ETH zürich

spcl.inf.ethz.ch
@spcl_eth

TORSTEN HOEFLER

Automatic Performance Models for the Masses

Static and dynamic techniques for application performance modeling

in collaboration with Alexandru Calotoiu and Felix Wolf @ RWTH Aachen
with students Arnamoy Bhattacharyya and Grzegorz Kwasniewski @ SPCL
presented at Co-Design 2015, HPC China, Wuxi, China

PASC16 Platform for Advanced Scientific Computing Conference
Lausanne Switzerland | 08-10 June 2016

CLIMATE & WEATHER
SOLID EARTH
LIFE SCIENCE
CHEMISTRY & MATERIALS
PHYSICS
COMPUTER SCIENCE & MATHEMATICS
ENGINEERING
EMERGING DOMAINS

sighpc
acm

IBM

CRAY
ECOphlex

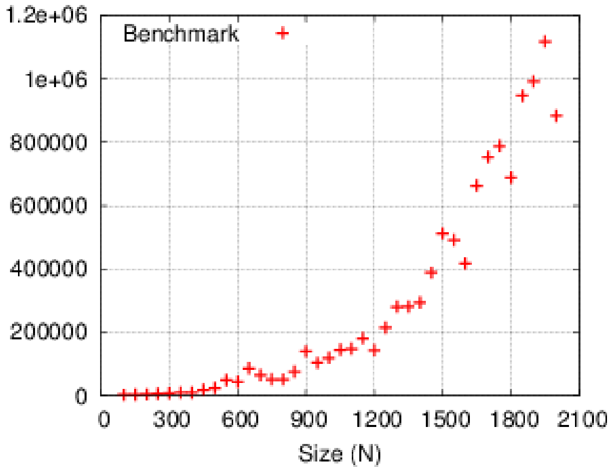
Use-cases for performance modeling

1. **Scalability bug prediction**
   - Find latent scalability bugs early on (before machine deployment)

     SC13: A. Calotoiu, TH, M. Poke, F. Wolf: Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes
2. **Automated performance (regression) testing**
   - Performance modeling as part of a software engineering discipline in HPC

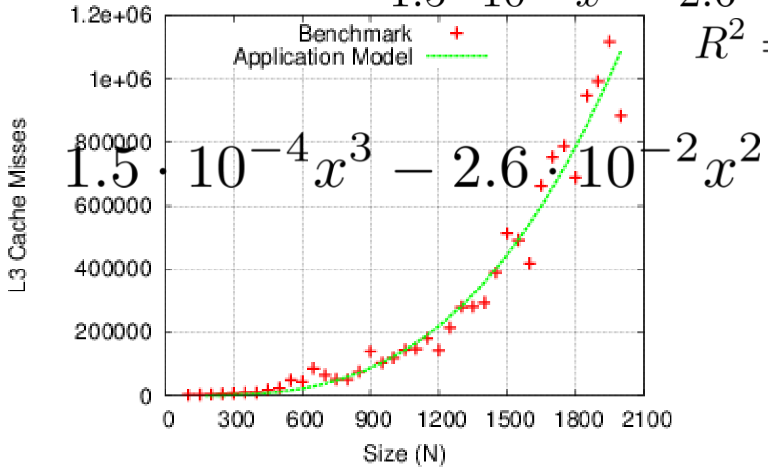
     ICS'15: S. Shudler, A. Calotoiu, T. Hoefler, A. Strube, F. Wolf: Exascaling Your Library: Will Your Implementation Meet Your Expectations?
3. **Guided or automated performance optimization**
   - E.g., near-optimal job scheduling

     Nan Ding, Wei Xue, et al. (forthcoming)
4. **Hardware/Software co-design (how to architect systems)**
   - Zhiwei Xu's "efficiency first" design



vs.

$$1.5 \cdot 10^{-4} x^3 - 2.6 \cdot 10^{-2} x^2$$
$$R^2 = 0.93$$



But how to measure and report performance?

- **We all think we know it but it's harder than I thought!**

  How many measurements?

  How to summarize data?

  How to summarize many processes?

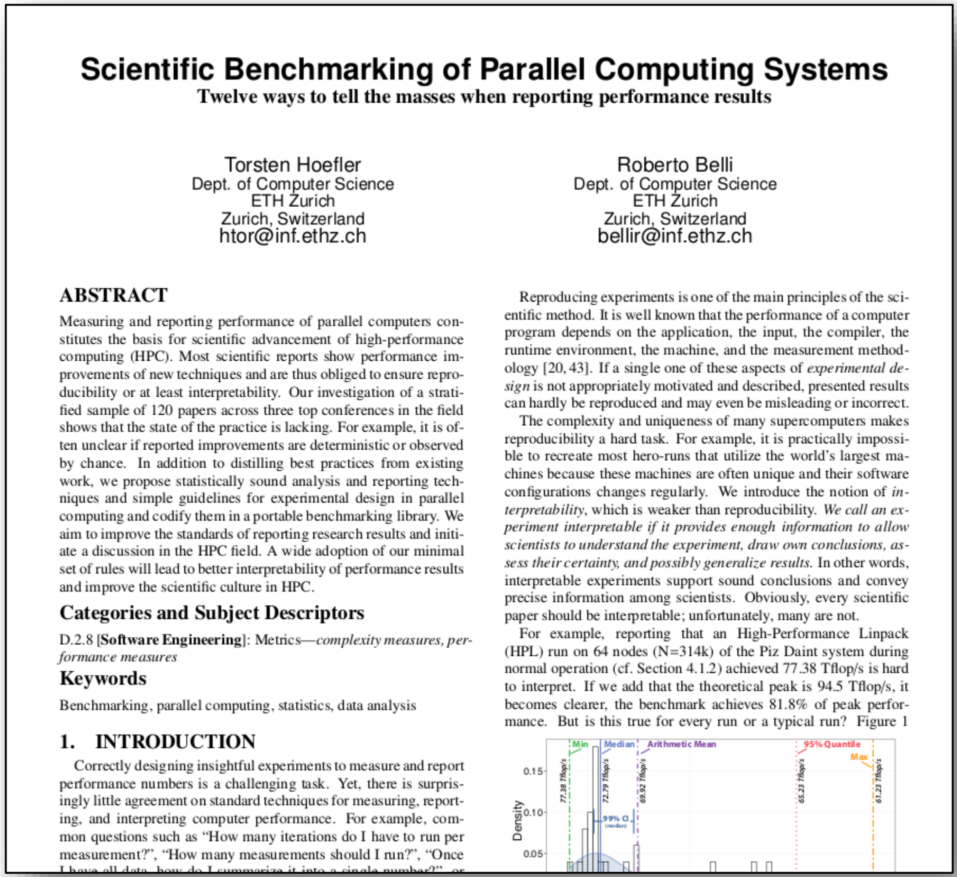
  …

- **Attempt to establish a rigorous practice**
  - Clarify common problems

    e.g., Which mean to use when, common statistics issues, …
  - Good start for students

