TORSTEN HOEFLER

How fast will your application go? 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

Performance modeling

- **What is this all about???**
  - A wide-spread practitioner's view on performance modeling:

(replace "meeting" with performance optimization and "premeeting" with performance modeling)


IBM


CRAY

Analytical application performance modeling

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

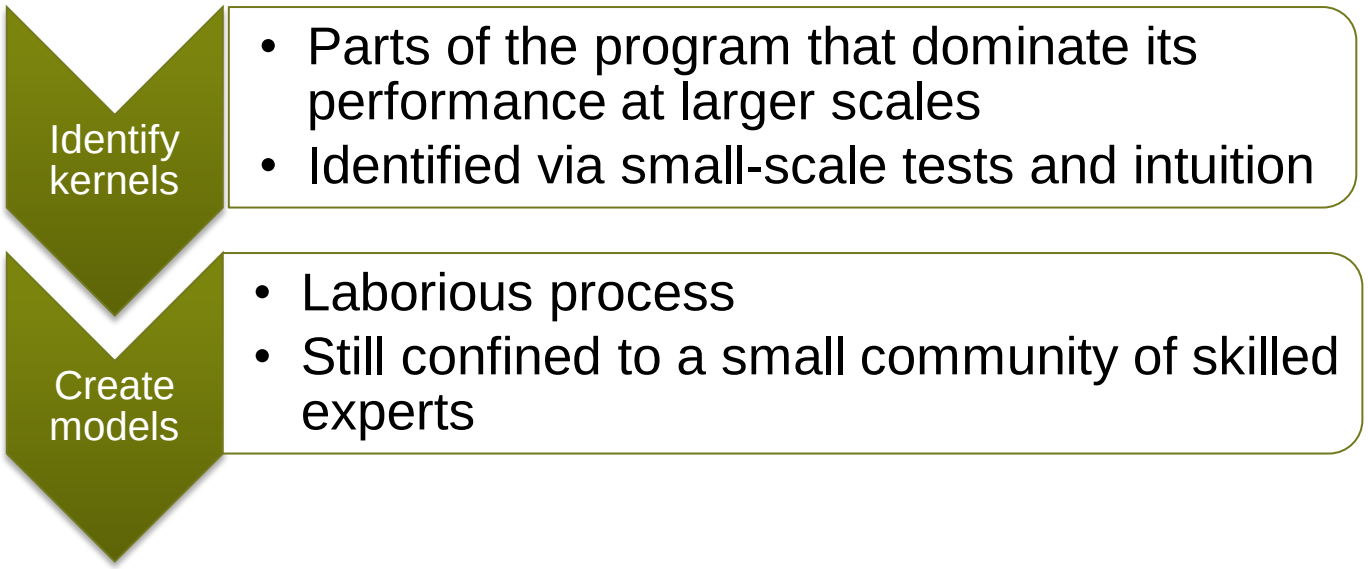
  A. Calotoiu, TH, M. Poke, F. Wolf: Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes
- **Automated performance testing**
  - Performance modeling as part of a software engineering discipline in HPC

  ICS'15: S. Shudler, A. Calotoiu, S. Hoefler, A. Strube, F. Wolf: Exascaling Your Library: Will Your Implementation Meet Your Expectations?
- **Hardware/Software co-design**
  - Decide how to architect systems
- **Making performance development intuitive**

Performance Modeling = Performance Analysis v 2.0 (the next logical step)

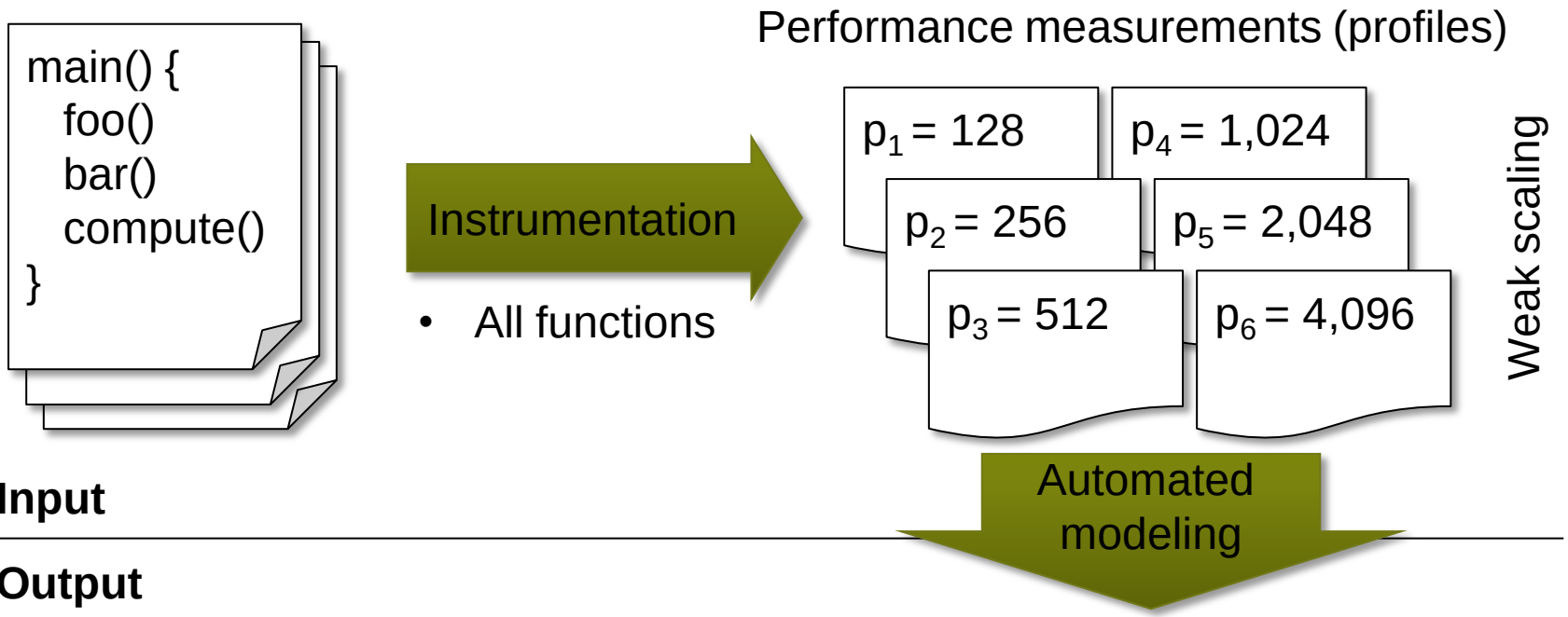
vs.

Manual analytical performance modeling

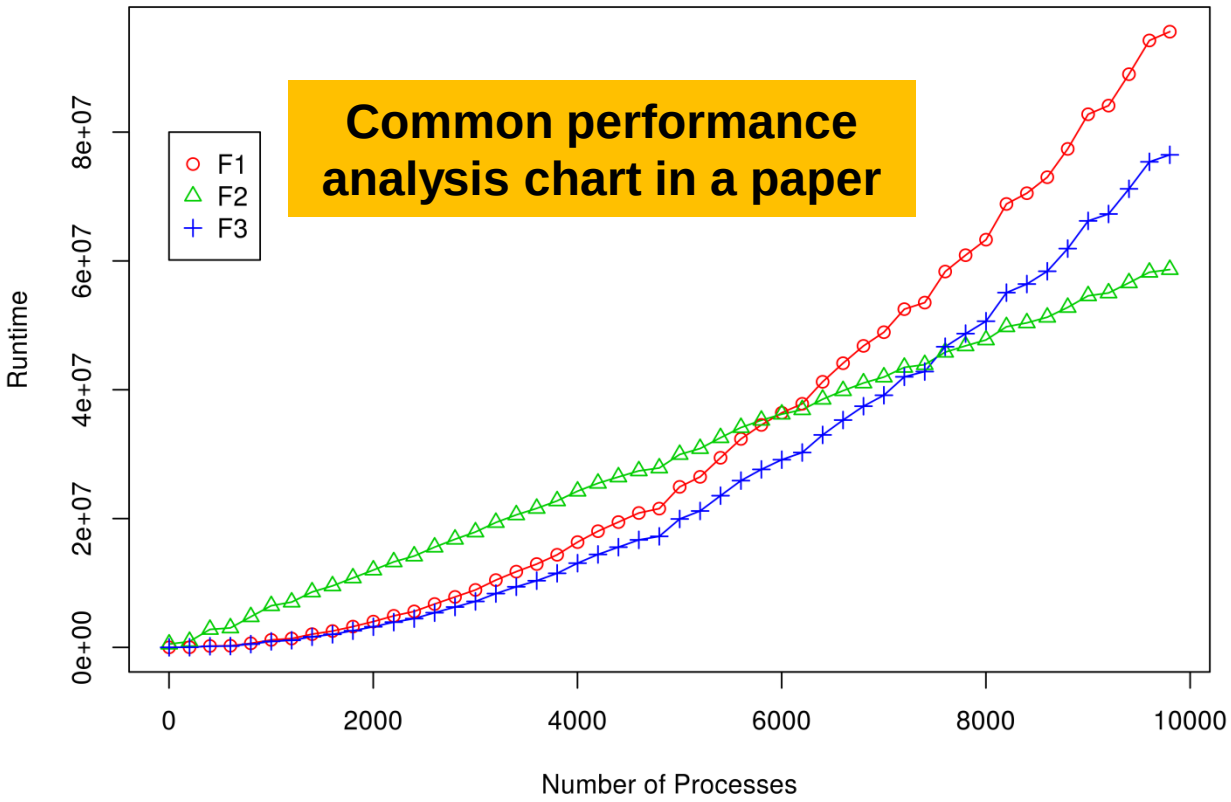


- **Disadvantages**
  - Time consuming
  - Error-prone, may overlook unscalable code

