Parallel Scalability Suite: Profile and visualize parallel applications for scalability

HiPEAC Tutorial 2023

Anderson B. N. da Silva, Vitor R. G. da Silva,
Carlos A. Valderrama, and **Samuel Xavier-de-Souza**

Laboratory of Parallel Architectures for Signal Processing

Toulouse, January 18, 2023

Agenda

1. 10:00 AM: Session I - Introduction to PaScal Suite
   - Performance versus Scalability
   - Introduction to PaScal Analyzer
   - Introduction to PaScal Viewer
2. 11:00 AM: Coffee break
3. 11:30 AM: Session II - Hands-on (bring your laptop computer)
   - Evaluating scalability with PaScal Suite
   - Profiling an OpenMP binary for scalability
   - Instrumenting source code manually
   - Final remarks

Agenda

1. 10:00 AM: Session I - Introduction to PaScal Suite
   - Performance versus Scalability
   - Introduction to PaScal Analyzer
   - Introduction to PaScal Viewer
2. 11:00 AM: Coffee break
3. 11:30 AM: Session II - Hands-on (bring your laptop computer)
   - Evaluating scalability with PaScal Suite
   - Profiling an OpenMP binary for scalability
   - Instrumenting source code manually
   - Final remarks

Universidade Federal do Rio Grande do Norte

Brazil - Rio Grande do Norte - Natal

- **UFRN:** 64-year old university. 40k students, 2.5k profs.
- **IMD/UFRN:** IT Institute, graduate programs, technology park ~90 companies
- **Lab. of Parallel Architectures for Signal Processing**[1]
  - **Staff**: 5 Profs, 4 postdocs, visiting researchers
  - **Students**: 19 grads, half-a-dozen undergrads
  - **Collaboration**: many universities and companies around the globe

[1] http://lapps.imd.ufrn.br

Laboratory of Parallel Architectures for Signal Processing

- **Basic research:** high-performance computing, numerical algorithms, information theory, analysis of cyclostacionary processes, data analytics, and machine learning.
- **Applied research:** high-performance geophysics, fault-tolerant computing for aerospace, parallel GNSS receivers, energy-efficient parallel software, energy-efficient communications, **parallel scalability profiling tools**, computational load balancing, block recursive matrix inversion, software-performance, and software-energy models, correntropy, automatic classification of modulations, and channel and source encodings.

Performance versus Scalability

Objectives of this tutorial

- Present the difference between performance and scalability
- Show existing performance-profile tools are unsuited for profiling scalability
- Emphasize the growing need for scalable parallel programs
- Demonstrate how PaScal Suite can assist developers with scalability analysis
- Teach in practice how to use it to identify scalability hotspots and bottlenecks

Performance versus Scalability

A metaphor for a program

- Performance is related to how fast the task is executed
- Scalability is concerned with how efficiently the task is performed

Performance versus Scalability

Here:

- Performance is the inverse of Time, i.e.
  - Performance $=$ 1/Time
- Efficiency is how much work is performed per resource, i.e.
  - Efficiency $=$ Work x Performance / Resources, or
  - Efficiency $=$ Work / (Time x Resources)

The unity task performed with one resource in the unity time is 100% efficient

🟩

Work = 1
Time = 1
Resource = 1
Efficiency = 1

Performance versus Scalability

The first task is sequential, and the second is parallel. Both perform 100% efficiently.

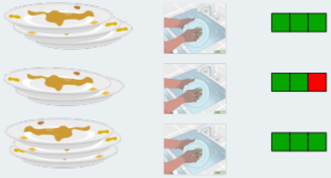
Work = 8
Time = 8
Resource = 1
Efficiency = 1

Work = 8
Time = 4
Resource = 2
Efficiency = W/(TxR) = 1

Note: Resources can scale for the sequential task as well, which would relate to using more sophisticated hardware. Mixing both types of resource scaling would relate to heterogeneous processing.

Performance versus Scalability

Scaling up resources alone often harms efficiency



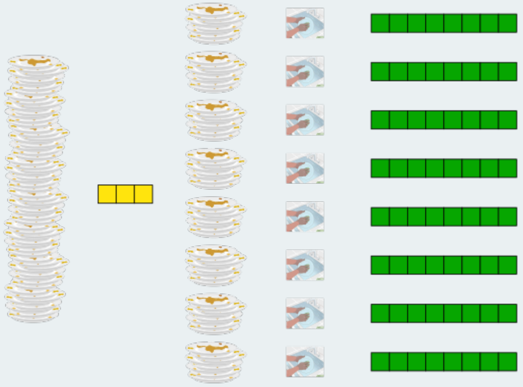
Work = 8
Time = 3
Resource = 3
Efficiency = 8/(3x3) = 0.89

Work = 8
Time = 1
Resource = 20
Efficiency = 8/(1x20) = 0.4

Performance versus Scalability

Larger machines, like supercomputers, require larger problems to sustain efficiency

