Predicting the Energy-Consumption of MPI Applications at Scale Using Only a Single Node

F. *Christian* Heinrich,
Arnaud Legrand, Tom Cornebize, Augustin Degomme,
Alexandra Carpen-Amarie, Sascha Hunold,
Anne-Cécile Orgerie and Martin Quinson

INRIA, University of Grenoble, University of Rennes, University of Vienna

September 6, 2017

Motivation

- You want to run your application on a new or modified system
- Questions you will ask yourself: How is my code going to perform? What should I change?
- Energy is expensive; how much does my application consume?
- Programmers will need access to that machine to optimize your application; this is expensive and a waste of resources

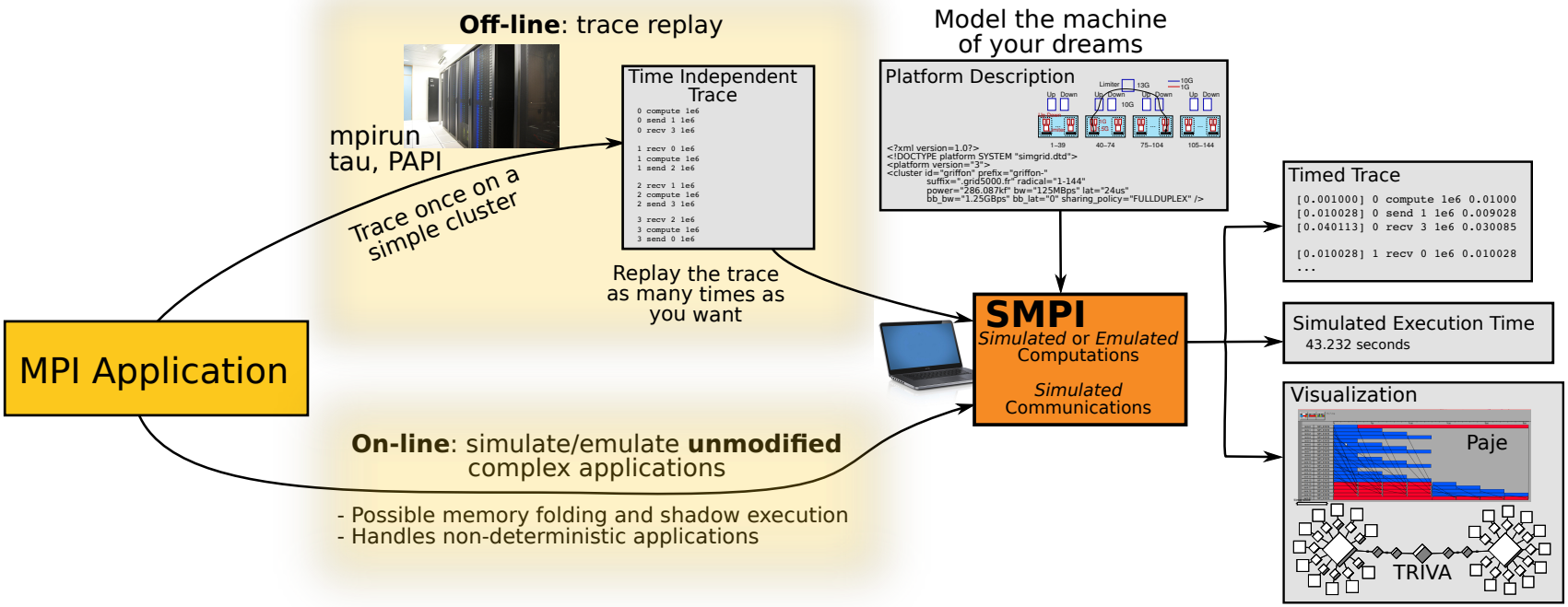
Evaluate the system

- Our approach allows you to evaluate the performance (and energy consumption) of an MPI-based application using only a single node.

Quick overview:

1. Obtain a platform model
   - How does MPI perform on this platform?
2. Run the application on one node, all cores
   - We're interested in the interference of MPI processes (memory contention; contention on L1-L3 caches)
   - Measure the energy consumption
3. Run the application on one node, one core
   - Measure the energy consumption
4. Feed measurements / platform model into simulator

Simulation with SimGrid

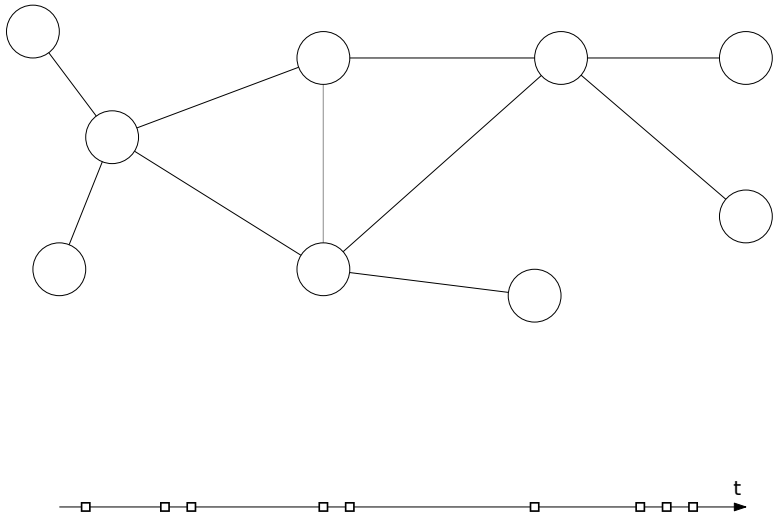


- SimGrid: an open source framework for building simulators.
- Key features: Hybrid LogGP / fluid network models; accounts for contention and (possibly complex) topologies!
- Offline / online simulation of *real (unmodified) applications*