    12 simple concise rules

- **My thesis: give up on (performance) reproducibility?**



Scientific Benchmarking of Parallel Computing Systems

Twelve ways to tell the masses when reporting performance results

Torsten Hoefler
Dept. of Computer Science
ETH Zurich
Zurich, Switzerland
htor@inf.ethz.ch

Roberto Belli
Dept. of Computer Science
ETH Zurich
Zurich, Switzerland
bellir@inf.ethz.ch

ABSTRACT

Measuring and reporting performance of parallel computers constitutes the basis for scientific advancement of high-performance computing (HPC). Most scientific reports show performance improvements of new techniques and are thus obliged to ensure reproducibility or at least interpretability. Our investigation of a stratified sample of 120 papers across three top conferences in the field shows that the state of the practice is lacking. For example, it is often unclear if reported improvements are deterministic or observed by chance. In addition to distilling best practices from existing work, we propose statistically sound analysis and reporting techniques and simple guidelines for experimental design in parallel computing and codify them in a portable benchmarking library. We aim to improve the standards of reporting research results and initiate a discussion in the HPC field. A wide adoption of our minimal set of rules will lead to better interpretability of performance results and improve the scientific culture in HPC.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

Keywords

Benchmarking, parallel computing, statistics, data analysis

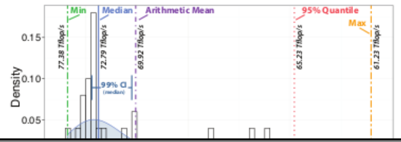
1. INTRODUCTION

Correctly designing insightful experiments to measure and report performance numbers is a challenging task. Yet, there is surprisingly little agreement on standard techniques for measuring, reporting, and interpreting computer performance. For example, common questions such as "How many iterations do I have to run per measurement?", "How many measurements should I run?", "Once I have all data, how do I summarize it into a single number?", or

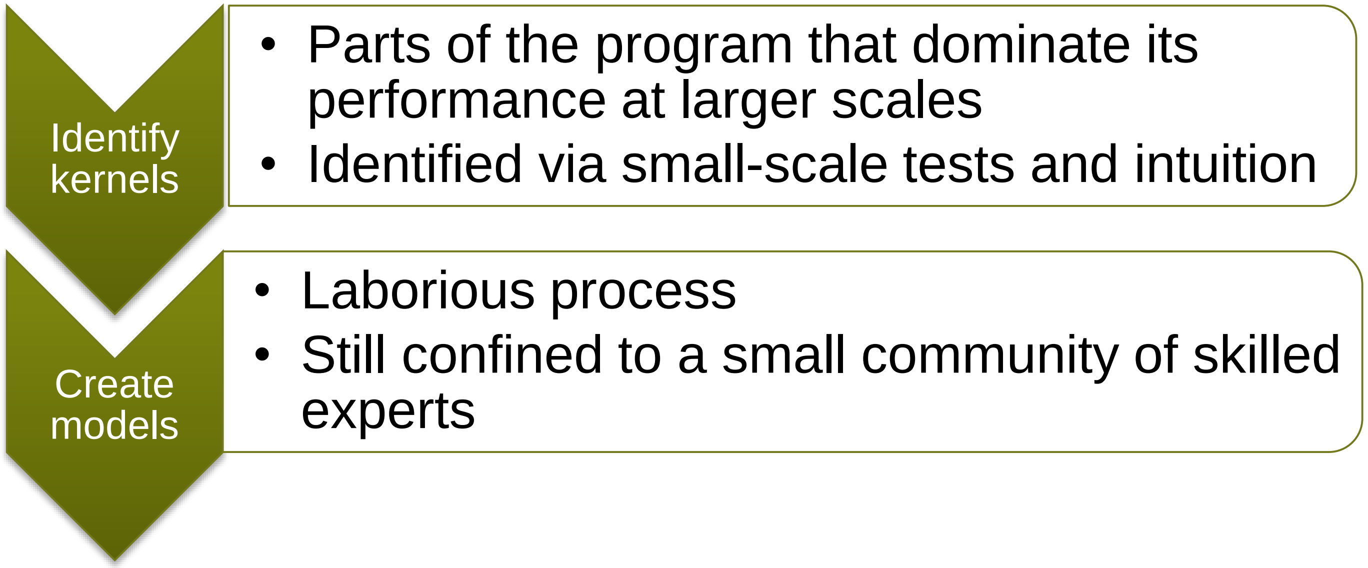
Reproducing experiments is one of the main principles of the scientific method. It is well known that the performance of a computer program depends on the application, the input, the compiler, the runtime environment, the machine, and the measurement methodology [20, 43]. If a single one of these aspects of *experimental design* is not appropriately motivated and described, presented results can hardly be reproduced and may even be misleading or incorrect.

The complexity and uniqueness of many supercomputers makes reproducibility a hard task. For example, it is practically impossible to recreate most hero-runs that utilize the world's largest machines because these machines are often unique and their software configurations changes regularly. We introduce the notion of *interpretability*, which is weaker than reproducibility. *We call an experiment interpretable if it provides enough information to allow scientists to understand the experiment, draw own conclusions, assess their certainty, and possibly generalize results.* In other words, interpretable experiments support sound conclusions and convey precise information among scientists. Obviously, every scientific paper should be interpretable; unfortunately, many are not.

For example, reporting that an High-Performance Linpack (HPL) run on 64 nodes (N=314k) of the Piz Daint system during normal operation (cf. Section 4.1.2) achieved 77.38 Tflop/s is hard to interpret. If we add that the theoretical peak is 94.5 Tflop/s, it becomes clearer, the benchmark achieves 81.8% of peak performance. But is this true for every run or a typical run? Figure 1



