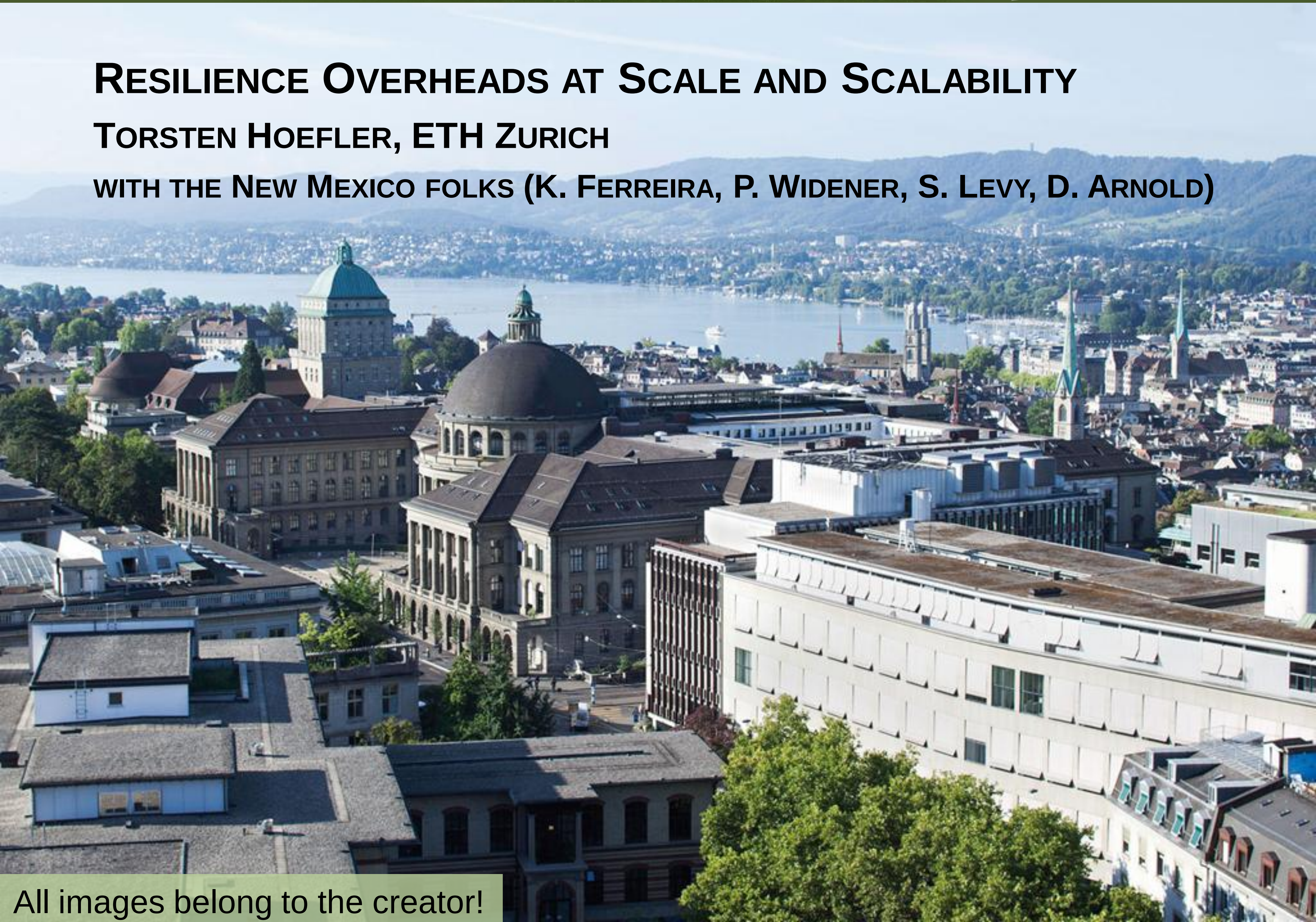


Resilience Overheads at Scale and Scalability

Torsten Hoefler, ETH Zurich

with the New Mexico folks (K. Ferreira, P. Widener, S. Levy, D. Arnold)

All images belong to the creator!

Fault-tolerance Interfaces

- **Very simple**
  - Coordinated Checkpointing

    *void take_coordinated_checkpoint(void *data, int size, char* output)*
  - Uncoordinated Checkpointing

    *void take_uncoordinated_checkpoint(void *data, int size, char* output)*

- **But complex to use**
  - Which option? Coordinated, uncoordinated?
  - Where to write files to (HD, SSD, parallel FS)?

Overall Goal of the Project: Exascale Analysis

- **Evaluate overall scalability of resilience techniques**
  - For very large scale systems [PMBS'13]

- **Offer a freely available framework for reproducible work**
  - Provide traces for key DOE workloads [trace repo]
  - Enables cross-validation of results [LSAP'10]

- **Evaluate scalability of uncoordinated checkpoint/restart (uCR) for DOE workloads [SC14]**
  - Identify issues
  - Investigate solutions

    Clustered checkpointing [SC14]

    Nonblocking collectives [EuroMPI'14]

[LSAP'10]: TH, Schneider, Lumsdaine: LogGOPSim - Simulating Large-Scale Applications in the LogGOPS Model
[PMBS'13]: Widener, Ferreira, Levy, Hoefler: Exploring the effect of noise on the performance benefit of nonblocking allreduce
[SC14]: Ferreira, Widener, Levy, Arnold, TH: Understanding the Effects of Communication and Coordination on Checkpointing at Scale
[trace repo]: http://htor.inf.ethz.ch:8888/

Take Away Messages

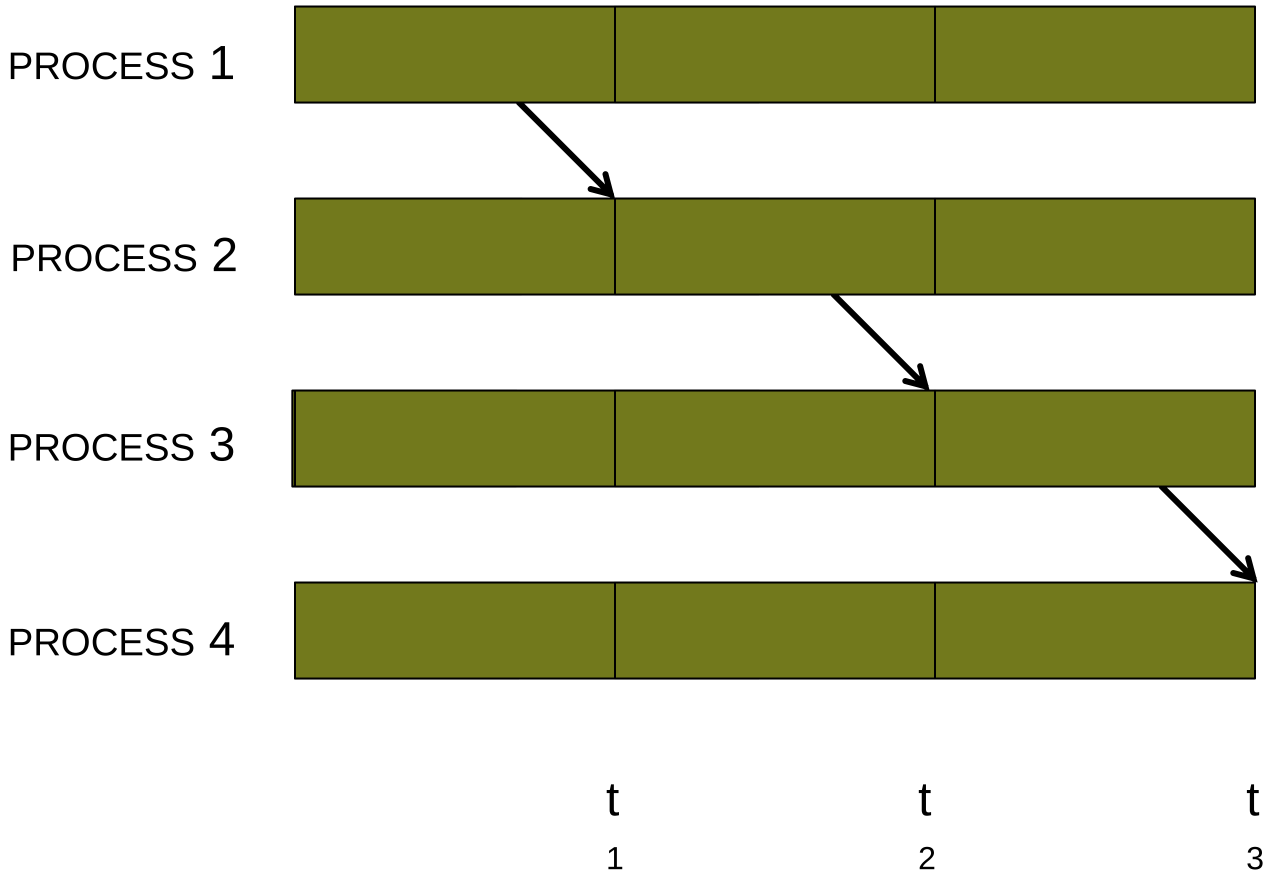
- **The effect of happens-before delay chains:**
  1. Local checkpoints can have a greater performance impact than message logging overheads for uCR;
  2. An application's communication pattern dictates whether uCR checkpoint overheads are amplified or absorbed;
  3. Collective communication limits the extent to which the execution run-ahead of surviving processes actually improves overall application execution time.

- **Mitigation strategies:**
  1. Checkpoint clustering protocols can be used to improve uCR performance
  2. Nonblocking collective communication

- **Reproducing results: LogGOPSim [LSAP'10,online]**

[LSAP'10]: TH, Schneider, Lumsdaine: LogGOPSim - Simulating Large-Scale Applications in the LogGOPS Model
[online]: http://spcl.inf.ethz.ch/Research/Performance/LogGOPSim/

Resilience Today: Coordinated Checkpoint/Restart (cCR)



Coordinated Checkpoint/Restart

- **Dominance due to a number of key assumptions**
  - Some of which may continue to hold true for future systems:

    Failure that do not crash the system (SDC) are rare

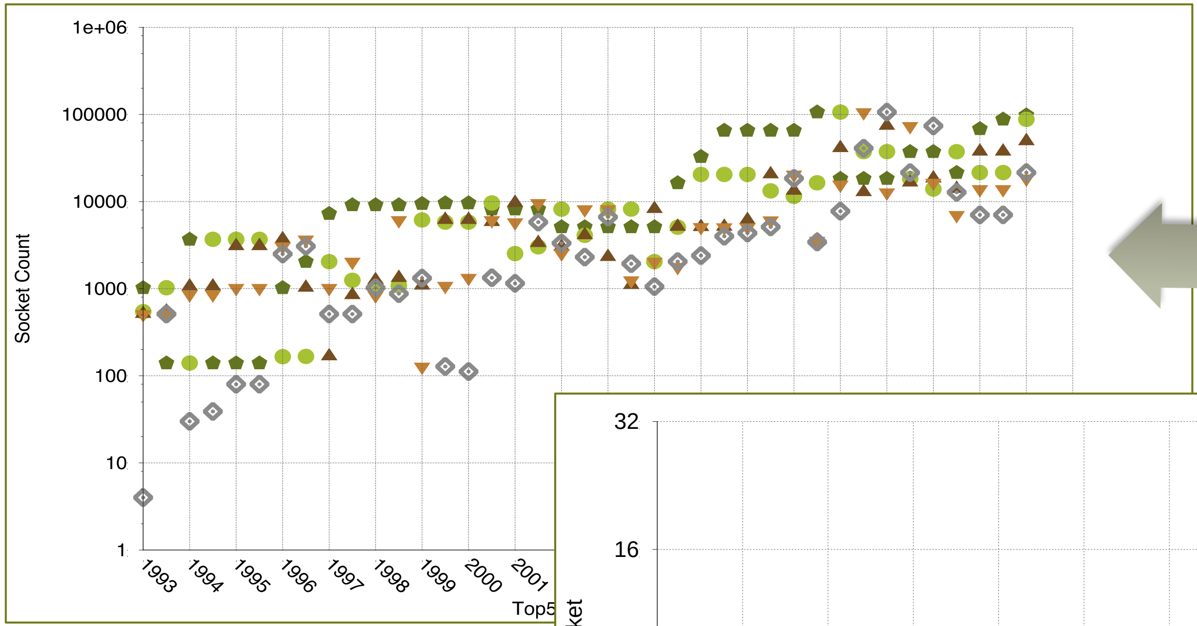
    Checkpoints used for other purposes (i.e , steering, viz)

  - Some of which may not:

    Application state can be saved and restored much more quickly than a system's mean time to interrupt (MTTI)

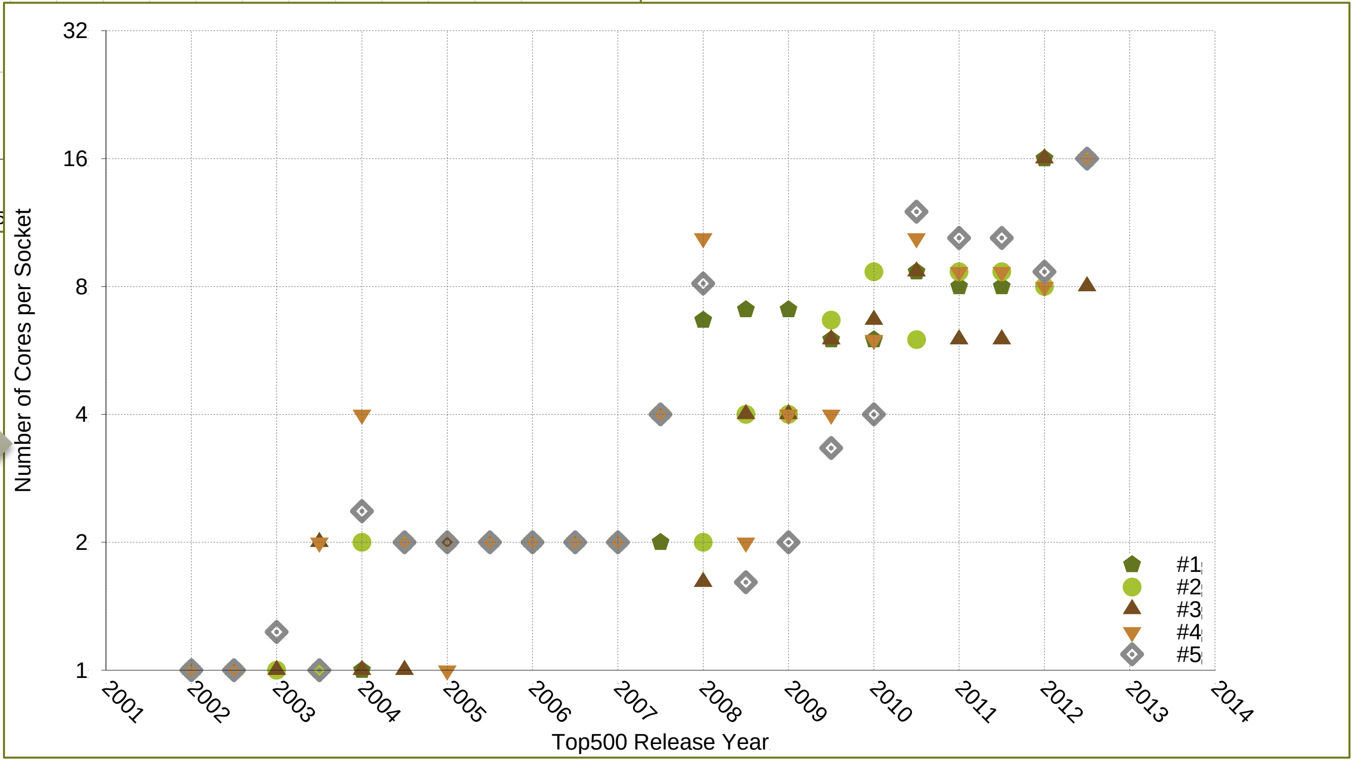
    The hardware and upkeep (e.g., power) costs of supporting frequent checkpointing is a modest portion (currently perhaps 10-20%) of the system's overall cost