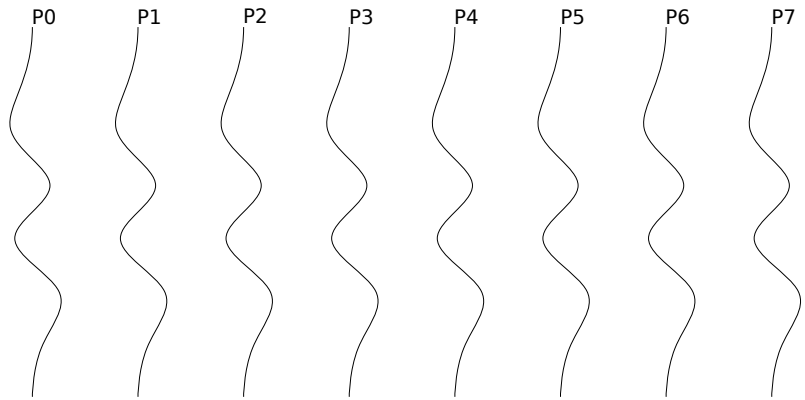
Offline / Online

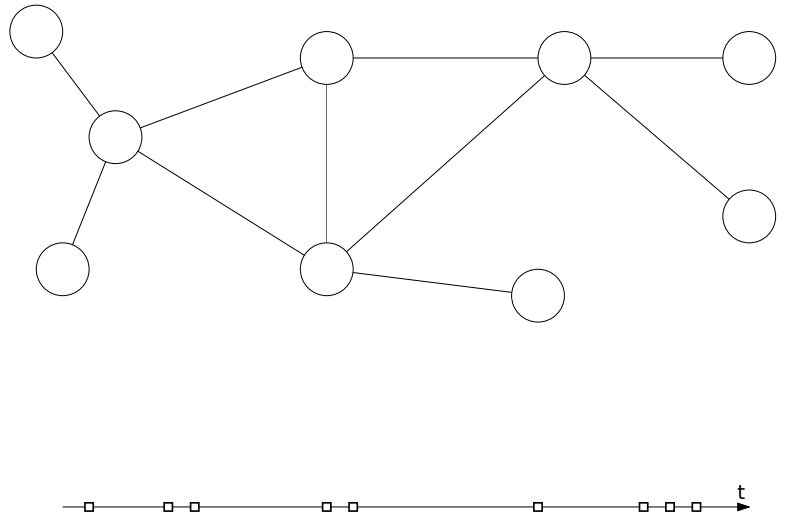
Off-Line

```
- P5: MPI_Recv at t=0.872s
- P3: MPI_Wait at t=0.881s
- P7: MPI_Send at t=1.287s
- P5: MPI_Recv at t=1.568s
- P7: MPI_Send at t=2.221s
- P0: MPI_Recv at t=2.559s
- P3: MPI_Wait at t=2.602s
- P0: MPI_Send at t=3.520s
- P1: MPI_Recv at t=4.257s
- P2: MPI_Recv at t=4.514s
- P6: MPI_Send at t=5.017s
- P7: MPI_Recv at t=5.989s
- P6: MPI_Recv at t=5.997s
- P4: MPI_Send at t=6.107s
- P6: MPI_Recv at t=6.534s
- P2: MPI_Send at t=7.152s
- P4: MPI_Recv at t=7.754s
[...]
```



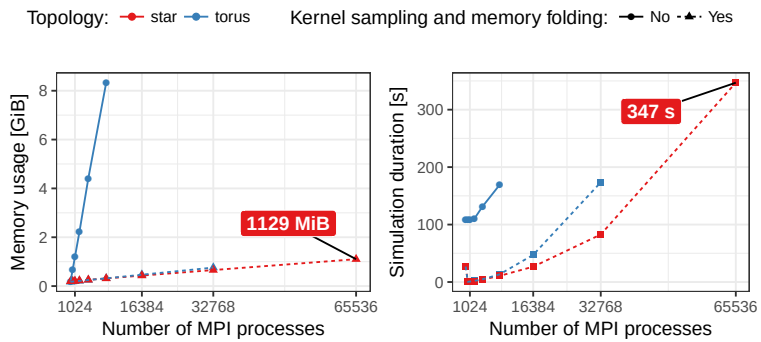
On-Line

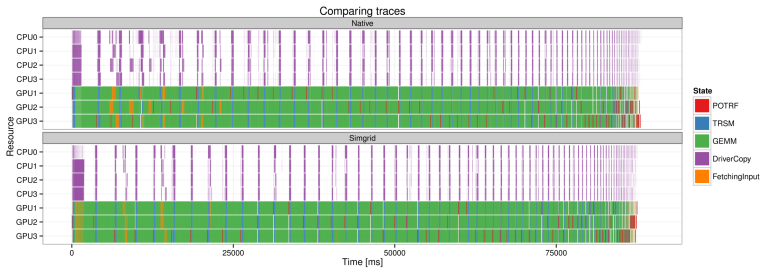




Past results

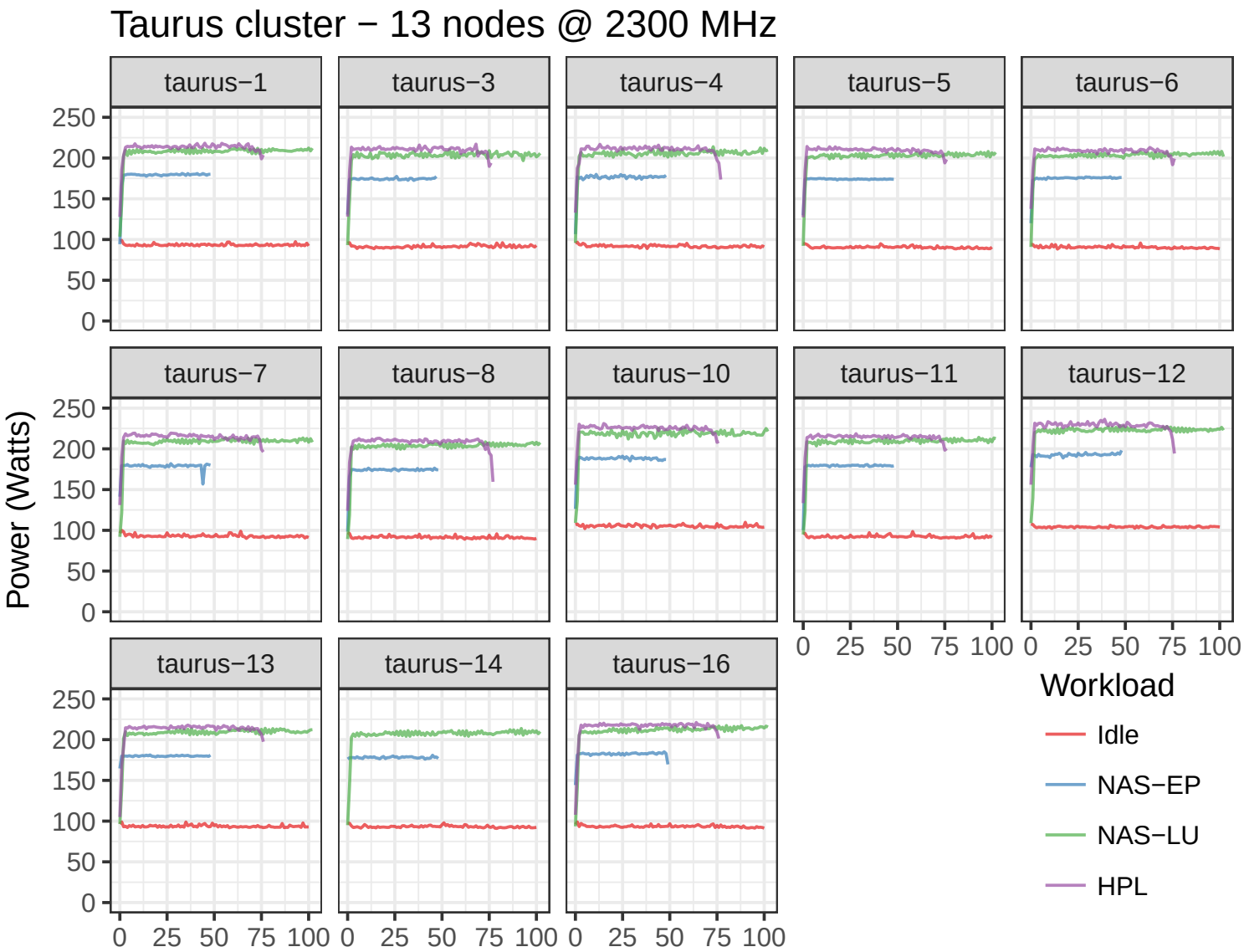
- MPI :
  - Validation at small scale (multi-node, single-core), also with real applications (BigDFT, Ondes3D, . . .)
  - Scalability demo with toy benchmarks
  - Accounting for the network topology and contention
  - Different software stacks such as OpenMPI, MPICH, . . . supported
- StarPU :
  - Hybrid node (Multi-GPU+multi-core) dense matrices
  - Small multi-core sparse matrices





