`bencherl`: A scalability benchmark suite for Erlang/OTP

Stavros Aronis[1] Nikolaos Papaspyrou[2] Katerina Roukounaki[2]
Konstantinos Sagonas[1,2] Yiannis Tsiouris[2] Ioannis Venetis[2]

[1]Department of Information Technology, Uppsala University, Sweden

[2]School of Electrical and Computer Engineering, National Technical University of Athens, Greece

Erlang Workshop 2012, Copenhagen

Motivation

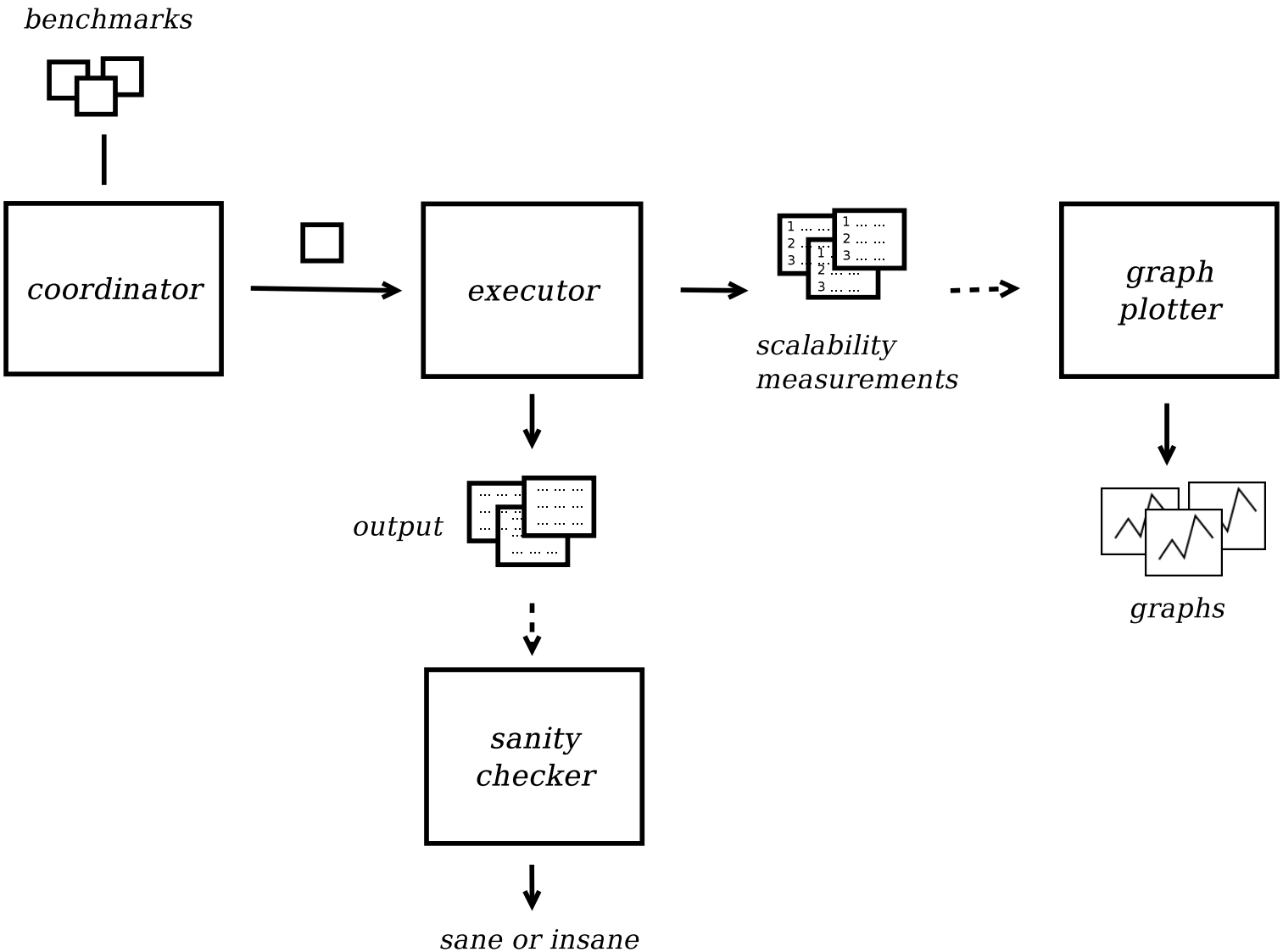
Frustrated Erlang programmer

I thought my Erlang program was **100% parallelizable**, but when I made it parallel and ran it on a machine with **N CPU cores**, I got a **speedup** that was **much lower than N**. Why?

bencherl

- runs benchmarks in an automatic way
- configurable extendable examines can examine can benchmark public available tool repository examines 5 dimensions

Architecture



Coordinator

The module that coordinates everything during a `bencherl` run.

- Determines the **benchmarks** that should be executed
- Determines the **runtime environments**, where each benchmark should be executed
- Sets up each runtime environment before a benchmark is executed in it
- Prepares instruction files for the executor
- Performs any benchmark-specific pre- and post-execution actions

Executor

The module that executes a particular benchmark in a particular runtime environment.

- Receives detailed instructions from the executor about what to do
- Starts any necessary Erlang slave nodes
- Executes the benchmark in a new process
- Stops the Erlang slave nodes it started
- Makes sure that the output that the benchmark produced during its execution is written in an output file
- Makes sure that the measurements that are collected during the execution of the benchmark are written in a measument file
  - Uses `erlang:now/0` and `timer:diff/2`