Manual analytical performance modeling

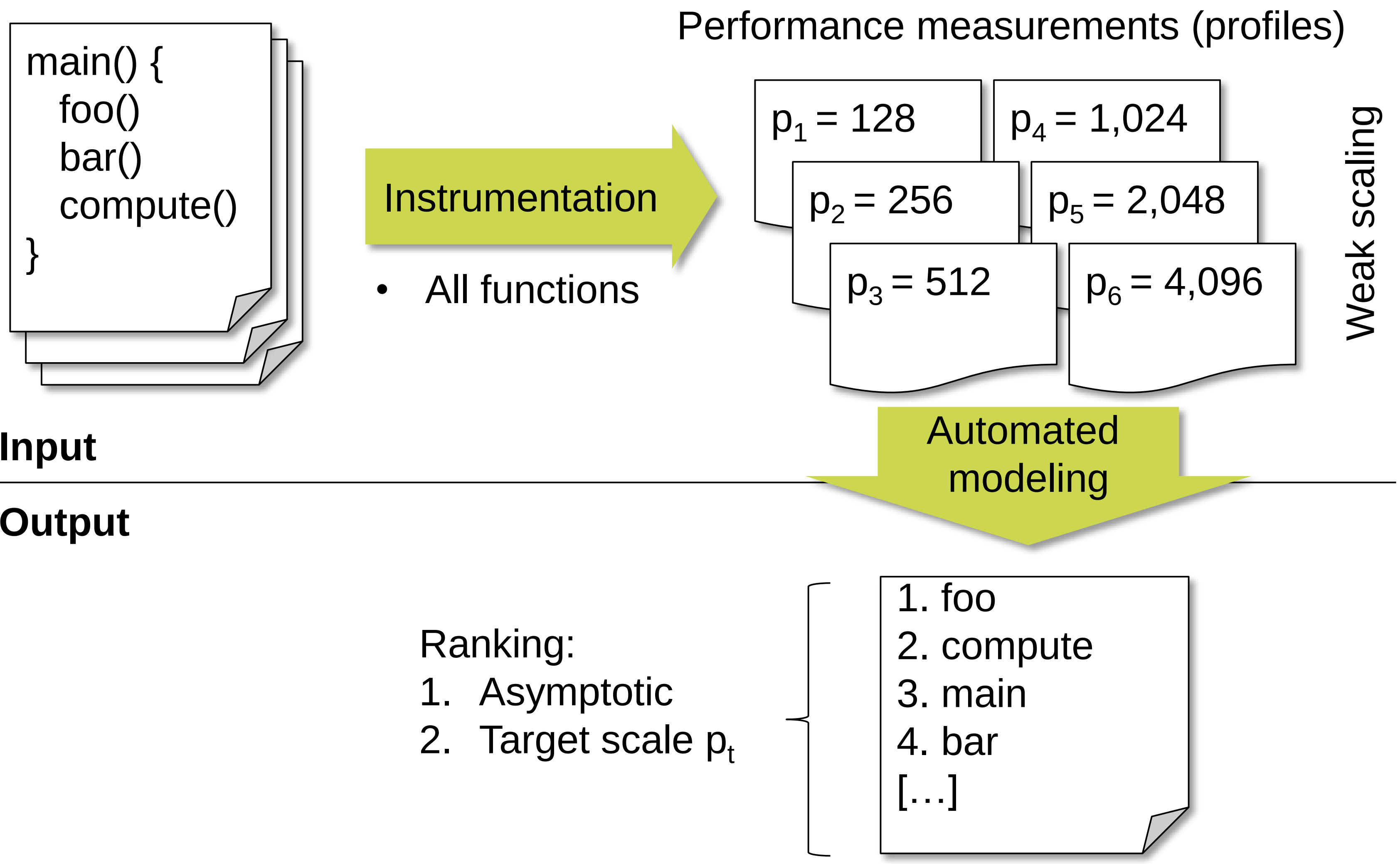


- Parts of the program that dominate its performance at larger scales
- Identified via small-scale tests and intuition

- Laborious process
- Still confined to a small community of skilled experts

- **Disadvantages**
  - Time consuming
  - Error-prone, may overlook unscalable code

Our first step: scalability bug detector

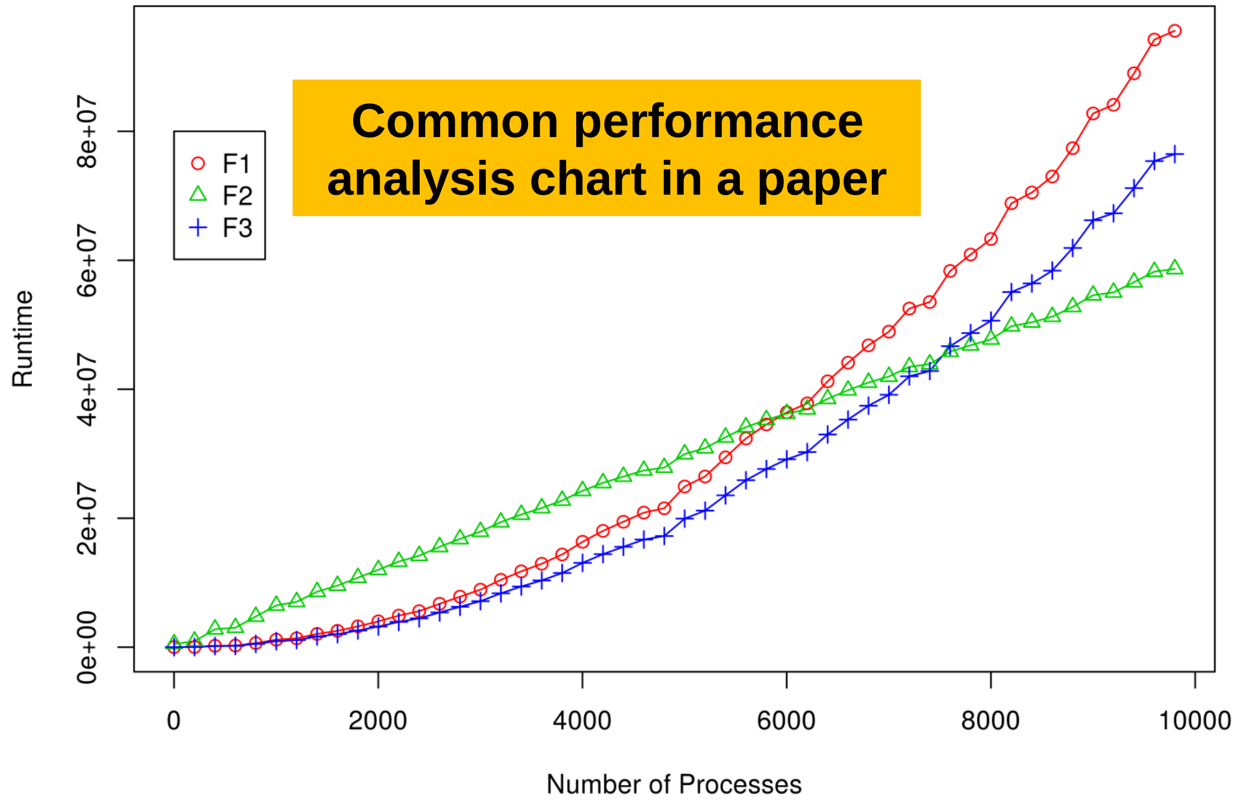


```
main() {
   foo()
   bar()
   compute()
}
```

Ranking:
1. Asymptotic
2. Target scale p_t

1. foo
2. compute
3. main
4. bar
[…]

