. Run the generated benchmarks in a simulated HPC system

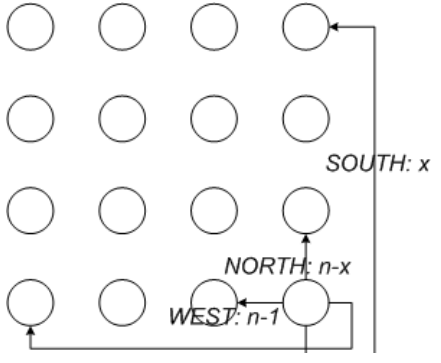
- Good accuracy at all scales (allows to trade-off accuracy/scale)
- Extremely fast in comparison to the other approaches
- Data-intensive trace processing offline in benchmark generation
- Benchmark execution scales are variable as communication patterns are abstracted

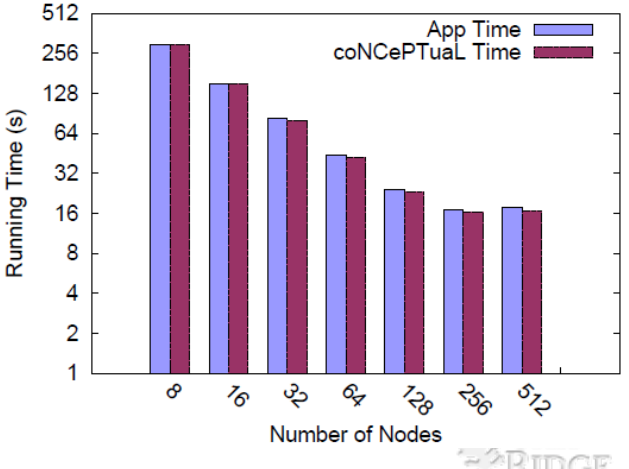
ScalaTrace: Fundamentals & Applications

Scalable tracing and trace-based performance analysis

- ScalaTrace
  - Scales: const. sized, lossless traces → PRSDs
  - Concise: single file (a few MBs)
  - Accurate: time preserving → reply resembles app
- ScalaExtrap
  - Trace-based extrapolation
  - Performance analysis and prediction at any scale
- Benchmark generation
  - Performance accurate stand-alone benchmark
  - Obfuscates source code: good for proprietary, classified, and expert-controlled codes
- Idea: Combine all 3
  → Feed into exascale simulator

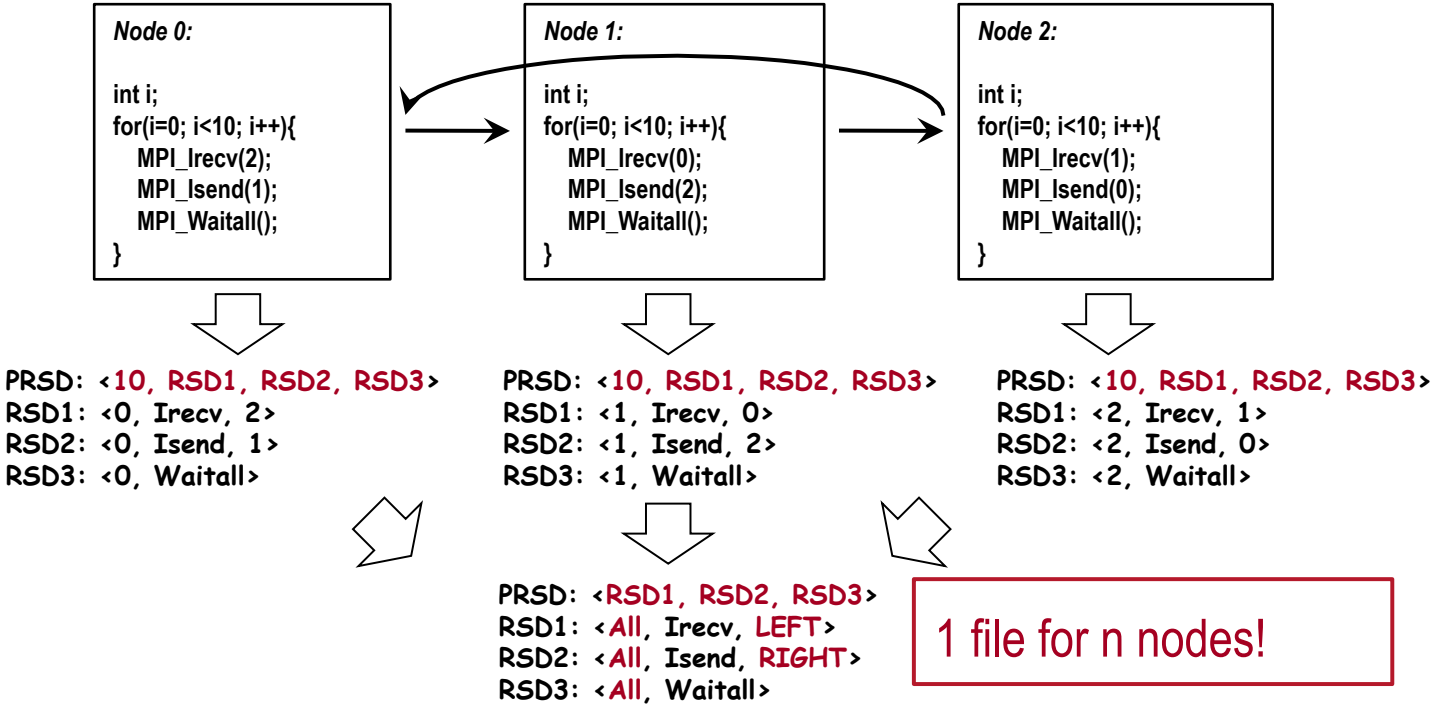
LOOP: <10, E1, E2, E3>
E1: <All, Irecv, LEFT>
E2: <All, Isend, RIGHT>
E3: <All, Waitall>