Sanity checker

The module that checks whether all executions of a particular benchmark produced the same output.

- Runs after a benchmark has executed in all desired runtime environments
- Examines the output that the benchmark produced in all runtime environments
- Decides whether the benchmark was successfully executed in all runtime environments
- Is based on the assumption that if a benchmark produces any output during its execution, then this output should be the same accross all runtime environments, where the benchmark was executed
  - Uses `diff`

Graph plotter

The module that plots scalability graps based on the collected measurements.

- Runs after a benchmark has executed in all desired runtime environments
- Processes the measurements that were collected during the execution of the benchmark
- Plots a set of scalability graphs
- Uses `Gnuplot`

Scalability graphs

Benchmarks

`bencherl` comes with an initial collection of benchmarks.

SYNTHETIC		REAL-WORLD
bang	orbit_int	dialyzer_bench
big	parallel	scalaris_bench
ehb	pcmark	
ets_test	ran	
genstress	serialmsg	
mbrot	timer_wheel	

This collection can be enhanced in a number of simple steps.

Step 1: Add in `bencherl` everything that the benchmark needs for its execution.

- The sources of the Erlang application that it benchmarks
- Any scripts to run before or after its execution
- Any data that it needs for its execution

Step 2: Write the handler for the benchmark.

A benchmark handler is a standard Erlang module that exports two functions.

A function that returns the different argument sets that should be used for running a specific version of the benchmark:

```
bench_args(Vrsn, Conf) -> Args
  when
    Vrsn :: 'short' | 'intermediate' | 'long',
    Conf :: [{Key :: atom(), Val :: term()}, ...],
    Args :: [[term()]].
```