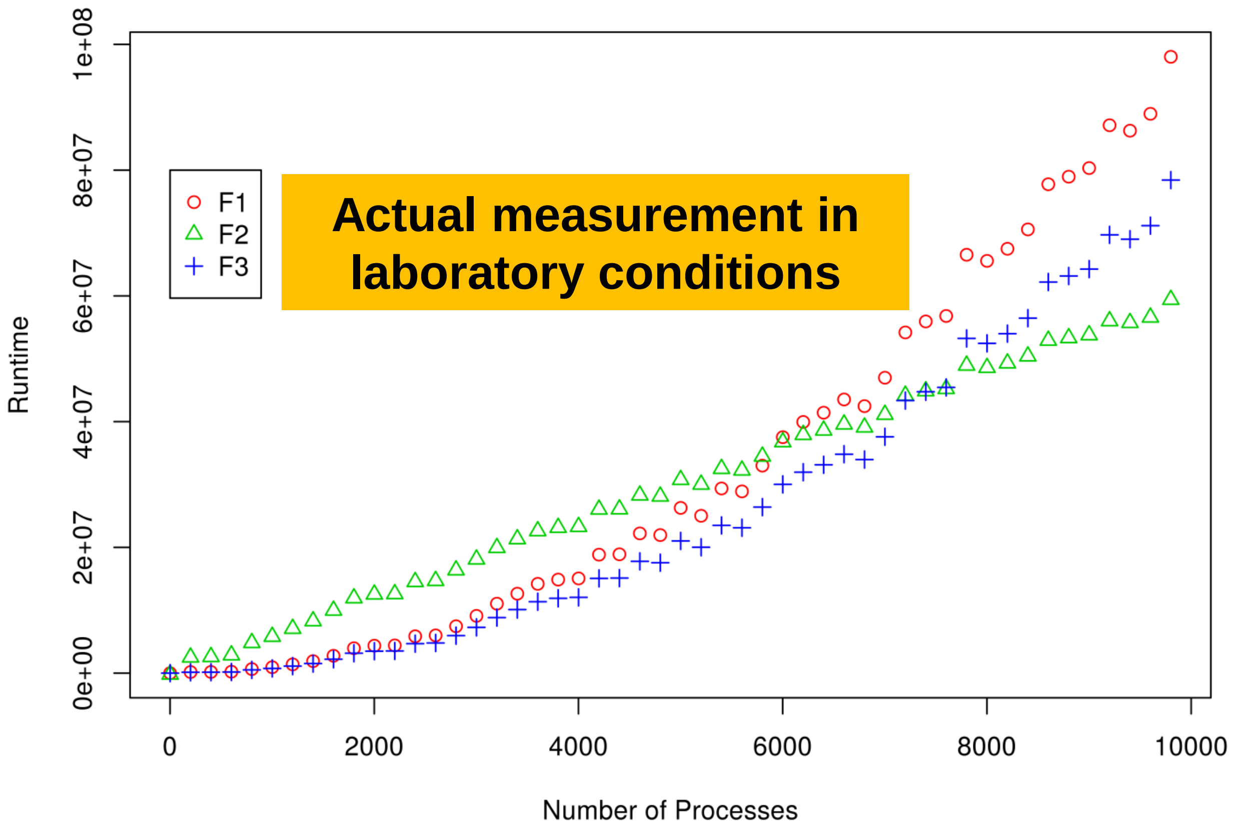
Primary focus on scaling trend



Our ranking

1. F_1
2. F_3
3. F_2

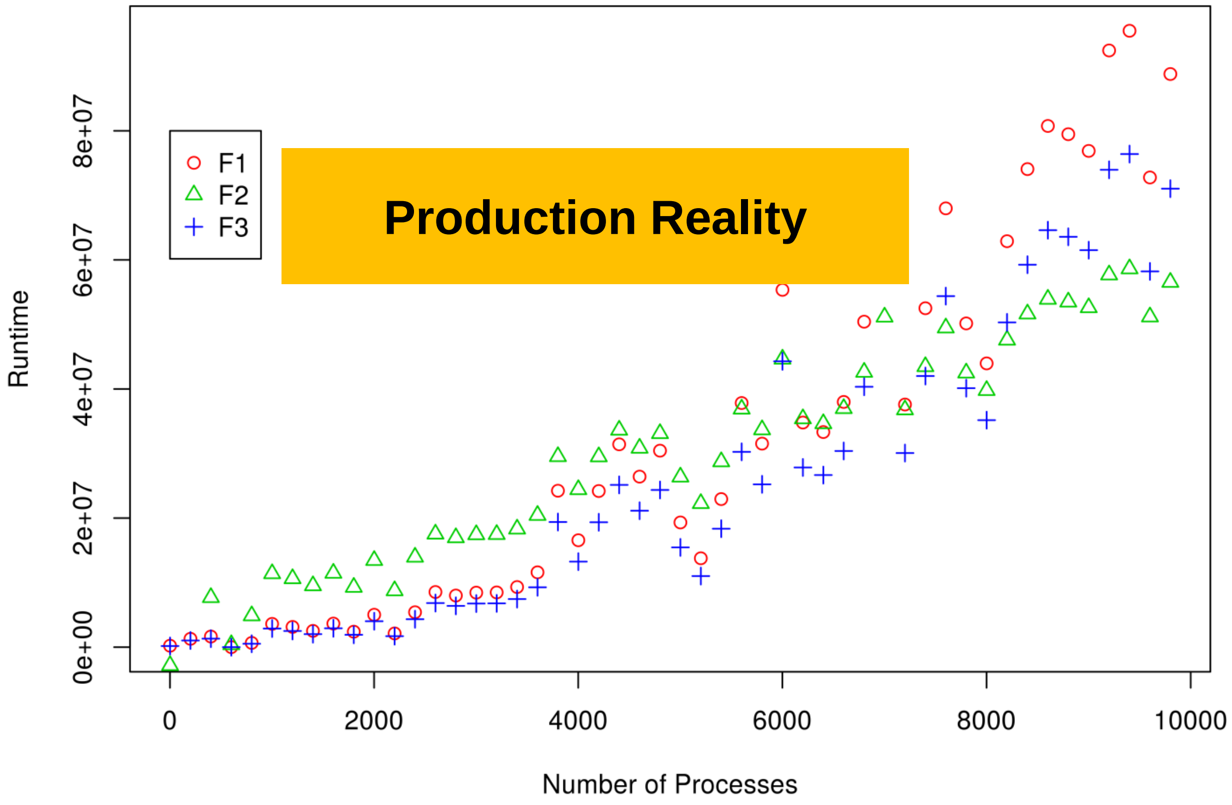
Primary focus on scaling trend



Our ranking

1. F_1
2. F_3
3. F_2

Primary focus on scaling trend



Our ranking

1. F_1
2. F_3
3. F_2

How to mechanize the expert? → Survey!