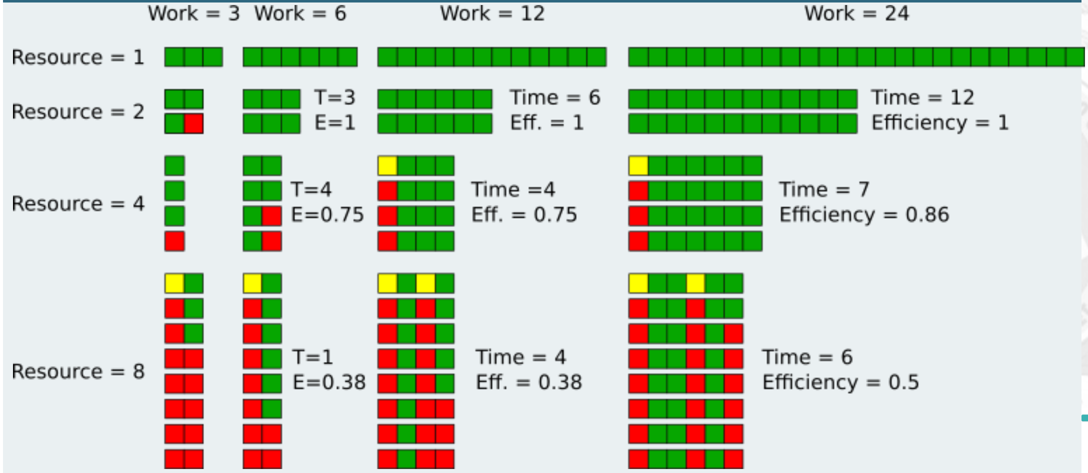
Parallelization overhead, dependencies, serial parts, and so on also harms efficiency



Work = 64
Time = 3 + 8
Resource = 8
Efficiency =
64/(11*8) = 0.73

Performance versus Scalability

Efficiency can vary with resource scaling and problem scaling



Performance versus Scalability

- Scalability analysis requires coarse evaluation of many scaling configurations
- Performance analysis involves a detailed evaluation of a given configuration

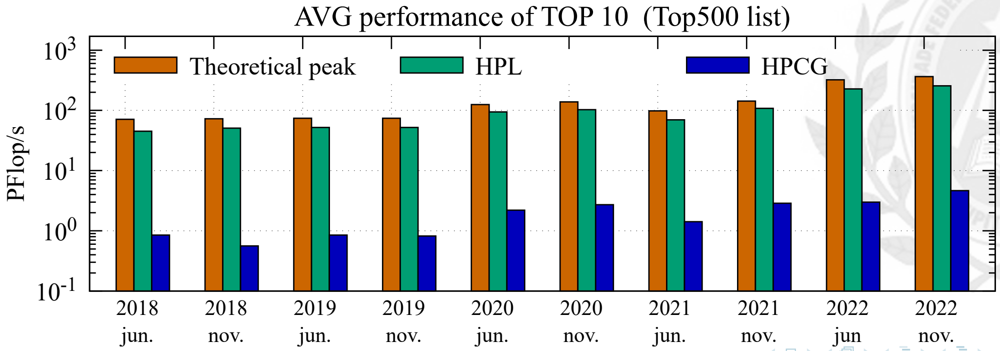
How unsuited are existing performance-profiling tools to assess scalability

- Require the user to run and collect information on many configurations manually
- Provide much detailed data beyond the necessary to evaluate scalability
- Have no native visual artifacts that guide and facilitate the scalability analysis

Despite the shortage of suitable analysis tools, parallel scalability is a growing concern.

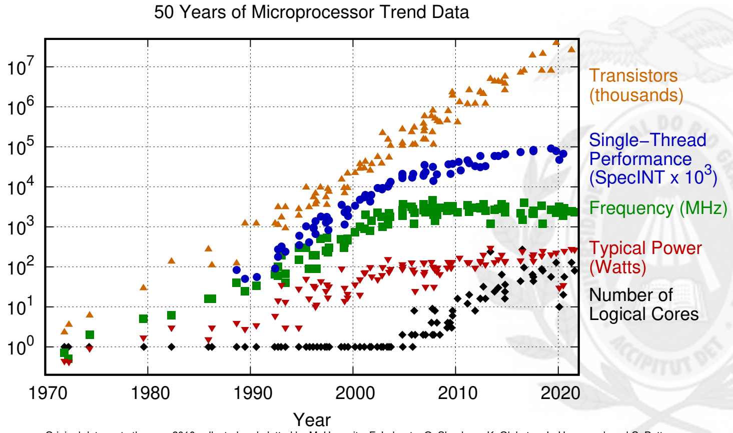
Performance versus Scalability

- Top 500 difference between peak, HPL, and HPGC performance.
- Very large machines are good for solving many problems at the same time but are not efficient to solve very large problems.



Single node scaling soon to face harder scaling problems

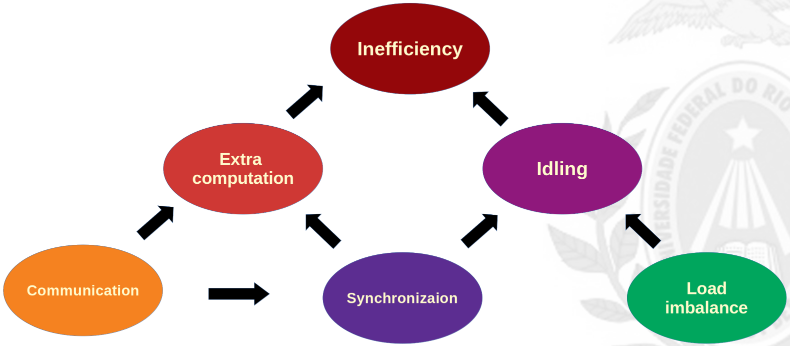
- Moore's law lives
- Single thread big slow down
- MHz plateaued
- Number of cores follows Moore's law