Our first step: scalability bug detector



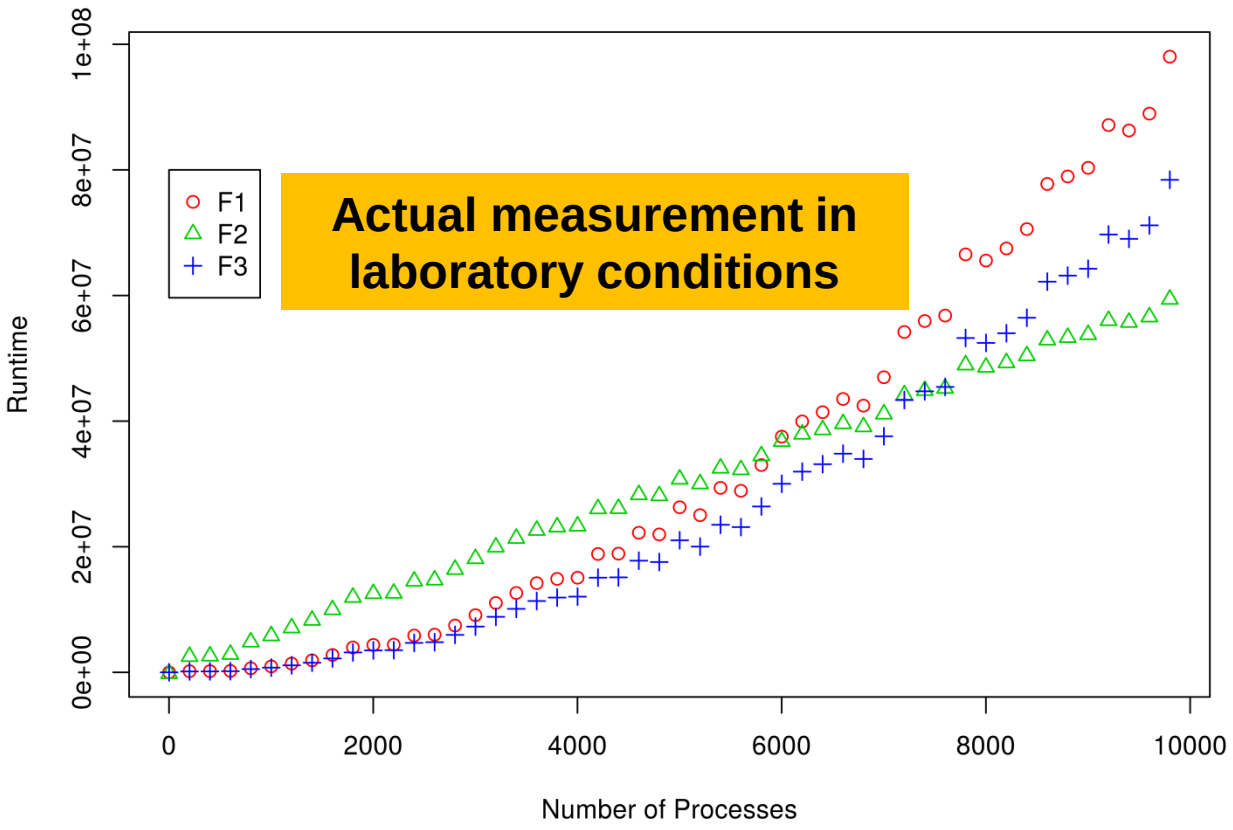
Primary focus on scaling trend



Our ranking

1. F_1
2. F_3
3. F_2

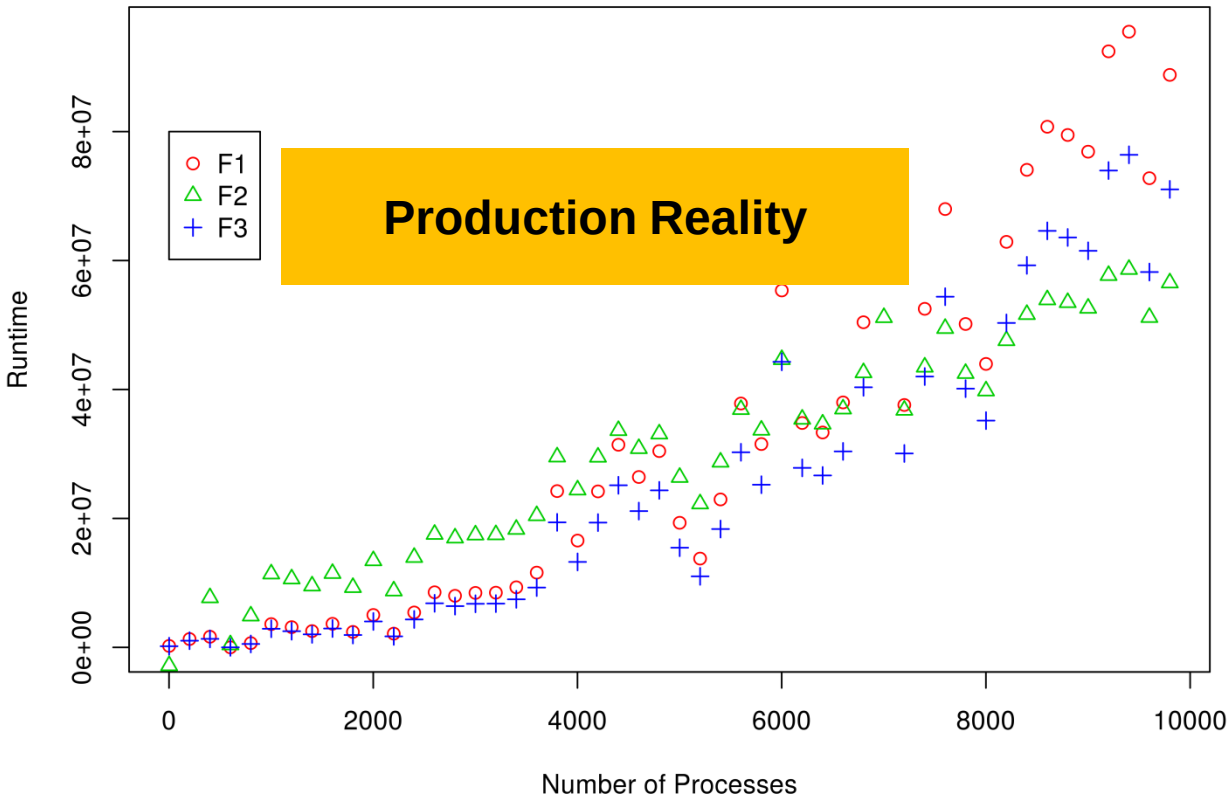
Primary focus on scaling trend



Our ranking

1. F_1
2. F_3
3. F_2

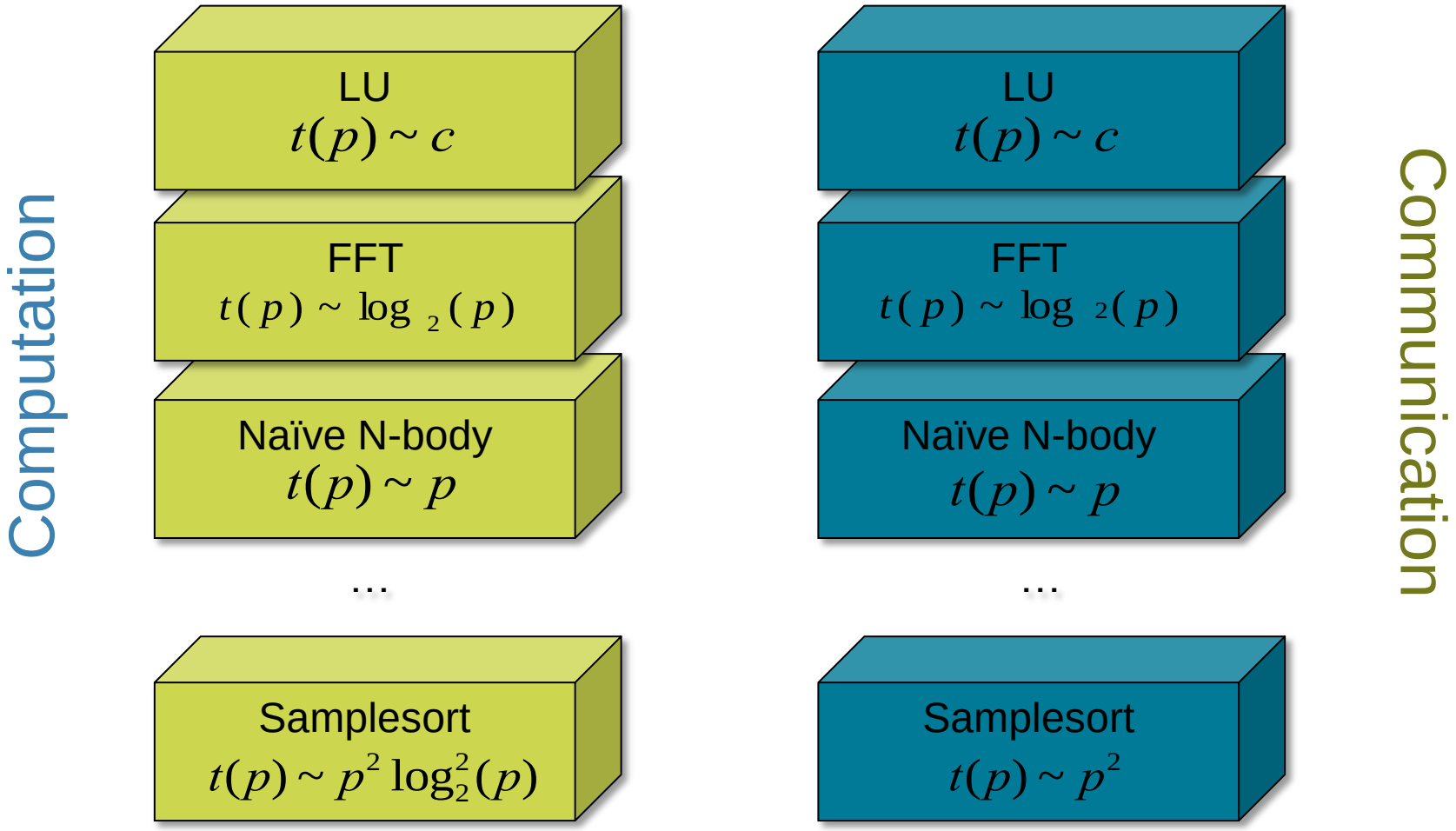
Primary focus on scaling trend



Our ranking

1. F_1
2. F_3
3. F_2

How to mechanize the expert? → Survey!