Computation

LU
$t(p) \sim c$

FFT
$t(p) \sim \log_2(p)$

Naïve N-body
$t(p) \sim p$

…

Samplesort
$t(p) \sim p^2 \log_2^2(p)$

Communication

LU
$t(p) \sim c$

FFT
$t(p) \sim \log_2(p)$

Naïve N-body
$t(p) \sim p$

…

Samplesort
$t(p) \sim p^2$

Survey result: performance model normal form

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

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

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

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

Survey result: performance model normal form

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

$n \in \mathbb{N}$

$i_k \in I$

$n = 2$

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

$J = \{0, 1\}$

$c_1 + c_2 \cdot p$

$c_1 + c_2 \cdot p^2$

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

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

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

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

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

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

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

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

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

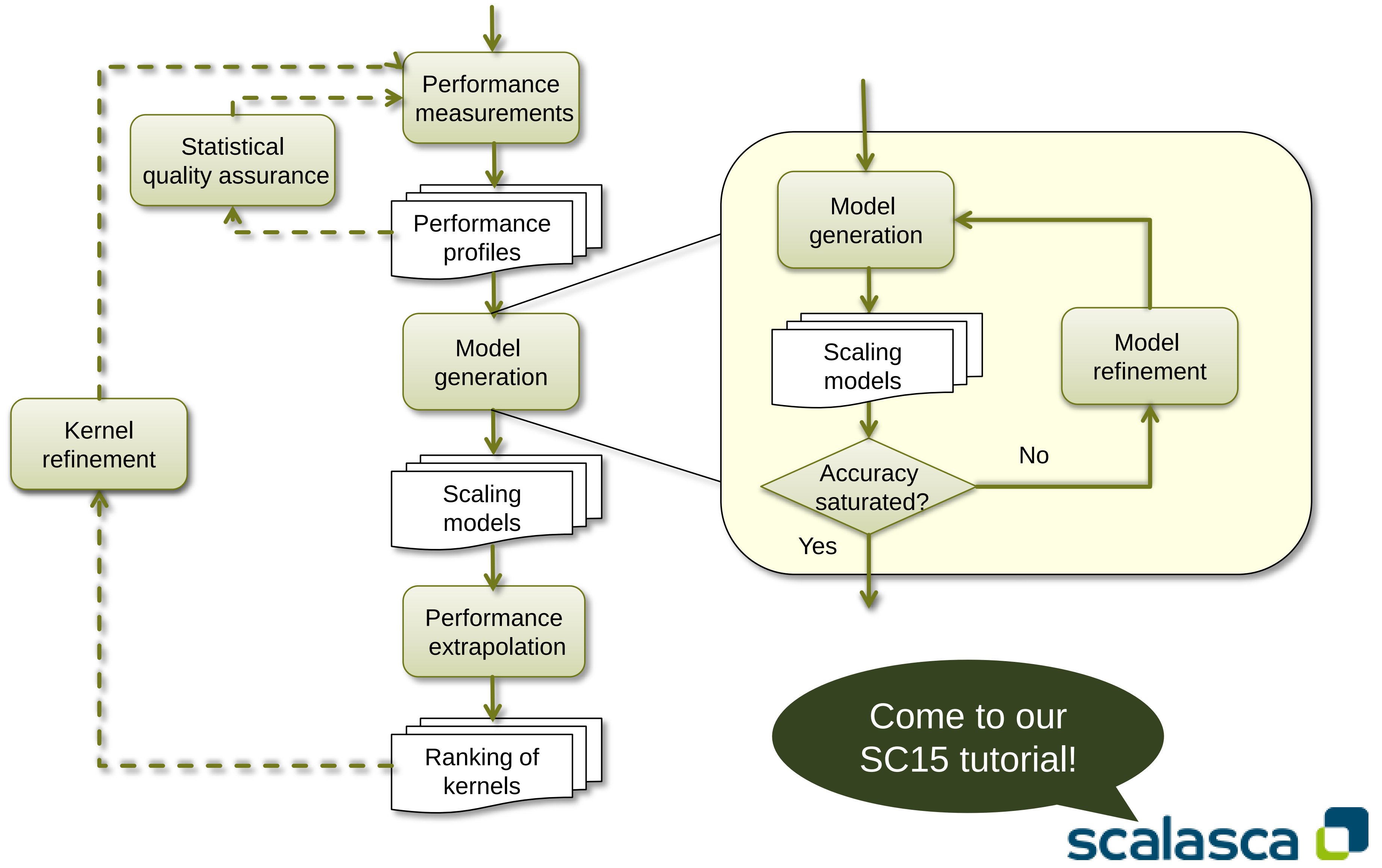
$c_1 \cdot p + c_2 \cdot p^2 \cdot \log(p)$

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

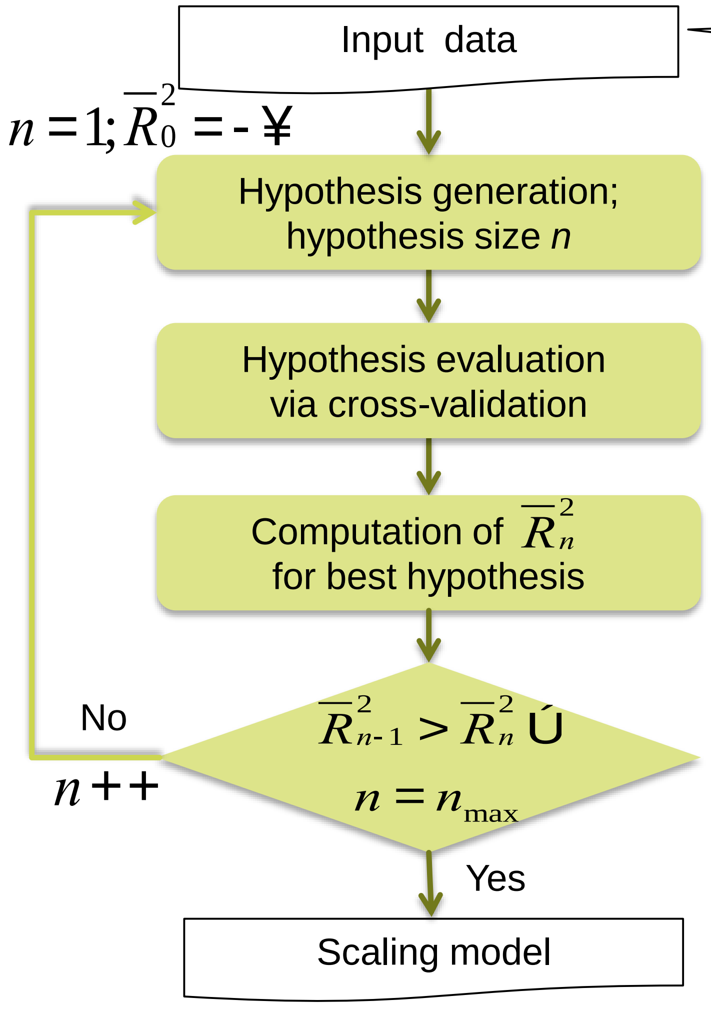
$c_1 \cdot p \cdot \log(p) + c_2 \cdot p^2 \cdot \log(p)$

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

Our automated generation workflow



Model refinement


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

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

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

$$c_1 \times \log(p)$$

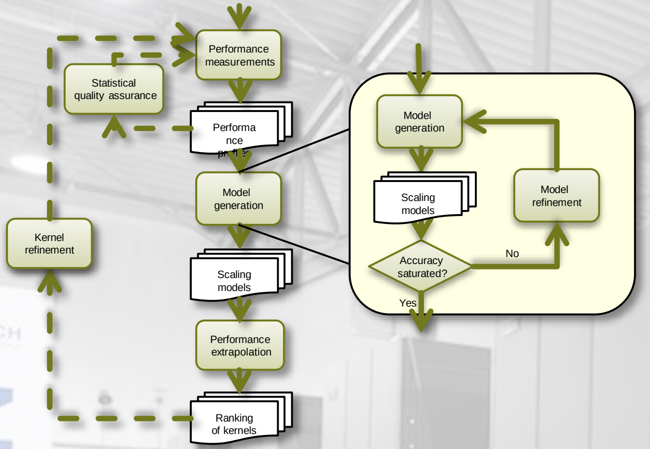
$$R^2 = 1 - \frac{residualSumSquares}{totalSumSquares}$$