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
Plot and data collected for 2010–2019 by K. Rupp

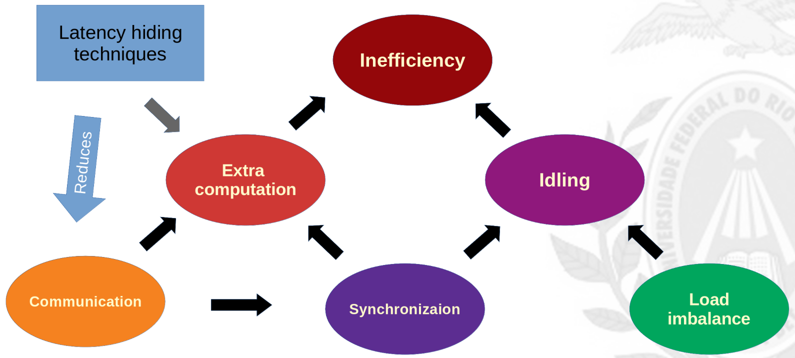
Sources of inefficiency in (parallel) computing

- Due to:
  - Load imbalance
  - Synchronization
  - Communication
- Lead to:
  - Extra computation
  - Idle time
- Optimization approaches
  - avoid costs
- Trade-off approaches
  - Trade costs



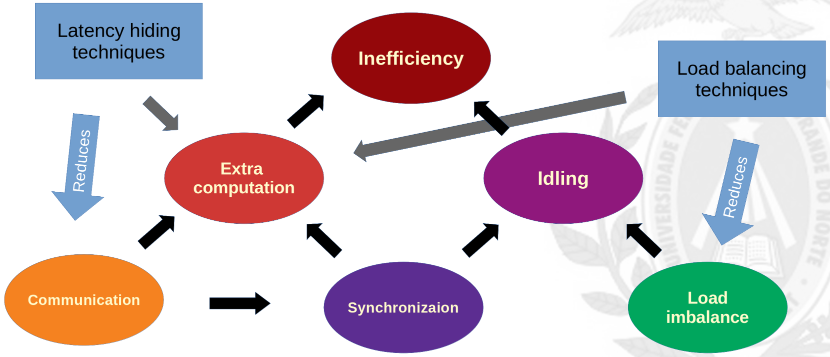
Sources of inefficiency in (parallel) computing

- Due to:
  - Load imbalance
  - Synchronization
  - Communication
- Lead to:
  - Extra computation
  - Idle time
- Optimization approaches
  - avoid costs
- Trade-off approaches
  - Trade costs



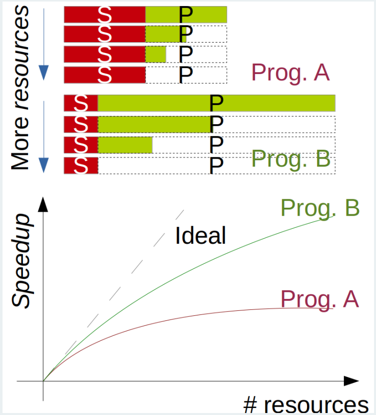
Sources of inefficiency in (parallel) computing

- Due to:
  - Load imbalance
  - Synchronization
  - Communication
- Lead to:
  - Extra computation
  - Idle time
- Optimization approaches
  - avoid costs
- Trade-off approaches
  - Trade costs