Systems Growing, Decreasing in Reliability

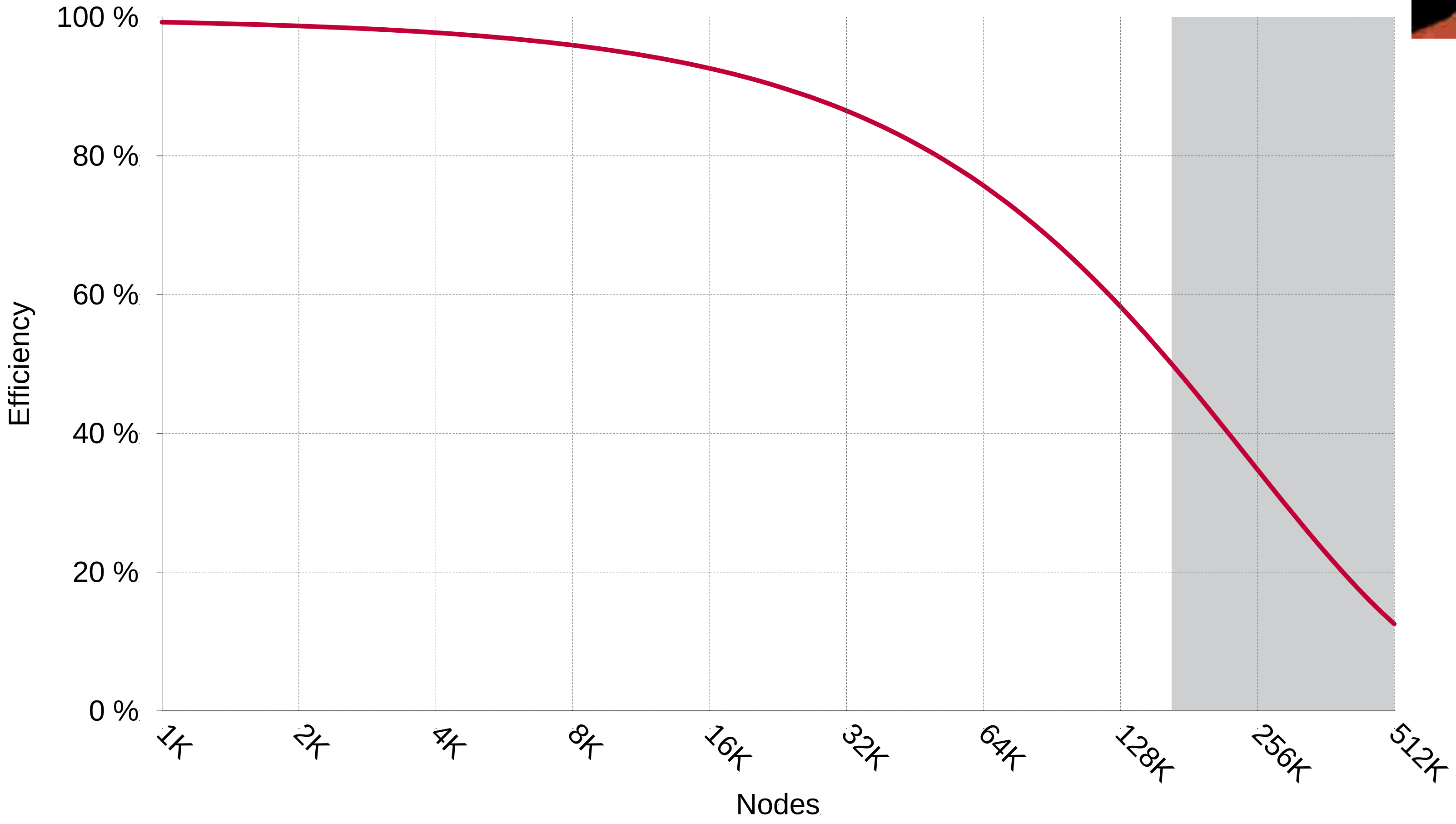


Systems are getting larger

Each node is getting more complicated

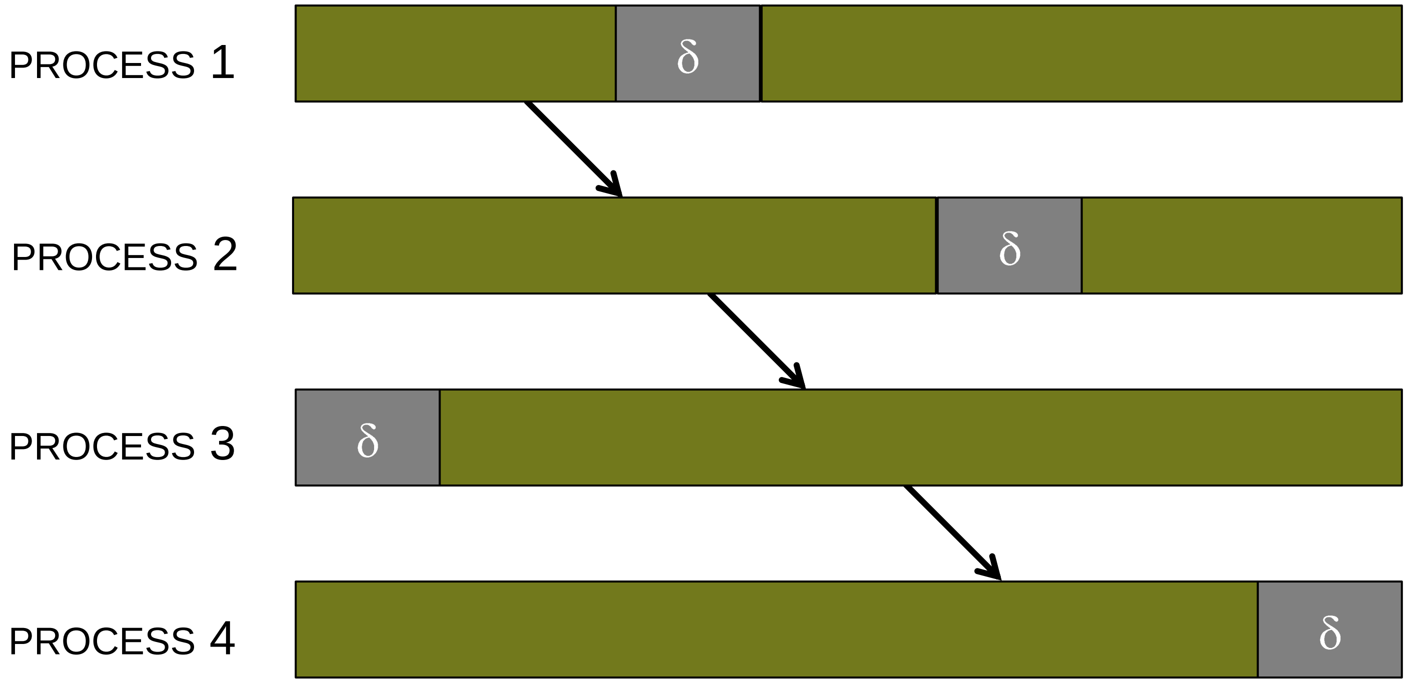


Therefore Coordinated CR will not Scale



Node MTBF: 25 years, Checkpoint/Restart Time: 5 minutes,
Checkpoint Frequency: Optimal interval due to Young, walltime due to Daly

Uncoordinated Checkpointing to the Rescue



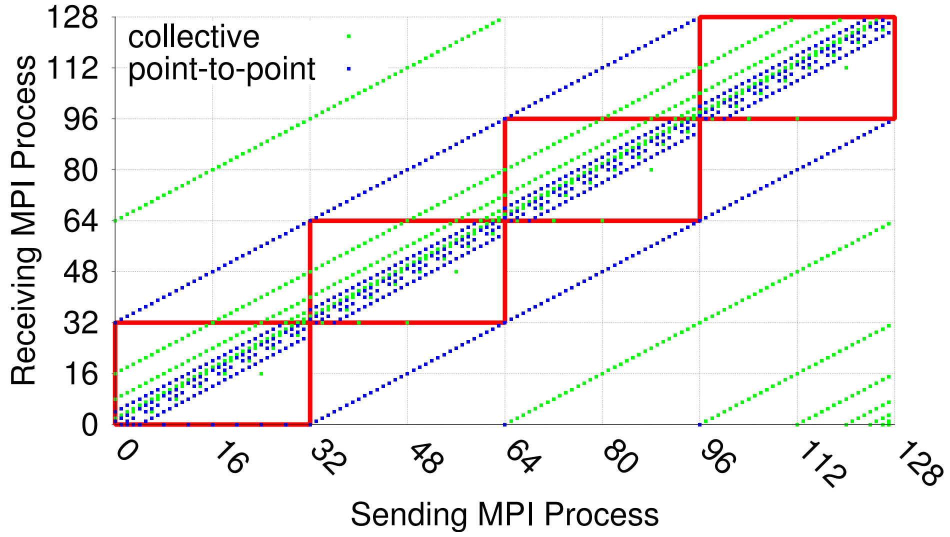
uCR to the Rescue (cont'd)

- **Advantages:**
  - Each node checkpoints independently, reducing expensive synchronization and possible resource contention
  - Upon failure, only failed nodes restart rather than all nodes (may save power)

- **Drawbacks:**
  - Potentially expensive message logging protocols needed to ensure checkpoint consistency

Related Work on Reducing Message Log Sizes

- **Send Determinism [IPDPS'11]**
  - Common deterministic property of applications that can be exploited to minimize message logging volume
- **Hierarchical (clustered) Checkpointing [IPDPS'12]**

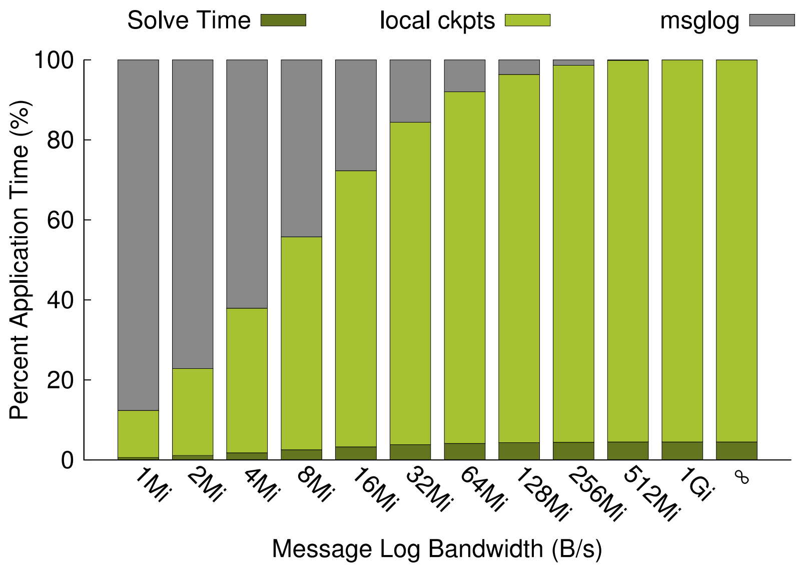


- cCR within a cluster
- uCR across clusters
- Only messages crossing clusters need to be logged

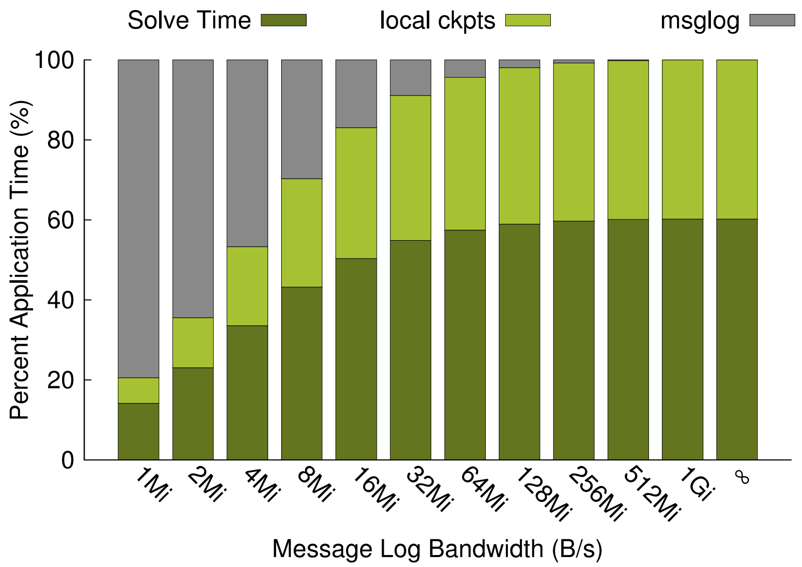
- **Demand checkpointing [HPDC'14]**
  - Reduce log size by forcing other processes to checkpoint

[IPDPS'11]: Guermouche, Ropars, Brunet, Snir, Cappello: Uncoordinated Checkpointing Without Domino Effect for Send-Deterministic MPI Applications
[IPDPS'12]: Guermouche, Ropars, Snir, Cappello: HydEE: Failure Containment without Event Logging for Large Scale Send-Deterministic MPI Application
[HPDC'14]: M. Besta, TH: Fault Tolerance for Remote Memory Access Programming Models

As storage bandwidth increases, uCR checkpoint overheads dominate



CTH @ 64K Processes



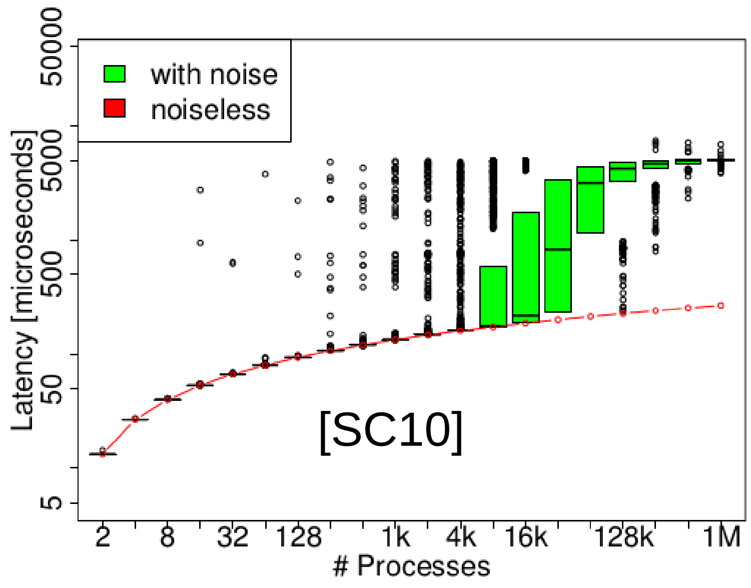
LAMMPS @ 64K Processes

Our Focus: uCR local checkpoint overheads

Questions to Consider

- **Question I: How does uCR perform at large scale?**
- **Question II: How does uCR compare with cCR at scale?**
- **Question III: What applications characteristics contribute to uCR’s performance?**
- **Question IV: How can we improve uCR performance?**