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

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

JÜLICH
JUQUEEN
IBM
supercomputer Blue Gene Q
Blue Gene Q supercomputer

Evaluation overview

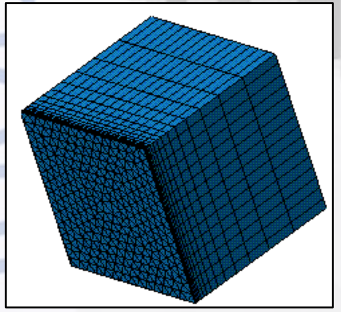


$$I = \{\tfrac{0}{2}, \tfrac{1}{2}, \tfrac{2}{2}, \tfrac{3}{2}, \tfrac{4}{2}, \tfrac{5}{2}, \tfrac{6}{2}\}$$

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

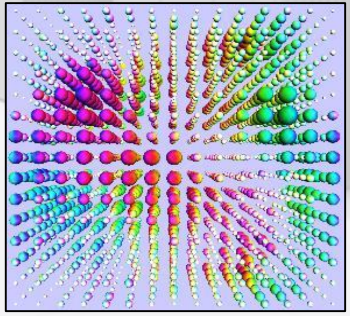
$$n = 5$$

Sweep3D



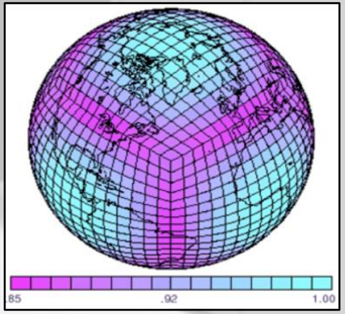
✓

MILC



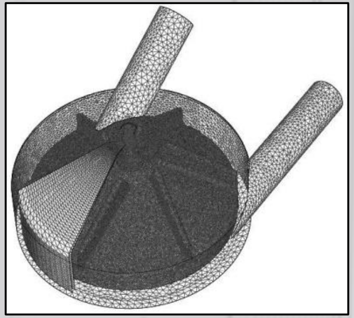
✗

HOMME



✓ ✓

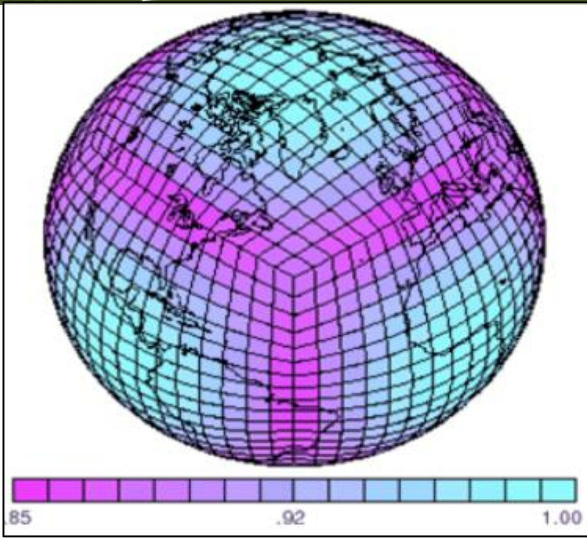
XNS



✓

HOMME

- **Core of the Community Atmospheric Model (CAM)**
- Spectral element dynamical core on a cubed sphere grid

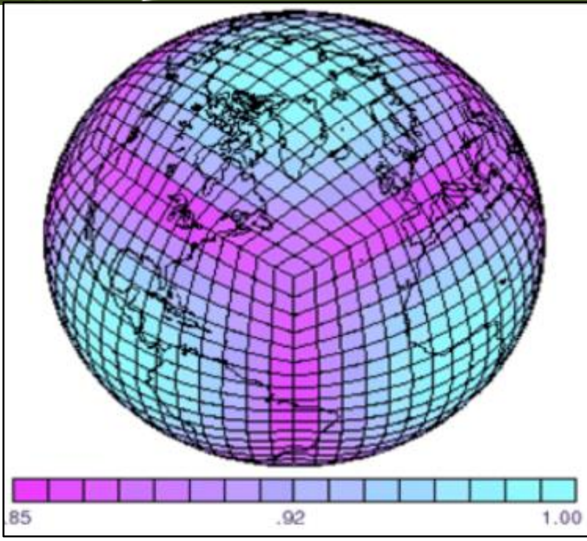


Kernel **[3 of 194]**	**Model [s]** **t = f(p)**	**Predictive error [%]** **p_t = 130k**
box_rearrange → MPI_Reduce	$0.026 + 2.53\times10^{-6} p\times\sqrt{p} + 1.24\times10^{-12} p^{3}$	57.02
vlaplace_sphere_vk	49.53	99.32
compute_and_apply_rhs	48.68	1.65

p_i £ 15k

HOMME

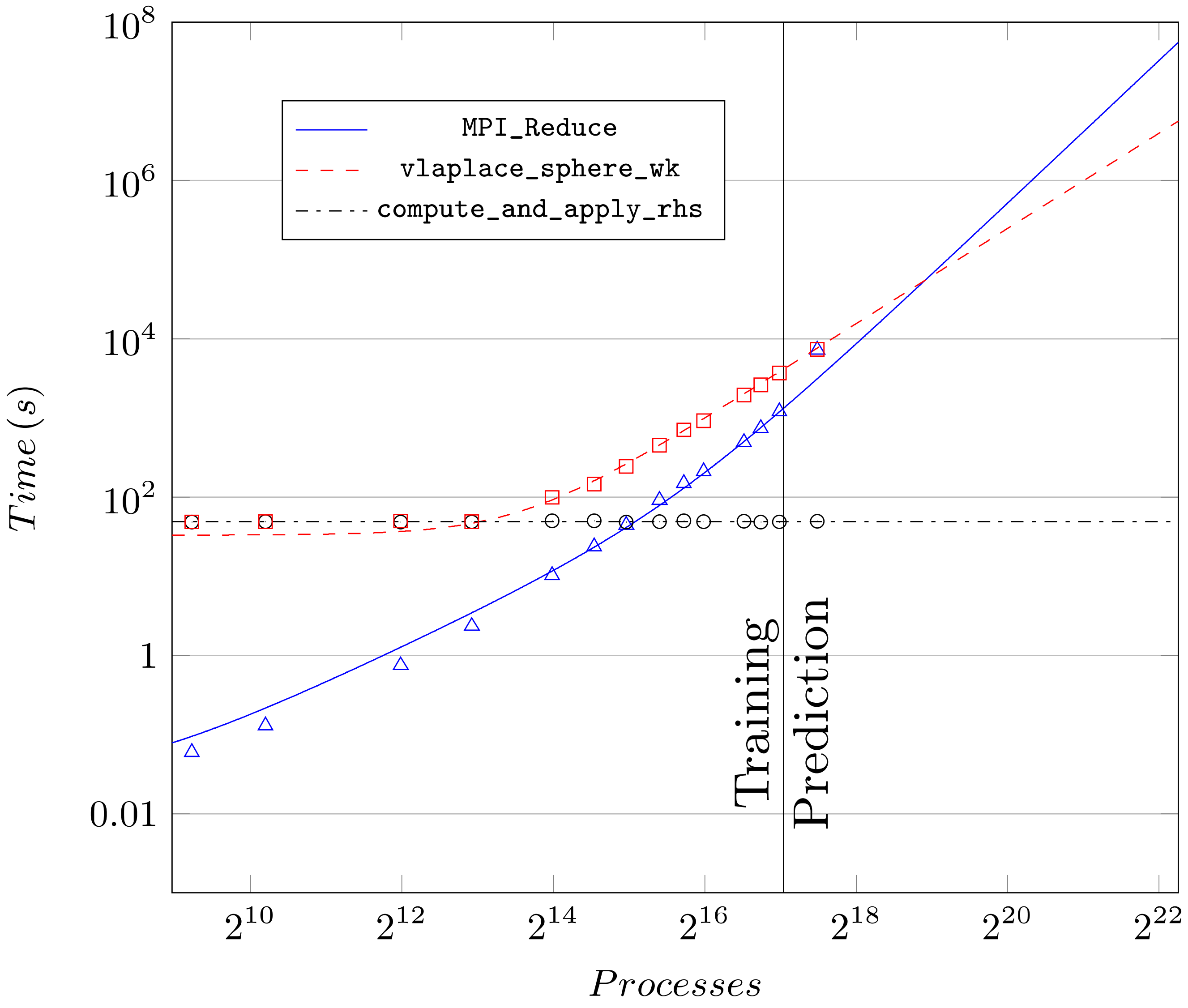
- **Core of the Community Atmospheric Model (CAM)**
- Spectral element dynamical core on a cubed sphere grid