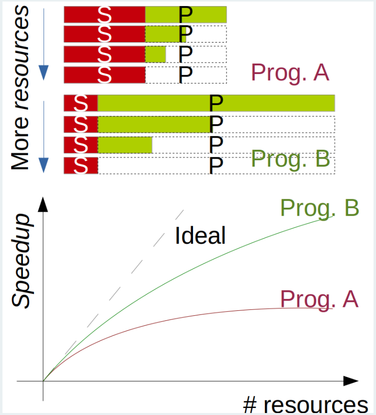
Speedup models

Amdahl's model

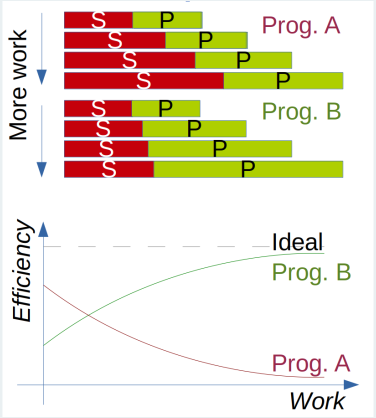


Speedup models

Amdahl's model

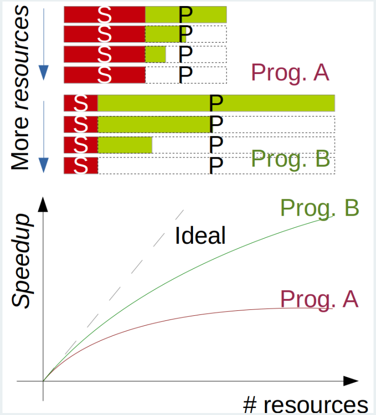


Gustafson's model

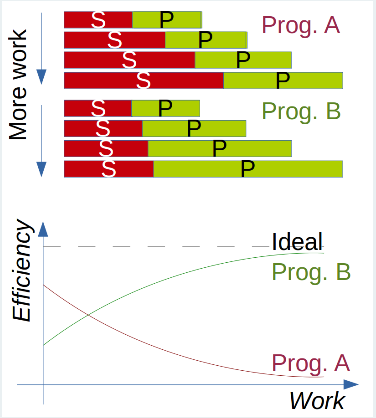


Speedup models

Amdahl's model



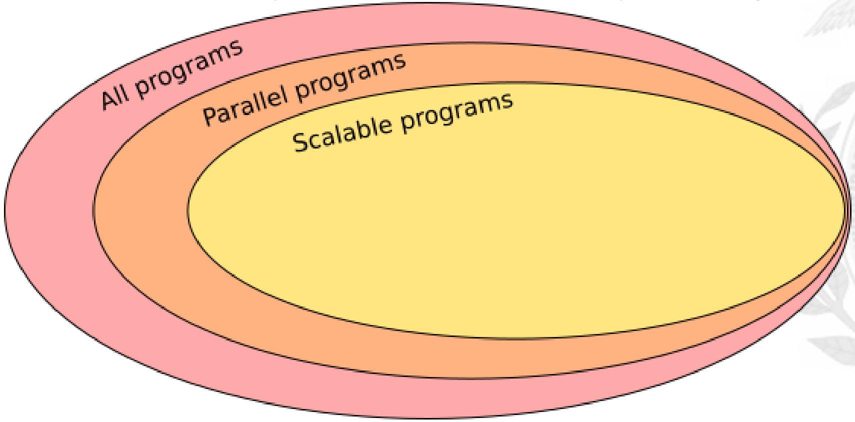
Gustafson's model



- Many other models

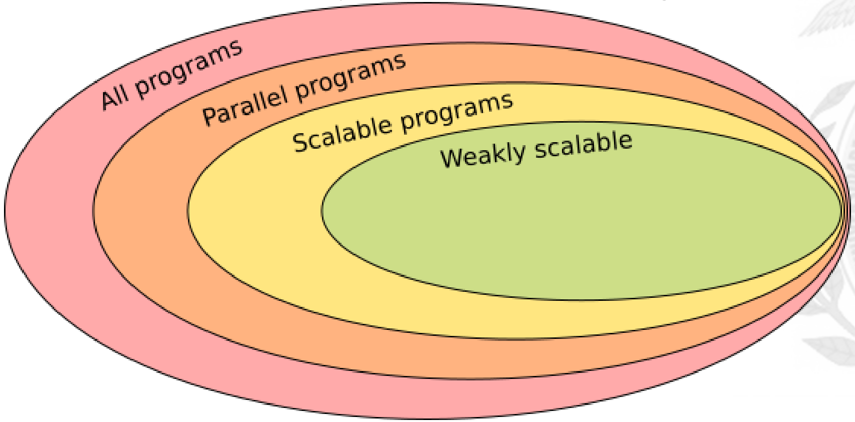
Parallel Efficiency and Scalability Analysis

Scalable: sustain efficiency for more resources with any-factor larger problem



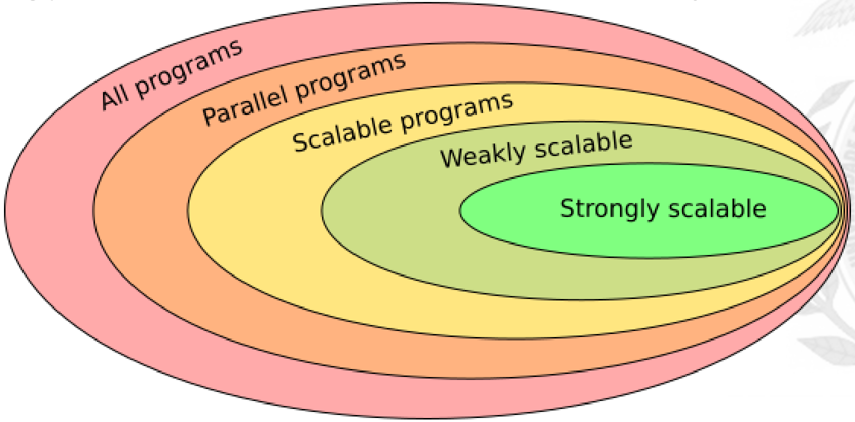
Parallel Efficiency and Scalability Analysis

Weakly scalable: Sustain efficiency even when problem size grows less than resources