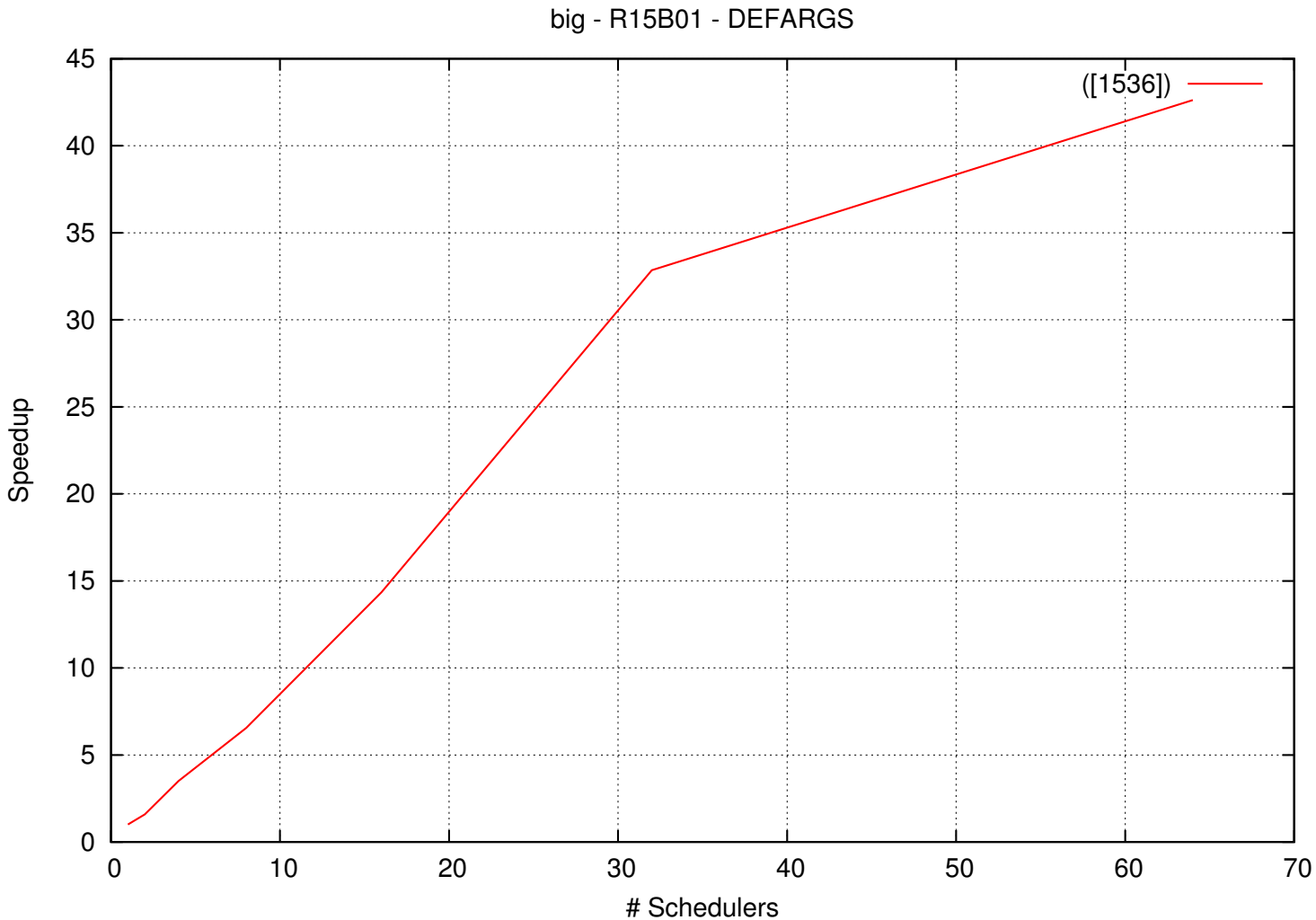
A function that uses a specific argument set, specific Erlang slave nodes and specific configuration settings to run the benchmark:

```
run(Args, Slaves, Conf) -> 'ok' | {'error', Reason}
  when
    Args   :: [term()],
    Slaves :: [node()],
    Conf   :: [{Key :: atom(), Val :: term()}, ...],
    Reason :: term().
```