Our Approach: Simulation

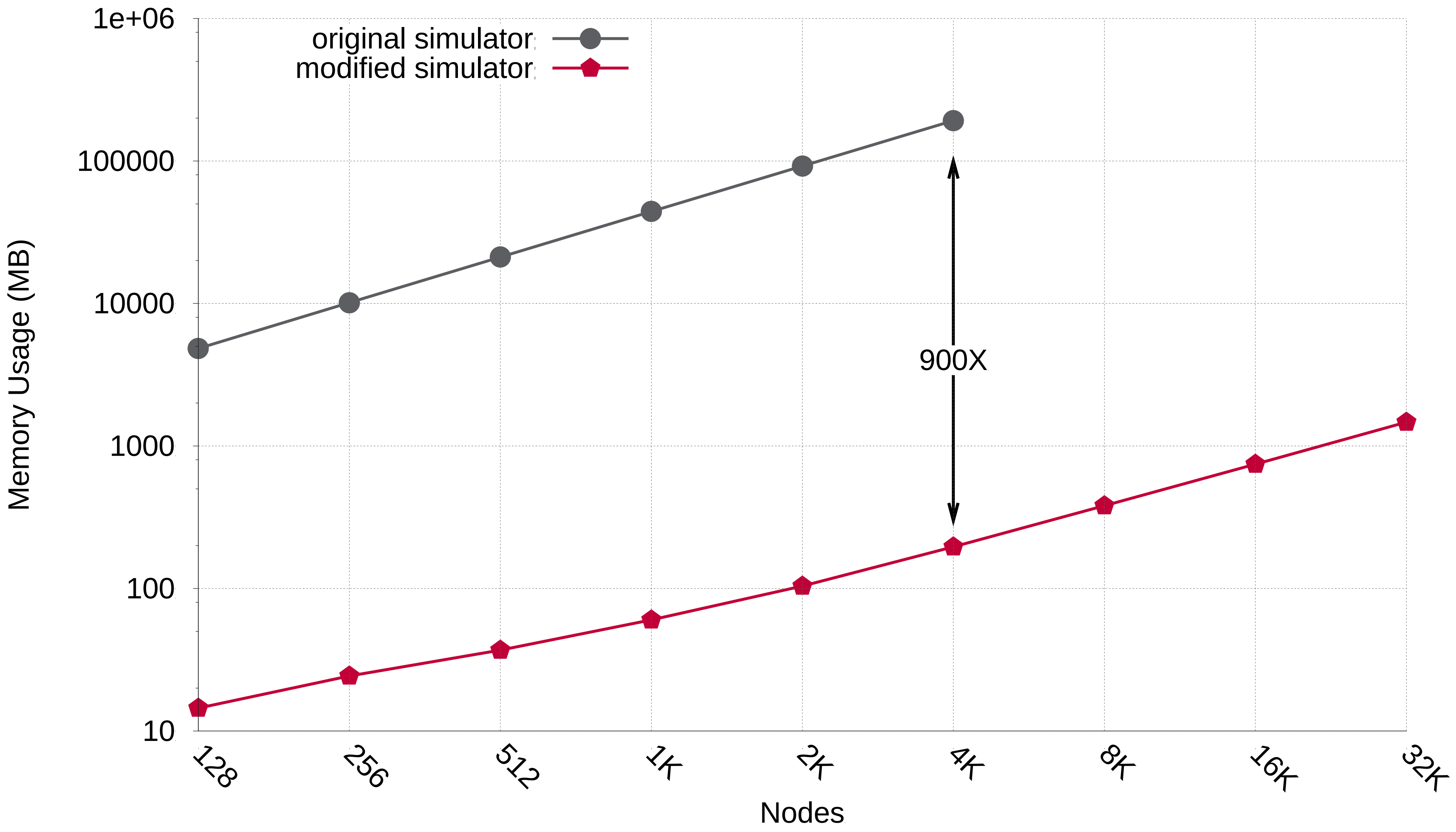


- **LogGOPSim-based simulation toolkit**
  - LogP-based simulator [LSAP'10]
  - Previously validated accurate for both cCR and uCR [PMBS'13]
  - Feed in application traces and overheads due to resilience mechanisms, get out per-node wall times
  - Increased scale achieved through trace extrapolation functionality

- **Extrapolation details:**
  - Collectives: Extrapolated accurately based on node count using well known algorithms
  - Point-to-point: Approximated using a weak-scaling application model

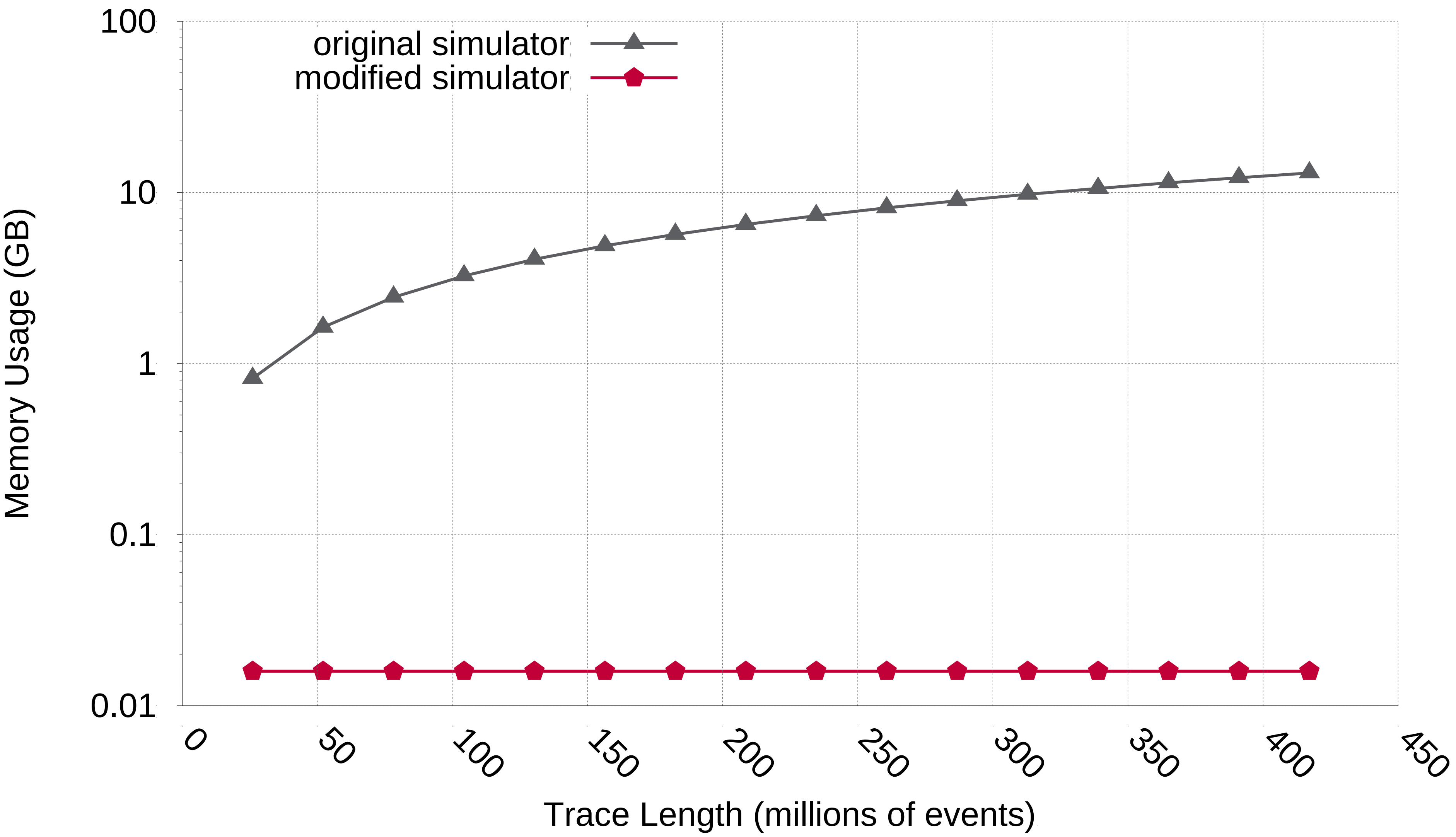
[LSAP'10]: TH, Schneider, Lumsdaine: LogGOPSim - Simulating Large-Scale Applications in the LogGOPS Model
[SC10]: TH, Schneider, Lumsdaine: Characterizing the Influence of System Noise on Large-Scale Applications by Simulation
[PMBS'13]: Widener, Ferreira, Levy, Hoefler: Exploring the effect of noise on the performance benefit of nonblocking allreduce

LogGOPSim Extensions: In-Memory Extrapolation (size)



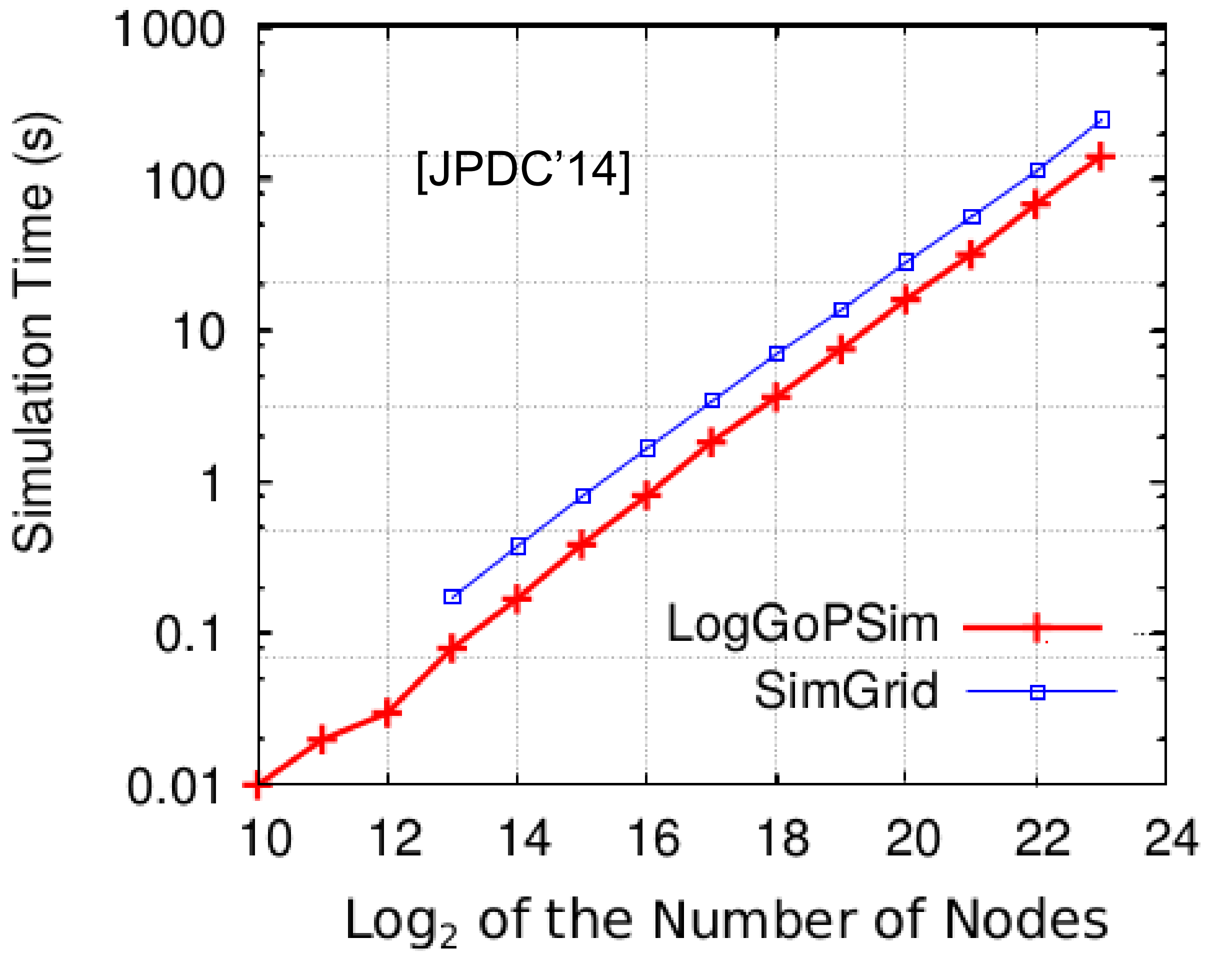
[PMBS'13]: Widener, Ferreira, Levy, Hoefler: Exploring the effect of noise on the performance benefit of nonblocking allreduce

LogGOPSim Extensions: In-Memory Extrapolation (time)

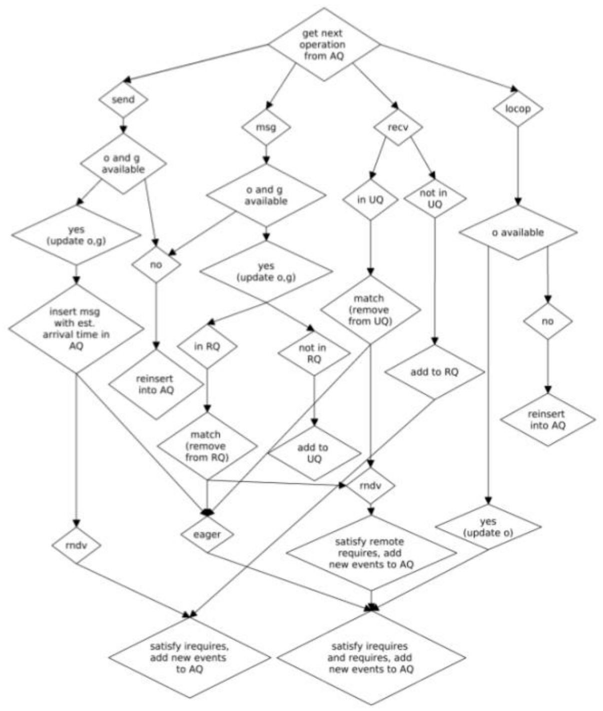


[PMBS'13]: Widener, Ferreira, Levy, Hoefler: Exploring the effect of noise on the performance benefit of nonblocking allreduce

LogGOPSim State of the Art Performance



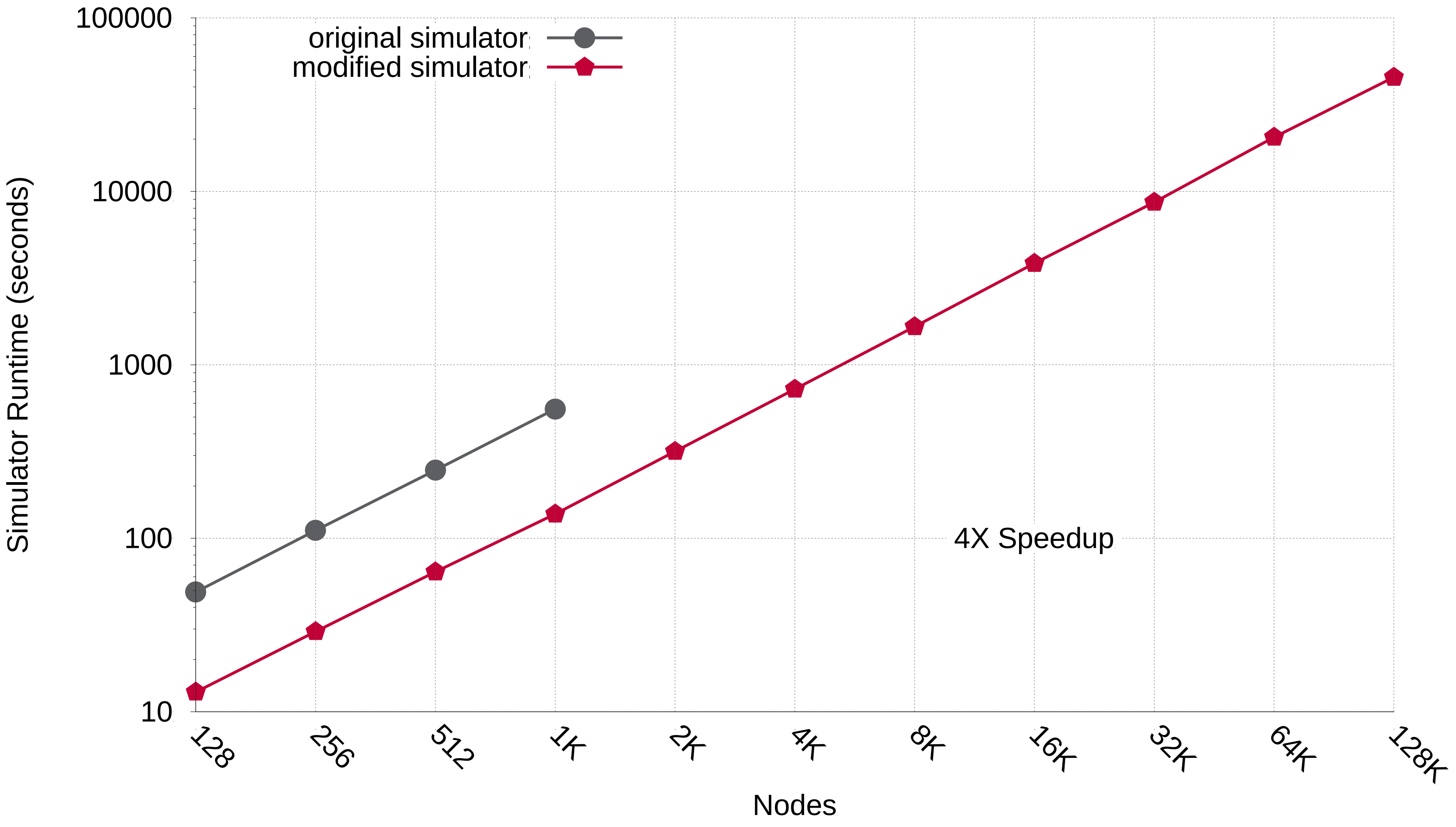
Source: [JPDC'14]