ScalaTrace

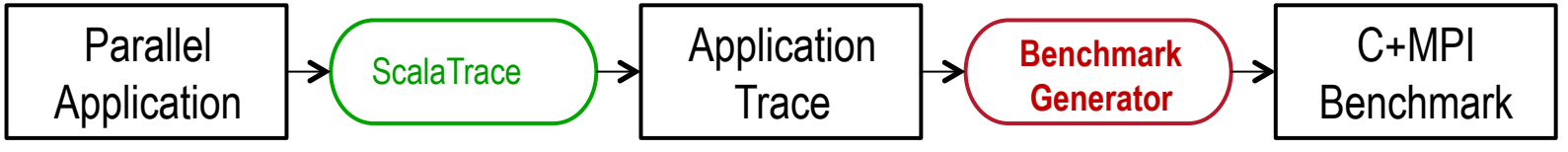
- Lossless, scalable: intra- & inter-node compression
  - E.g.: 3 nodes, ring style communication



  - Program structure preserved

ScalaBenchGen: Auto-Benchmark Creator

- Files generated:
  - main.h, main.c:
    - Skeleton code: loops, communication events, sleeps (to replace computation)
  - util.h util.c:
    - Memory management
    - Request handle management
    - Communicator management
    - Communication wrapper functions: facilitate customization for what-if test and profiling
  - Makefile



Code Generation from App Trace

Application → Application Trace → Benchmark in C+MPI

```
func1();
for(i = 0; i < 10; i++){
   func2();
   MPI_Irecv(LEFT);
   func3();
   MPI_Isend(RIGHT);
   func4();
   MPI_Waitall();
}
func5();
MPI_Barrier();
```

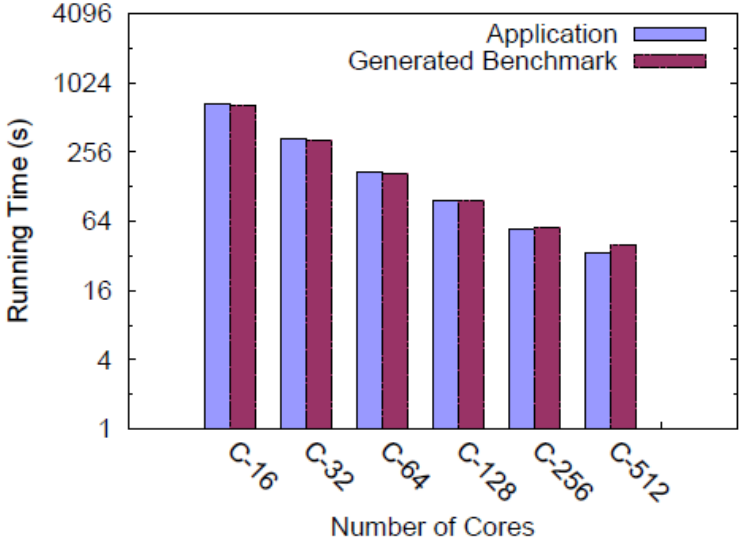
```
T1
PRSD1:  <10, RSD2, RSD3, RSD4>
        T2
RSD2:   <All, Irecv, LEFT>
        T3
RSD3:   <All, Isend, RIGHT>
        T4
RSD4:   <All, Waitall>
        T5
RSD5:   <All, Barrier>
```

```
for(i1=0;i1<10;i1++) {
   if(i1 == 0)
      compute(T1);
   if(i1 != 0)
      compute(T2);
   do_irecv(buffer2, 8, MPI_INT, 3, 0,
            get_comm(&comms, 2), reqs, …);
   compute(T3);
   do_isend(buffer, 8, MPI_INT, 1, 0,
         get_comm(&comms, 2), reqs, …);
   compute(T4);
   do_waitall("9998 9999", reqs);
}
compute(T5)
MPI_Barrier(get_comm(&comms, 2));
```