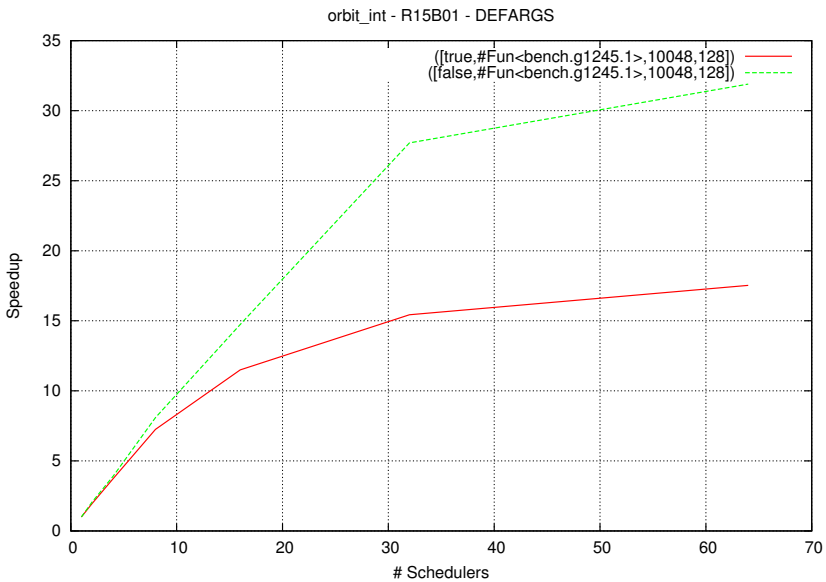
A benchmark handler example

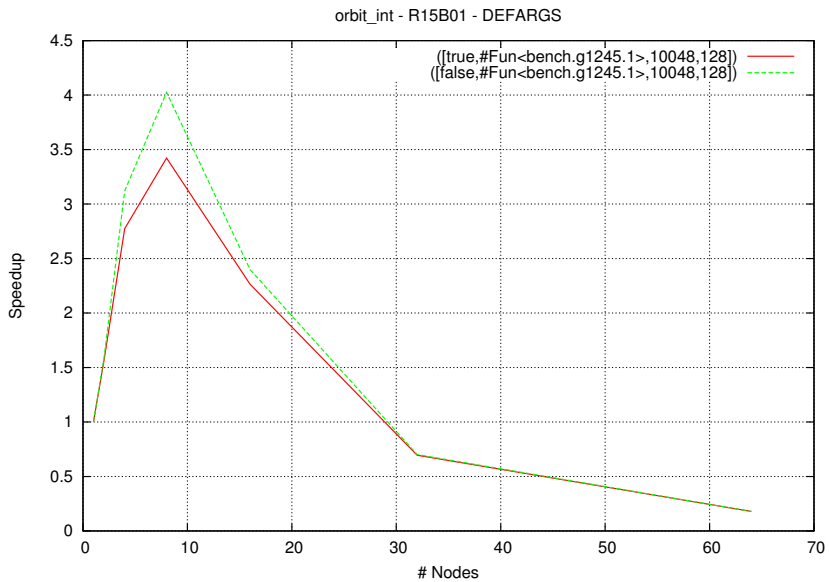
```
-module(scalaris_bench).

-include_lib("kernel/include/inet.hrl").

-export([bench_args/2, run/3]).

bench_args(Version, Conf) ->
    {_,Cores} = lists:keyfind(number_of_cores, 1, Conf),
[F1, F2, F3] = case Version of
short -> [1, 1, 0.5];
intermediate -> [1, 8, 0.5];
long -> [1, 16, 0.5]
end,
[[T,I,V] || T <- [F1 * Cores], I <- [F2 * Cores], V <- [trunc(F3 * Cores)]].

run([T,I,V|_], _, _) ->
{ok, N} = inet:gethostname(),
{ok, #hostent{h_name=H}}=inet:gethostbyname(N),
Node = "firstnode@" ++ H,
rpc:block_call(list_to_atom(Node), api_vm, add_nodes, [V]),
io:format("~p~n", [rpc:block_call(list_to_atom(Node), bench, quorum_read, [T,I])]),
ok.
```

Experience #1: Some benchmarks scale well.