[LSAP'10]

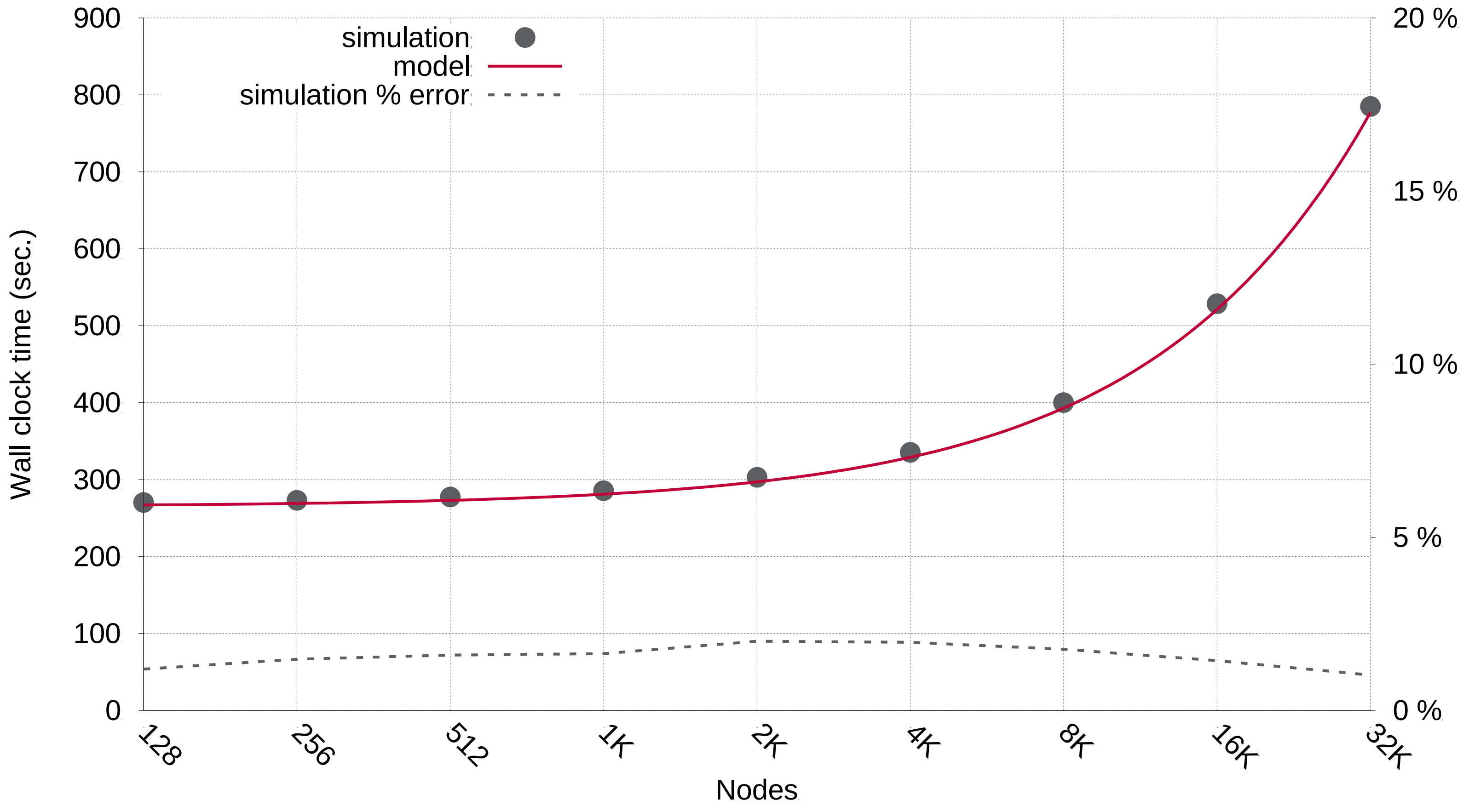
[JPDC'14]: Casanova et al.: Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms.
[LSAP'10]: TH, Schneider, Lumsdaine: LogGOPSim - Simulating Large-Scale Applications in the LogGOPS Model

LogGOPSim Extensions: Performance



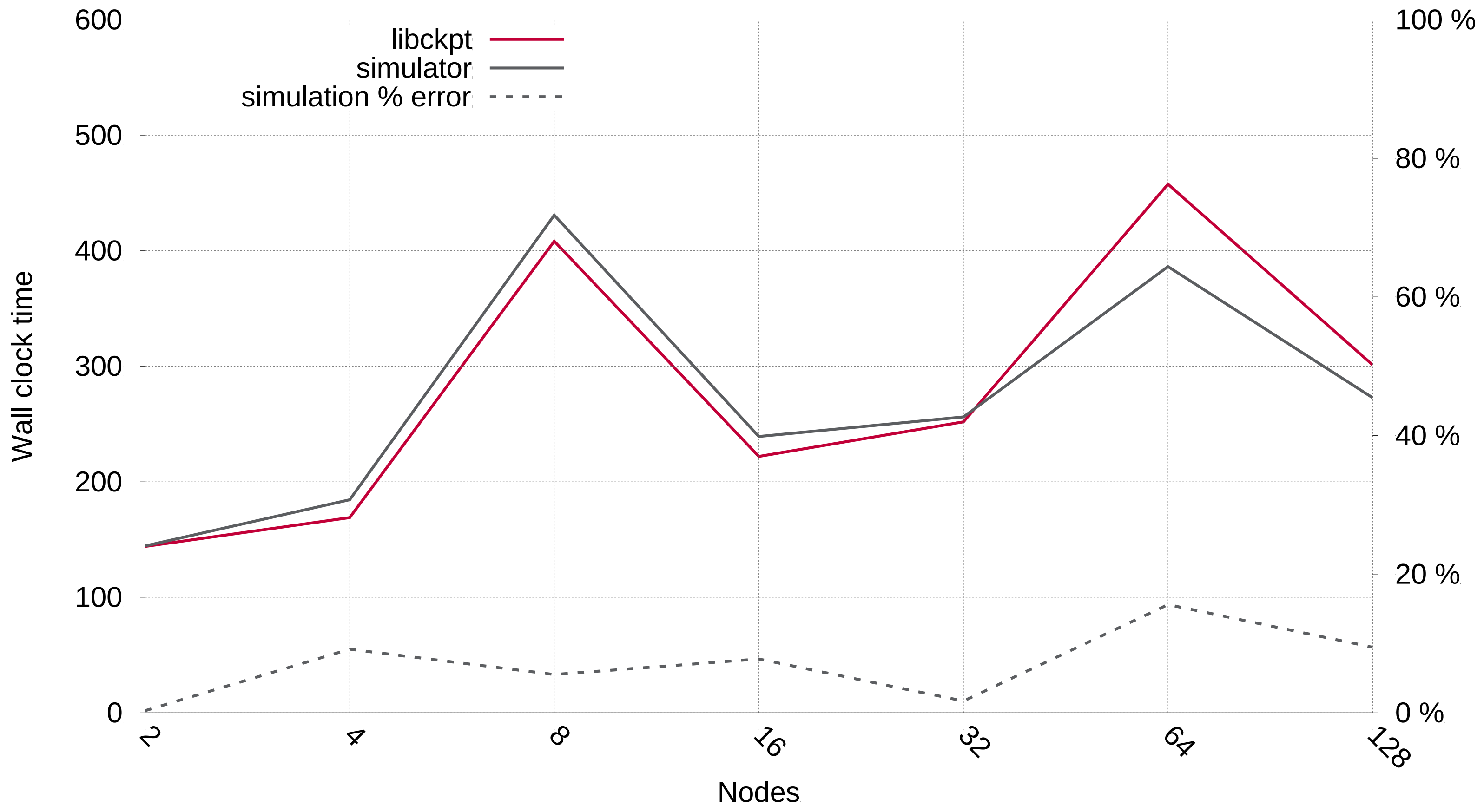
Accuracy validation: analytic model

- **Model of failure-free coordinated checkpointing**
  - LAMMPS within 1%
  - CTH within 3% (see below)

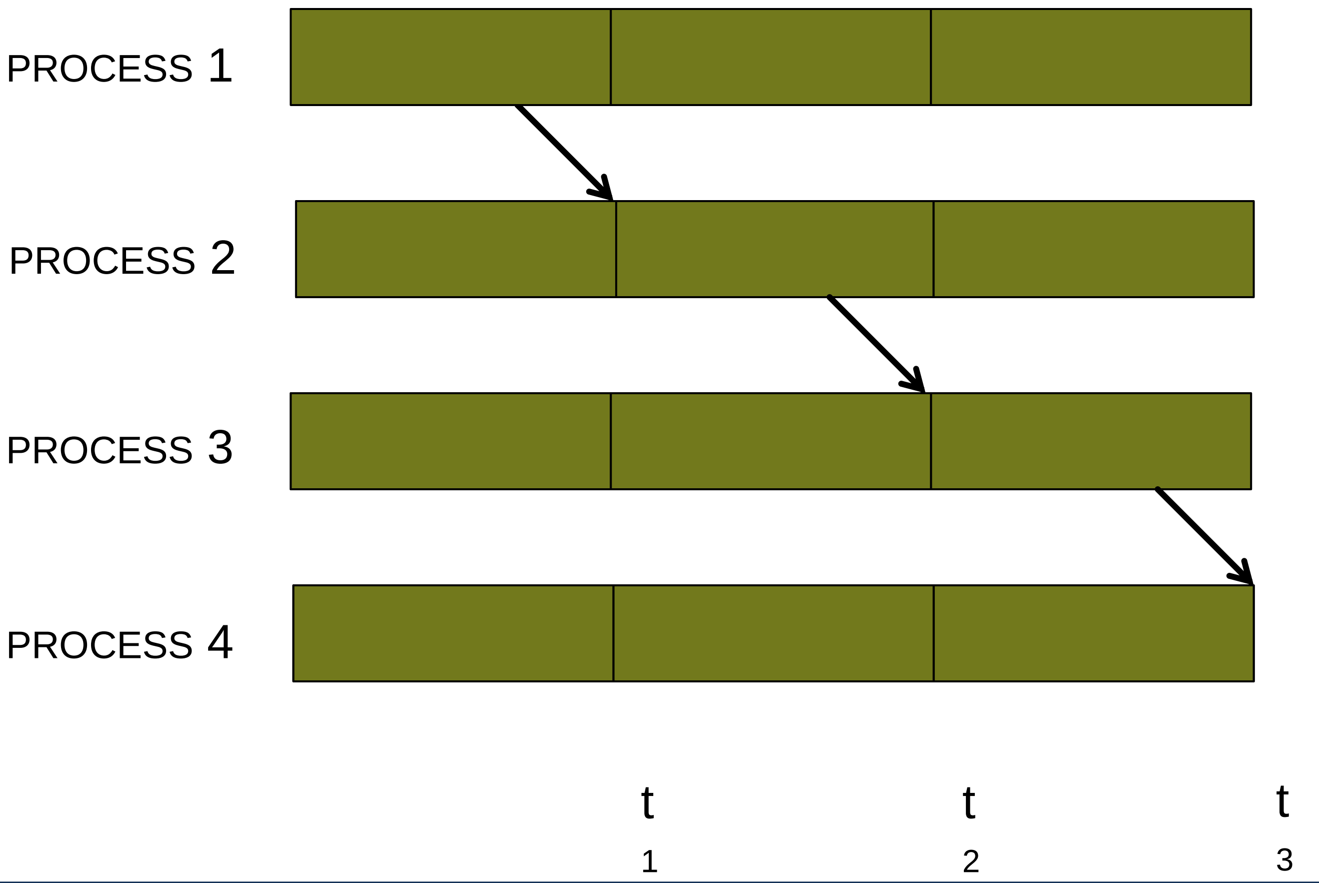


Validation: small-scale testing

- **Tests with coordinated & uncoordinated checkpointing**
  - LAMMPS within 5%
  - CTH within 16% (coordinated checkpointing results shown)



Key Insight: Model uCR as Application Jitter

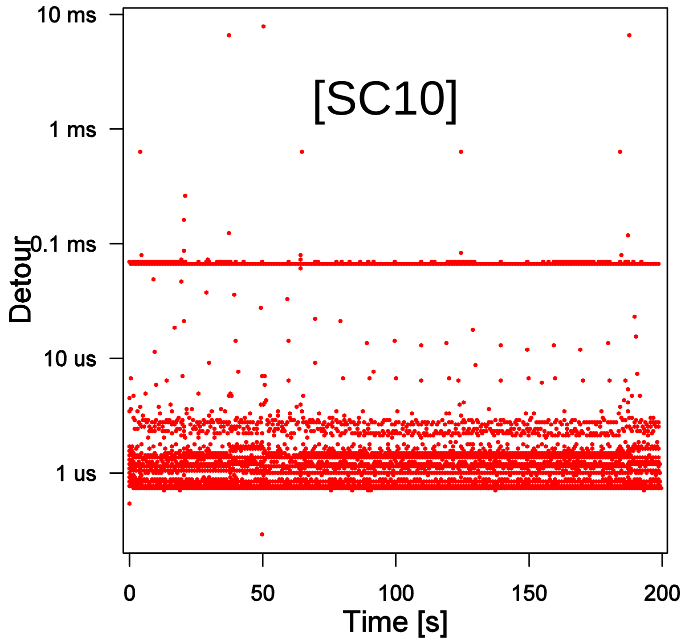


Overheads due to Analogy with OS "Jitter" cf. [SC10]

[SC10]: TH, Schneider, Lumsdaine: Characterizing the Influence of System Noise on Large-Scale Applications by Simulation

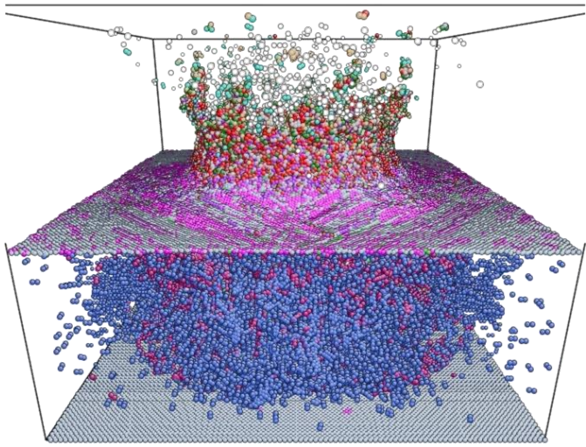
Our Approach (cont'd)

- **Differences between OS noise [SC10] and resilience noise [PMBS'13]**
  - Resilience events order magnitude larger than typical OS interference events.
  - Noise playback is synchronous with application unlike asynchronous OS noise.