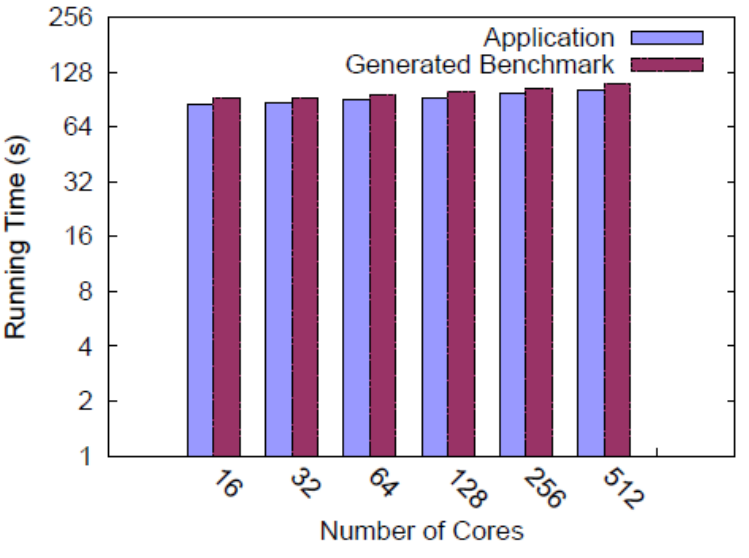
- All communication events & computation times generated
  - Compute times→sleep, distinguish loop/entry/exit paths
- Hide request handle and communicator management
  - Receive-triggered sends @ wildcards (avoid deadlocks)
- Contains loop structure: concise and easy to read

Accuracy of Generated Timings

- Time the application and the generated benchmark, and compare the results
- Mean absolute percentage error is only 6.4%
  → formula: |TGenBench – Tapp|/Tapp x 100%