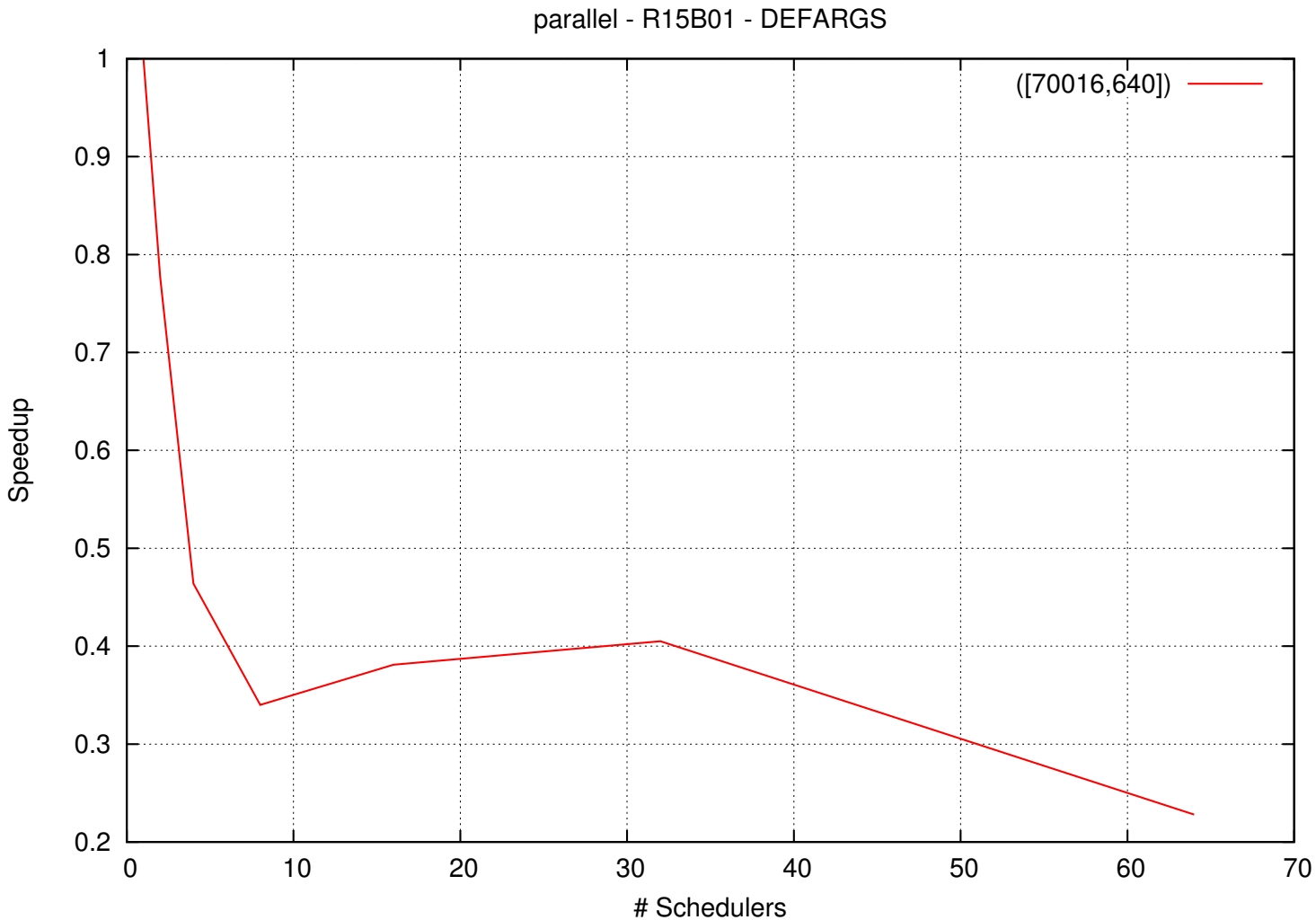


Experience #2: Some benchmarks scale do not scale as well on more than one nodes.

