Resilience Overheads at Scale and Scalability
Torsten Hoefler, ETH Zurich
with the New Mexico folks (K. Ferreira, P. Widener, S. Levy, D. Arnold)
Fault-tolerance Interfaces

- **Very simple**
  - Coordinated Checkpointing
    ```c
    void take_coordinated_checkpoint(void *data, int size, char* output)
    ```
  - Uncoordinated Checkpointing
    ```c
    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

Each node is getting more complicated

Systems are getting larger
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

PROCESS 1

PROCESS 2

PROCESS 3

PROCESS 4
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 Applications
[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

[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

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

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

[PMBS’13]: Widener, Ferreira, Levy, Hoefler: Exploring the effect of noise on the performance benefit of nonblocking allreduce
Validation: small-scale testing

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

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

A1: Slowdowns can be significant and increase with scale

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

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

---

![Image of a snail](image-url)
a) Time Spent Communicating?

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

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

Collective of choice is MPI_Allreduce()
c) Time in MPI_Allreduce()?

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

[SC14]: Ferreira, Widener, Levy, Arnold, TH: Understanding the Effects of Communication and Coordination on Checkpointing at Scale
d) Inter-arrival of MPI\_Allreduce() the Culprit!

<table>
<thead>
<tr>
<th>App</th>
<th>Interarrival Avg</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAMMPS</td>
<td>1.8 seconds</td>
<td>70%</td>
</tr>
<tr>
<td>MCCK</td>
<td>0.79 seconds</td>
<td>50%</td>
</tr>
<tr>
<td>miniFE</td>
<td>0.59 seconds</td>
<td>30%</td>
</tr>
<tr>
<td>LULESH</td>
<td>0.13 seconds</td>
<td>10%</td>
</tr>
<tr>
<td>HPCCG</td>
<td>0.04 seconds</td>
<td>8%</td>
</tr>
<tr>
<td>CTH</td>
<td>0.03 seconds</td>
<td>5%</td>
</tr>
</tbody>
</table>

Interval (sec.)

---

[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?

[SC14]: Ferreira, Widener, Levy, Arnold, TH: Understanding the Effects of Communication and Coordination on Checkpointing at Scale
A.IV: Clustering Improves uCR Performance

Clustering improves performance because it reduces potential of overlapping noise events

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

```c
j=10;
k=10;
while (j>0){
    j=j+k;
    k--;
}
```
Counting Arbitrary Affine Loop Nests

- **Affine loops**

  \[
  x = x_0; \quad \text{// Initial assignment}
  \]

  \[
  \text{while}(c^T x < g) \quad \text{// Loop guard}
  \]

  \[
  x = Ax + b; \quad \text{// Loop update}
  \]

- **Perfectly nested affine loops**

  \[
  \text{while}(c_1^T x < g_1) \{
  \]

  \[
  x = A_1 x + b_1;
  \]

  \[
  \text{while}(c_2^T x < g_2) \{
  \]

  \[
  \quad \ldots
  \]

  \[
  x = A_{k-1} x + b_{k-1};
  \]

  \[
  \text{while}(c_k^T x < g_k) \{
  \]

  \[
  x = A_k x + b_k;
  \]

  \[
  \quad \text{while}(c_{k+1}^T x < g_{k+1}) \{ \ldots \}
  \]

  \[
  x = U_k x + v_k; \}
  \]

  \[
  x = U_{k-1} x + v_{k-1};
  \]

  \[
  \ldots \}
  \]

  \[
  x = U_1 x + v_1; \}
  \]

  \[
  A_k, U_k \in \mathbb{R}^{m \times m}, \ b_k, v_k, c_k \in \mathbb{R}^m, \ g_k \in \mathbb{R} \ \text{and} \ k = 1 \ldots r.
  \]
Counting Arbitrary Affine Loop Nests

- Example

```c
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

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

```plaintext
while(c_1^T \times < 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;
}
```

T. Hoefler, G. Kwasniewski: Automatic Complexity Analysis of Explicitly Parallel Programs, SPAA’14
Counting Arbitrary Affine Loop Nests

- Example