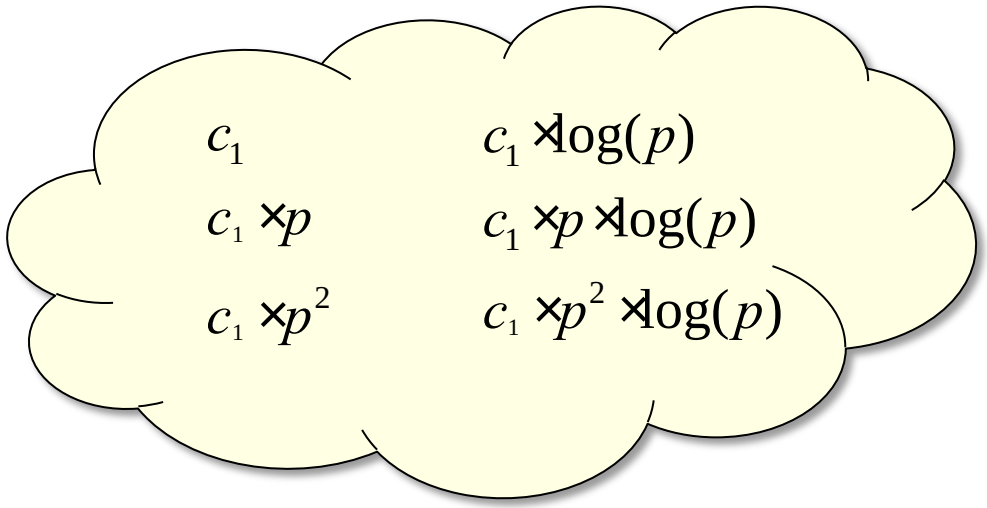


Survey result: performance model normal form

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

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

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



Survey result: performance model normal form

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

$n \in \mathbb{N}$

$i_k \in I$

$n = 2$

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

$J = \{0, 1\}$

$c_1 + c_2 \cdot p$

$c_1 + c_2 \cdot p^2$

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

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

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

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

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

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

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

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

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

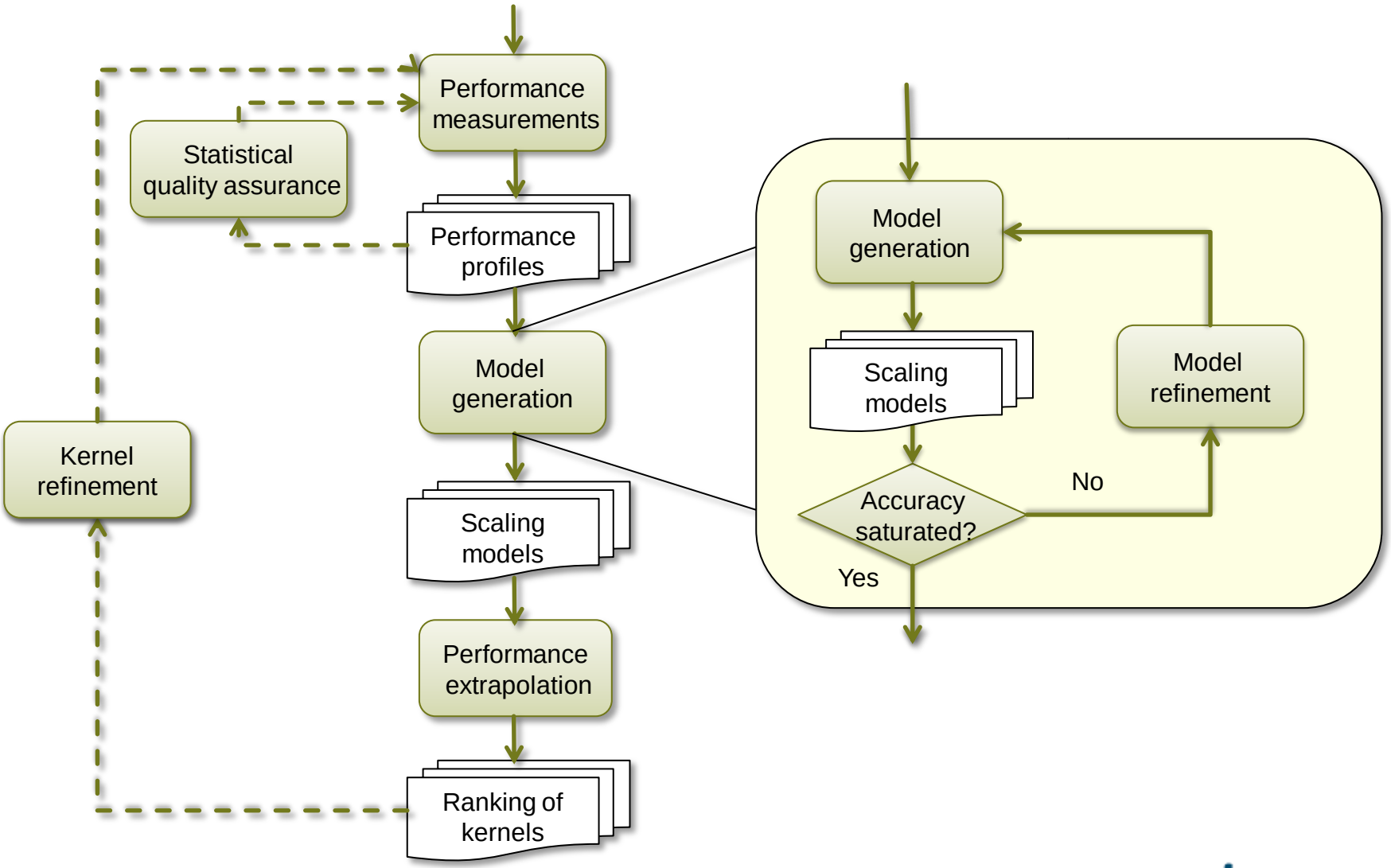
$c_1 \cdot p + c_2 \cdot p^2 \cdot \log(p)$

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

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

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

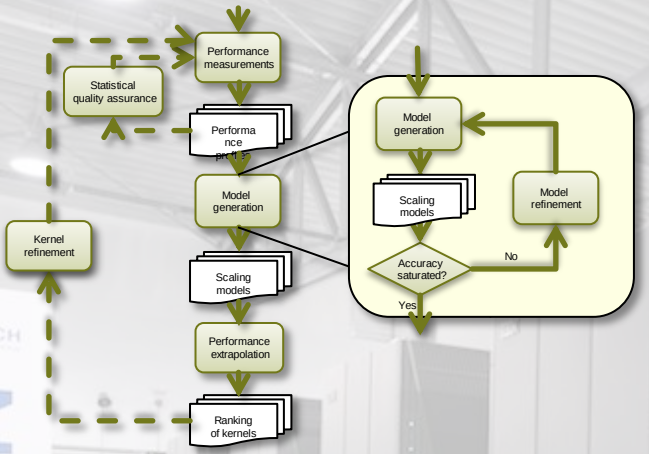
Our automated generation workflow



scalasca


JÜLICH
JUQUEEN
IBM
supercomputer Blue Gene Q

Evaluation overview

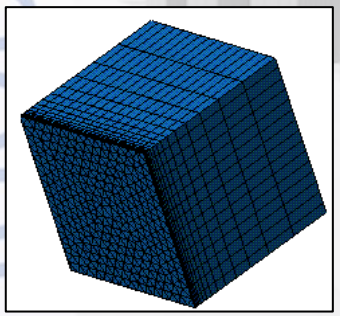


$$I = \{\frac{0}{2}, \frac{1}{2}, \frac{2}{2}, \frac{3}{2}, \frac{4}{2}, \frac{5}{2}, \frac{6}{2}\}$$

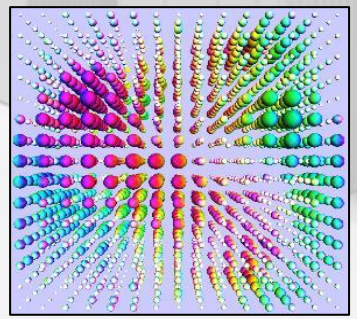
$$J = \{0, 1, 2\}$$

$$n = 5$$

Sweep3D

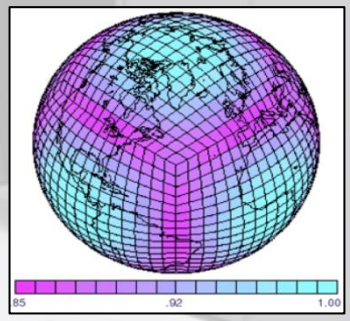


MILC

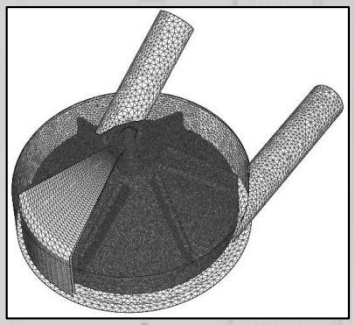


✖

HOMME

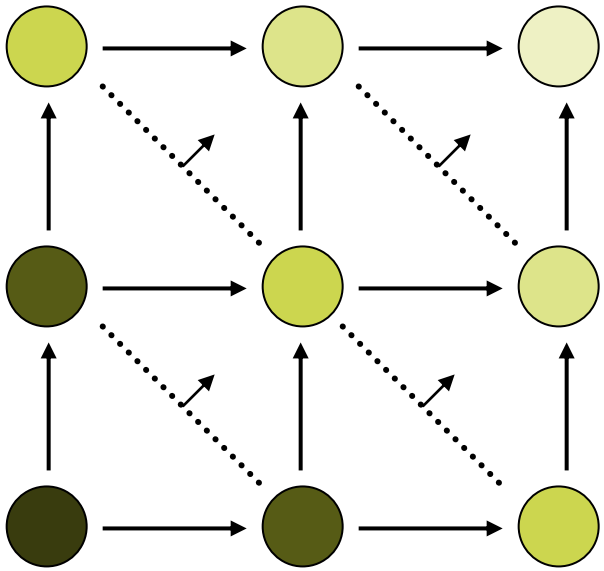


XNS



Sweep3D communication performance

- **Solves neutron transport problem**
- **3D domain mapped onto 2D process grid**
- **Parallelism achieved through pipelined wave-front process**

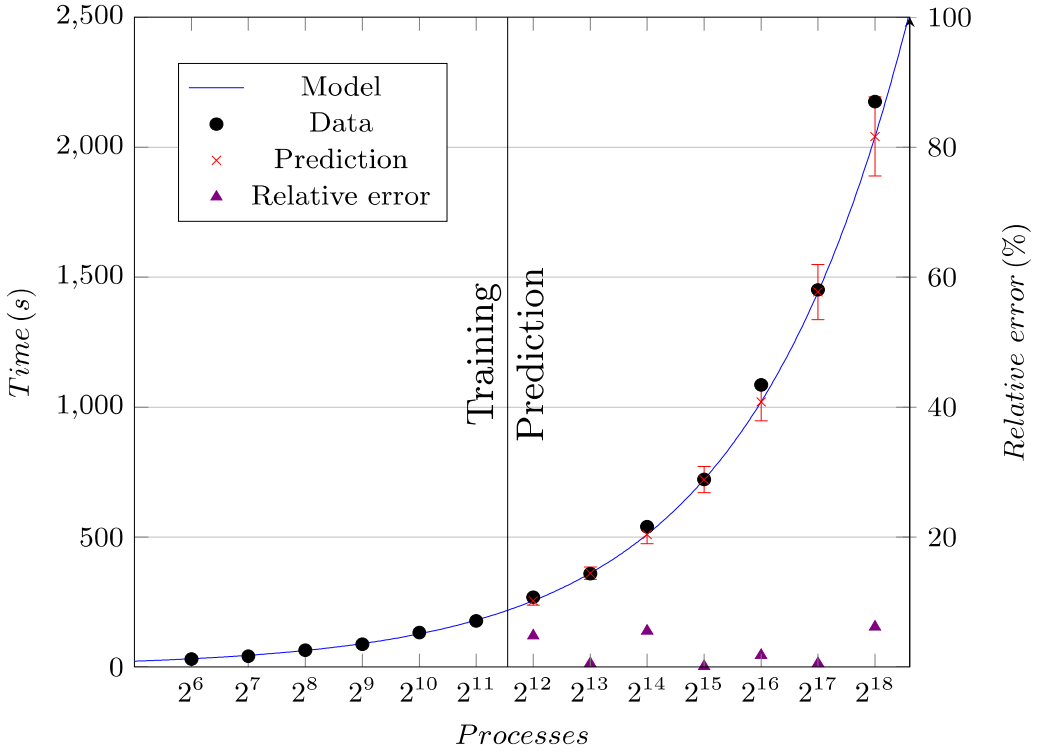


$$t^{comm} = c \cdot \sqrt{p}$$

- **LogGP model for communication developed by Hoisie et al.**
  - We assume $p=p_x*p_y \rightarrow$ Equation (6) in [1]

[1] A. Hoisie, O. M. Lubeck, and H. J. Wasserman. Performance analysis of wavefront algorithms on very-large scale distributed systems. In Workshop on Wide Area Networks and High Performance Computing, pages 171–187. Springer-Verlag, 1999.