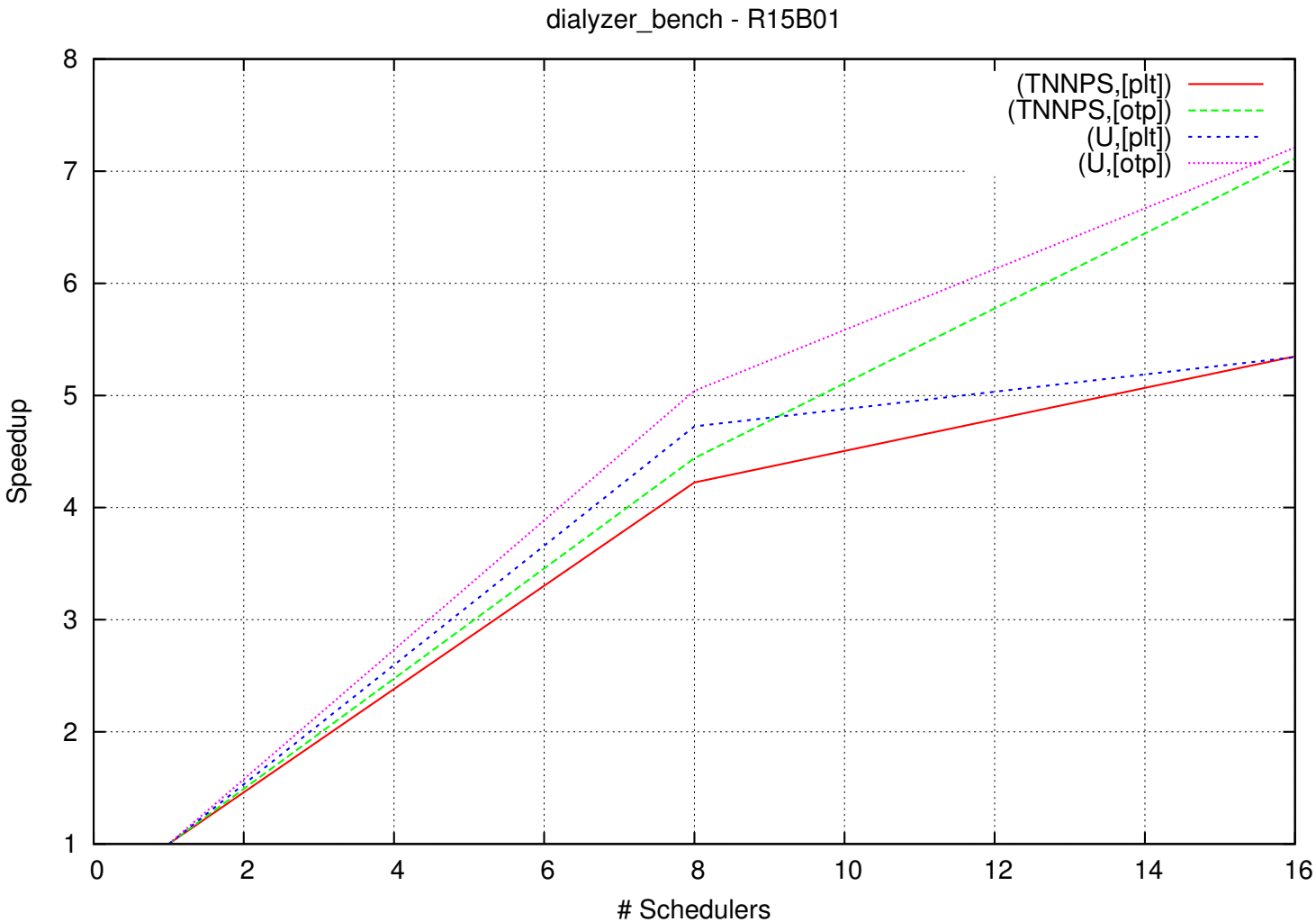




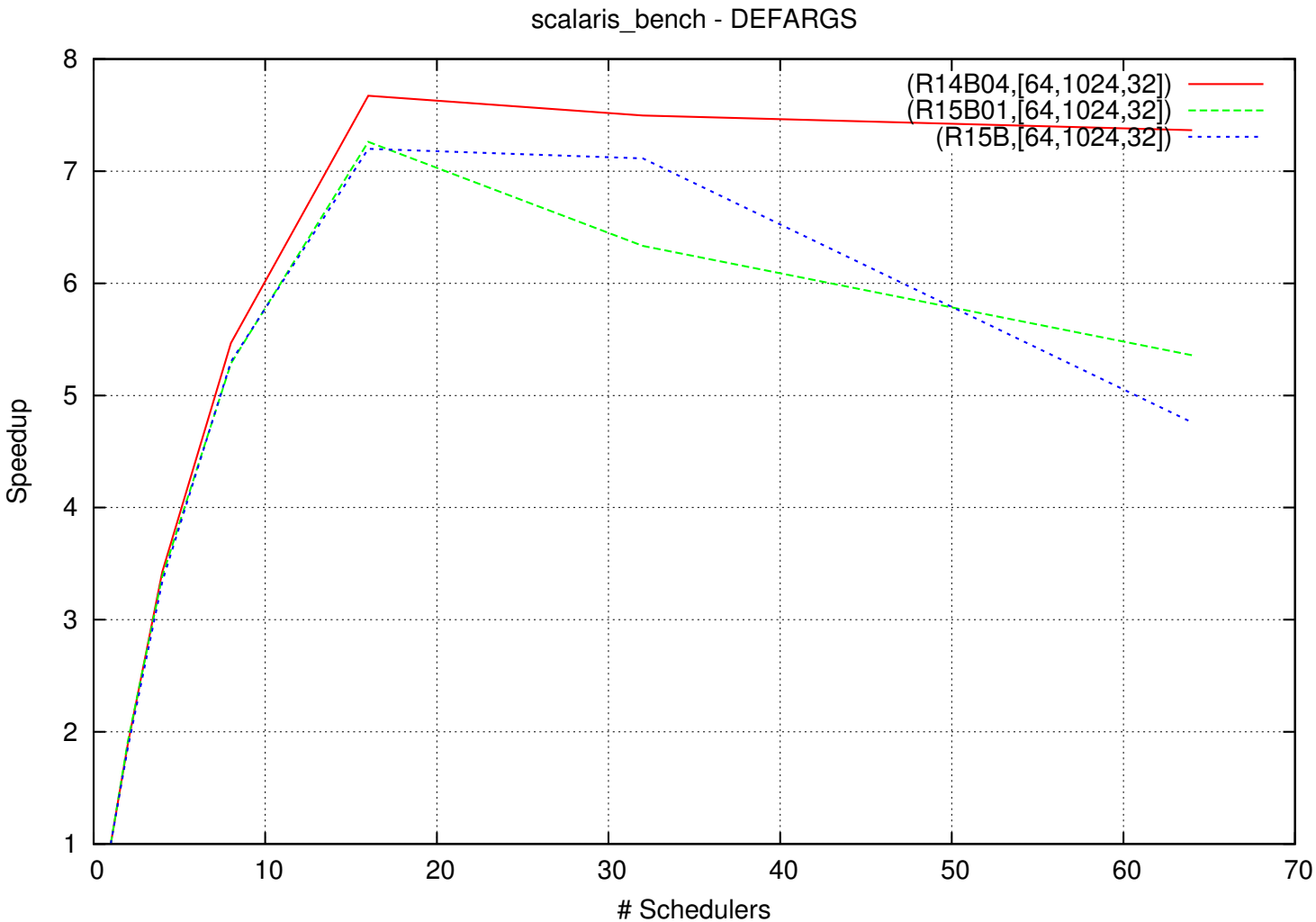
Experience #3: Some benchmarks do not scale.



Experience #4: Some benchmarks scale better with specific runtime options.



Experience #5: Some benchmarks scale better with specific Erlang/OTP releases.



Conclusion

Future work

Thank you!