[SC10]: TH, Schneider, Lumsdaine: Characterizing the Influence of System Noise on Large-Scale Applications by Simulation
[PMBS'13]: Widener, Ferreira, Levy, Hoefler: Exploring the effect of noise on the performance benefit of nonblocking allreduce

Our Workloads and Setup



- **Key current and future workloads:**
  - Current SNL Applications/Proxies:

    LAMMPS - molecular dynamics code from SNL

    CTH - a shock physics code from SNL

    HPCCG - conjugate gradient solver from mantevo suite
  - Exascale Proxy Applications

    miniFE - a finite element benchmark from mantevo suite

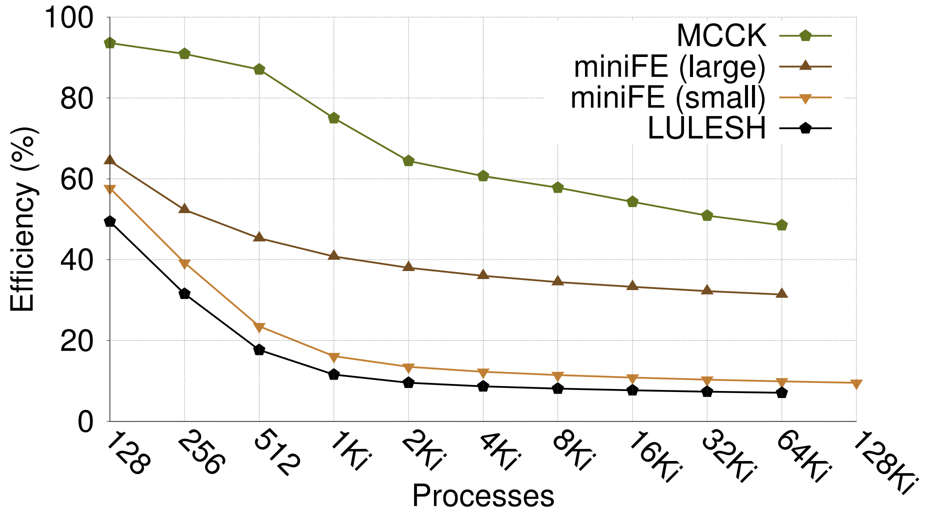
    LULESH – unstructured hydrodynamics benchmark

    MCCK - a neutronics proxy application

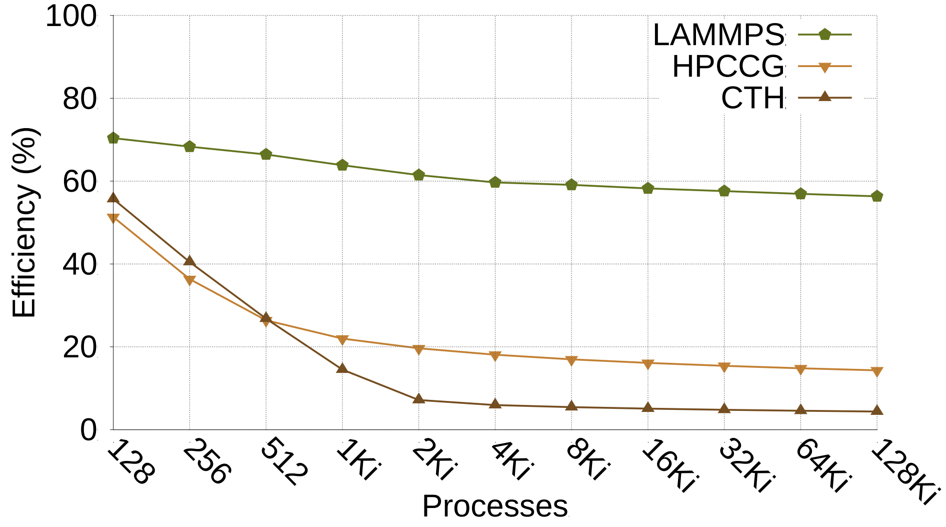
- **Experimental parameters**
  - uCR checkpoint duration: 1 second
  - uCR checkpoint interval: 120 seconds
  - Each node checkpoints independently beginning with a random offset (worst-case scenario)

[SC14]: Ferreira, Widener, Levy, Arnold, TH: Understanding the Effects of Communication and Coordination on Checkpointing at Scale

Q.I: How does uCR perform at scale?



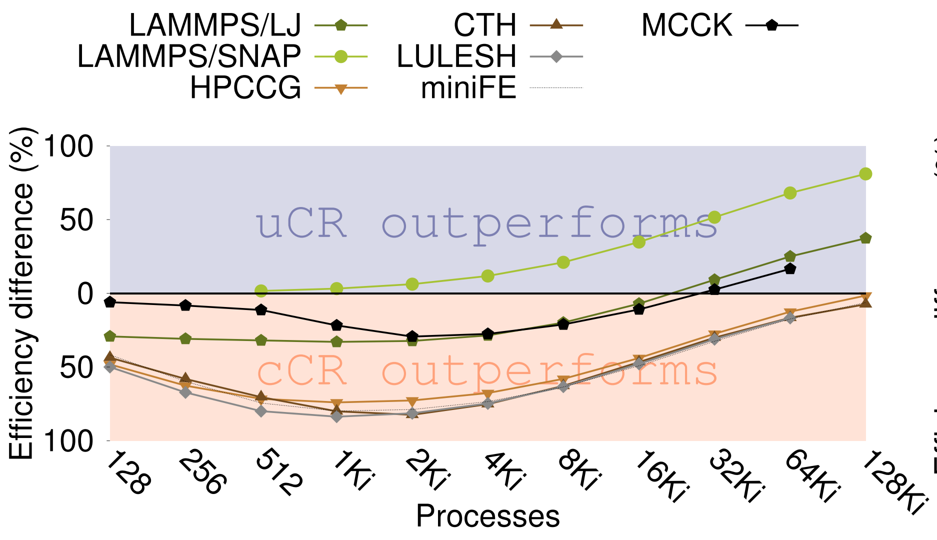
Co-Design Center Proxy Apps



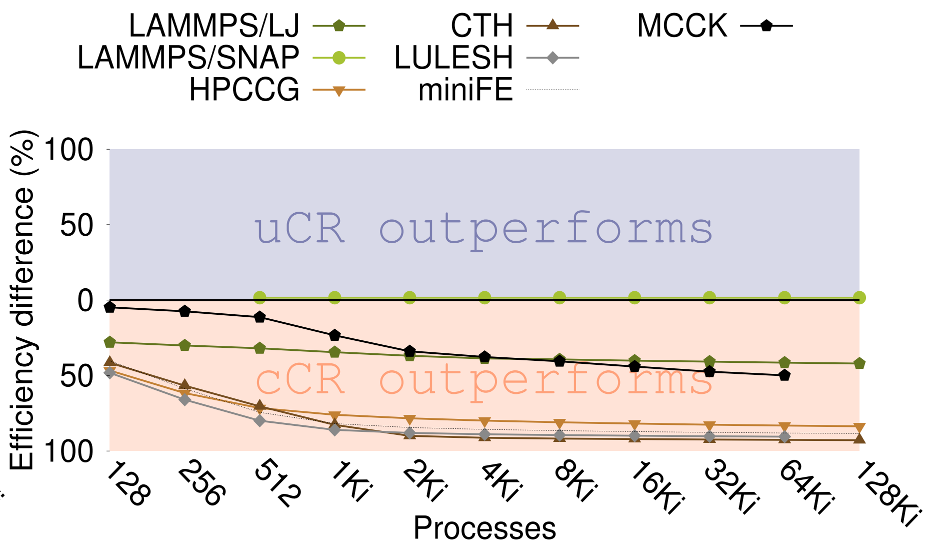
Sandia Workloads

A1: Slowdowns can be significant and increase with scale

Q.II: How does uCR compare to cCR?



Parallel File System – 512 MiB/sec aggregate BW



Local Stable Storage (e.g., SSD) – 2 GiB/sec/process

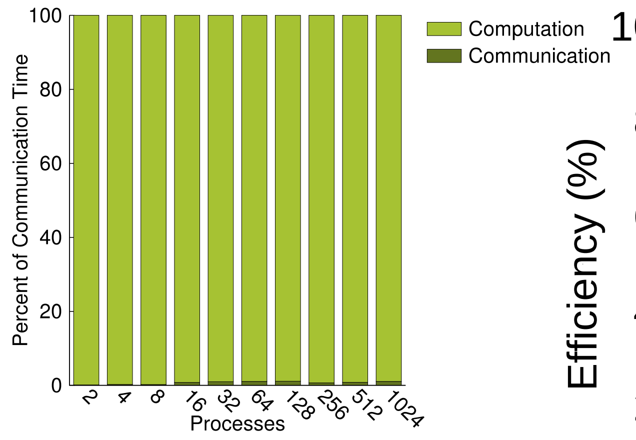
A2: In bandwidth limited scenarios, uCR may outperform cCR

What is causing uCR slowdown?

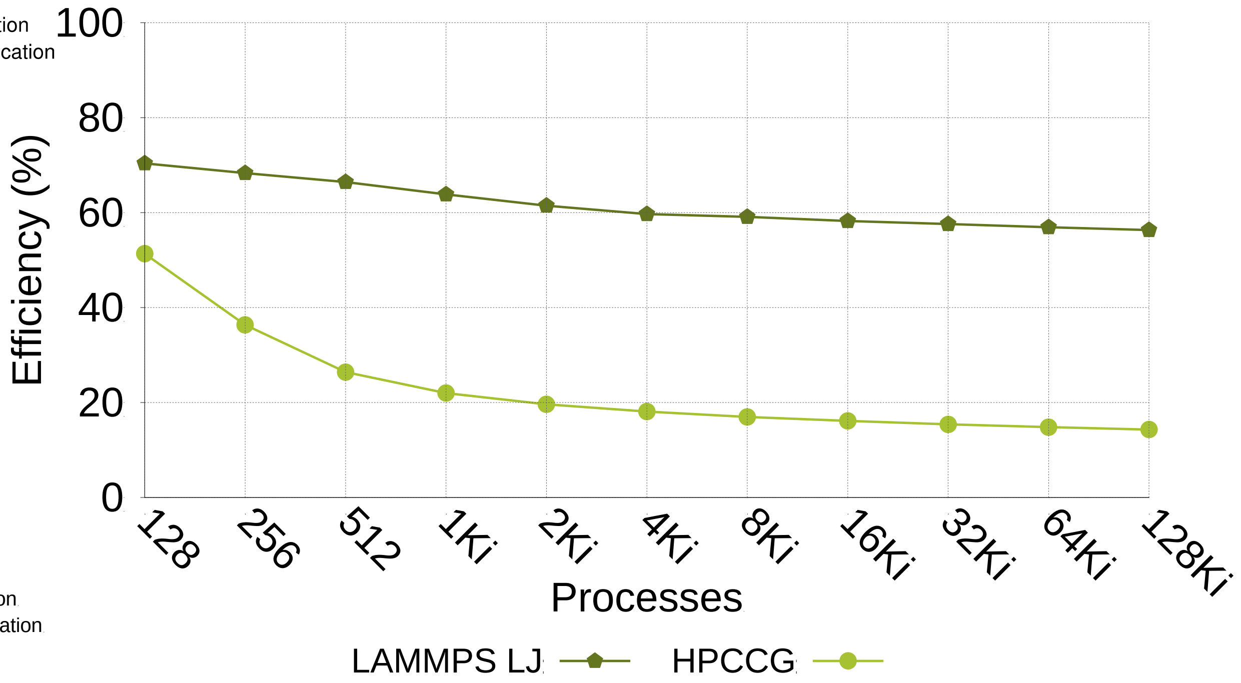
- **Previous experience in OS noise helps guide this search:**

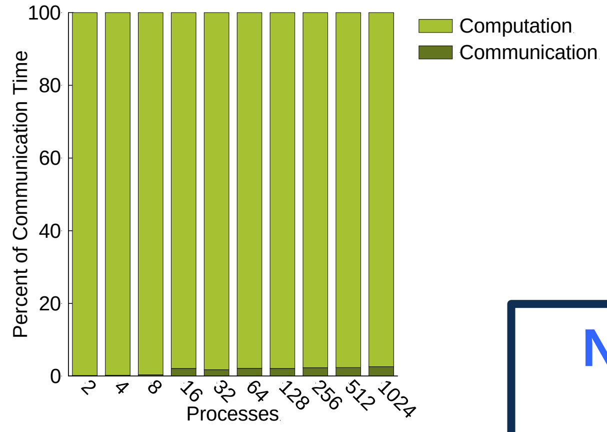
  a) Communication/Computation ratios?
  b) Breakdown of communication operations (i.e., collectives)?
  c) Algorithms used to implement collectives?
  d) ???

a) Time Spent Communicating?



HPCCG

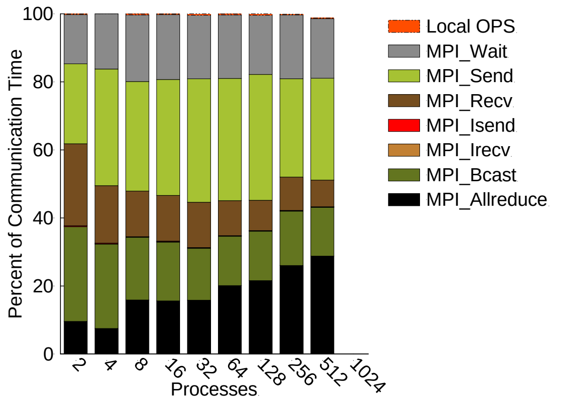




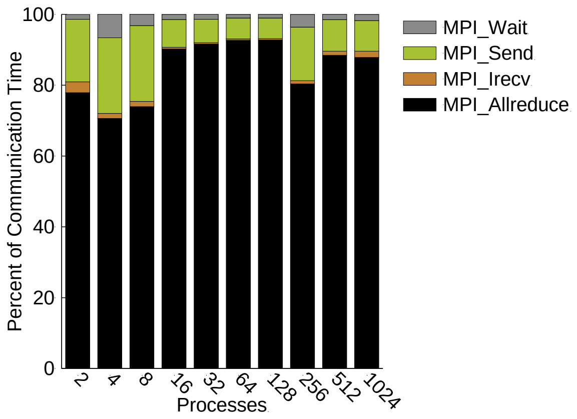
LAMMPS

Nope, does not appear to be correlated to time spent communicating

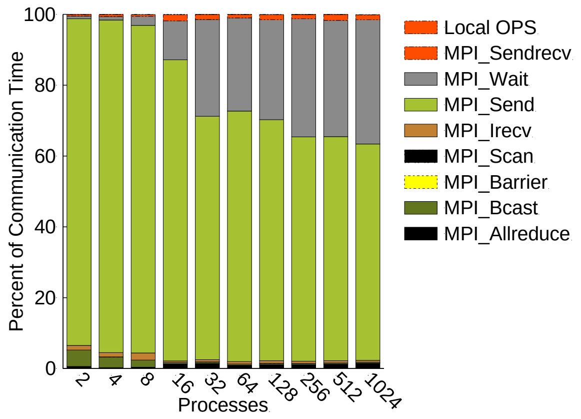
b) Type of Communication?