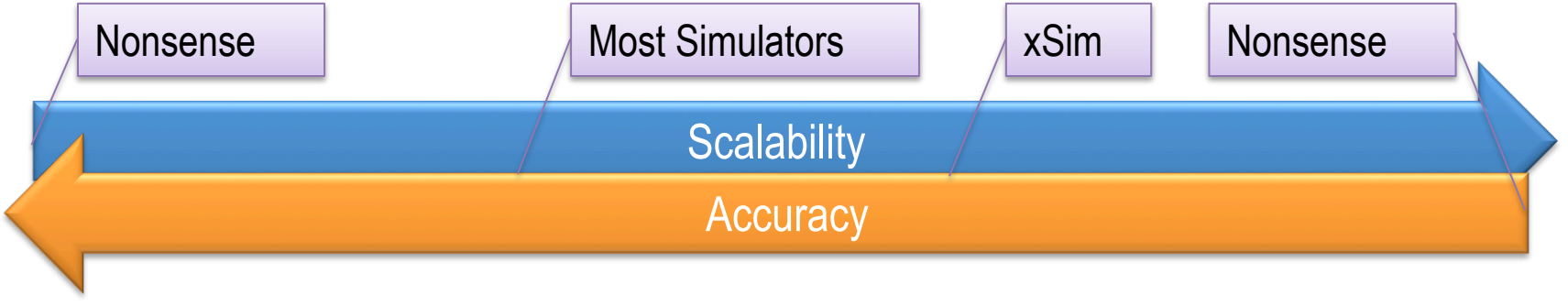
LU – strong scaling



Sweep3D: Weak Scaling

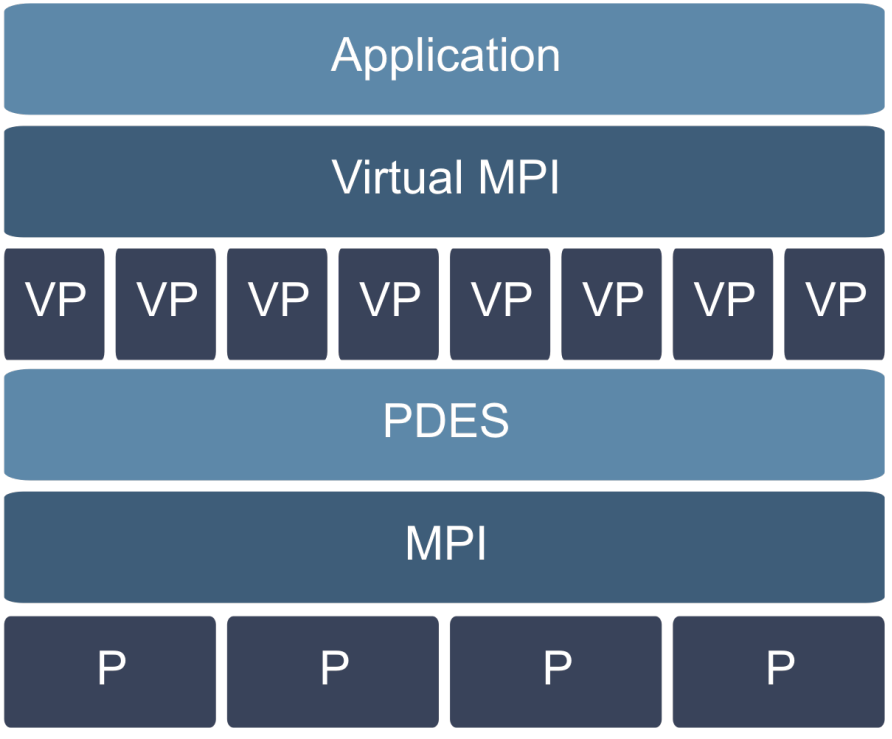
xSim: The Extreme-Scale Simulator

- Execution of real applications, algorithms or their models atop a simulated HPC environment for:
  - Performance evaluation, including identification of resource contention and underutilization issues
  - Investigation at extreme scale, beyond the capabilities of existing simulation efforts
- xSim: A highly scalable solution that trades off accuracy



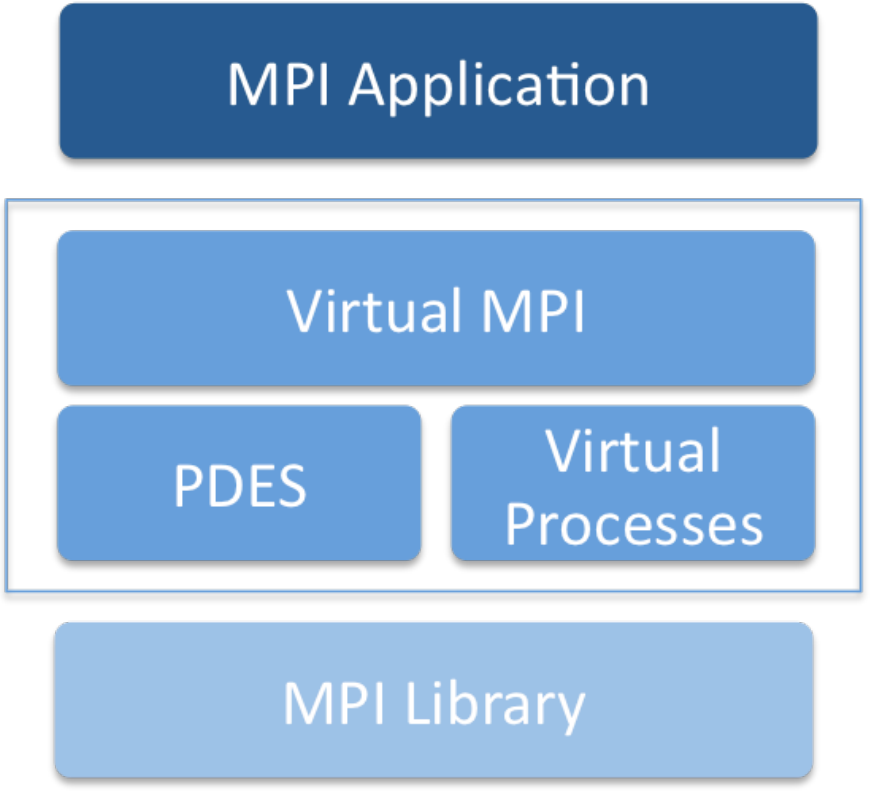
xSim: Technical Approach

- Combining highly oversub-scribed execution, a virtual MPI, & a time-accurate PDES
- PDES uses the native MPI and simulates virtual procs.
- The virtual procs. expose a virtual MPI to applications
- Applications run within the context of virtual processors:
  - Global and local virtual time
  - Execution on native processor
  - Processor and network model