Parallel Efficiency and Scalability Analysis

Strongly scalable: Sustain efficiency even without increasing problem size

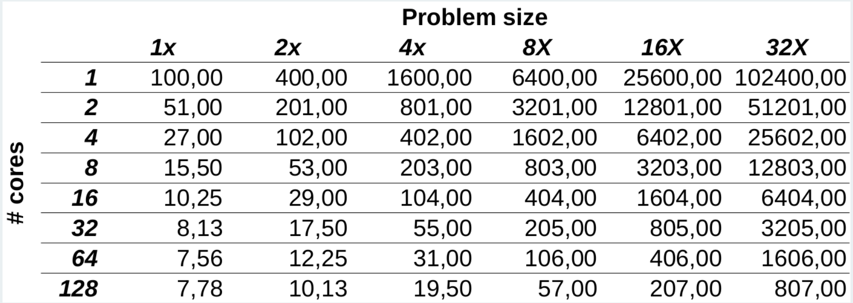


Assessing a program's scalability with tables

$$T_1 = n^2$$

$$T_P = \frac{n^2}{p} + log_2 p$$

Timings



		Problem size					
		1x	**2x**	**4x**	**8X**	**16X**	**32X**
# cores	**1**	100,00	400,00	1600,00	6400,00	25600,00	102400,00
	2	51,00	201,00	801,00	3201,00	12801,00	51201,00
	4	27,00	102,00	402,00	1602,00	6402,00	25602,00
	8	15,50	53,00	203,00	803,00	3203,00	12803,00
	16	10,25	29,00	104,00	404,00	1604,00	6404,00
	32	8,13	17,50	55,00	205,00	805,00	3205,00
	64	7,56	12,25	31,00	106,00	406,00	1606,00
	128	7,78	10,13	19,50	57,00	207,00	807,00

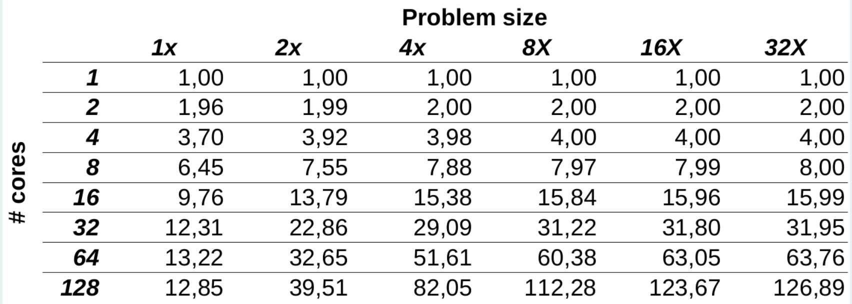
Assessing a program's scalability with tables

$$T_1 = n^2$$

$$T_{\mathrm{P}} = \frac{n^2}{p} + log_2 p$$

$$= \frac{T_1}{T_p}$$

Speedups



# cores	Problem size 1x	2x	4x	8X	16X	32X
1	1,00	1,00	1,00	1,00	1,00	1,00
2	1,96	1,99	2,00	2,00	2,00	2,00
4	3,70	3,92	3,98	4,00	4,00	4,00
8	6,45	7,55	7,88	7,97	7,99	8,00
16	9,76	13,79	15,38	15,84	15,96	15,99
32	12,31	22,86	29,09	31,22	31,80	31,95
64	13,22	32,65	51,61	60,38	63,05	63,76
128	12,85	39,51	82,05	112,28	123,67	126,89

Assessing a program's scalability with tables

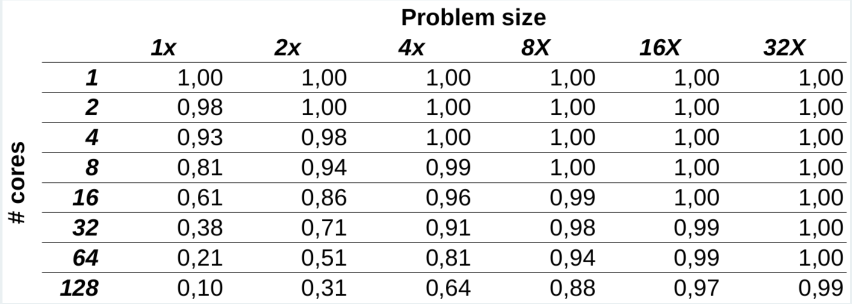
$$T_1 = n^2$$

$$T_{\mathrm{P}} = \frac{n^2}{p} + log_2 p$$

$$= \frac{T_1}{T_p}$$

$$E = \frac{S}{p}$$

Efficiencies



		Problem size					
		1x	***2x***	***4x***	***8X***	***16X***	***32X***
# cores	***1***	1,00	1,00	1,00	1,00	1,00	1,00
	2	0,98	1,00	1,00	1,00	1,00	1,00
	4	0,93	0,98	1,00	1,00	1,00	1,00
	8	0,81	0,94	0,99	1,00	1,00	1,00
	16	0,61	0,86	0,96	0,99	1,00	1,00
	32	0,38	0,71	0,91	0,98	0,99	1,00
	64	0,21	0,51	0,81	0,94	0,99	1,00
	128	0,10	0,31	0,64	0,88	0,97	0,99