```cpp
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};
```

```cpp
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;
}
```

T. Hoefler, G. Kwasniewski: Automatic Complexity Analysis of Explicitly Parallel Programs, SPAA’14
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;
  }
  ```

T. Hoefler, G. Kwasniewski: Automatic Complexity Analysis of Explicitly Parallel Programs, SPAA’14
Counting Arbitrary Affine Loop Nests

- Example

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

while(c1^T x < g1) {
  x = A1x + b1;
  while(c2^T x < g2) {
    ... 
    x = A_k-1x + b_k-1;
    while(c_k^T x < g_k) {
      x = A_kx + 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_1x + v_1;
}
```

T. Hoefler, G. Kwasniewski: Automatic Complexity Analysis of Explicitly Parallel Programs, SPAA’14
Counting Arbitrary Affine Loop Nests

Example

\[
\begin{align*}
\text{for } (j=1; j < n/p + 1; j= j*2) \quad &
\text{for } (k=j; k < m; k = k + j ) \\
\text{veryComplicatedOperation}(j,k); \\
\end{align*}
\]

\[
\begin{align*}
\begin{pmatrix} j \\ k \end{pmatrix} &= \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} j \\ k \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix}; \\
\end{align*}
\]

\[
\begin{align*}
\text{while}(1 \ 0) \begin{pmatrix} j \\ k \end{pmatrix} < n/p +1 \\ {\text{do}} \begin{align*}
\begin{pmatrix} j \\ k \end{pmatrix} &= \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} j \\ k \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix}; \\
\end{align*} \\
\text{while}(0 & 1) \begin{pmatrix} j \\ k \end{pmatrix} < m \\
\begin{pmatrix} j \\ k \end{pmatrix} &= \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} j \\ k \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix}; \\
\end{align*}
\]

\[
\begin{align*}
\begin{pmatrix} j \\ k \end{pmatrix} &= \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} j \\ k \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix}; \\
\end{align*}
\]

\[
\begin{align*}
\text{while}(1 \ 0) \begin{pmatrix} j \\ k \end{pmatrix} < n/p +1 {\text{do}} \begin{align*}
\begin{pmatrix} j \\ k \end{pmatrix} &= \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} j \\ k \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix}; \\
\end{align*} \}
\]

T. Hoefler, G. Kwasniewski: Automatic Complexity Analysis of Explicitly Parallel Programs, SPAA’14
Counting Arbitrary Affine Loop Nests

Example

\[
\begin{align*}
\text{for} \quad (j=1; \quad j < n/p + 1; \quad j= j*2) \\
\quad \text{for} \quad (k=j; \quad k < m; \quad k = k + j) \\
\quad \text{veryComplicatedOperation}(j, k);
\end{align*}
\]

\[
\begin{align*}
\text{while}(c_1^T x < g_1) \{ \\
x = A_1 x + b_1; \\
\text{while}(c_2^T x < g_2) \{ \\
\quad x = A_{k-1} x + b_{k-1}; \\
\quad \text{while}(c_{k}^T x < g_k) \{ \\
\quad \quad x = A_k x + b_k; \\
\quad \quad \text{while}(c_{k+1}^T x < g_{k+1}) \{ \\
\quad \quad \quad x = U_k x + v_k; \\
\quad \quad \} x = U_{k-1} x + v_{k-1}; \\
\quad \} x = U_1 x + v_1; \\
\} \\
\end{align*}
\]

where \( x = \begin{pmatrix} j \\ k \end{pmatrix} \)
Current Workflow

Parallel program

```c
for i = 0 to procSize
    call mpi_recv(buf, ..., dp_type, reduce_each_proc(i),
        i, mpi_comm_world, request, ierr)
enddo

for i = 0 to procSize
    call mpi_send(buf2, ..., dp_type, reduce_each_proc(i),
        i, mpi_comm_world, ierr)
enddo

wait request, status, ierr
```

Closed form representation

\[ x(i_1, \ldots, i_r) = A_{final}(i_1, \ldots, i_r) \cdot x_0 + b_{final}(i_1, \ldots, i_r) \]

with

\[ i_r = 0 \ldots n_k(x_{0,k}), k = 1 \ldots r \]

Affine loop synthesis