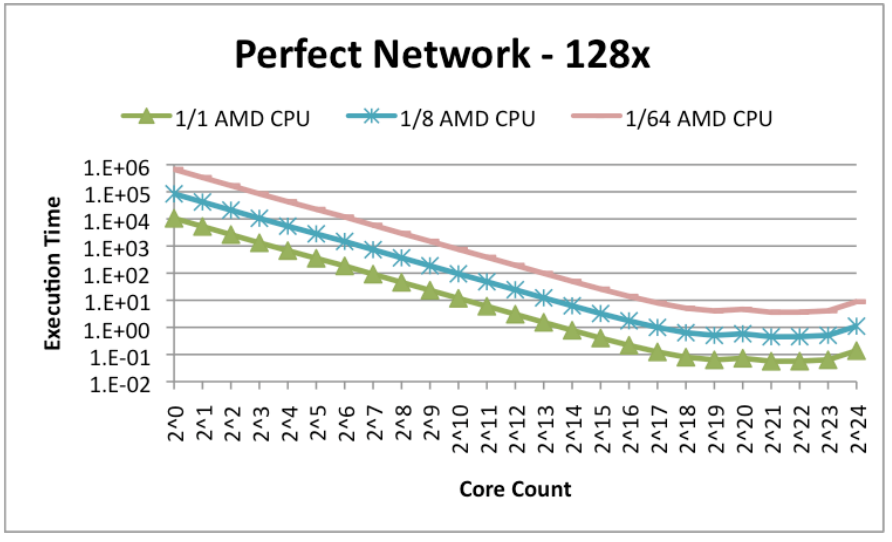


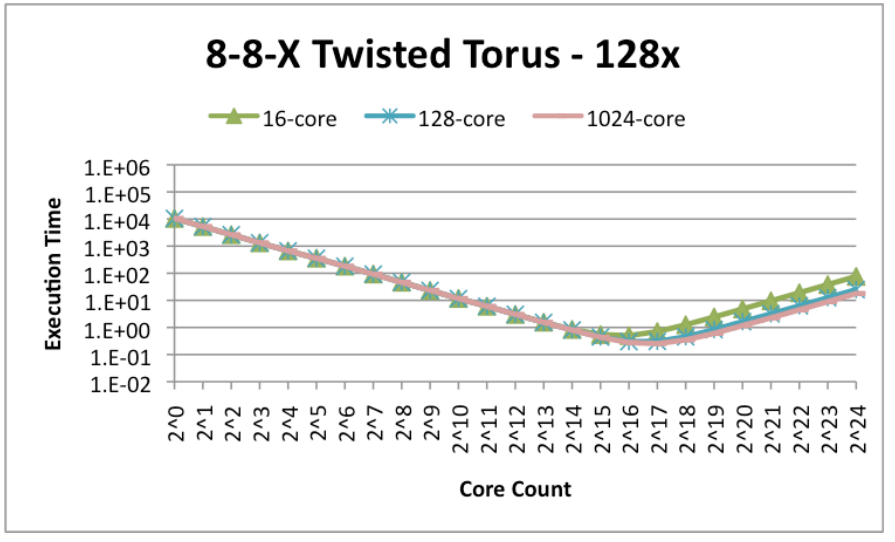
xSim: Design

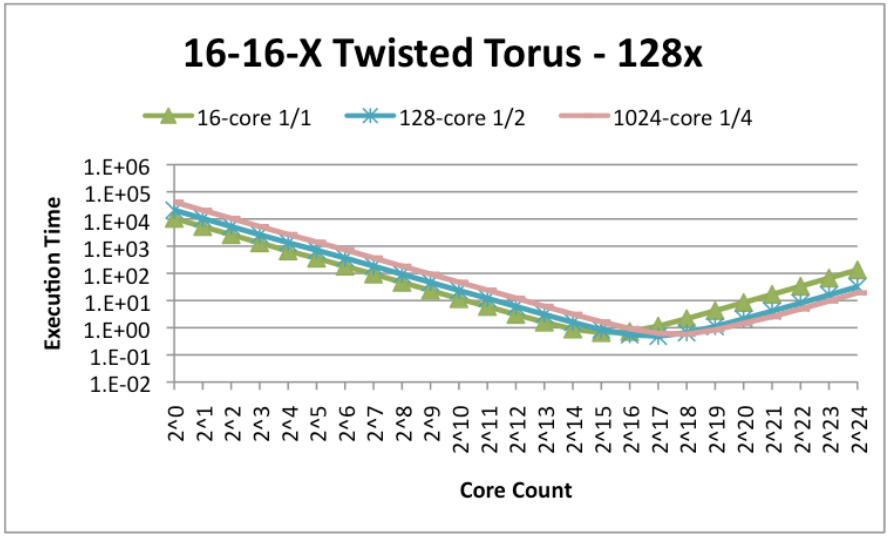
- The simulator is a library
- Utilizes PMPI to intercept MPI calls and to hide the PDES
- Implemented in C with 2 threads per native process
- Support for C/Fortran MPI
- Easy to use:
  - Compile with xSim header
  - Link with the xSim library
  - Execute: *mpirun -np <np> <application> -xsim-np <vp>*