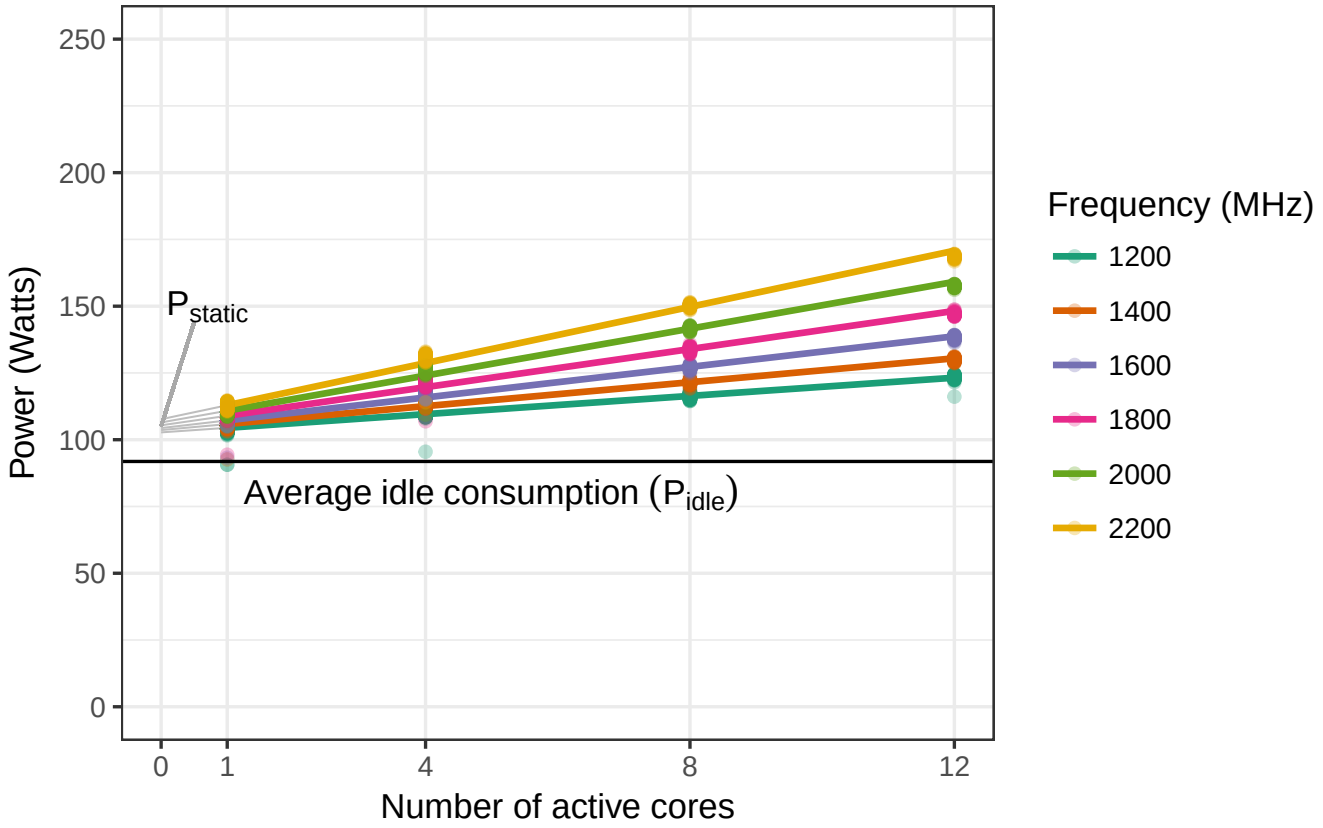
Contribution 1: Problem

- Problem: No energy model. It should be application-dependent.



Contribution 1: Solution (1/2)

- Our solution allows users to account for different frequencies, hosts and applications



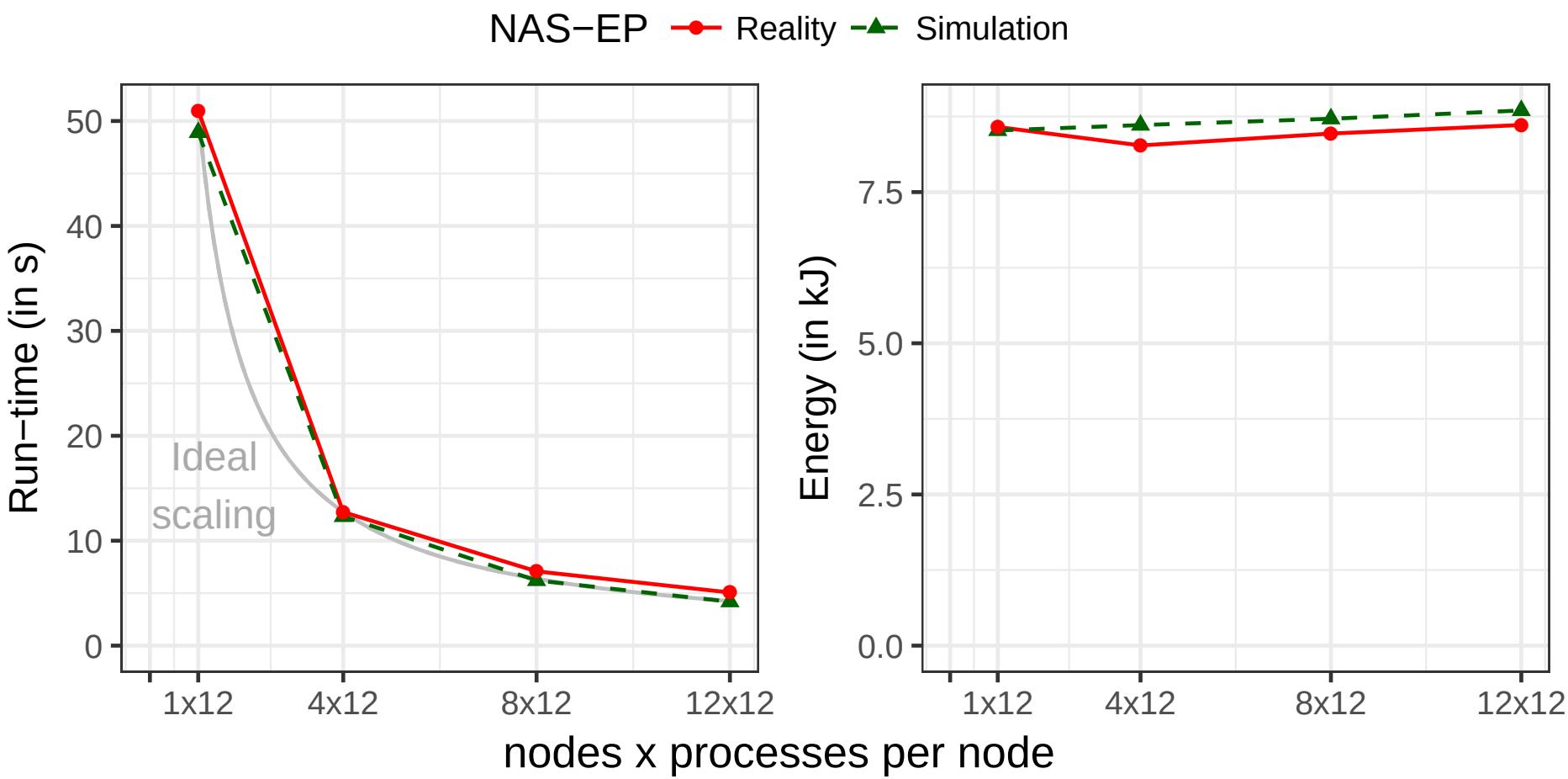
$$P_{i,f,w}(u) = P^{\text{static}}_{i,f} + P^{\text{dynamic}}_{i,f,w} \times u$$