Kernel [3 of 194]	Model [s] t = f(p)	Predictive error [%] p_t = 130k
box_rearrange → MPI_Reduce	$3.63\times10^{-6} p\times\sqrt{p} + 7.21\times10^{-13} p^3$	30.34
vlaplace_sphere_vk	$24.44 + 2.26\times10^{-7} p^2$	4.28
compute_and_apply_rhs	49.09	0.83

$\boxed{p_i \text{ £ } 43\text{k}}$

HOMME


10^8
10^6
10^4
10^2
1
0.01
$Time\ (s)$
MPI_Reduce
vlaplace_sphere_wk
compute_and_apply_rhs
Training
Prediction
2^{10}
2^{12}
2^{14}
2^{16}
2^{18}
2^{20}
2^{22}
$Processes$

It works, great! Or not?

- **We face several problems:**
  - Multiple models – when did we collect enough data?
    if(np < 1.000) a(); else b();
  - Multiparameter modeling – search space explosion
    Interesting instance of the curse of dimensionality
  - Modeling overheads – traces do not scale
    Cross validation (leave-one-out) is slow and
    Our current profiling requires a lot of storage (>TBs)

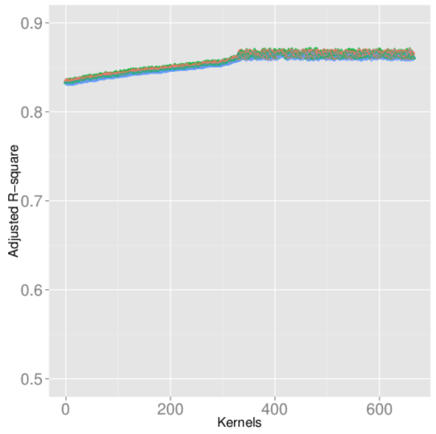
First step: simple compiler analyses

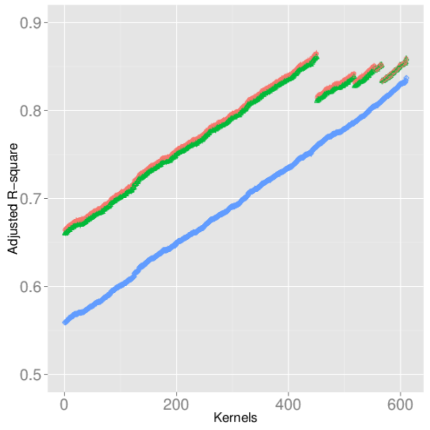
- **Automatic kernel detection in LLVM**
  - Loop call graph – each loop/function as kernel (recursively)
  - Determine relevant input parameters for each kernel

    Massive pruning possible!
- **Online model generation (using online LASSO)**

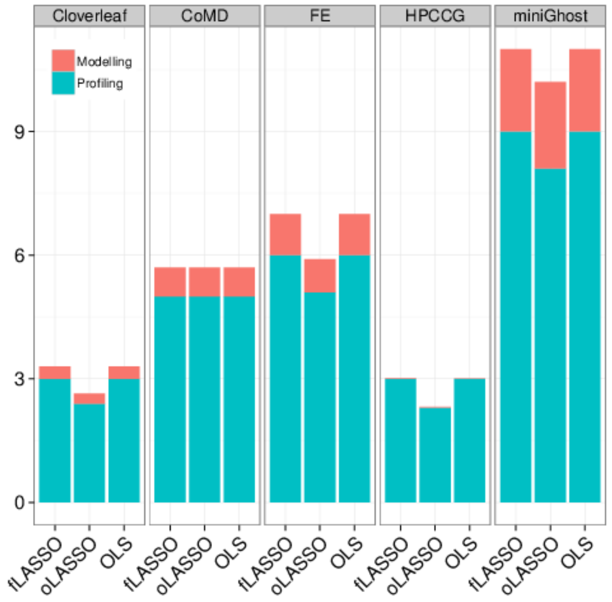
  Constant (little) amount of data stored
- **Automatic profiling rate limiting**

  Profile less as models gain confidence





Quality: NAS UA and Mantevo MiniFE

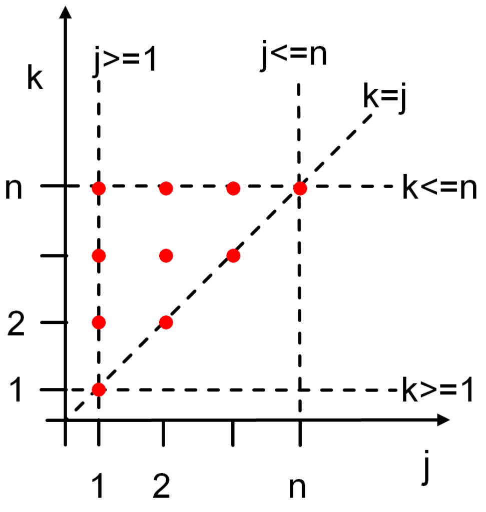


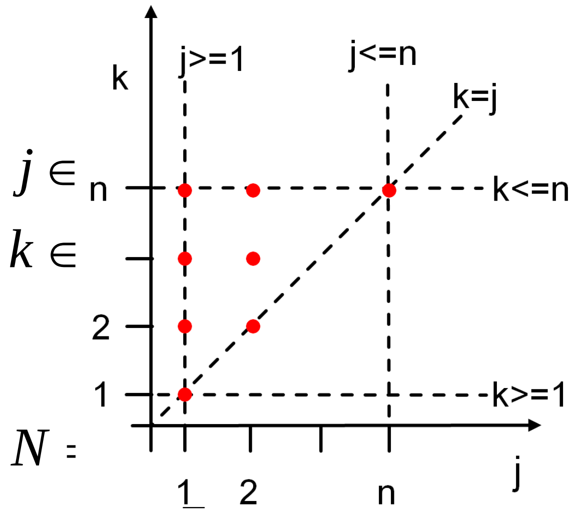
Overhead: Mantevo

Second step: counting loop iterations

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

Polyhedral model





$$N = (n+1)\log_2 n - n + 2$$

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($c_{k+1}^T x < g_{k+1}$) {... }
           $x = U_k x + v_k$; }
       $x = U_{k-1} x + v_{k-1}$;
      ...}
   $x = U_1 x + v_1$;}
```

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

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);
```