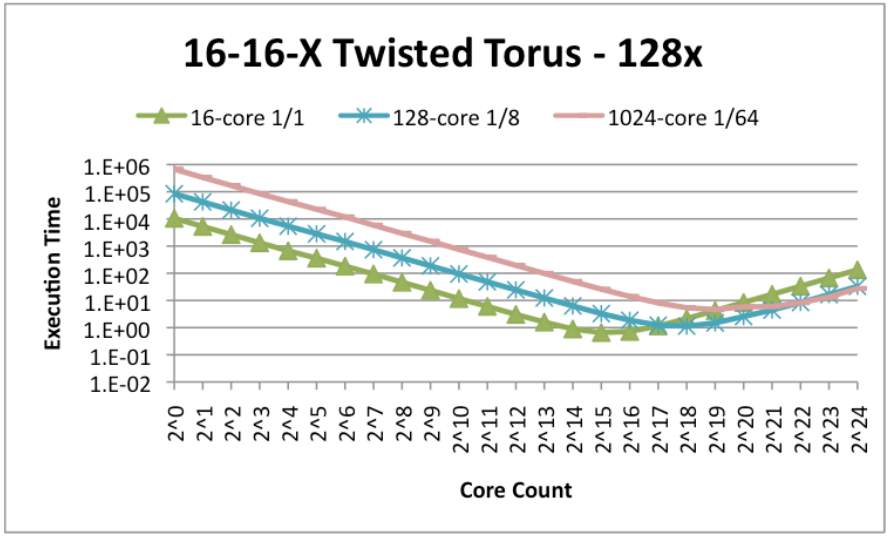


Scaling a Monte Carlo Solver to 2^24 Ranks









Investigating Performance Under Fault

- Parameterized MPI process failure injection/detection
- Parameterized MPI job failure injection/detection and restart
- First co-design toolkit supporting checkpoint/restart:

Table 1: Varying the checkpoint interval and system MTTF

$MTTF_S$	C	E_1	E_2	F	$MTTF_A$
—	1,000	5,248 s	—	0	—
6,000 s	500	5,258 s	7,957 s	1	3,978 s
6,000 s	250	6,377 s	7,074 s	1	3,537 s
6,000 s	125	6,601 s	6,750 s	1	3,375 s
3,000 s	500	5,258 s	10,584 s	2	3,528 s
3,000 s	250	6,377 s	8,618 s	2	2,872 s
3,000 s	125	6,601 s	7,948 s	2	2,649 s

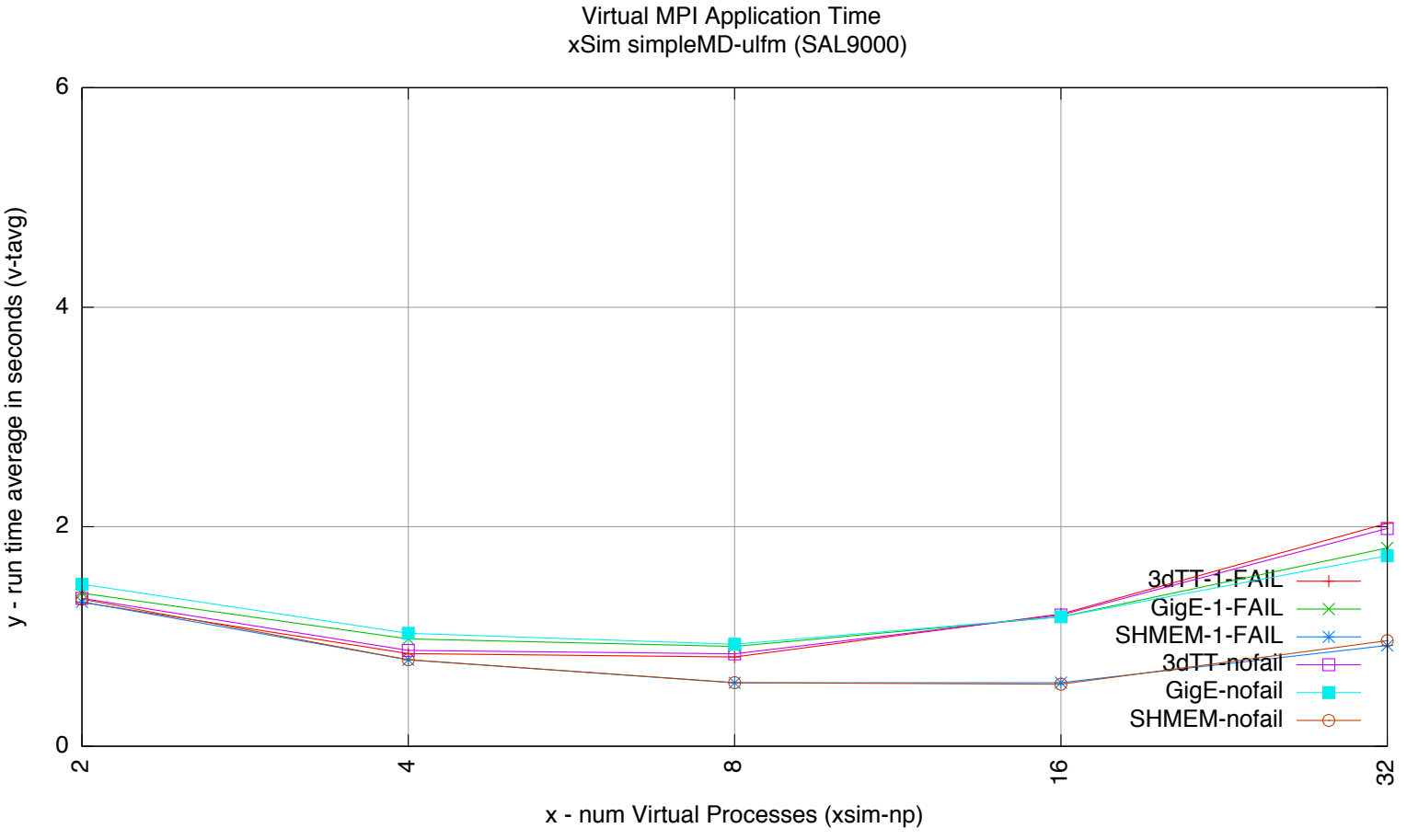
User-Level Failure Mitigation (ULFM)