Assessing a program's scalability with tables

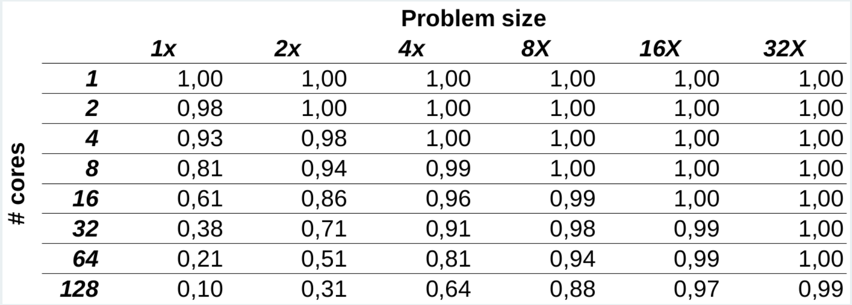
$$T_1 = n^2$$

$$T_{\mathrm{P}} = \frac{n^2}{p} + log_2 p$$

$$= \frac{T_1}{T_p}$$

$$E = \frac{S}{p}$$

Efficiencies



# cores	Problem size 1x	2x	4x	8X	16X	32X
1	1,00	1,00	1,00	1,00	1,00	1,00
2	0,98	1,00	1,00	1,00	1,00	1,00
4	0,93	0,98	1,00	1,00	1,00	1,00
8	0,81	0,94	0,99	1,00	1,00	1,00
16	0,61	0,86	0,96	0,99	1,00	1,00
32	0,38	0,71	0,91	0,98	0,99	1,00
64	0,21	0,51	0,81	0,94	0,99	1,00
128	0,10	0,31	0,64	0,88	0,97	0,99

Scalable? Weaky scalable? Or strongly scalable?

Assessing a program's scalability with tables

$$T_1 = n^2$$

$$T_{\mathrm{P}} = \frac{n^2}{p} + log_2 p$$

$$= \frac{T_1}{T_p}$$

$$E = \frac{S}{p}$$

Efficiencies

# cores	Problem size					
	1x	***2x***	***4x***	***8X***	***16X***	***32X***
1	1,00	1,00	1,00	1,00	1,00	1,00
2	0,98	1,00	1,00	1,00	1,00	1,00
4	0,93	0,98	1,00	1,00	1,00	1,00
8	0,81	0,94	0,99	1,00	1,00	1,00
16	0,61	0,86	0,96	0,99	1,00	1,00
32	0,38	0,71	0,91	0,98	0,99	1,00
64	0,21	0,51	0,81	0,94	0,99	1,00
128	0,10	0,31	0,64	0,88	0,97	0,99

Scalable? Weaky scalable? Or strongly scalable?

Analysis still doable with small core count and quadratically increasing rates

Assessing a program's scalability with tables

Problem size

# cores	1x	2x	3x	4x	5x	6x	7x	8x	9x	10x	11x	12x	13x	14x	15x	16x	17x	18x	19x	20x	21x	22x	23x	24x	25x	26x	27x	28x	29x
1	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	11	21	31	41	51	61	71	81	91	101	111	121	131	141	151	161	171	181	191
2	0,98	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
3	0,95	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
4	0,93	0,98	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
5	0,90	0,97	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
6	0,87	0,96	0,98	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
7	0,84	0,95	0,98	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
8	0,81	0,94	0,97	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
9	0,78	0,93	0,97	0,98	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
10	0,75	0,92	0,96	0,98	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
11	0,72	0,91	0,96	0,98	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
12	0,70	0,90	0,95	0,97	0,98	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
13	0,68	0,89	0,95	0,97	0,98	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
14	0,65	0,88	0,94	0,97	0,98	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
15	0,63	0,87	0,94	0,96	0,98	0,98	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
16	0,61	0,86	0,93	0,96	0,98	0,98	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
17	0,59	0,85	0,93	0,96	0,97	0,98	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
18	0,57	0,84	0,92	0,96	0,97	0,98	0,98	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
19	0,55	0,83	0,92	0,95	0,97	0,98	0,98	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
20	0,54	0,82	0,91	0,95	0,97	0,98	0,98	0,99	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
21	0,52	0,81	0,91	0,95	0,96	0,98	0,98	0,99	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
22	0,50	0,80	0,90	0,94	0,96	0,97	0,98	0,98	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
23	0,49	0,79	0,90	0,94	0,96	0,97	0,98	0,98	0,99	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
24	0,48	0,78	0,89	0,94	0,96	0,97	0,98	0,98	0,99	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
25	0,46	0,78	0,89	0,93	0,96	0,97	0,98	0,98	0,99	0,99	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
26	0,45	0,77	0,88	0,93	0,95	0,97	0,98	0,98	0,99	0,99	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
27	0,44	0,76	0,88	0,93	0,95	0,97	0,97	0,98	0,98	0,99	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
28	0,43	0,75	0,87	0,92	0,95	0,96	0,97	0,98	0,98	0,99	0,99	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
29	0,42	0,74	0,86	0,92	0,95	0,96	0,97	0,98	0,98	0,99	0,99	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
30	0,40	0,73	0,86	0,92	0,94	0,96	0,97	0,98	0,98	0,99	0,99	0,99	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
31	0,39	0,72	0,85	0,91	0,94	0,96	0,97	0,98	0,98	0,98	0,99	0,99	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
32	0,38	0,71	0,85	0,91	0,94	0,96	0,97	0,98	0,98	0,98	0,99	0,99	0,99	0,99	0,99	0,99	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00

1/4 of the table of the previous slide with linear rates

Assessing a program's scalability with tables

- Building tables manually is tedious and time-consuming
  - Remember: performance-profiling tools do not build these tables. They generate data (way too much) for a single table cell.
  - The scalability-caring developer is left alone to run, collect and, somehow, find a way to visualize (scalability) hot spots and bottlenecks
- Tables hide scalability trends
- Line or bar plots are unsuited for visualizing this type of data. Cloaks the trends.
- What about profiling for scalability?
  - Requires a table for each code region of interest
  - Manually: more tedious and time-consuming
  - Using profiling tools: even more tedious and time-consuming

Introduction to PaScal Analyzer

PaScal Analyzer

A tool and library from the Parallel Scalability Suite toolset to control, measure, and collect information from multiple executions of a parallel application automatically.