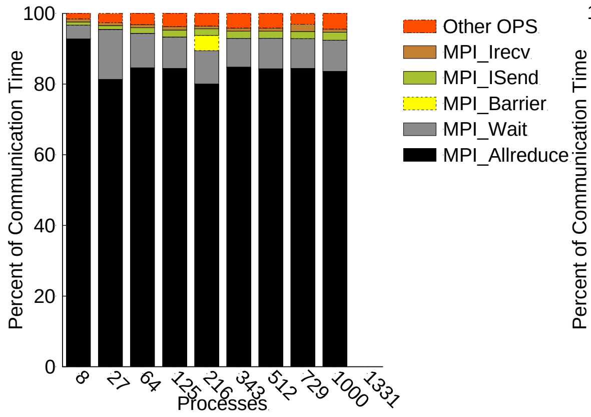
CTH



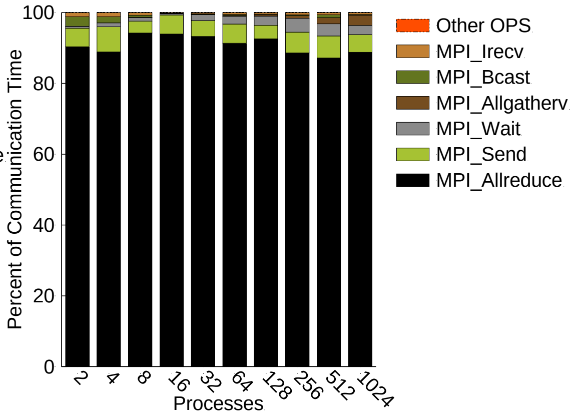
HPCCG



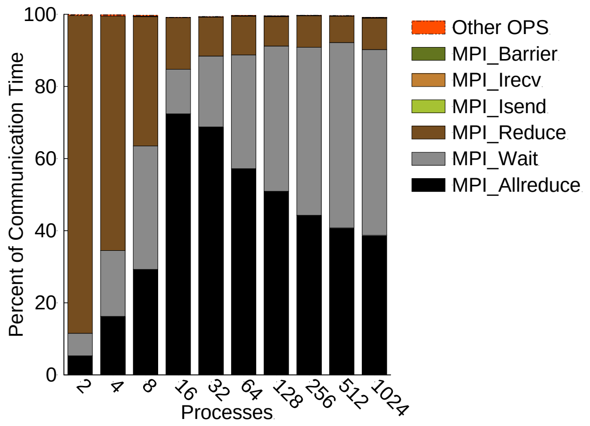
LAMMPS



LULESH



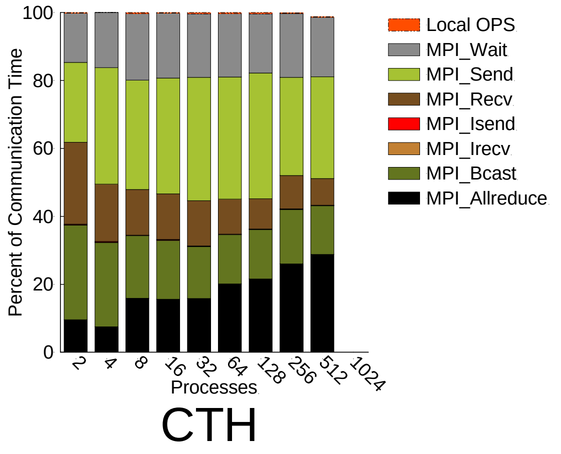
miniFE

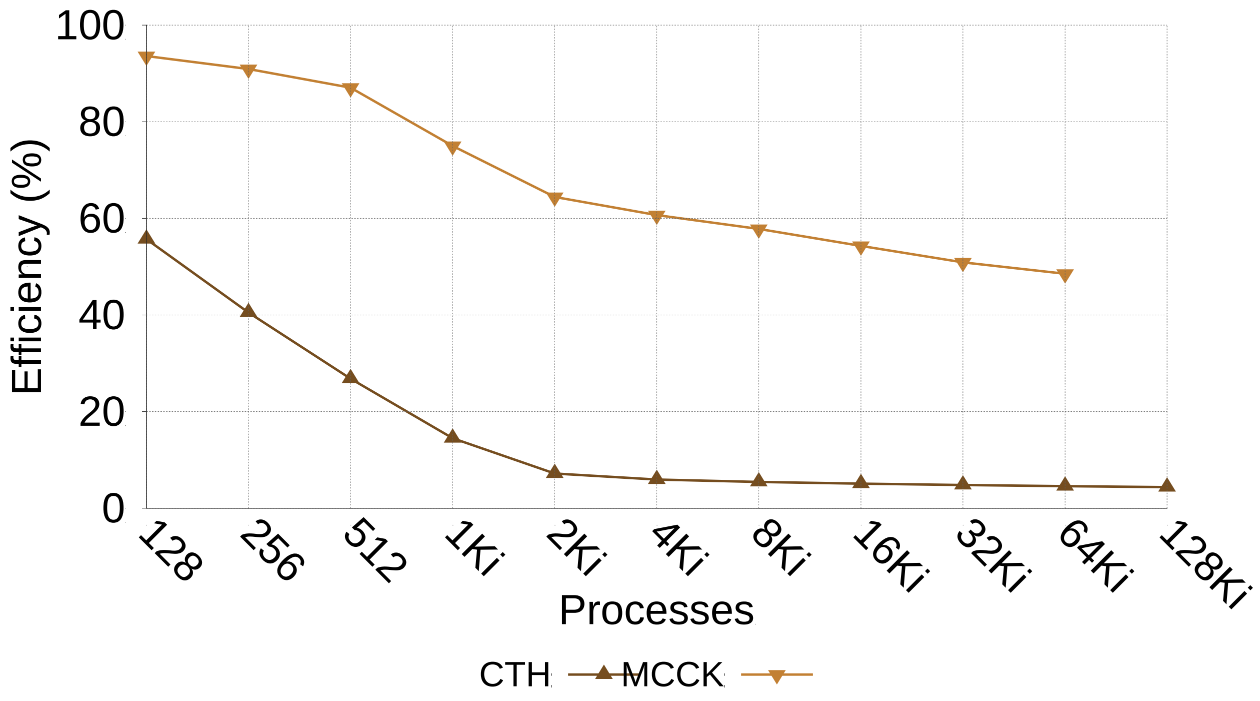


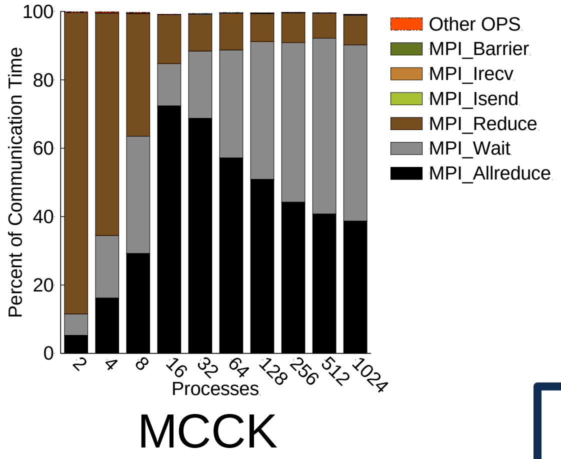
MCCK

Collective of choice is MPI_Allreduce()

c) Time in MPI_Allreduce()?



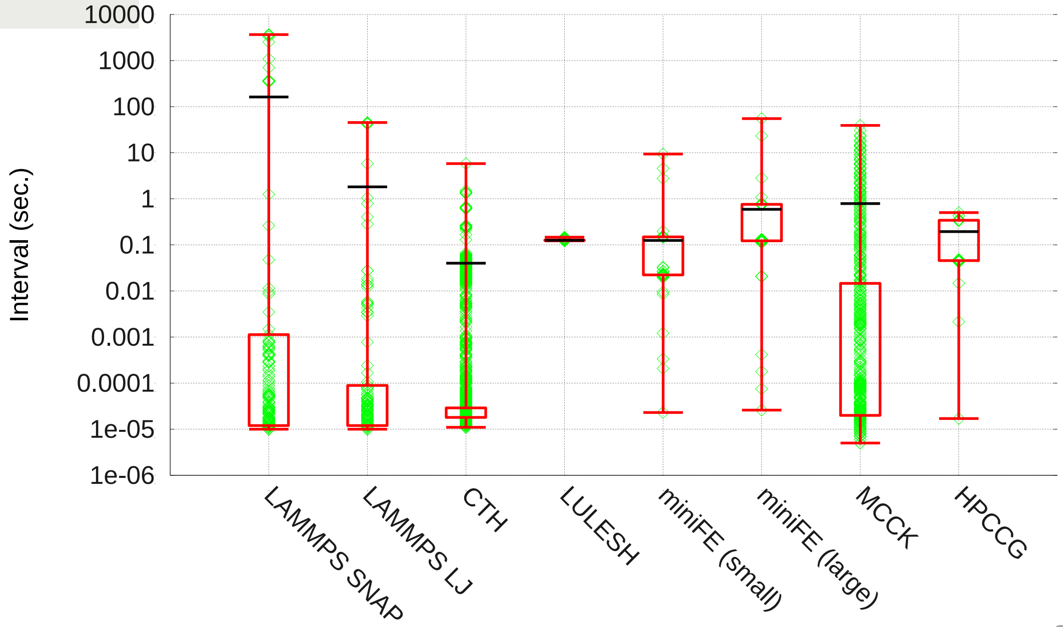




Nope, does not appear to be correlated to time in MPI_Allreduce()

d) Inter-arrival of MPI_Allreduce() the Culprit!

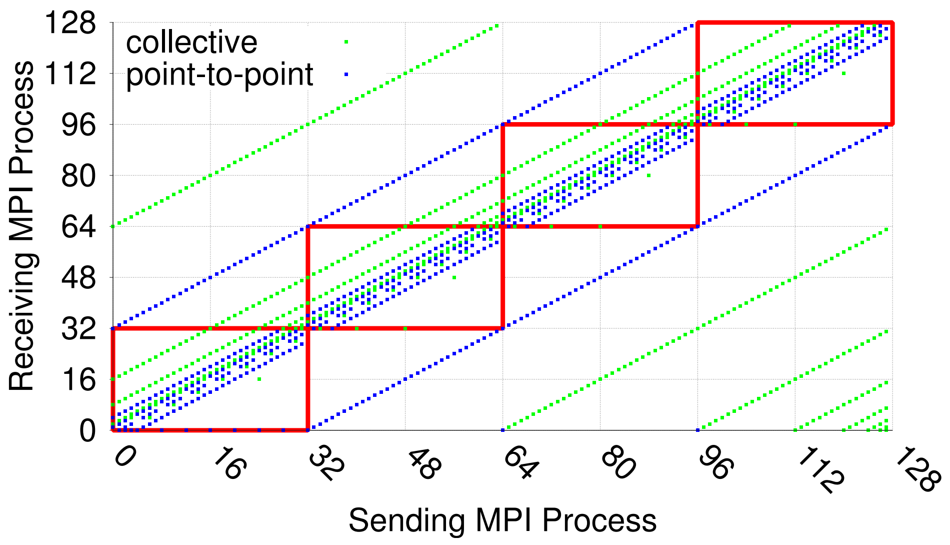
App	Interarrival Avg	Efficiency
LAMMPS	1.8 seconds	70%
MCCK	0.79 seconds	50%
miniFE	0.59 seconds	30%
LULESH	0.13 seconds	10%
HPCCG	0.04 seconds	8%
CTH	0.03 seconds	5%

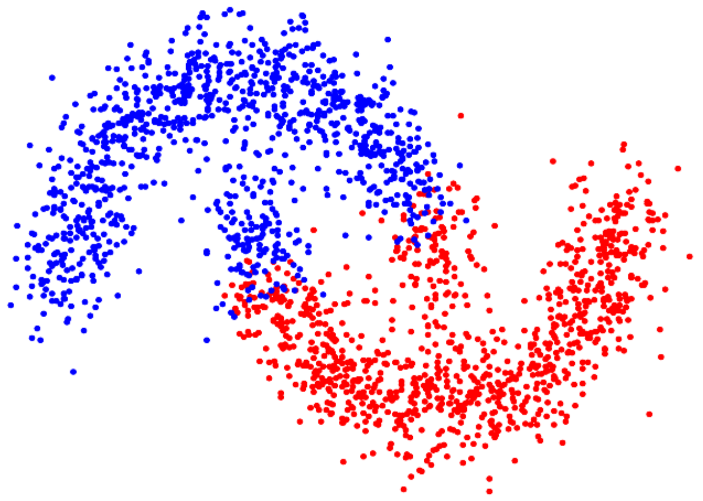


[SC14]: Ferreira, Widener, Levy, Arnold, TH: Understanding the Effects of Communication and Coordination on Checkpointing at Scale

Q.IV: How can we improve uCR performance?

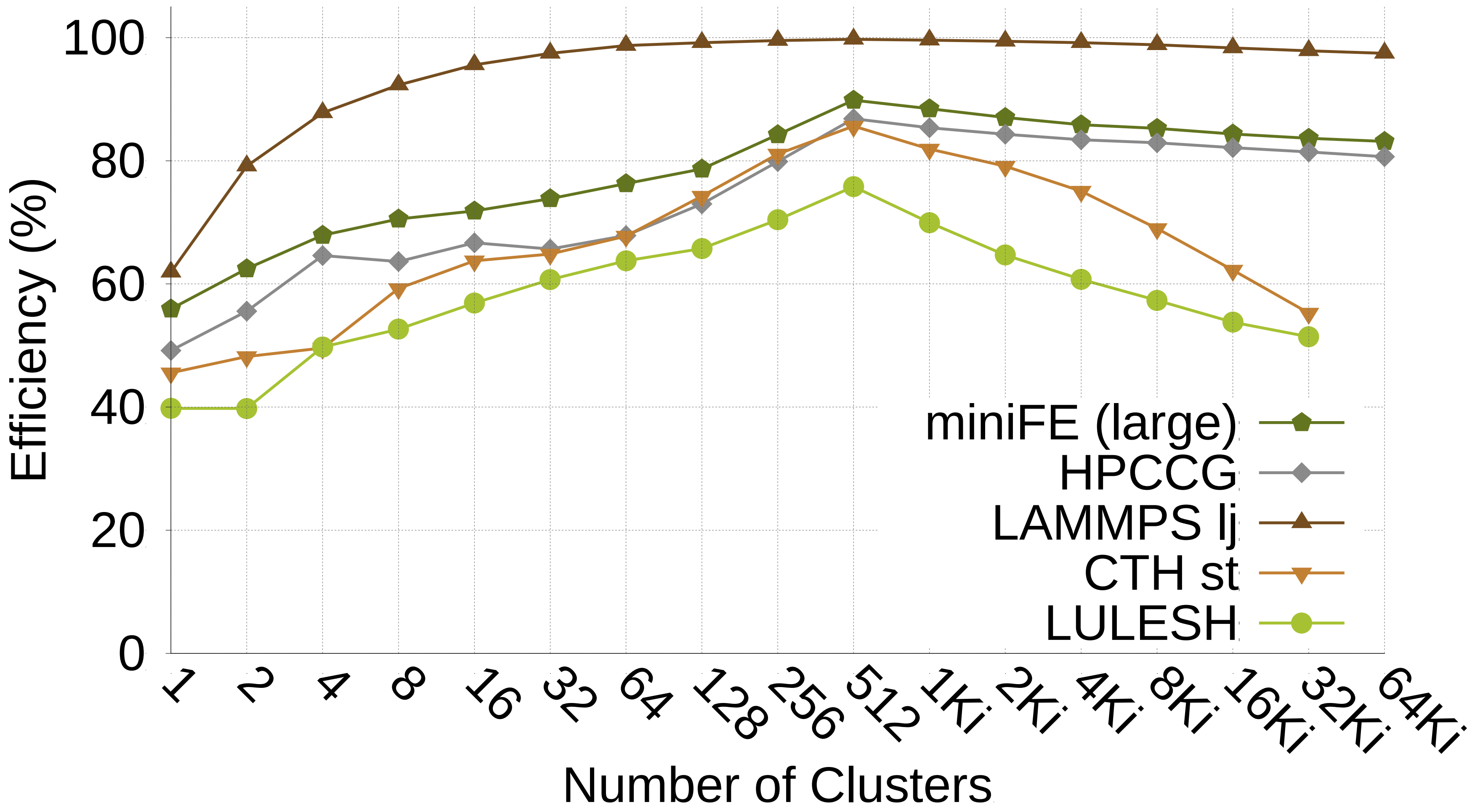
- **cCR within a cluster**
  - Hierarchical (clustered) checkpointing approaches
- **uCR across clusters**
- **Only messages crossing clusters are logged**