Sweep3D communication performance

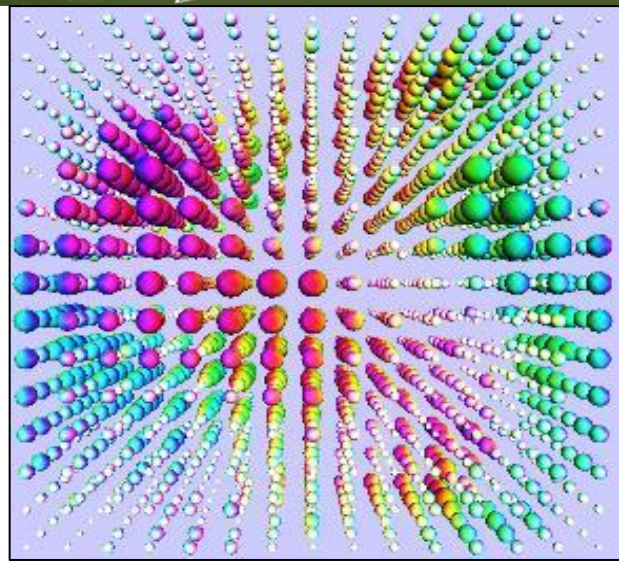


Kernel [2 of 40]	Runtime[%] p_t=262k	Model [s] t = f(p)	Predictive error [%] p_t=262k
sweep → MPI_Recv	65.35	$4.03\sqrt{p}$	5.10
sweep	20.87	582.19	.01

#bytes = const.
#msg = const.

$p_i \pounds 8k$

MILC



- **MILC/su3_rmd – from MILC suite of QCD codes with performance model manually created**

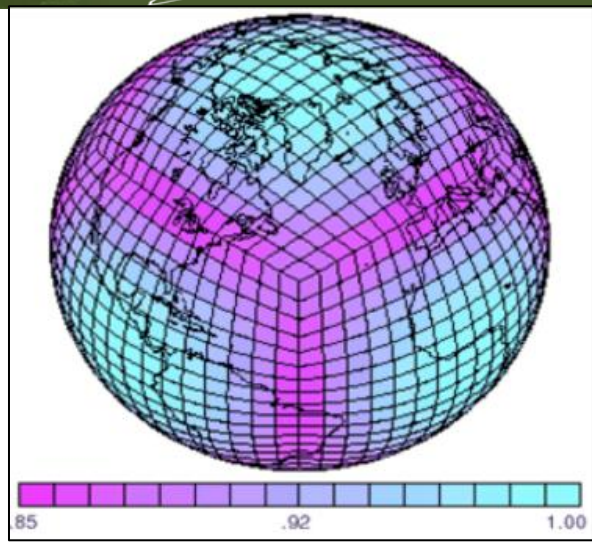
- **Time per process should remain constant except for a rather small logarithmic term caused by global convergence checks**

Kernel [3 of 479]	Model [s] t=f(p)	Predictive Error [%] p_t=64k
compute_gen_staple_field	2.40×10^{-2}	0.43
g_vecdoublesum → MPI_Allreduce	$6.30 \times 10^{-6} \times \log_2^2(p)$	0.01
mult_adj_su3_fieldlink_lathwec	3.80×10^{-3}	0.04

$$p_i \pounds 16k$$

HOMME

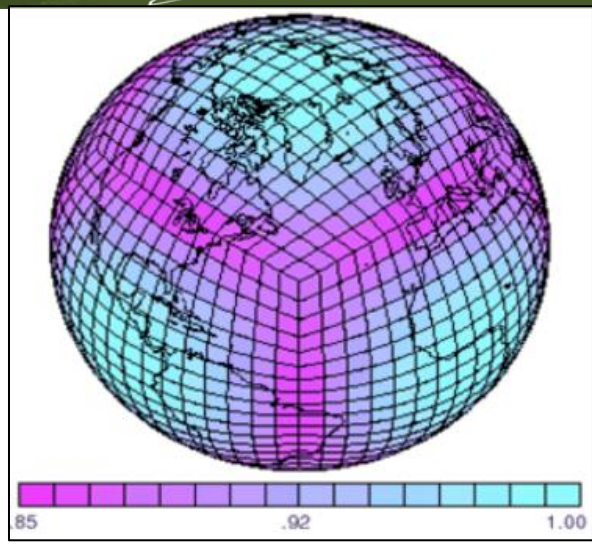
- **Core of the Community Atmospheric Model (CAM)**
- Spectral element dynamical core on a cubed sphere grid



Kernel [3 of 194]	Model [s] t = f(p)	Predictive error [%] p_t = 130k
box_rearrange → MPI_Reduce	$0.026 + 2.53 \times 10^{-6} p \times \sqrt{p} + 1.24 \times 10^{-12} p^3$	57.02
vlaplace_sphere_vk	49.53	99.32
compute_and_apply_rhs	48.68	1.65

p_i £15k

HOMME (2)

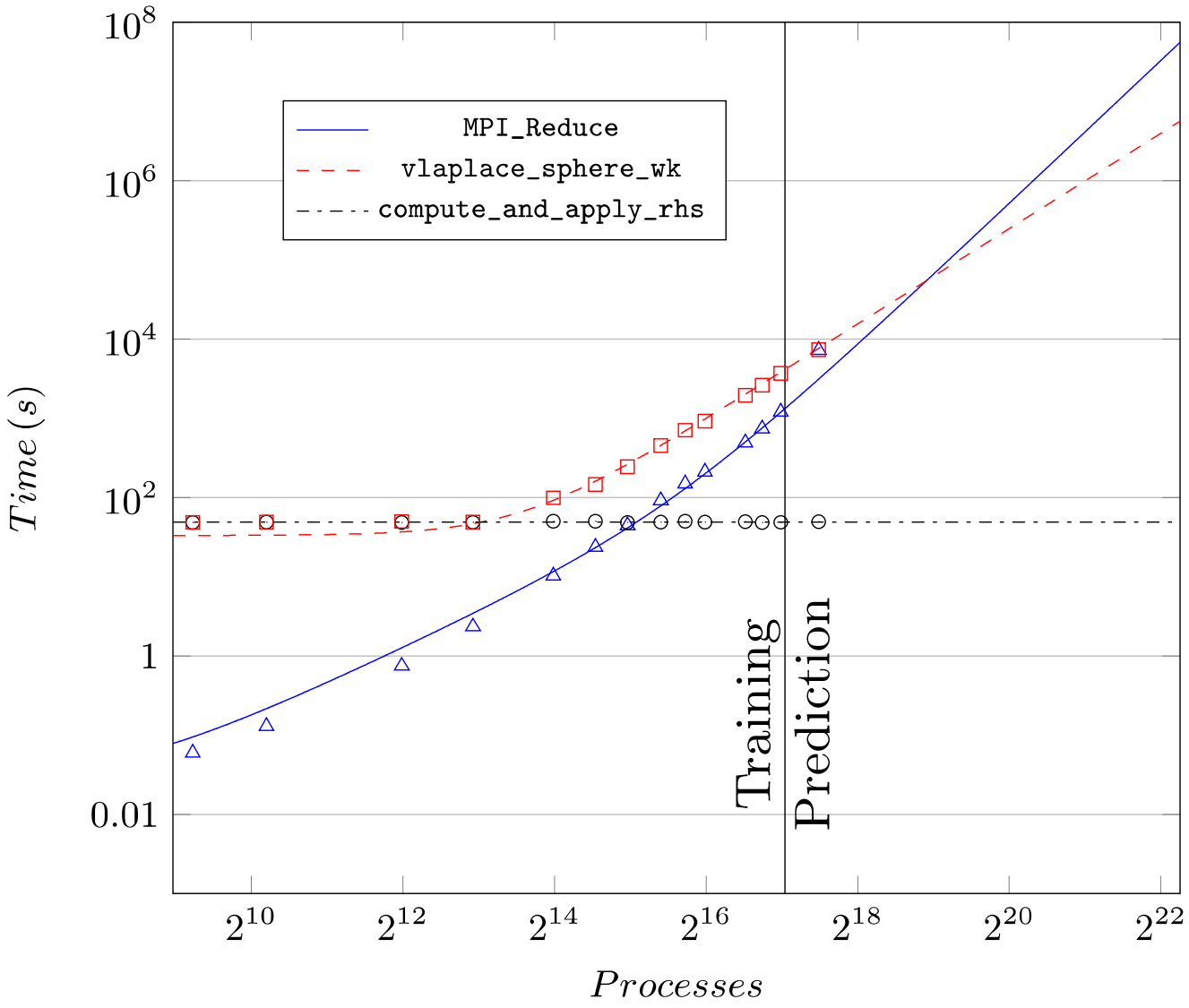


- **Core of the Community Atmospheric Model (CAM)**
- Spectral element dynamical core on a cubed sphere grid

Kernel [3 of 194]	Model [s] t = f(p)	Predictive error [%] p_t = 130k
box_rearrange → MPI_Reduce	$3.63 \times 10^{-6} p \times \sqrt{p} + 7.21 \times 10^{-13} p^3$	30.34
vlaplace_sphere_vk	$24.44 + 2.26 \times 10^{-7} p^2$	4.28
compute_and_apply_rhs	49.09	0.83

$p_i \pounds 43\text{k}$

HOMME (3)


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

What about strong scaling?

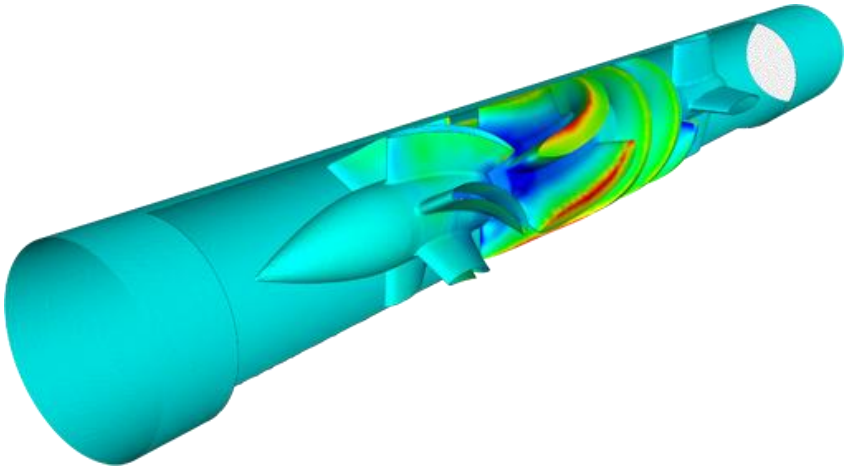
- **Wall-clock time not necessarily monotonically increasing – harder to capture model automatically**
  - Different invariants require different reductions across processes

	Weak scaling	Strong scaling
Invariant	Problem size per process	Overall problem size
Model target	Wall-clock time	Accumulated time
Reduction	Maximum / average	Sum

- **Superlinear speedup through cache effects**
  - Measure and model re-use distance?