- User-Level Failure Mitigation (ULFM)
  - Enhancements to the MPI standard proposed by the MPI-FTWG
  - Permits application-level handling of MPI process faults

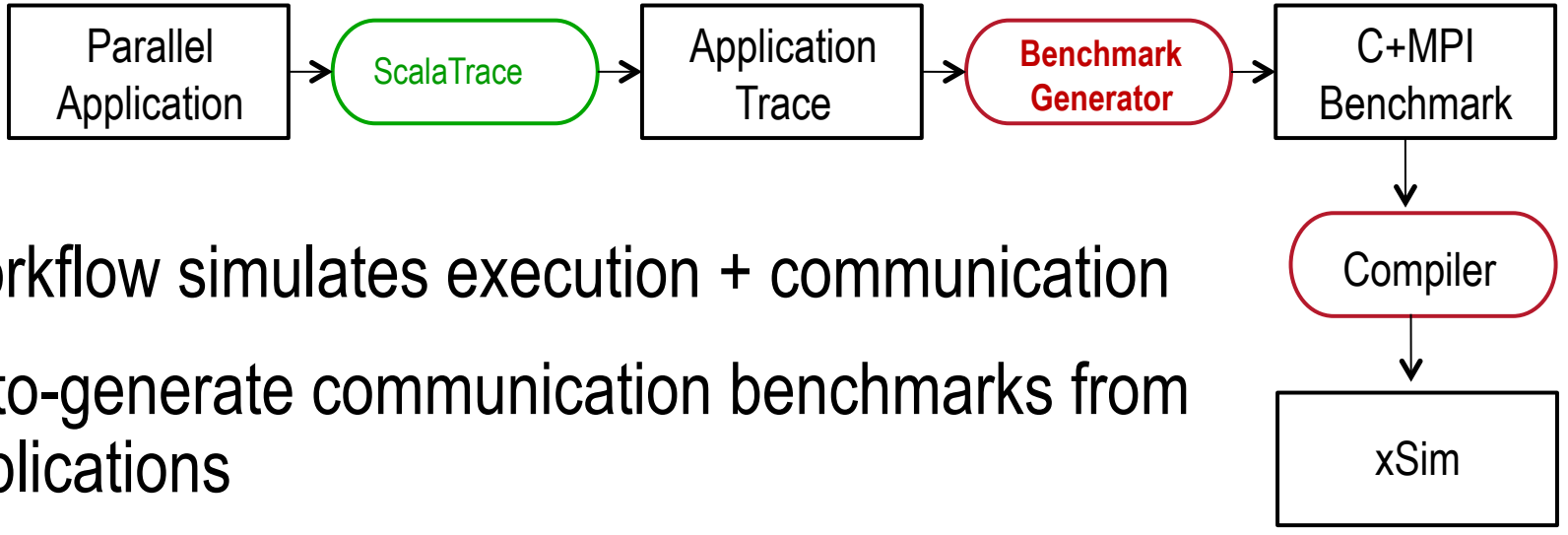
    - **MPI_ERR_PROC_FAILED/MPI_ERR_PENDING**
    - **MPI_Comm_failure_ack(MPI_Comm comm)**
    - **MPI_Comm_failure_get_acked(MPI_Comm comm, MPI_Group *failed_grp)**
    - **MPI_Comm_revoke(MPI_Comm comm)**
    - **MPI_Comm_shrink(MPI_Comm *comm, MPI_Comm *newcomm)**
    - **MPI_Comm_agree(MPI_comm comm, int *flag);**