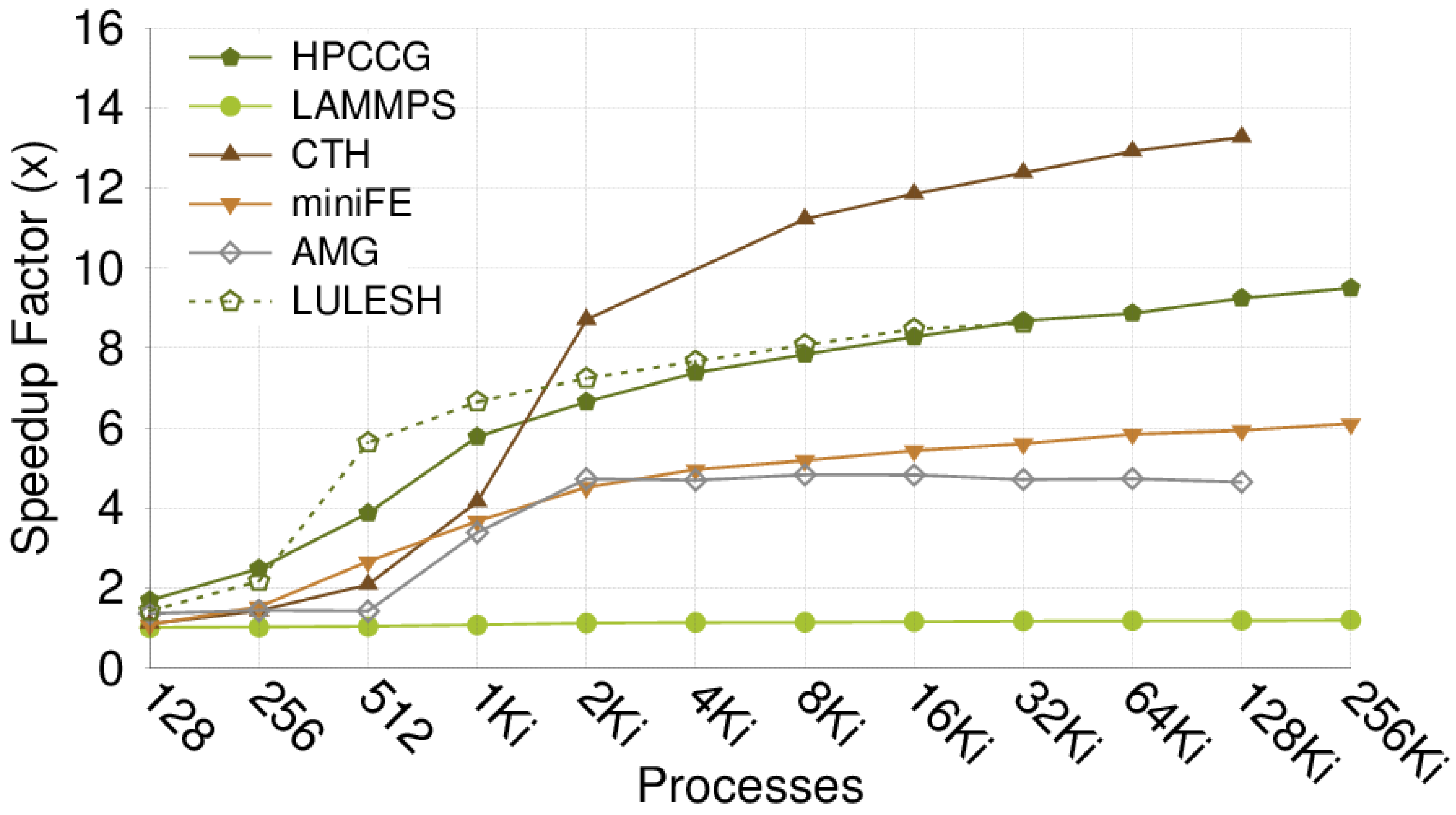
In addition to reducing message log volumes can this technique improve performance?

A.IV: Clustering Improves uCR Performance



Clustering improves performance because it reduces potential of overlapping noise events

A.IV: Nonblocking Collectives to the Rescue?



Nonblocking collectives can improve the runtime substantially

[EuroMPI'14]: Widener, Ferreira, Levy, TH: Exploring the effect of noise on the performance benefit of nonblocking allreduce

What does this all mean?

- **What if …**
  - I do not use collectives?

    Point-to-point operations can also create dependencies which may lead to significant (30%) slowdowns with uCR [SC14]

  - I use non-blocking collectives?

    If your code is capable of enough overlap, uCR may work well [EuroMPI'14]

  - I have an over-decomposed, many-task model?

    Similar to non-blocking collectives, uCR may work well but your runtime may need to consider the dependencies created in communication.

[SC14]: Ferreira, Widener, Levy, Arnold, TH: Understanding the Effects of Communication and Coordination on Checkpointing at Scale
[EuroMPI'14]: Widener, Ferreira, Levy, TH: Exploring the effect of noise on the performance benefit of nonblocking allreduce

Key Take Home Messages

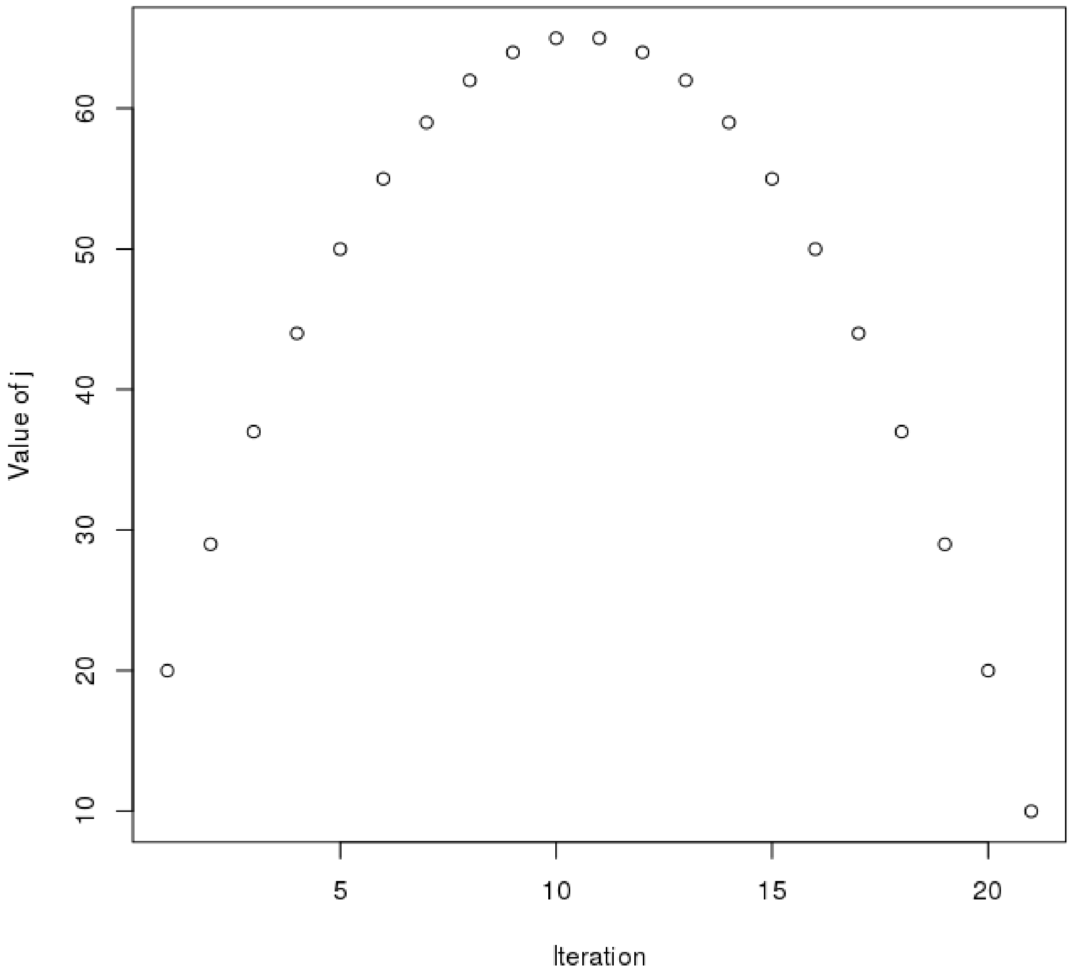
- At current and future stable storage bandwidths, the cost of local checkpoints for uCR can have a greater impact than message logging overheads
- This cost is dictated by happens-before chains created by an application's communication pattern
- uCR protocols based on process clustering can be used to tune an application's performance sensitivity to local checkpointing activities
- In uCR protocols, collective communication limits the progress which surviving processes make once a failure has occurred

Application Scalability: Counting Loop Iterations

- **When the polyhedral model cannot handle it**

```
j=10;
k=10;
while (j>0){
   j=j+k;
   k--;
}
```

?



Counting Arbitrary Affine Loop Nests

- Affine loops

```
$x$=$x_0$;                 // Initial assignment
while($c^T x < g$)         // Loop guard
    $x$=$Ax + b$;          // Loop update
```

- Perfectly nested affine loops

```
while($c_1^T x < g_1$) {
   $x = A_1 x + b_1$;
   while($c_2^T x < g_2$) {
      ...
       $x = A_{k-1} x + b_{k-1}$;
       while($c_k^T x < g_k$) {
           $x = A_k x + b_k$;
           while($c_{k+1}^T x < g_{k+1}$) {... }
           $x = U_k x + v_k$; }
       $x = U_{k-1} x + v_{k-1}$;
      ...}
   $x = U_1 x + v_1$;}
```

$$A_k, U_k \in \mathbb{R}^{m \times m},\ b_k, v_k, c_k \in \mathbb{R}^m,\ g_k \in \mathbb{R} \text{ and } k = 1 \dots r.$$

Counting Arbitrary Affine Loop Nests

- **Example**

```
for (j=1; j < n/p + 1; j= j*2)
        for (k=j; k < m; k = k + j )
                veryComplicatedOperation(j,k);
```

Counting Arbitrary Affine Loop Nests

- **Example**

```
for (j=1; j < n/p + 1; j= j*2)
    for (k=j; k < m; k = k + j )
        veryComplicatedOperation(j,k);
```

```
while($c_1^T x < g_1$) {
  $x = A_1 x + b_1$;
  while($c_2^T x < g_2$) {
    ...
     $x = A_{k-1} x + b_{k-1}$;
     while($c_k^T x < g_k$) {
        $x = A_k x + b_k$;
        while($c_{k+1}^T x < g_{k+1}$) {... }
        $x = U_k x + v_k$; }
     $x = U_{k-1} x + v_{k-1}$;
    ...}
  $x = U_1 x + v_1$;}
```

Counting Arbitrary Affine Loop Nests

- **Example**

```
for (j=1; j < n/p + 1; j= j*2)
        for (k=j; k < m; k = k + j )
                veryComplicatedOperation(j,k);
```

$$\begin{pmatrix} j \\ k \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} j \\ k \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix};$$

```
while($c_1^T x < g_1$) {
  $x = A_1 x + b_1$;
  while($c_2^T x < g_2$) {
    ...
     $x = A_{k-1} x + b_{k-1}$;
     while($c_k^T x < g_k$) {
        $x = A_k x + b_k$;
        while($c_{k+1}^T x < g_{k+1}$) {... }
        $x = U_k x + v_k$; }
     $x = U_{k-1} x + v_{k-1}$;
    ...}
  $x = U_1 x + v_1$;}
```

Counting Arbitrary Affine Loop Nests

- **Example**

```
for (j=1; j < n/p + 1; j= j*2)
    for (k=j; k < m; k = k + j )
        veryComplicatedOperation(j,k);
```

$$\begin{pmatrix} j \\ k \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} j \\ k \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix};$$

$$while(\begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} j \\ k \end{pmatrix} < {}^{n}\!/_{p} + 1)\{$$

$$\}$$

```
while($c_1^T x < g_1$) {
  $x = A_1 x + b_1$;
  while($c_2^T x < g_2$) {
    ...
     $x = A_{k-1} x + b_{k-1}$;
     while($c_k^T x < g_k$) {
        $x = A_k x + b_k$;
        while($c_{k+1}^T x < g_{k+1}$) {... }
        $x = U_k x + v_k$; }
     $x = U_{k-1} x + v_{k-1}$;
    ...}
  $x = U_1 x + v_1$;}
```

Counting Arbitrary Affine Loop Nests

- **Example**

```
for (j=1; j < n/p + 1; j= j*2)
        for (k=j; k < m; k = k + j )
                veryComplicatedOperation(j,k);
```

$$\begin{pmatrix} j \\ k \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} j \\ k \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix};$$

```
while($(1\;\;0)\begin{pmatrix} j \\ k \end{pmatrix} < n/p + 1$){
    $\begin{pmatrix} j \\ k \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} j \\ k \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix};$
    while($(0\;\;1)\begin{pmatrix} j \\ k \end{pmatrix} < m$){



    }
}
```

```
while($c_1^T x < g_1$) {
  $x = A_1 x + b_1;$
  while($c_2^T x < g_2$) {
    ...
     $x = A_{k-1} x + b_{k-1};$
     while($c_k^T x < g_k$) {
        $x = A_k x + b_k;$
        while($c_{k+1}^T x < g_{k+1}$) {... }
        $x = U_k x + v_k;$ }
     $x = U_{k-1} x + v_{k-1};$
    ...}
  $x = U_1 x + v_1;$}
```

Counting Arbitrary Affine Loop Nests

- **Example**

```
for (j=1; j < n/p + 1; j= j*2)
     for (k=j; k < m; k = k + j )
          veryComplicatedOperation(j,k);
```

```
while($c_1^T x < g_1$) {
  $x = A_1 x + b_1$;
  while($c_2^T x < g_2$) {
    ...
     $x = A_{k-1} x + b_{k-1}$;
     while($c_k^T x < g_k$) {
        $x = A_k x + b_k$;
        while($c_{k+1}^T x < g_{k+1}$) {... }
        $x = U_k x + v_k$; }
     $x = U_{k-1} x + v_{k-1}$;
    ...}
  $x = U_1 x + v_1$;}
```

$$\begin{pmatrix} j \\ k \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} j \\ k \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix};$$

$$while(\begin{pmatrix} 1 & 0 \end{pmatrix}\begin{pmatrix} j \\ k \end{pmatrix} < {}^{n}\!/_{p} + 1)\{$$

$$\begin{pmatrix} j \\ k \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} j \\ k \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix};$$

$$while(\begin{pmatrix} 0 & 1 \end{pmatrix}\begin{pmatrix} j \\ k \end{pmatrix} < m)\{$$

$$\begin{pmatrix} j \\ k \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} j \\ k \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix};$$

$$\}\begin{pmatrix} j \\ k \end{pmatrix} = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} j \\ k \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix};$$

$$\}$$

Counting Arbitrary Affine Loop Nests

- **Example**

```
for (j=1; j < n/p + 1; j= j*2)
     for (k=j; k < m; k = k + j )
          veryComplicatedOperation(j,k);
```

```
while($c_1^T x < g_1$) {
  $x = A_1 x + b_1$;
  while($c_2^T x < g_2$) {
    ...
     $x = A_{k-1} x + b_{k-1}$;
     while($c_k^T x < g_k$) {
        $x = A_k x + b_k$;
        while($c_{k+1}^T x < g_{k+1}$) {... }
        $x = U_k x + v_k$; }
     $x = U_{k-1} x + v_{k-1}$;
    ...}
  $x = U_1 x + v_1$;}
```

```
$x = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} x + \begin{pmatrix} 1 \\ 0 \end{pmatrix};$
$while(\begin{pmatrix} 1 & 0 \end{pmatrix} x < n/p + 1)\{$
    $x = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} x + \begin{pmatrix} 0 \\ 0 \end{pmatrix};$
    $while(\begin{pmatrix} 0 & 1 \end{pmatrix} x < m)\{$
        $x = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} x + \begin{pmatrix} 0 \\ 0 \end{pmatrix}$
    $\} x = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} x + \begin{pmatrix} 0 \\ 0 \end{pmatrix};$
$\}$
```

where $x = \begin{pmatrix} j \\ k \end{pmatrix}$

Current Workflow

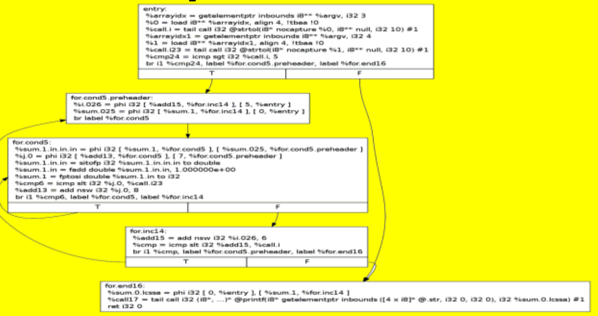
Parallel program

```
    do i = 1, procCols
       call mpi_irecv( buff, 2, dp_type, reduce_exch_proc(i),
>                      i, mpi_comm_world, request, ierr )
       call mpi_send( buff2, 2, dp_type, reduce_exch_proc(i),
>                      i, mpi_comm_world, ierr )
       call mpi_wait( request, status, ierr )
    enddo

    do i = id *n/p, ( id +1)* n/p
       do j = 1, nSize
            call compute
```