Contribution 1: Solution (2/2)

- Very simple (but effective) energy model on per-host basis.

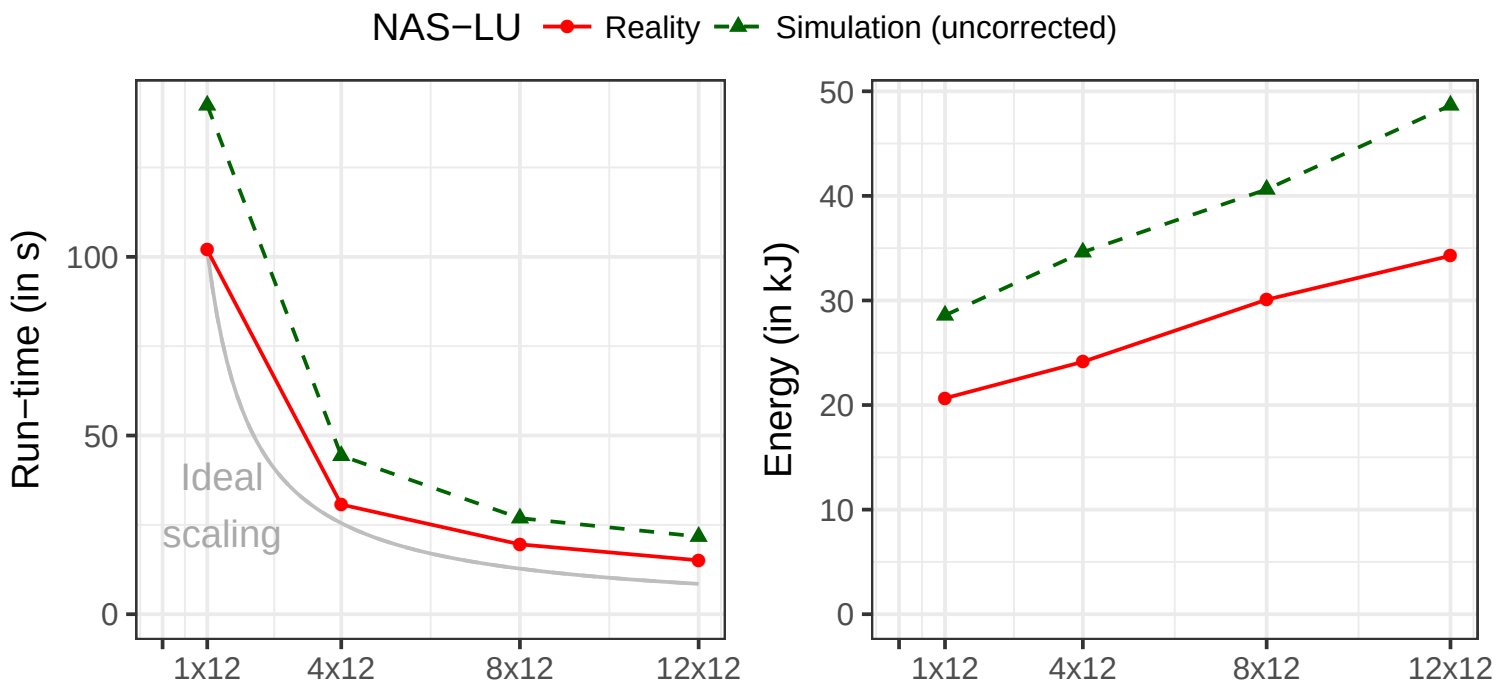
```
<host id="taurus-1.lyon.grid5000.fr" speed="20000"
      core="12">
    <prop id="watt_per_state"
        value="94.53:120.75:214.75,94.62:119.38:20
               ..."/>
    <prop id="watt_off" value="10" />
</host>
```

Contribution 1: Outcome



Contribution 2: Problem

- Previous benchmark (NAS-EP) uses almost no communication. What about more complicated applications?
- NAS-LU uses collective communications and is memory bound
- Applications often contend e.g., on L1 or L3 cache



Contribution 2: Solution

- We unbias by computing speedup factors through trace alignment

```
28  ...
    Region 43 if( iex .eq. 0 ) then
30  Region 43          if( north .ne. -1 ) then
                          call MPI_RECV( dum1(1,jst),
32                >                       5*(jend-jst+1),
                  >                       dp_type,
34                >                       north,
                  >                       from_n,
36                >                       MPI_COMM_WORLD,
                  >                       status,
38                >                       IERROR )
    Region 2              do j=jst,jend
40  Region 2                 g(1,0,j,k) = dum1(1,j)
    Region 2                 g(2,0,j,k) = dum1(2,j)
42  Region 2                 g(3,0,j,k) = dum1(3,j)
    Region 2                 g(4,0,j,k) = dum1(4,j)
44  Region 2                 g(5,0,j,k) = dum1(5,j)
    Region 2              enddo
46  Region 2           endif
    Region 2
48  Region 2           if( west .ne. -1 ) then
                          call MPI_RECV( dum1(1,ist),
50                >                       5*(iend-ist+1),
                  >                       dp_type,
52                >                       west,
                  >                       from_w,
54                >                       MPI_COMM_WORLD,
                  >                       status,
56                >                       IERROR )
    Region 3              do i=ist,iend
58  Region 3                 g(1,i,0,k) = dum1(1,i)
    Region 3                 g(2,i,0,k) = dum1(2,i)
60  Region 3                 g(3,i,0,k) = dum1(3,i)
    Region 3                 g(4,i,0,k) = dum1(4,i)
62  Region 3                 g(5,i,0,k) = dum1(5,i)
    Region 3              enddo
64  Region 3           endif
    ...
```

Calibration^{RL} trace (MPI)

rank	start (s)	duration (mus)	state
...	...	...	...
1	1.643388	1293	mpi_allreduce
1	1.644681	62	Computing
1	1.644743	82	mpi_barrier
1	1.644825	6454	Computing
1	1.651279	549	mpi_recv
1	1.651828	474	Computing
1	1.652302	53	mpi_send
1	1.652355	2	Computing
1	1.652357	15	mpi_send
1	1.652372	359	Computing
1	1.652731	11	mpi_recv
1	1.652742	462	Computing
1	1.653204	15	mpi_send
1	1.653219	1	Computing
1	1.653220	9	mpi_send
1	1.653229	376	Computing
1	1.653605	22	mpi_recv
1	1.653627	465	Computing
1	1.654092	16	mpi_send
1	1.654108	1	Computing
...	...	...	...