XNS

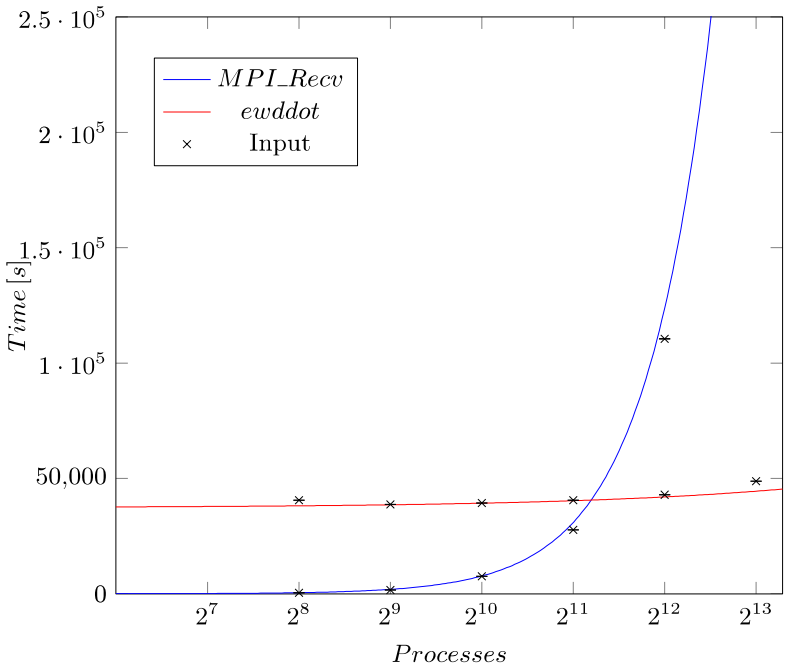
- **Finite element flow simulation program with numerous equations represented:**
  - Advection diffusion
  - Navier-Stokes
  - Shallow water



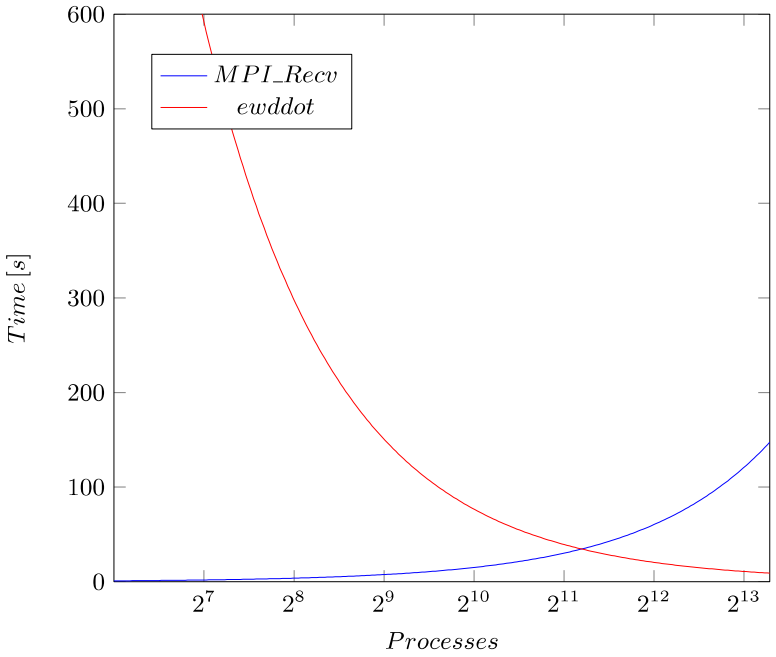
- **Strong scaling analysis**
  - P = {128; …; 4,096}
  - 5 measurements per p_i
  - Using accumulated time across processes as metric

XNS (2)

Accumulated time



Wallclock time



Kernel	Runtime[%] p=128	Runtime[%] p=4,096	Model [s] t = f(p)
ewdgennprm->MPI_Recv	0.46	51.46	$0.029 \times p^2$
ewddot	44.78	5.04	$p \times \log(p)$

#bytes = ~p
#msg = ~p

Is this all? No, it's just the beginning …

- **We face several problems:**
  - Multiparameter modeling – search space explosion
    Interesting instance of the curse of dimensionality
  - Modeling overheads
    Cross validation (leave-one-out) is slow and
    Our current profiling requires a lot of storage (>TBs)

Step back – what do we really care about?

- **Work**

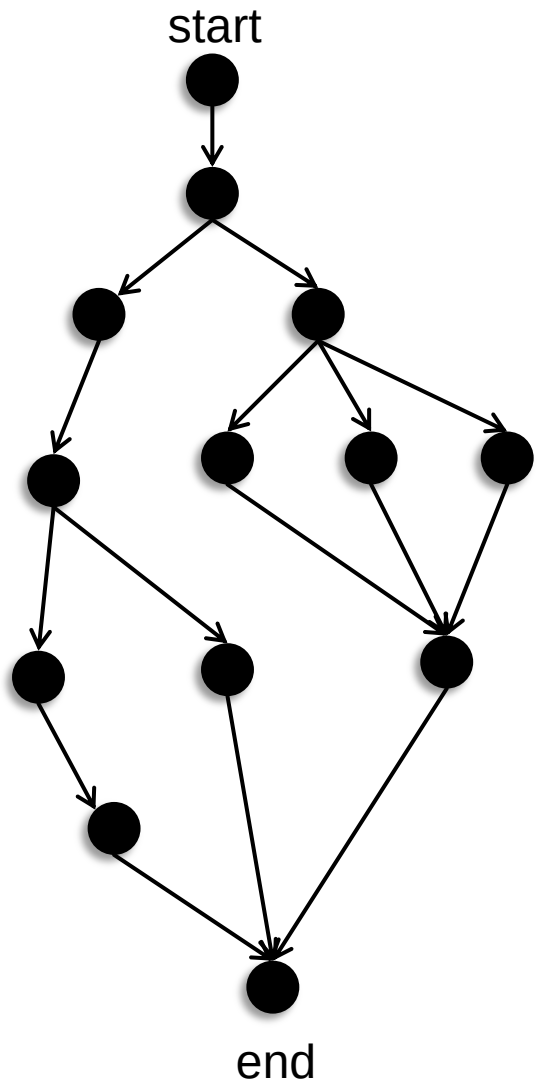
$$W = T_1$$

- **Depth**

$$D = T_\infty$$

- **Parallel efficiency**

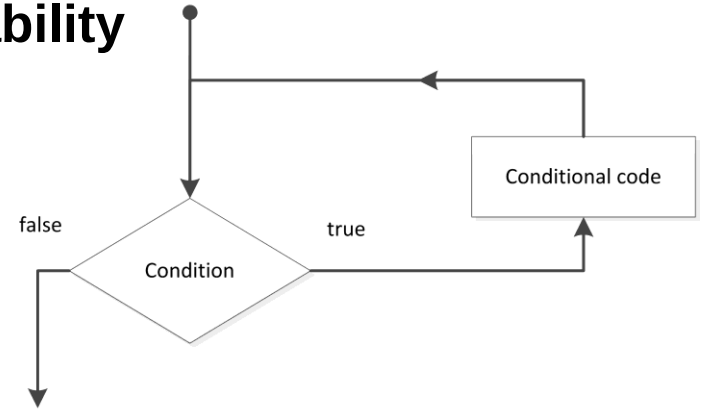
$$E_p = \frac{T_1}{pT_p}$$



Static analysis of explicitly parallel programs

- **Structures that determine program scalability**

LOOPS



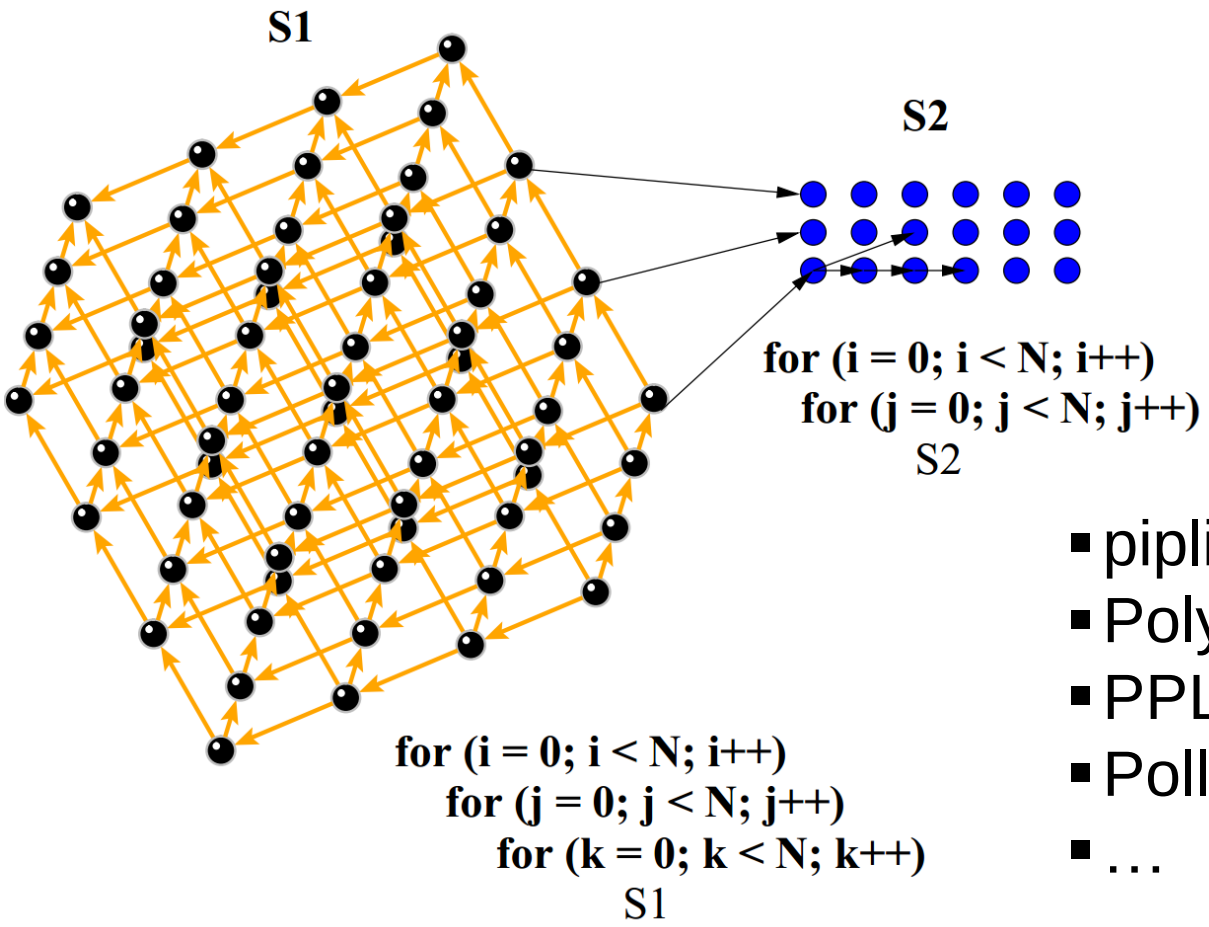
- **Assumption:**
  Other instructions do not influence it

- **Example:**

```
for (x=0; x < n/p; x++)
    for (y=1; y < n; y=2*y )
        veryComplicatedOperation(x,y);
```

Related work: counting loop iterations

- **Polyhedral model**



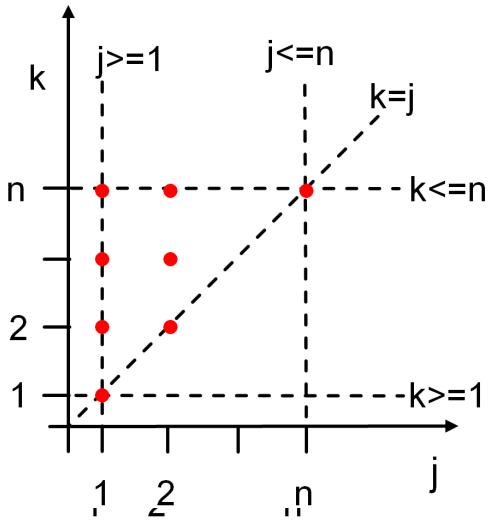
- piplib
- PolyLib
- PPL
- Polly
- . . .