Note: Currently suited for shared-memory environments.

Introduction to PaScal Analyzer

Objectives of the Parallel Scalability Analyzer

- Read the definitions of multiple configurations of a parallel program
- Measure data from those configurations automatically
- Collect the measured data in a non-intrusive way
- Organize the collected data for future visual analysis

Introduction to PaScal Analyzer

Objectives of the Parallel Scalability Analyzer

- Read the definitions of multiple configurations of a parallel program
- Measure data from those configurations automatically
- Collect the measured data in a non-intrusive way
- Organize the collected data for future visual analysis

Which data?

- The PaScal Analyzer can deal with timing, power, and performance counters
- In this tutorial, we will be only concerned with timing.

Requirements to use PaScal Analyzer

- Linux operating system
- For profiling:
  - applications written in C and C++
  - OpenMP or PThreads (no MPI yet)
  - profiling binary: GNU C Compiler

Instrumenting source code

- PThreads code and OpenMP code can be manually instrumented
  - Include the `pascalops.h` header file in our source files.
  - Use `pascal_start` to mark the beginning of a region for measurement. Use the single integer argument as the identification of the region.
  - Use `pascal_stop` to mark the end of the region identified by the single integer argument.

Instrumenting source code

Example of instrumentation of a OpenMP C program

```
#include "pascalops.h"
...
int main()
{
    ...
        pascal_start(1);
        #pragma omp parallel
        {
                ...
        }
        pascal_stop(1);
        ...
}
```

Compiling our code with the PaScalOps library

- After installation, the pascalops library should be visible to GCC
- Simply add the flag `-lmpascalops` for compiling manually instrumented sources.

Example compilation

```
$ gcc myapp.c -fopenmp -lmpascalops
```

Using PaScal Analyzer

Usage

```
pascalanalyzer [-h] [-c CORES] [-f FREQUENCIES] [-i INPUTS] [-o OUTPUT] [-r RPTS] [-g]
               [-t {aut,AUT,man,MAN}] [-a LEVEL] [--mpi RUNTIME] [--ragt TYPE] [-v VERBOSE]
               [--dcrs] [--dhpt] [--domp] [--dout] [--govr GOVERNOR] [--idtm TIME]
               [--fgpe EVENT] [--fgps EVENT] [--rple {sysfs,perf}]
               [--rpls {sysfs,scontrol,perf}] [--ipmi SERVER USER PASSWORD] [--modl NPTS MODE]
               [--prcs] [--lpcs] [--imnt PATH EXTENSIONS]
               [application]
```

Examples

```
$ pascalanalyzer ./myapp --inst aut --idtm 5 --cors 1:4 --ipts 10,20,30,40  --verb INFO
$ pascalanalyzer ./myapp -t man -g -r 5 -c 1,32 -i 10,320 -v 2 -o myoutput.json
```

Introduction to PaScal Analyzer

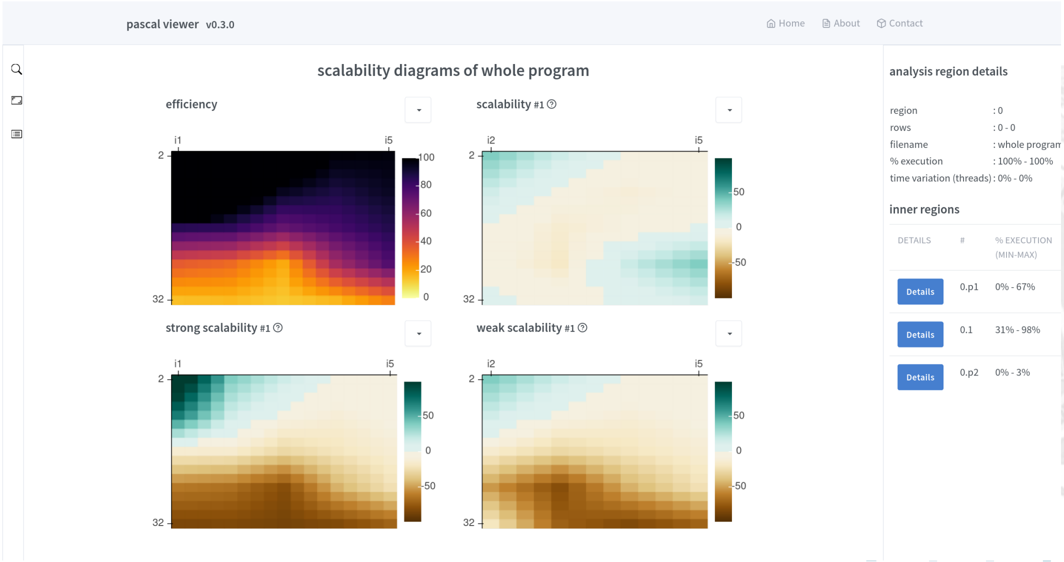
PaScal Viewer

A web tool from the Parallel Scalability Suite that loads PaScal Analyzer's outputs and presents four diagrams to assist the user in understanding the program's scaling behavior with visual elements that emphasize scalability hotspots and bottlenecks.