Calibration^{SMPI} trace (uncorrected SMPI)

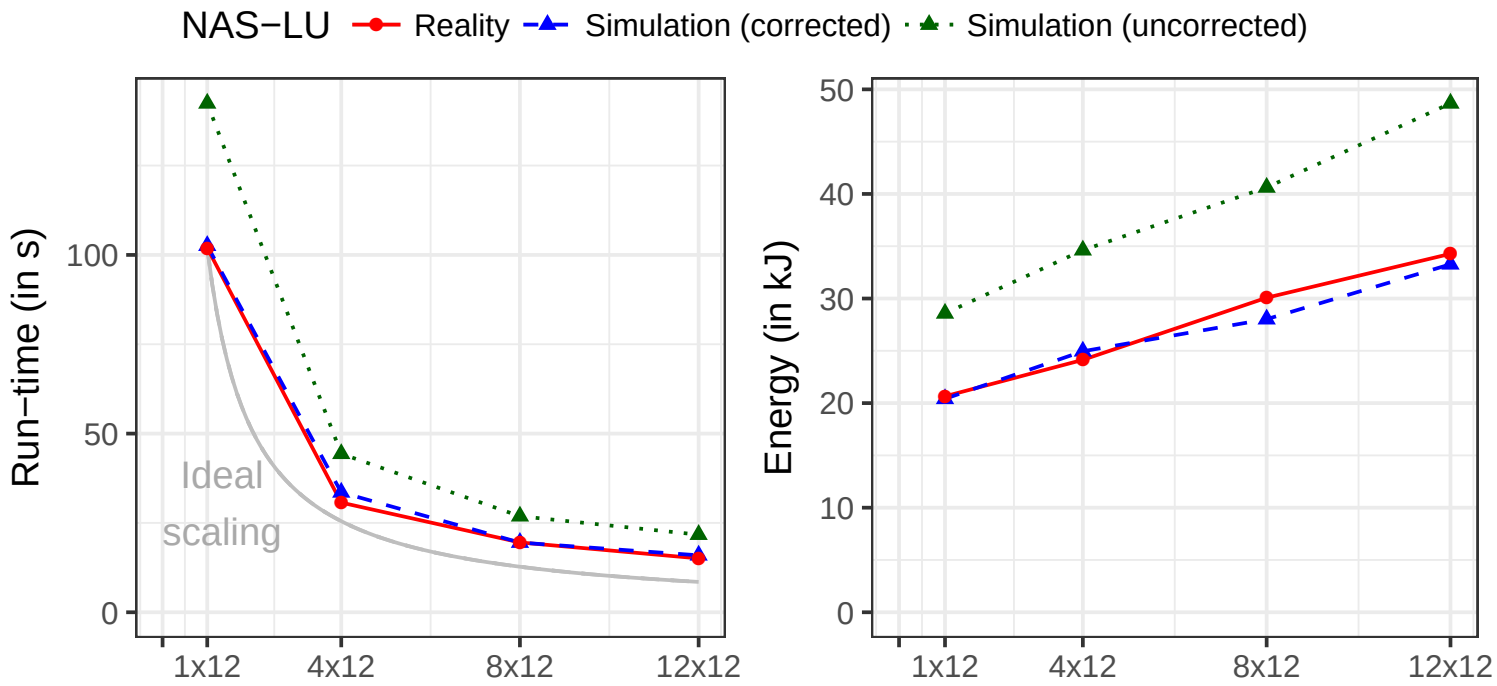
start (s)	duration (mus)	state	Filename	Line
...	...	...	...	...
0.550426	1130	mpi_allreduce	l2norm.f	57
0.551556	18	Computing		
0.551574	47	mpi_barrier	ssor.f	74
0.551621	5303	Computing		
0.556924	617	mpi_recv	exchange_1.f	30
0.557541	608	Computing	Region 3	
0.558149	4	mpi_send	exchange_1.f	113
0.558153	12	Computing	Region 17	
0.558165	4	mpi_send	exchange_1.f	130
0.558169	652	Computing	Region 18	
0.558821	8	mpi_recv	exchange_1.f	30
0.558829	587	Computing	Region 3	
0.559416	5	mpi_send	exchange_1.f	113
0.559421	12	Computing	Region 17	
0.559433	5	mpi_send	exchange_1.f	130
0.559438	699	Computing	Region 18	
0.560137	9	mpi_recv	exchange_1.f	30
0.560146	597	Computing	Region 3	
0.560743	4	mpi_send	exchange_1.f	113
0.560747	14	Computing	Region 18	
...	...	...	...	...

Merging traces

Region-based speedup/slowdown factors

```
"bcast_inputs.f:37:exchange_3.f:42",0.1655      Region 1
"exchange_1.f:30:exchange_1.f:48",14.6704       Region 2
"exchange_1.f:30:exchange_1.f:113",1.2967       Region 3
"exchange_1.f:30:exchange_1.f:130",1.2994       Region 4
. . .
"exchange_1.f:113:exchange_1.f:130",11.7101     Region 17
"exchange_1.f:130:exchange_1.f:30",1.9696       Region 18
...
"exchange_3.f:288:exchange_1.f:30",0.8933       Region 43
...
```