R. Karp, R. Miller, and S. Winograd. The organization of computations for uniform recurrence equations. J. ACM, 14(3):563–590, July 1967.

Related work: counting loop iterations

- **Polyhedral model**

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



$$j \in [1, n]$$

$$k \in [j, n]$$

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

$$N = \frac{n(n+1)}{2}$$

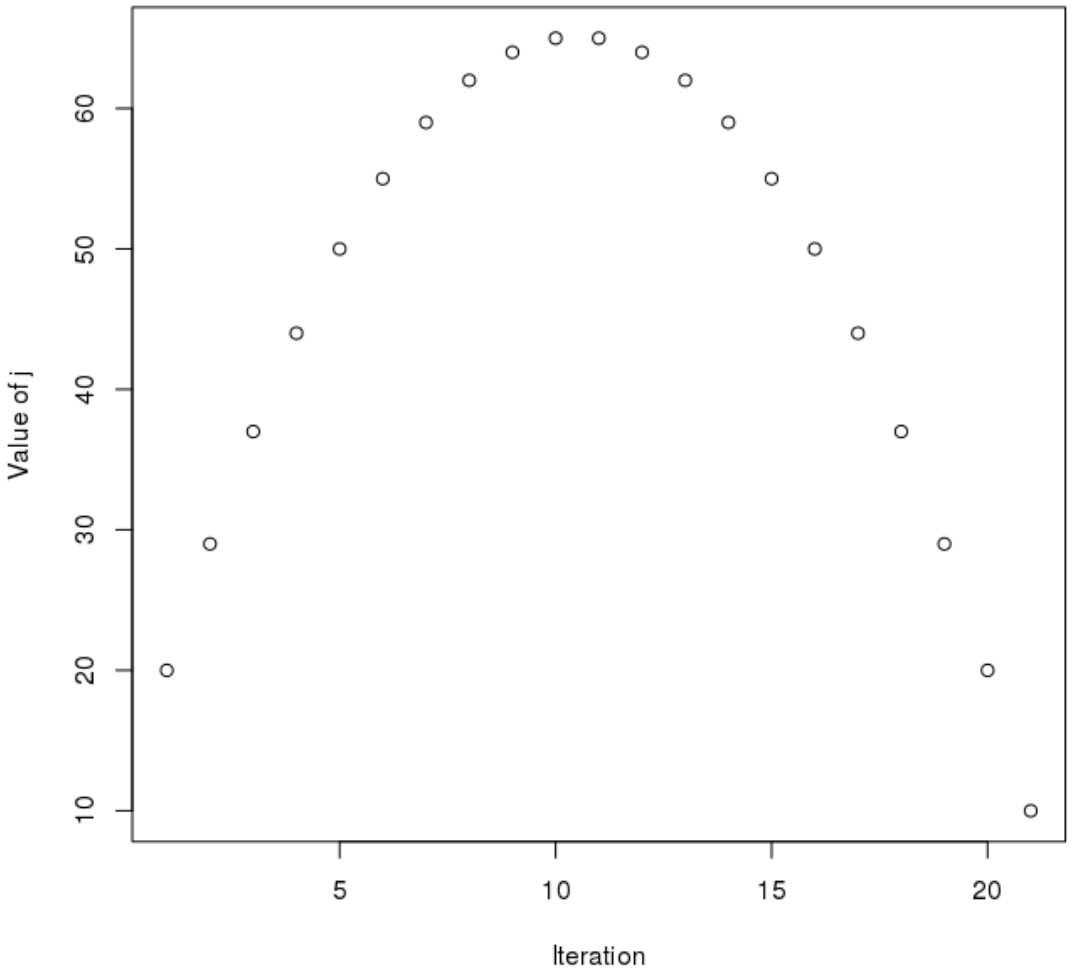
A.I. Barvinok. A polynomial time algorithm for counting integral points in polyhedra when the dimension is fixed, Math. Oper. Res., 1994

Related work: 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;}
```

$A_k, U_k \in \mathbb{R}^{m \times m}$, $b_k, v_k, c_k \in \mathbb{R}^m$, $g_k \in \mathbb{R}$ 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);
```

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

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

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

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

$$\}$$

$$\}$$

---

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

Overview of the whole system

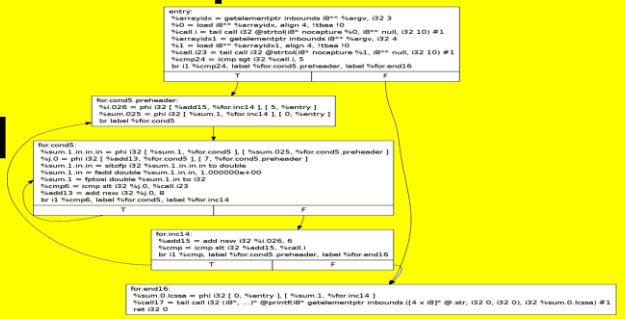
Parallel program

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

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

LLVM

Loop extraction



Affine loop synthesis

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

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

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

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

Algorithm in details

Closed form representation of a loop

- Single affine statement

$$x = Lx + p$$

- Counting function

$$n(x_0)$$

$$x = x_0;$$

$$while\ (c^T x < g)$$

$$x = Ax + b;$$

$$x(i, x_0) = A^i x_0 + \sum_{j=0}^{i-1} A^j \cdot b$$

- Example

$$x(i, x_0) = L(i) \cdot x_0 + p(i)$$

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

$$n(x_0, c, g) := \arg\min_i (c^T \cdot x(i, x_0) \geq g)$$

```
while (c^T x < g) {
    x = [1 0; 1 1]^i x + [0; 0];
}
```

$$x(i, x_0) = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}^i x_0 + \sum_{j=0}^{i-1} \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}^j \begin{pmatrix} 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ i & 1 \end{pmatrix} x_0 + \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$n(x_0) = \left\lceil \frac{m - k_0}{j_0} \right\rceil$$

Algorithm in details

Folding the loops

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

while ($0 \widehat{x} < n/p$){

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

while ($1 \widehat{x} < m$){

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

}

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

}

Algorithm in details

Folding the loops

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

while ($0 \leq x < n/p$){

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

while ($1 \leq x < m$){

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

}

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

}

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

while ($0 \leq x < n/p$){

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

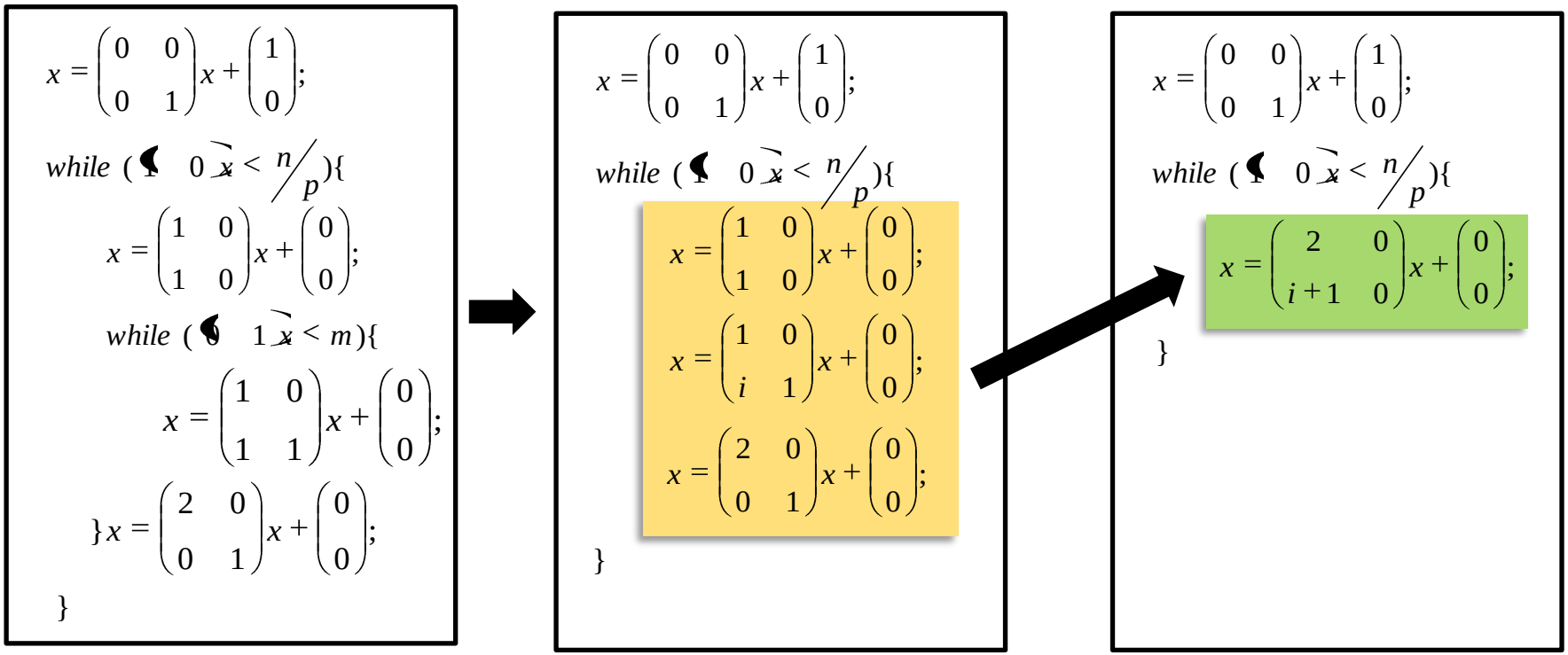
$$x = \begin{pmatrix} 1 & 0 \\ i & 1 \end{pmatrix} x + \begin{pmatrix} 0 \\ 0 \end{pmatrix};$$

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

}

Algorithm in details

Folding the loops



Algorithm in details

Starting conditions

$x_{0,1}$ ⟶ $x = x_0;$

$while\ (c_1^T x < g_1)\{$

$x_{0,2}$ ⟶ $x = A_1 x + b_1;$

$while\ (c_2^T x < g_2)\{$

$x_{0,3}$ ⟶ $x = A_2 x + b_2;$

$while\ (c_3^T x < g_3)\{$

$x = A_3 x + b_3;$

$\}x = U_2 x + v_2;$

$\}x = U_1 x + v_1;$

$\}$

Algorithm in details

Counting the number of iterations

We have:

Algorithm in details

Counting the number of iterations

We have:

- The closed form for each loop:
  - *Single affine statement*
  - *Counting function*
- Starting condition for each loop

Algorithm in details

Counting the number of iterations

We have:

- The closed form for each loop:
  - *Single affine statement*
  - *Counting function*
- Starting condition for each loop

Number of iterations:

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

Algorithm in details

Counting the number of iterations

- The equation computes the precise number of iterations

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

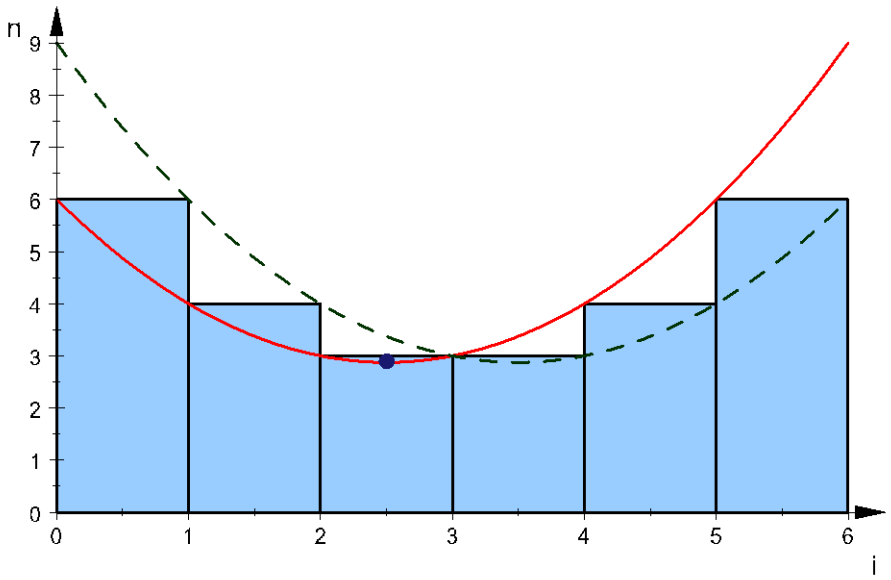
Algorithm in details

Counting the number of iterations

- The equation gives precise number of iterations

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

- But simplification may fail → Sum approximation
  - *Approximate sums by integrals*
    → lower and upper bounds



Solving more general problems

Solving more general problems

- **Multipath loops**

Solving more general problems

- **Multipath loops**
- **Conditional statements**

Solving more general problems

- Multipath loops
- Conditional statements
- Non-affine loops

```
do j=1,lastrow-firstrow+1
   sum = 0.d0
   do k=rowstr(j),rowstr(j+1)-1
      sum = sum + a(k)*p(colidx(k))
   enddo
   w(j) = sum
enddo
```

$$\texttt{lastrow-firstrow+1} = \texttt{row\_size} = \frac{\texttt{na}}{\texttt{nprows}}$$

$$\texttt{rowstr(j+1)-1-rowstr(j)} = u$$

$$N = \frac{\texttt{na} \cdot u}{\texttt{nprows}}$$

Case studies

- **NAS Parallel Benchmarks: EP**

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

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

Case studies

- **NAS Parallel Benchmarks: EP**

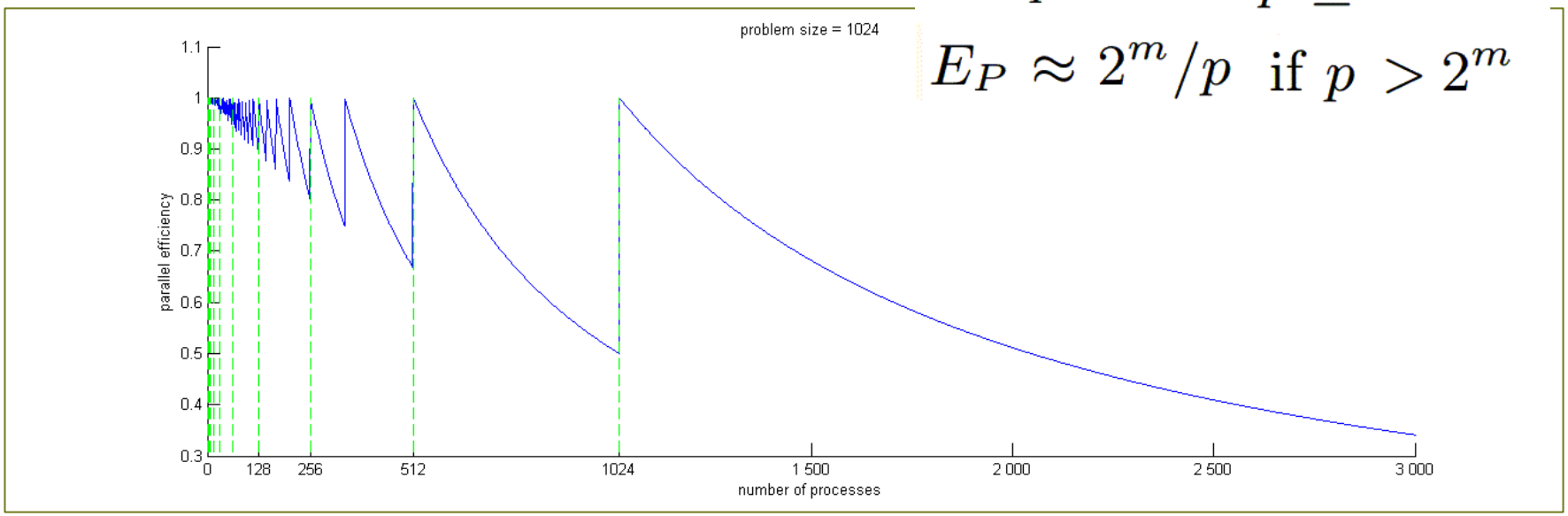
$$N(m,p) = \left\lceil \frac{2^{m-16} \cdot (u + 2^{16})}{p} \right\rceil$$

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

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

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

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



Case studies

CG – conjugate gradient

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

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

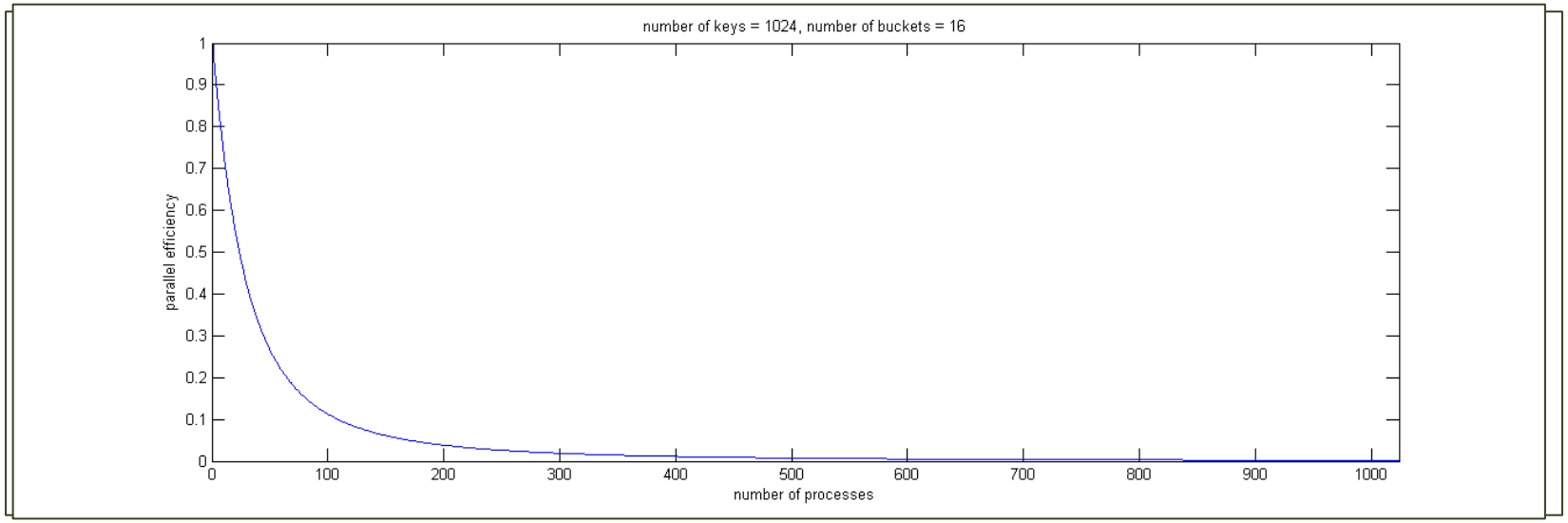
$$E_p = \frac{k_4}{p \left(k_1 \left\lceil \frac{m}{p} \right\rceil + k_2 \sqrt{\left\lceil \frac{m}{p} \right\rceil} + k_3 \log_2 \left(\sqrt{p} \right) \right)}$$

IS – integer sort

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

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

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



What problems are remaining?

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

$$N = \frac{\texttt{na} \cdot u}{\texttt{nprows}}$$

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

    a) Bigger search-space ☹

    b) Bigger trace files ☹

- **Ad-hoc (partial) solution: online machine learning – PEMOGEN**
  - Replace cross-validation with LASSO (regression with L_1 regularizer)

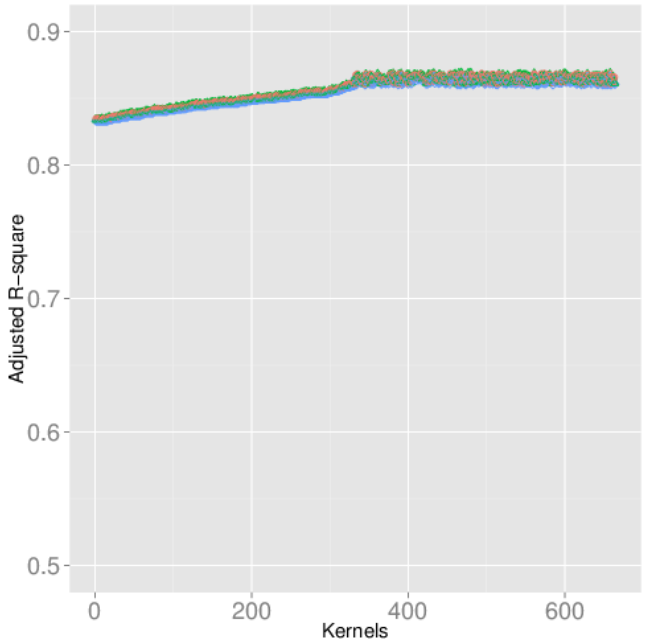
    Much cheaper!
  - Replace LASSO with online LASSO [1]

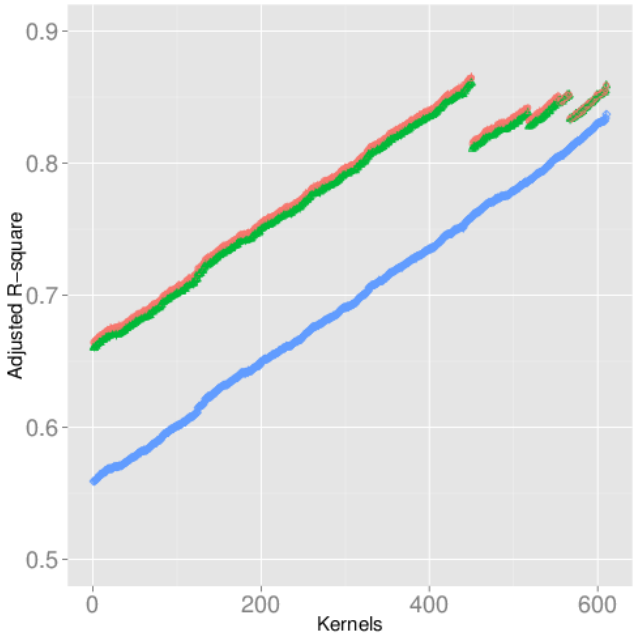
    No traces!