```c
while(c_1 x < g_1) {
    x = A_1 x + b_1;
    i_1 = i_1 + 1;
    while(c_2 x < g_2) {
        x = A_2 x + b_2;
        i_2 = i_2 + 1;
        while(c_3 x < g_3) {
            x = A_3 x + b_3;
            i_3 = i_3 + 1;
            ...}
        x = U_{i_2,1} x + v_{i_2};
        i_2 = i_2 + 1;
    x = U_{i_1,1} x + v_{i_1};
    i_1 = i_1 + 1;
}
```

Loop extraction

```
while(c_x < g) {
    x = A x + b;
    i = i + 1;
    while(c_{x+1} x < g_{x+1}) {
        x = U_{i,1} x + v_i;
        i = i + 1;
    }
```

Number of iterations

\[ N = \sum_{i_1=0}^{n_1(x_{0,1})} \sum_{i_2=0}^{n_2(x_{0,2})} \cdots \sum_{i_r=0}^{n_r(x_{0,r})} n_r(x_{0,r}) \]

Program analysis

\[ W = N \bigg|_{p=1} \]

\[ D = N \bigg|_{p\to\infty} \]
Static Loop Counting: Case studies

CG – conjugate gradient

\[ N \approx k_1 \left[ \frac{m}{p} \right] + k_2 \sqrt{\frac{m}{p}} + k_3 \log_2 \sqrt{p} \]

\[ D = T_\infty \approx n \left( 3 + t + \frac{m}{p} + p + u_1 + u_2 \right) \]

\[ E_p = \frac{D = T_\infty = \infty}{k_4} \]

\[ p \left[ k_1 \left[ \frac{m}{p} \right] + k_2 \sqrt{\frac{m}{p}} + k_3 \log_2 \sqrt{p} \right] \]

IS – integer sort

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!

\[
f(p) = \sum_{k=1}^{n} c_k \times p^{i_k} \times \log_{2}^{j_k}(p)
\]

\[
\begin{align*}
  n &= 1 \\
  I &= \{0, 1, 2\} \\
  J &= \{0, 1\}
\end{align*}
\]

[A. Calotoiu, T. Hoefler, M. Poke, F. Wolf: Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes]
Application Scalability – PMNF!

\[ f(p) = \sum_{k=1}^{n} c_k \times p^{i_k} \times \log_{2}^{j_k}(p) \]

\[ n, i_k, j_k \in \mathbb{N} \]

\[ c_1 + c_2 \times p \]
\[ c_1 + c_2 \times p^2 \]
\[ c_1 + c_2 \times \log(p) \]
\[ c_1 + c_2 \times p \times \log(p) \]
\[ c_1 + c_2 \times p^2 \times \log(p) \]

\( n = 2 \)
\( I = \{0, 1, 2\} \)
\( J = \{0, 1\} \)
Our automated generation workflow

- Performance measurements
- Performance profiles
- Model generation
- Scaling models
- Performance extrapolation
- Accuracy saturated?
- Ranking of kernels
- Statistical quality assurance
- Kernel refinement

A. Calotoiu, T. Hoefler, M. Poke, F. Wolf: Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes, SC13
Model refinement

1. **Input data**
2. **Hypothesis generation; hypothesis size \( n \)**
3. **Hypothesis evaluation via cross-validation**
4. **Computation of \( R^2 \) for best hypothesis**

- \( n = 1; R_0 = n \)
- \( c_1 \times \log(p) \)
- \( c_1 \times p \)
- \( c_1 \times p^2 \)

- \( R^2 = 1 - \frac{\text{residualSumSquares}}{\text{totalSumSquares}} \)

- \( R^2 = 1 - (1 - R^2) \times \frac{6}{6 - n - 2} \)

- \( I = \{0, 1, 2\}; J = \{0, 1\}; n_{\text{max}} = 2 \)

A. Calotoiu, T. Hoefler, M. Poke, F. Wolf: Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes, SC13