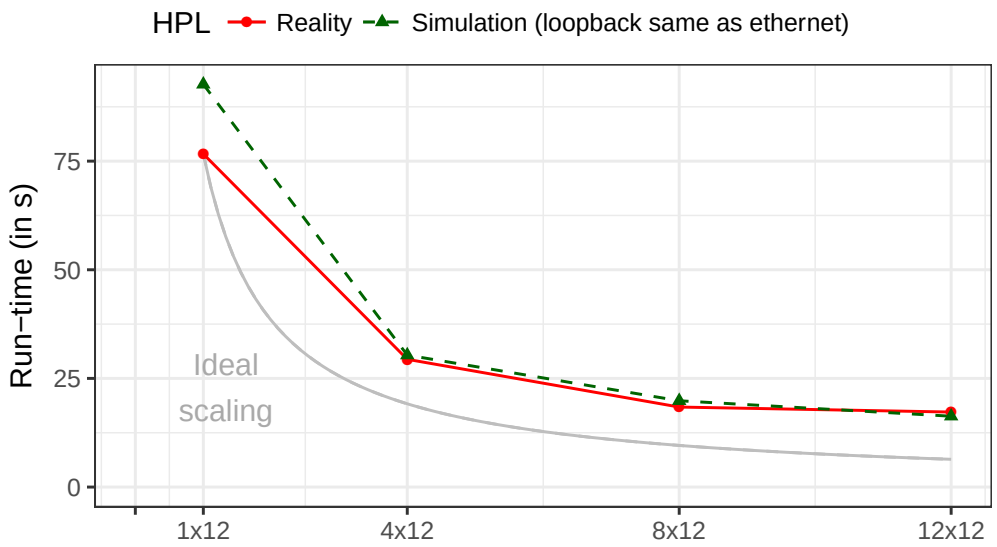
Contribution 2: Outcome



Contribution 3: Problem

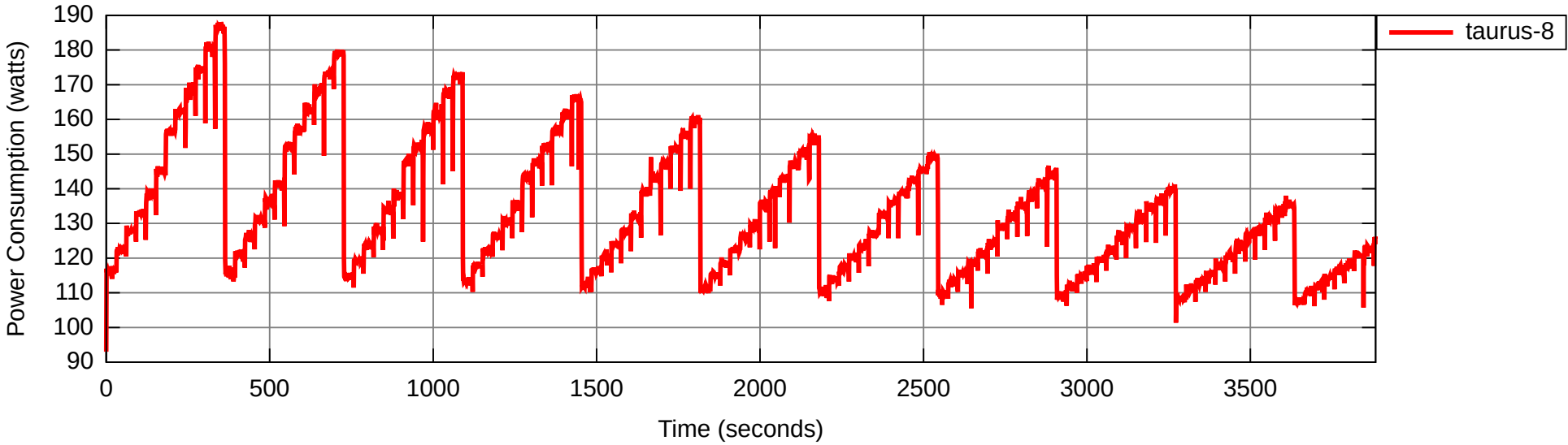
HPL is more complicated than this. Two main issues:

1. HPL sends many large messages from rank to rank.
   $\rightsquigarrow$ faster intra-node communications should be accounted for



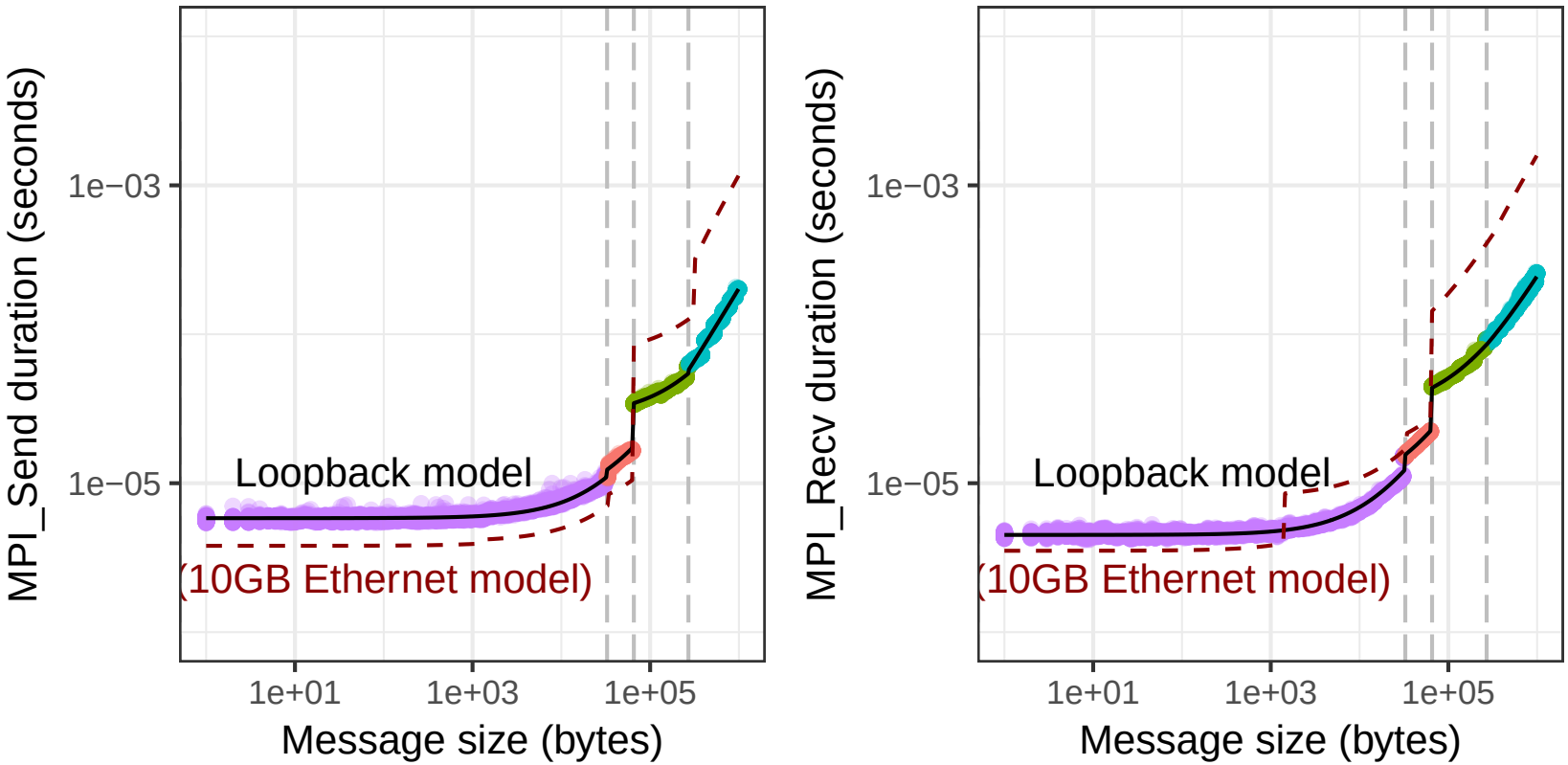
Contribution 3: Problem (2/2)

2. Makes heavy use of `MPI_Iprobe` in order to run computations while waiting for data.
   - But Iprobes do consume significant amounts of energy!
   - We hence cannot ignore Iprobes!