Introduction to PaScal Viewer

Objectives of the Parallel Scalability Viewer

- Understand PaScal Analyzer's outputs
- Generate heatmap-like diagrams for efficiency, scalability, weak scalability, and strong scalability
- Provide support for hierarchical regions
- Allow side-by-side comparisons of the parent and sibling regions among each other and the whole program.



pascal viewer v0.3.0

Home About Contact

scalability diagrams of whole program

analysis region details

region	: 0
rows	: 0 - 0
filename	: whole program
% execution	: 100% - 100%
time variation (threads)	: 0% - 0%

inner regions

DETAILS	#	% EXECUTION (MIN-MAX)
Details	0.p1	0% - 67%
Details	0.1	31% - 98%
Details	0.p2	0% - 3%

Agenda

1. 10:00 AM: Session I - Introduction to PaScal Suite
   - Performance versus Scalability
   - Introduction to PaScal Analyzer
   - Introduction to PaScal Viewer
2. 11:00 AM: Coffee break
3. 11:30 AM: Session II - Hands-on (bring your laptop computer)
   - Evaluating scalability with PaScal Suite
   - Profiling an OpenMP binary for scalability
   - Instrumenting source code manually
   - Final remarks

Agenda

1. 10:00 AM: Session I - Introduction to PaScal Suite
   - Performance versus Scalability
   - Introduction to PaScal Analyzer
   - Introduction to PaScal Viewer
2. 11:00 AM: Coffee break
3. 11:30 AM: Session II - Hands-on (bring your laptop computer)
   - Evaluating scalability with PaScal Suite
   - Profiling an OpenMP binary for scalability
   - Instrumenting source code manually
   - Final remarks

Setting up your laptop

- This is a hands-on session, but if you do not have laptop with you, hang around to see PaScal Suite in action.
- We will use NPAD, UFRN's HPC cluster, to do the tutorial exercises.
- NPAD's team has reserved 3 nodes with 32 cores each for this tutorial.
- Access the link:
  `http://npad.ufrn.br/npad/Tutorial`
  and follow the steps to create your tutorial account.
- In order to login, use the following command with your newly created credentials:
  `ssh -p4422 tutorial[YOUR NUMBER]@sc2.npad.ufrn.br`

Download the tutorial files and Install the PaScal Analyzer

- Once you have logged in, download the files you are going to use by executing the following command:

```
git clone git@gitlab.com:lappsufrn/pascal-suite-tutorial.git
```

- A public version of the Analyzer is available at https://gitlab.com/lappsufrn/pascal-suite-tutorial
- For this tutorial, follow these steps to install it locally in your tutorial account.

```
wget -c https://bit.ly/3kqkZiw
unzip 3kqkZiw
cd pascal-releases-master/
source env.sh
```

- Verify your installation is working by executing:

```
pascalanalyzer -h
```

Compile and evaluate the scalability of our application

- The application we will use solves the linear system $Ax = b$ using LU decomposition with in-place storage.
- It takes one argument: the order of the A matrix.
- To compile it, simply change to the tutorial folder and use Make:

```
cd pascal-suite-tutorial/
make
```

- To make our first scalability analysis, let's use a quadratic rate for scaling the number of cores and problem size.
- We will establish the number of elements in the A matrix as our problem size, starting with size 250k (matrix order 500)

Compile and evaluate the scalability of our application

- Use the following command to run your first analysis job:
  ```
  sbatch pascalanalyzer_job.sh
  ```
- Inspect the status of your job with:
  ```
  squeue -u $USER
  ```
- Now, copy the output file from NPAD to your machine
  ```
  scp -P 4422 tutorial[YOUR NUMBER]@sc2.npad.ufrn.br\
  :pascal-suite-tutorial/*.json .
  ```
- And load it in the PaScal Viewer webtool at the link:
  https://pascalsuite.imd.ufrn.br

Profiling an OpenMP binary for scalability

- The previous analysis involved the whole program and the parallel regions of the code because the Analyzer identifies the OpenMP parallel regions in the binary as inner regions.
- To analyze the gaps between regions and generate a complete scalability profile, use the -g option.
- The -g option considers these parts of the program as regions of interest:
  - the region between the start of the program and the first identified region;
  - the gaps between identified regions; and
  - the region between the last identified region and the end of the program.
- Add the -g option in the `pascalanalyzer_job.sh` script, run the job again, and load the resulting .json file on the webtool again.

Instrumenting source code manually

- Include the header `pascalops.h` in the source `LU_rand_omp.cpp`
- Instrument the region(s) of your interest with:
  - `pascal_start(id)`, and
  - `pascal_stop(id)`,
  - where `id` is an integer that uniquely identifies a region.
- Then, include the flag `-lmpascalops` in the compilation line in the `Makefile` file to link our program with the PaScal library.
- Change the `-t` option in the job script from `aut` to `man`.
- Compile, submit the job, and visualize the result with PaScal Viewer again.

Final remarks

- **Disclaimer**: PaScal Suite is a prototype and should be used with no guarantees or liability to the authors.
- You can contribute by reporting bugs but we are also seeking partners to make further developments.
- Wish list of features:
    - MPI support
    - Reduce analysis costs by building scalability models from samples of the scaling space
    - Reduce analysis cost by using performance models to avoid running the application until termination
    - Provide ML-assisted hints to help developers find root causes of scalability bottlenecks

That's all folks!

`http://dca.ufrn.br/~samuel`