P. Garrigues and L. El Ghaoui. An homotopy algorithm for the Lasso with online observations. NIPS 2008

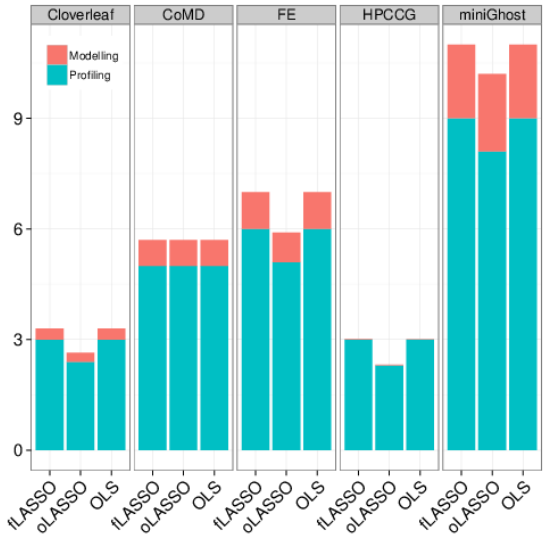
PEMOGEN – static analysis

- **Also integrated into LLVM compiler**
  - Automatic kernel detection and instrumentation (Loop Call Graph)
  - Static dataflow analysis reduces parameter space for each kernel



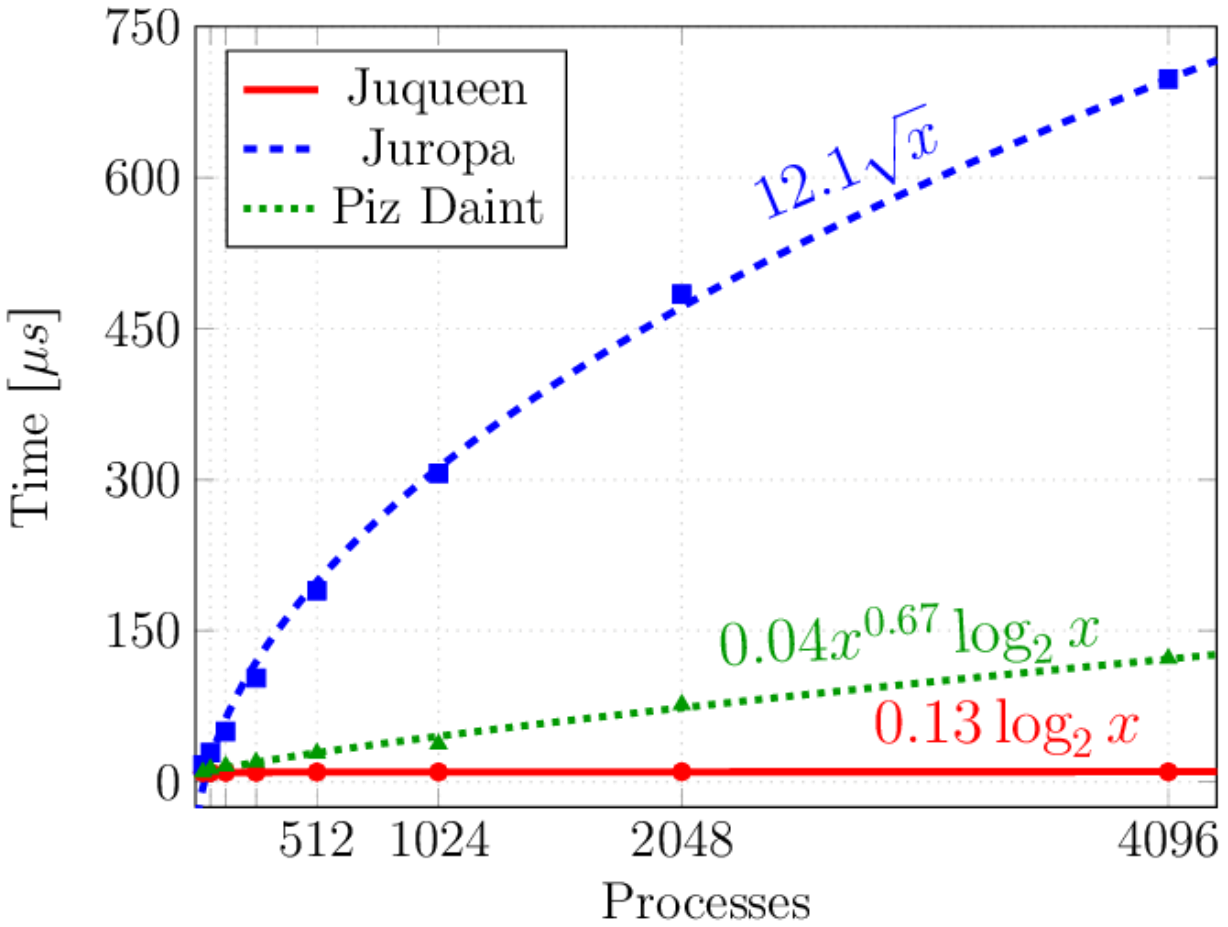


Quality: NAS UA and Mantevo MiniFE



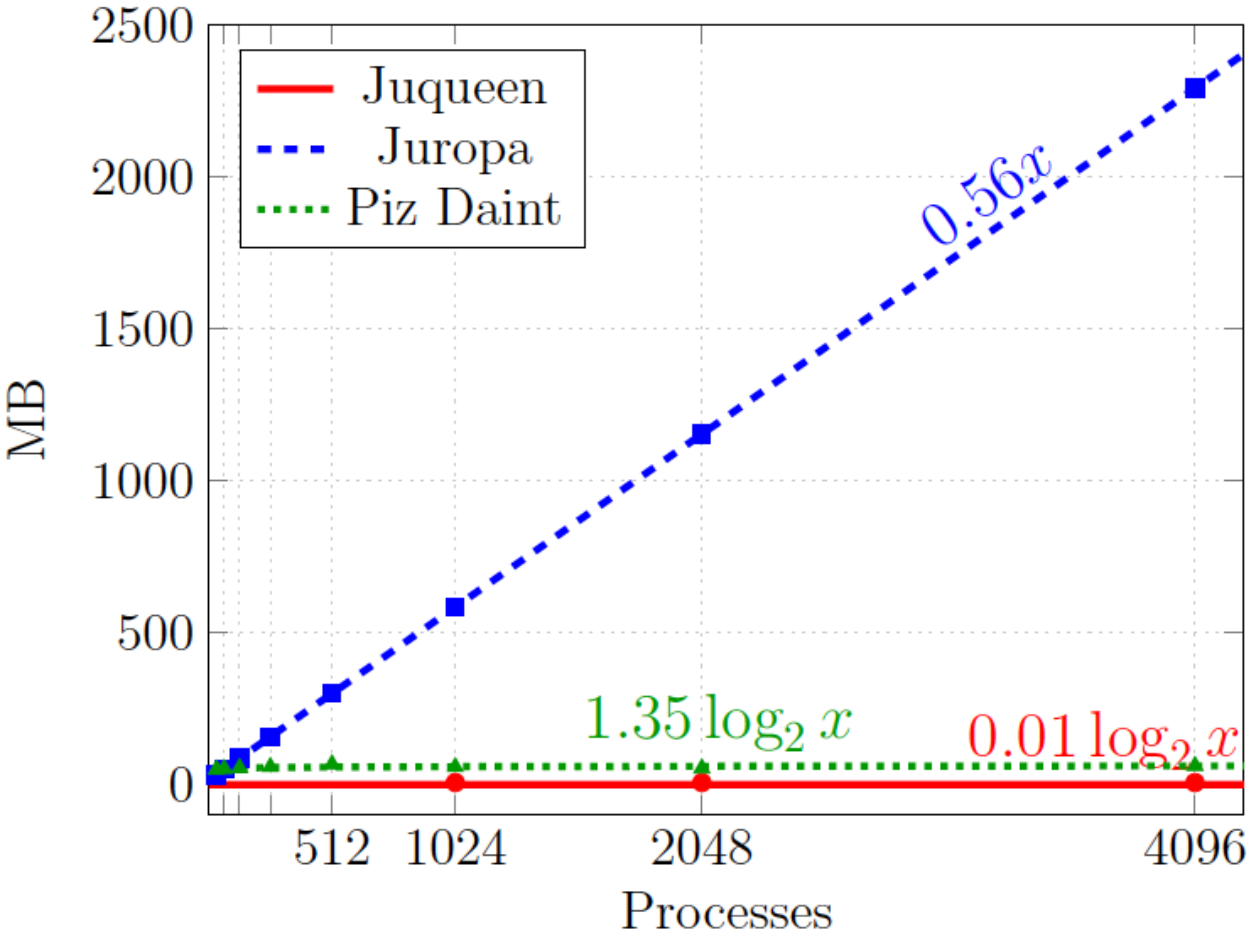
Overhead: Mantevo

Use-case A: automatic testing (Allreduce time)



- **Divergence on Piz**
- **Daint is $O(p^{0.67})$, the highest of all three**

Use-case B: automatic testing (MPI memory size)



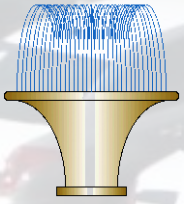
- **Linear memory consumption on Juropa**
- **ParaStation MPI**
- **uses RC over IB**

S. Shudler, A. Calotoiu, T. Hoefler, A. Strube, F. Wolf: Exascaling Your Library: Will Your Implementation Meet Your Expectations?, ICS'15

Performance Analysis 2.0 – Automatic Models

- **Is feasible**
  Still a long way to go …
- **Offers insight**
- **Requires low effort**
- **Improves code coverage**

SPAA 2014



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*

Backup

Counting Arbitrary Affline Loop Nests

- **Why affine loops?**
  - Closed form representation of the loop

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

$$x(i, x_0) = L(i) \cdot x_0 + p(i)$$

$$n(x_0, c, g) = \arg\min_d \left(c^T \cdot x(d, x_0) \geq g\right)$$

Counting Arbitrary Affline Loop Nests

- **Why affine loops?**
  - Closed form representation of the loop

```
x=x0;              // Initial assignment
while(c^T x < g)   // Loop guard
    x=Ax + b;      // Loop update
```

$$x(i, x_0) = L(i) \cdot x_0 + p(i)$$

$$n(x_0, c, g) = \arg\min_d \left(c^T \cdot x(d, x_0) \geq g\right)$$

- **Example**

```
for ( k=j; k < m; k = k + j )
        veryComplicatedOperation(j,k);
```

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

while ($1 \cdot x < m$){

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

}

$$x(i, x_0) = \begin{pmatrix} 1 & 0 \\ i & 1 \end{pmatrix} x_0 + \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$n(x_0) = \left\lceil \frac{m - k_0}{j_0} \right\rceil$$

where $x_0 = \begin{pmatrix} j_0 \\ k_0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} x + \begin{pmatrix} 0 \\ 0 \end{pmatrix}$

Loops

- **Multipath affine loops**

```
x=1;
while(x < n/p + 1) {
  y=x;
  while(y < m) { S1; y=2*y; }
  z=x;
  while(z < m) { S2; z= z + x; }
  x=2*x;
}
```