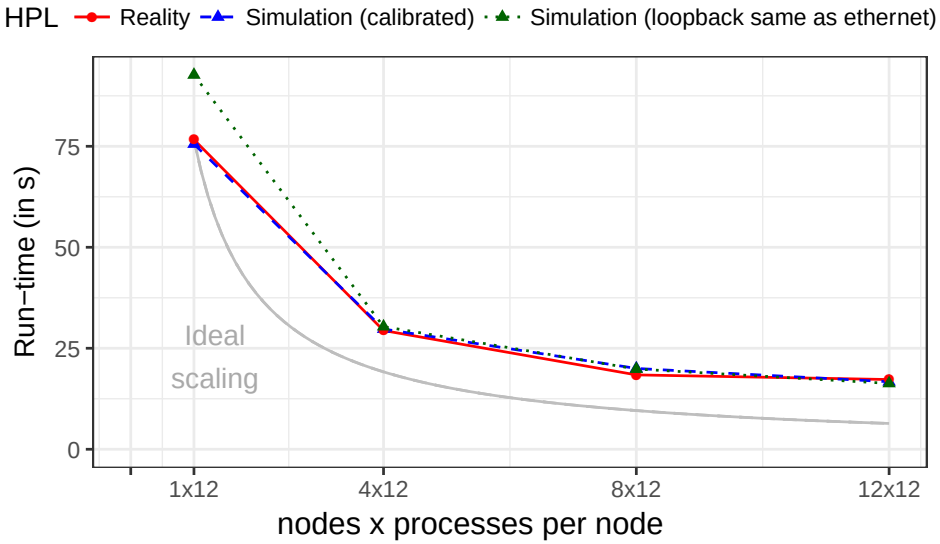
Contribution 3: Solution

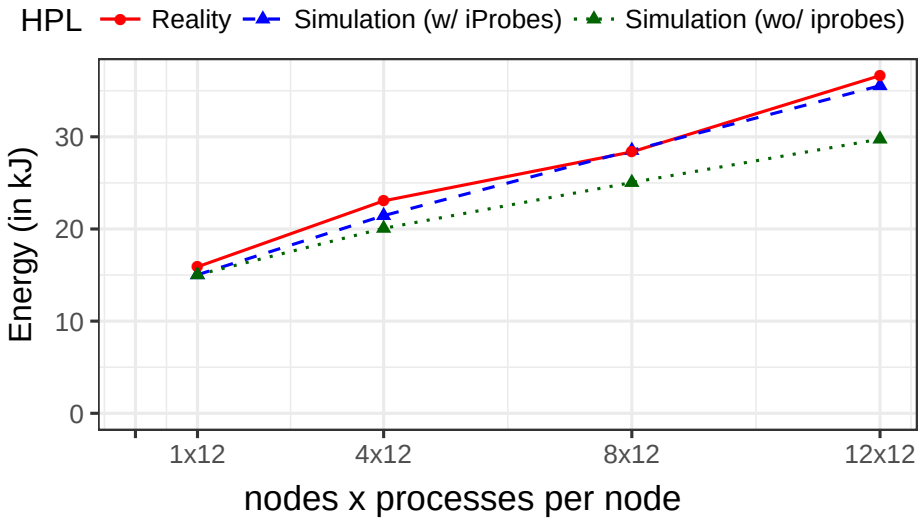
1. Calibrate loopback usage by sending local messages



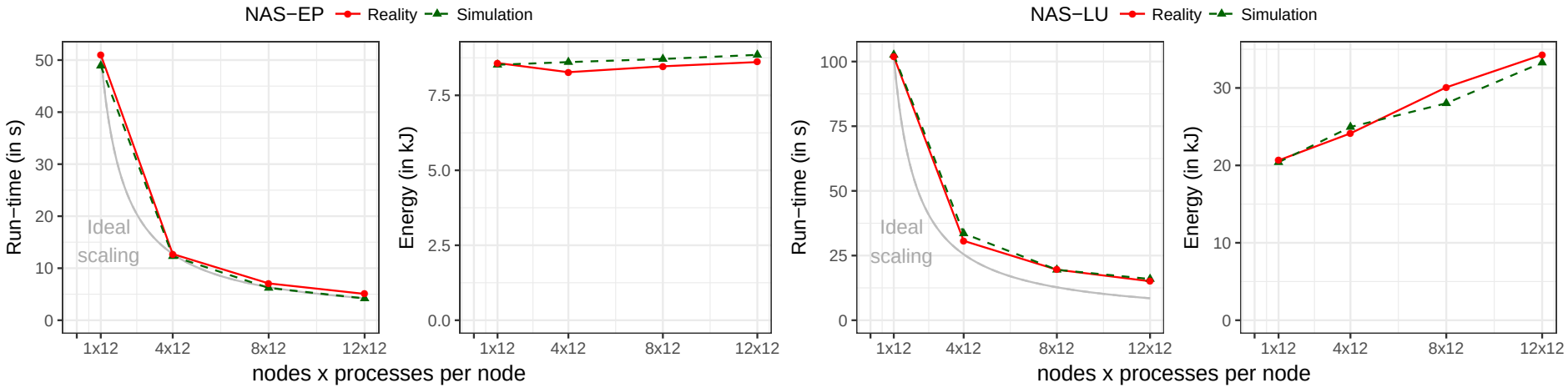
2. Iprobe issue is simple: Scale CPU usage while iprobeing via parameter `--cfg=smpi/iprobe-cpu-usage` (for us: 0.61)