Investigating Performance Under Fault

- Implemented ULFM support in xSim

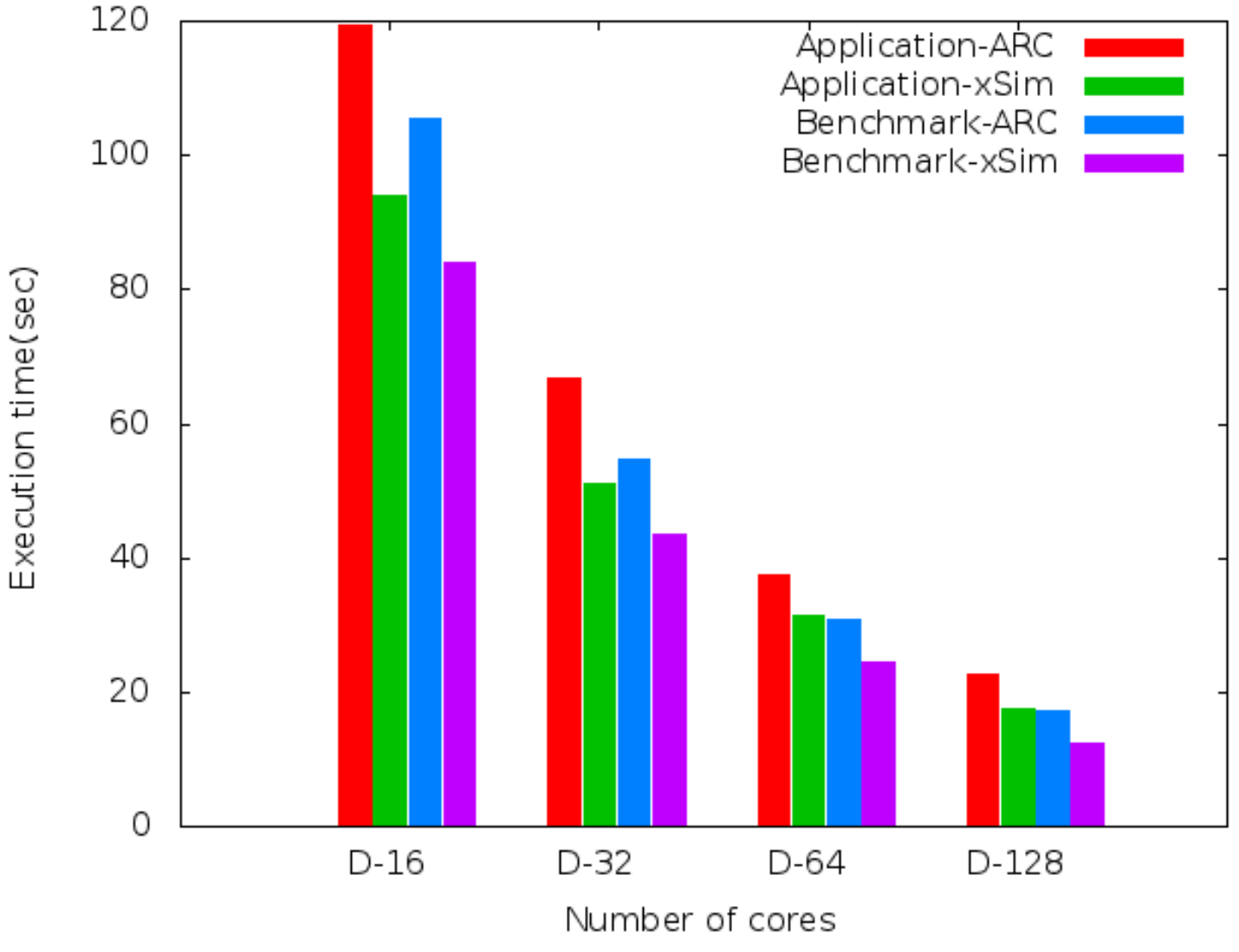


ScalaBenchGen + xSim



- Workflow simulates execution + communication
- Auto-generate communication benchmarks from applications
  - Communication pattern preserved
  - Concise, performance-accurate benchmark code
  - Obfuscates proprietary/controlled codes
- Platform agnostic → exascale performance analysis
  - Vary the sleep times to reflect the difference in CPU speed

Preliminary Results – NAS IS



Conclusion & Future Work

- Initial implementation of a novel way to investigate MPI application performance on future-generation HPC systems
  1. Run applications on existing HPC systems to extract MPI traces
  2. Generate simplified benchmarks from the extracted MPI traces
  3. Run the generated benchmarks in a simulated HPC system
- We are currently running more numbers for the NAS BP
- Implementation will be extended in the future to offer:
  - More MPI support
  - Better accuracy
- Big open question:
  - Can we generate benchmarks from fault tolerant MPI applications?

Questions