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

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

    }
}
```

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

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

$$while\left(\begin{pmatrix} 1 & 0 \end{pmatrix}\begin{pmatrix} j \\ k \end{pmatrix} < \tfrac{n}{p} + 1\right)\{$$

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

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

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

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

$$\}$$

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

Overview of the whole system

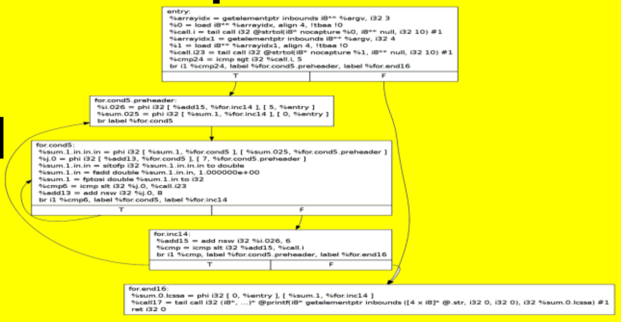
Parallel program

```
    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

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

with

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

Number of iterations

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

Program analysis

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

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

Case study: NAS EP

$$N(m,p) = \left\lceil \frac{2^{m-16}\cdot(u+2^{16})}{p} \right\rceil$$

$$W = T_1 \approx 2^m$$
$$D = T_\infty \approx 1$$

$$E_P = \frac{2^m}{p\left\lceil \frac{2^m}{p} \right\rceil}$$

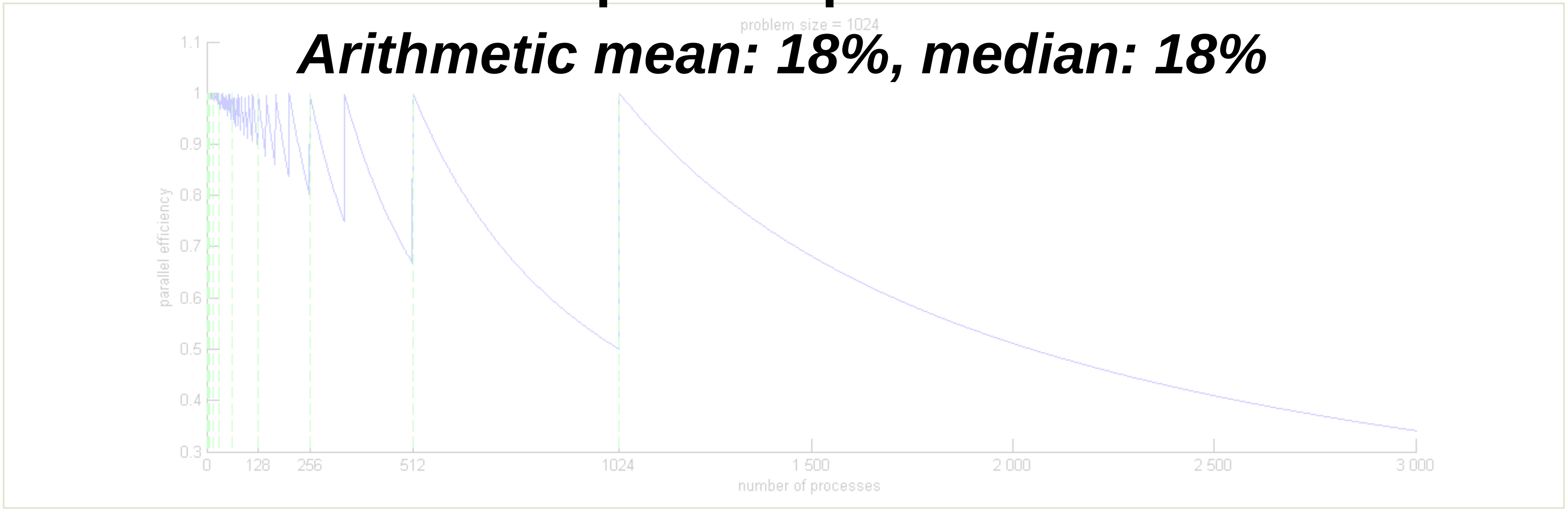
$$E_P = 1 \text{ if } p \leq 2^m$$
$$E_P = 2^m/p \text{ if } p > 2^m$$

```
u:    do i=1,100
        ik =kk/2
        if (ik .eq. 0) goto 130
        [illegible]
      continue
```

15 applications (NAS/Mantevo/Mibench):

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

Arithmetic mean: 18%, median: 18%



What problems are remaining?

- **Well, what about non-affine loops?**
  - More general abstract interpretation (next step)
  - Not decidable → will always have undefined terms

$$\boxed{N = \frac{\mathtt{na} \cdot u}{\mathtt{nprows}}}$$

- **Back to PMNF?**
  - Generalize to multiple input parameters

    a) Bigger search-space ☹

    b) Bigger trace files ☹

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

- **Combine static (loop counting) and dynamic approach (PMNF)**
  - Find number of loop iterations, replace u_x with PMNF_x
  - Find similar kernels (use only one PMNF for similar ones)
  - Remove irrelevant input parameters for each PMNF
  - Other simple optimizations: batch model update, etc.
  - Result: higher accuracy, lower overheads

Performance Analysis 2.0 – Automatic Models

A call for action: use performance modeling for rigorous designs

- **Especially for co-design (systems and applications)!**
  New architectures, e.g., FPGA assessment
- **High-performance programming as a science**
  Learn from natural sciences!
- **Start with the students: teach rigorous analysis and modeling**

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

T. Hoefler, G. Kwasniewski: Automatic Complexity Analysis of Explicitly Parallel Programs. *SPAA 2014.*

A. Bhattacharyya, T. Hoefler: PEMOGEN: Automatic Adaptive Performance Modeling during Program Runtime, *PACT 2014*

S. Shudler, A. Calotoiu, T. Hoefler, A. Strube, F. Wolf: Exascaling Your Library: Will Your Implementation Meet Your Expectations? *ICS 2015*

A. Bhattacharyya, G. Kwasniewski, T. Hoefler: Using Compiler Techniques to Improve Automatic Performance Modeling, *PACT 2015*