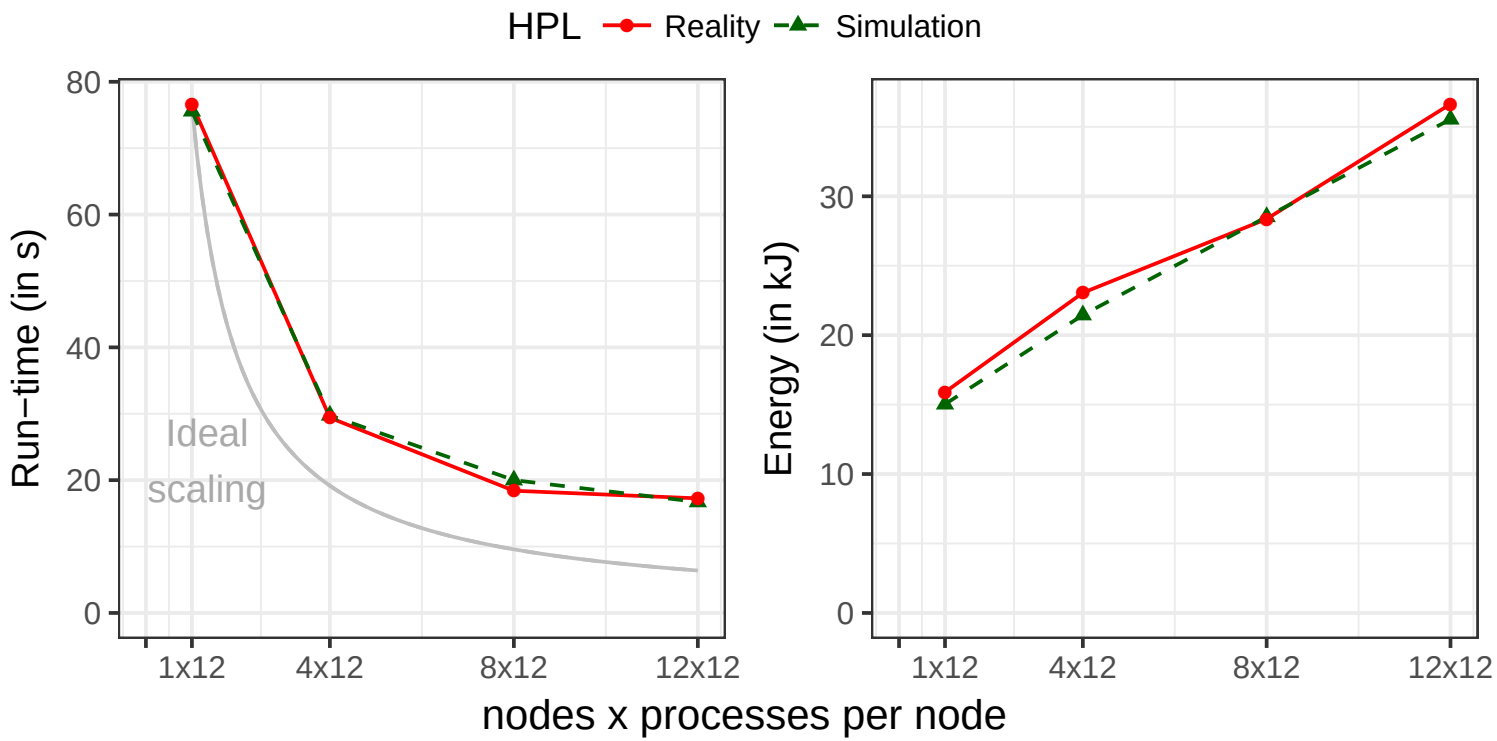
Contribution 3: Outcome





Validation


NAS−EP
Reality
Simulation
Run−time (in s)
Energy (in kJ)
Ideal
scaling
1x12
4x12
8x12
12x12
nodes x processes per node
NAS−LU
Reality
Simulation
Run−time (in s)
Energy (in kJ)
Ideal
scaling
1x12
4x12
8x12
12x12
nodes x processes per node


HPL
Reality
Simulation
Run−time (in s)
Energy (in kJ)
Ideal
scaling
1x12
4x12
8x12
12x12
nodes x processes per node

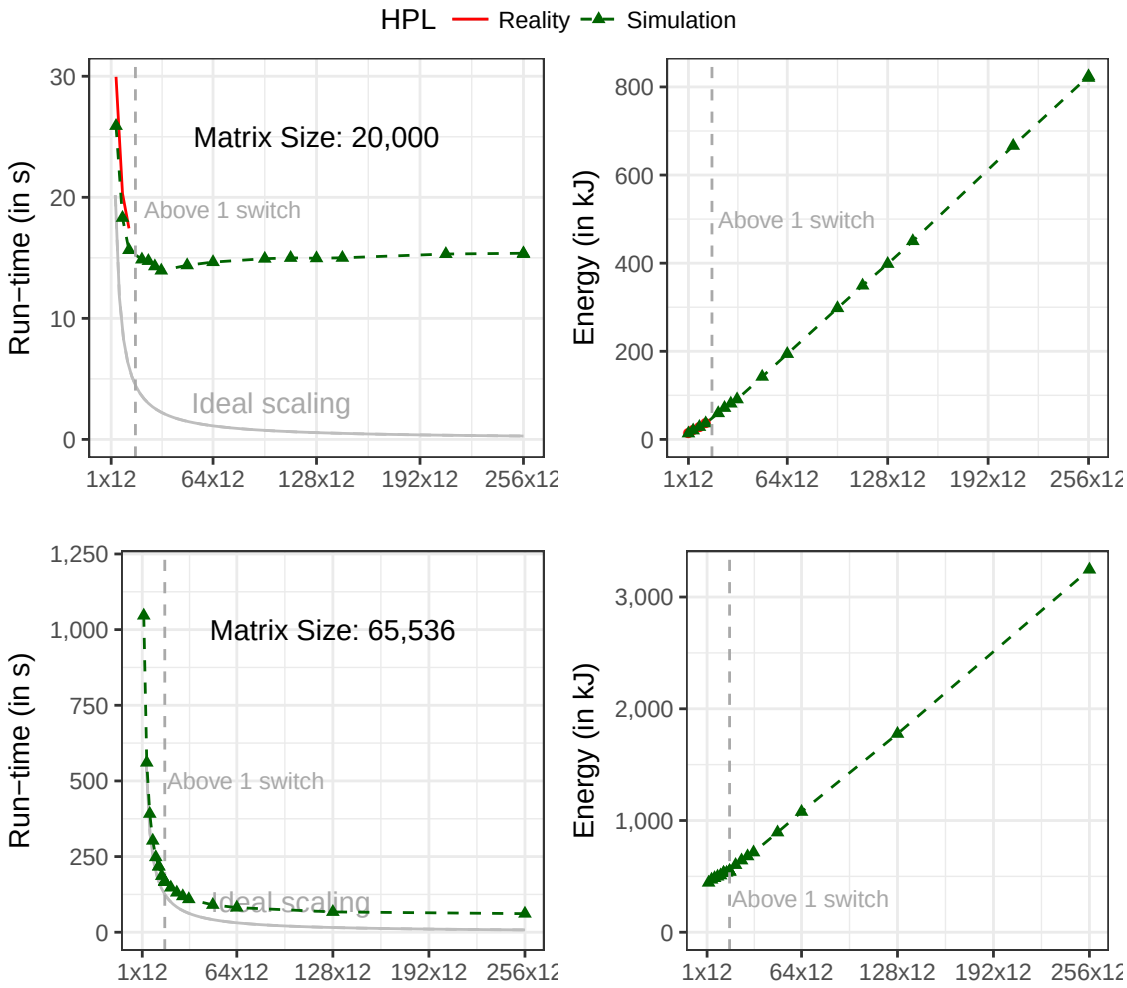
Limitations

- Only regular MPI applications; no OpenMPI!
- Only one model for the whole app - we need to find out if app has phases with significantly different power consumption, collect this data and inject it into the simulator
- The unbiasing (as used for NAS-LU) is a bit of a hack: For applications that wrap these MPI-calls, one needs to inspect the whole call-stack

Scalability results

- Emulation is very slow; with 144 processes and input size `20,000`, HPL takes hours but real execution takes 20 seconds
- Solution (for regular apps):
  1. Exploit regularity by sampling kernels (automatic / on-the-fly) or manually modeling computation kernels;
  2. Tweak memory allocations (especially for large-scale simulations)

Scalability results (2/2)



Future work

- Further validate and strengthen this approach by investigating other apps
- Try to re-obtain some of the results from the Top500 / Green500!

Thank You!

Thank you for your attention![1]

Contact: franz-christian.heinrich@inria.fr

[1] If you're interested: Our experiments, scripts, notes, figure generation code, ... is available online at https://gitlab.inria.fr/fheinric/paper-simgrid-energy