LLVM

Loop extraction



```
entry:
 %arrayidx = getelementptr inbounds i8** %argv, i32 3
 %0 = load i8** %arrayidx, align 4, !tbaa !0
 %call.i = tail call i32 @strtol(i8* nocapture %0, i8** null, i32 10) #1
 %arrayidx1 = getelementptr inbounds i8** %argv, i32 4
 %1 = load i8** %arrayidx1, align 4, !tbaa !0
 %call.i23 = tail call i32 @strtol(i8* nocapture %1, i8** null, i32 10) #1
 %cmp24 = icmp sgt i32 %call.i, 5
 br i1 %cmp24, label %for.cond5.preheader, label %for.end16
T | F

for.cond5.preheader:
 %i.026 = phi i32 [ %add15, %for.inc14 ], [ 5, %entry ]
 %sum.025 = phi i32 [ %sum.1, %for.inc14 ], [ 0, %entry ]
 br label %for.cond5

for.cond5:
 %sum.1.in.in.in = phi i32 [ %sum.1, %for.cond5 ], [ %sum.025, %for.cond5.preheader ]
 %j.0 = phi i32 [ %add13, %for.cond5 ], [ 7, %for.cond5.preheader ]
 %sum.1.in.in = sitofp i32 %sum.1.in.in.in to double
 %sum.1.in = fadd double %sum.1.in.in, 1.000000e+00
 %sum.1 = fptosi double %sum.1.in to i32
 %cmp6 = icmp slt i32 %j.0, %call.i23
 %add13 = add nsw i32 %j.0, 8
 br i1 %cmp6, label %for.cond5, label %for.inc14
T | F

for.inc14:
 %add15 = add nsw i32 %i.026, 6
 %cmp = icmp slt i32 %add15, %call.i
 br i1 %cmp, label %for.cond5.preheader, label %for.end16
T | F

for.end16:
 %sum.0.lcssa = phi i32 [ 0, %entry ], [ %sum.1, %for.inc14 ]
 %call17 = tail call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]* @.str, i32 0, i32 0), i32 %sum.0.lcssa) #1
 ret i32 0
```

Affine loop synthesis

```
while($c_1^T x < g_1$) {
  $x = A_1 x + b_1$;
  while($c_2^T x < g_2$) {
    ...
     $x = A_{k-1} x + b_{k-1}$;
     while($c_k^T x < g_k$) {
        $x = A_k x + b_k$;
        while($c_{k+1}^T x < g_{k+1}$) {... }
        $x = U_k x + v_k$; }
     $x = U_{k-1} x + v_{k-1}$;
    ...}
  $x = U_1 x + v_1$;}
```

Closed form representation

$$x(i_1,\dots,i_r) = A_{final}(i_1,\dots,i_r)\cdot x_0 + b_{final}(i_1,\dots,i_r)$$

with

$$i_r = 0 \dots n_k(x_{0,k}),\ k = 1 \dots r$$

Number of iterations

$$N = \sum_{i_1=0}^{n_1(x_{0,1})} \sum_{i_2=0}^{n_2(x_{0,2})} \dots \sum_{i_{r-1}=0}^{n_{r-1}(x_{0,r-1})} n_r(x_{0,r}).$$

Program analysis

$$W = N\Big|_{p=1}$$

$$D = N\Big|_{p\to\infty}$$

Static Loop Counting: Case studies

CG – conjugate gradient

$$N \approx k_1 \left\lceil \frac{m}{p} \right\rceil + k_2 \sqrt{\left\lceil \frac{m}{p} \right\rceil} + k_3 \log_2\left(\sqrt{p}\right)$$

$$D = T_\infty = \infty$$

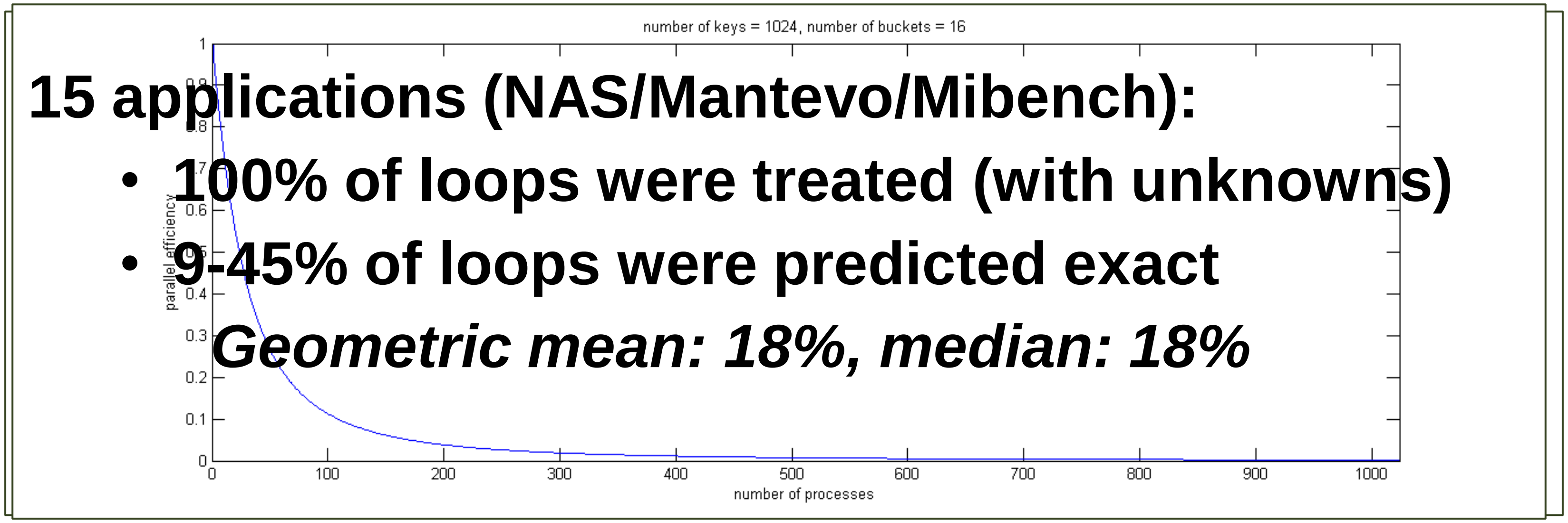
$$N \approx n\left(3(b+t) + 2\left\lceil \frac{m}{p} \right\rceil + p + u_1 + u_2\right)$$

IS – integer sort

$$D = T_\infty = \infty$$

$$E_p = \frac{k_4}{p\left(k_1 \left\lceil \frac{m}{p} \right\rceil + k_2 \sqrt{\left\lceil \frac{m}{p} \right\rceil} + k_3 \log_2\left(\sqrt{p}\right)\right)}$$

$$E_p = \frac{k_1}{p\left(p + k_2 \sqrt{\left\lceil \frac{m}{p} \right\rceil} + k_3\right)}$$



15 applications (NAS/Mantevo/Mibench):

- **100% of loops were treated (with unknowns)**
- **9-45% of loops were predicted exact**

Geometric mean: 18%, median: 18%

When Static doesn't work – PMNF!

$$f(p) = \sum_{k=1}^{n} c_k \cdot p^{i_k} \cdot \log_2^{j_k}(p)$$

$n \in \mathbb{N}$
$i_k \in I$
$j_k \in J$
$I, J \subset \mathbb{Q}$

$n = 1$
$I = \{0, 1, 2\}$
$J = \{0, 1\}$

c_1	$c_1 \cdot \log(p)$
$c_1 \cdot p$	$c_1 \cdot p \cdot \log(p)$
$c_1 \cdot p^2$	$c_1 \cdot p^2 \cdot \log(p)$

Application Scalability – PMNF!

$$f(p) = \sum_{k=1}^{n} c_k \cdot p^{i_k} \cdot \log_2^{j_k}(p)$$

$n \in \mathbb{N}$

$i_k \in I$

$n = 2$

$I = \{0, 1, 2\}$

$J = \{0, 1\}$

$c_1 + c_2 \cdot p$

$c_1 + c_2 \cdot p^2$

$c_1 + c_2 \cdot \log(p)$

$c_1 + c_2 \cdot p \cdot \log(p)$

$c_1 + c_2 \cdot p^2 \cdot \log(p)$

$c_1 \cdot \log(p) + c_2 \cdot p$

$c_1 \cdot \log(p) + c_2 \cdot p \cdot \log(p)$

$c_1 \cdot \log(p) + c_2 \cdot p^2$

$c_1 \cdot \log(p) + c_2 \cdot p^2 \cdot \log(p)$

$c_1 \cdot p + c_2 \cdot p \cdot \log(p)$

$c_1 \cdot p + c_2 \cdot p^2$

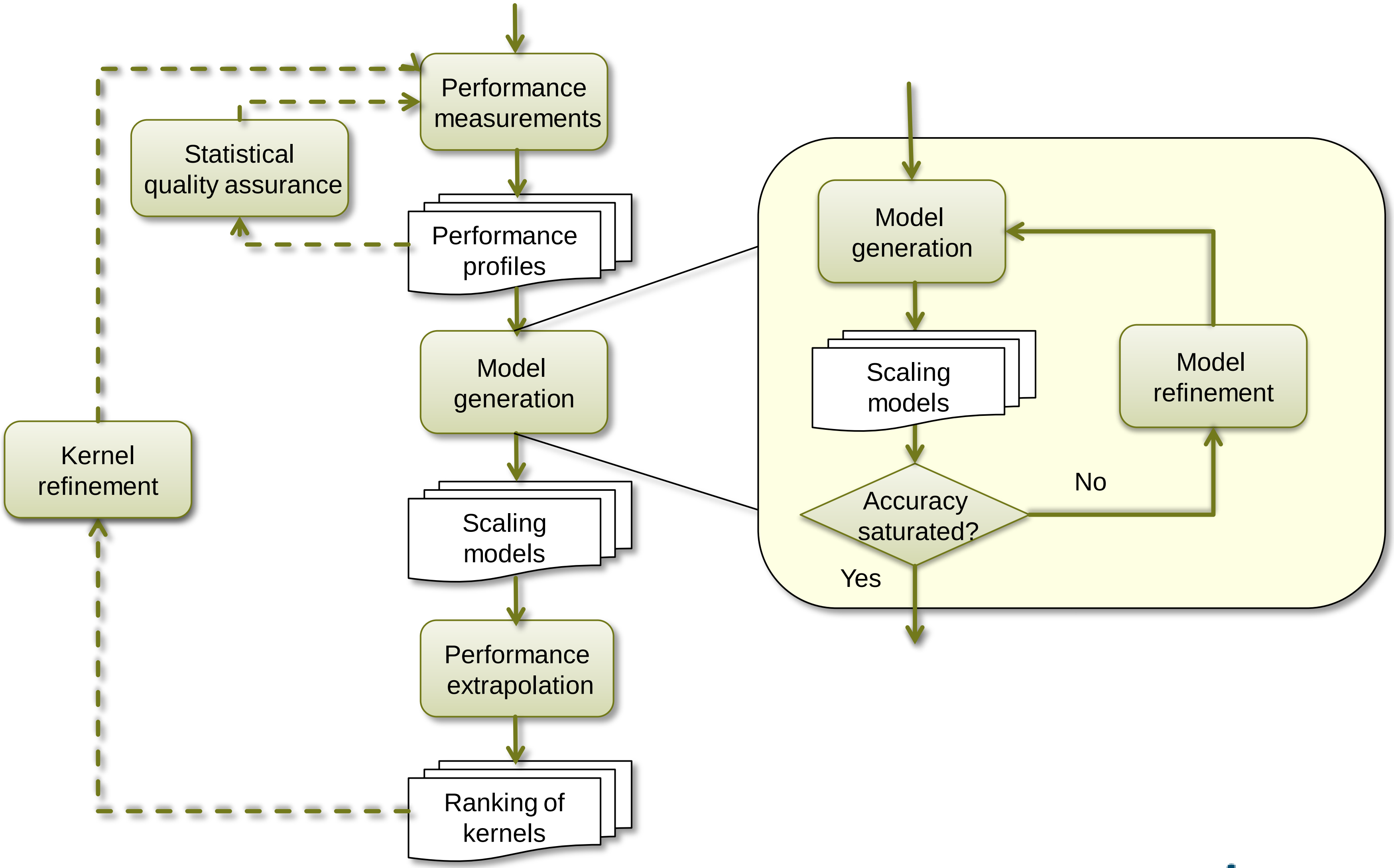
$c_1 \cdot p + c_2 \cdot p^2 \cdot \log(p)$

$c_1 \cdot p \cdot \log(p) + c_2 \cdot p^2$

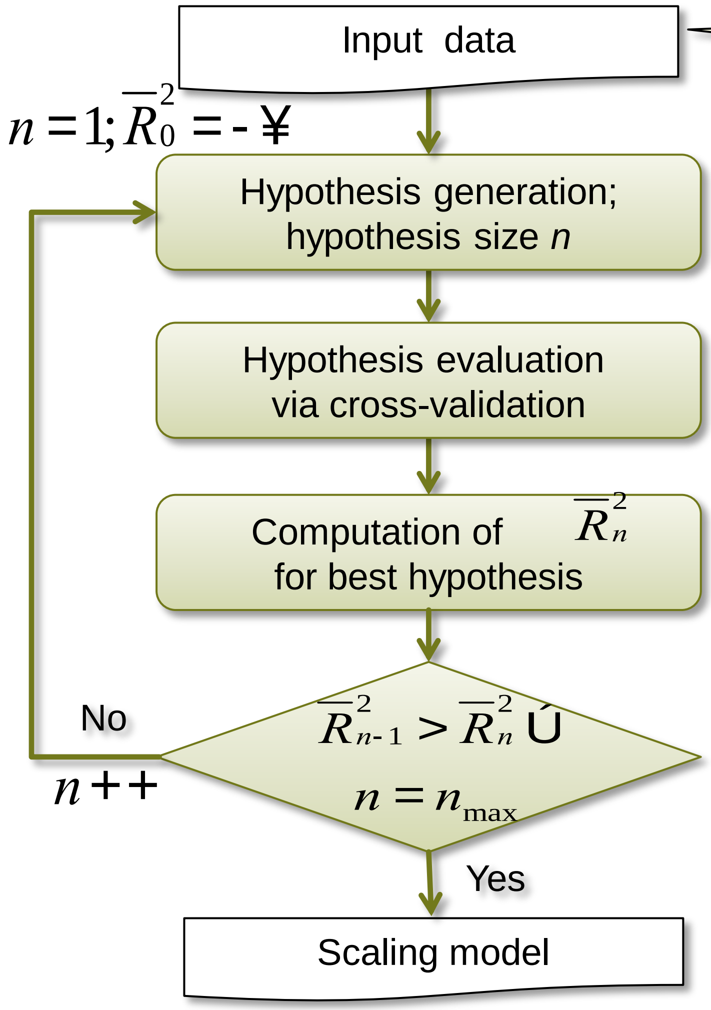
$c_1 \cdot p \cdot \log(p) + c_2 \cdot p^2 \cdot \log(p)$

$c_1 \cdot p^2 + c_2 \cdot p^2 \cdot \log(p)$

Our automated generation workflow



Model refinement


Input data
$n = 1; \bar{R}_0^2 = -\infty$
Hypothesis generation; hypothesis size *n*
Hypothesis evaluation via cross-validation
Computation of $\bar{R}_n^2$ for best hypothesis
$\bar{R}_{n-1}^2 > \bar{R}_n^2 \vee n = n_{\max}$
No
$n{+}{+}$
Yes
Scaling model

$$\{(p_1, t_1), \ldots, (p_6, t_6)\}$$

c_1	$c_1 \cdot \log(p)$
$c_1 \cdot p$	$c_1 \cdot p \cdot \log(p)$
$c_1 \cdot p^2$	$c_1 \cdot p^2 \cdot \log(p)$

$$c_1 \cdot \log(p)$$

$$R^2 = 1 - \frac{residualSumSquares}{totalSumSquares}$$

$$\bar{R}^2 = 1 - (1 - R^2) \cdot \frac{6 - 1}{6 - n - 2}$$

$$I = \{0, 1, 2\}; J = \{0, 1\}; n_{\max} = 2$$

scalasca