Capacity Planning of Supercomputers

Simulating MPI Applications at Scale

Tom Cornebize
Under the supervision of Arnaud Legrand

21 June 2017

Laboratoire d'Informatique de Grenoble
Ensimag - Grenoble INP

Introduction

Top500

Sunway TaihuLight, China, #1
93 Pflops
Custom five level hierarchy
$40,950 \times 260$ cores

Tianhe-2, China, #2
34 Pflops
Fat tree
$32,000 \times 12$ cores $+ 48,000$ Xeon Phi

Piz Daint, Switzerland, #3
20 Pflops
Dragonfly
$5,272 \times (8$ cores $+ 1$ GPU$)$

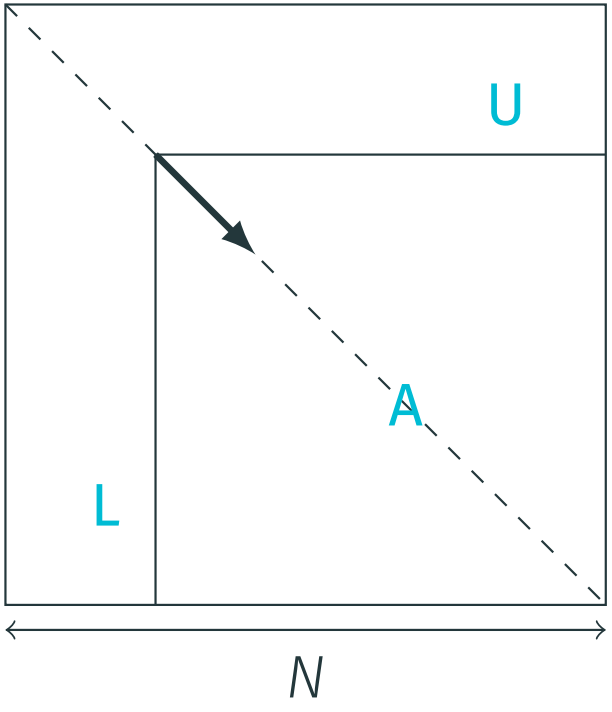
Stampede, United States, #20
5 Pflops
Fat tree
$6,400 \times (8$ cores $+ 1$ Xeon Phi$)$

High Performance LINPACK (HPL)

Benchmark used to establish the Top500

LU factorization, $A = L \times U$

Complexity: $\mathrm{flop}(N) = \frac{2}{3}N^3 + 2N^2 + O(N)$



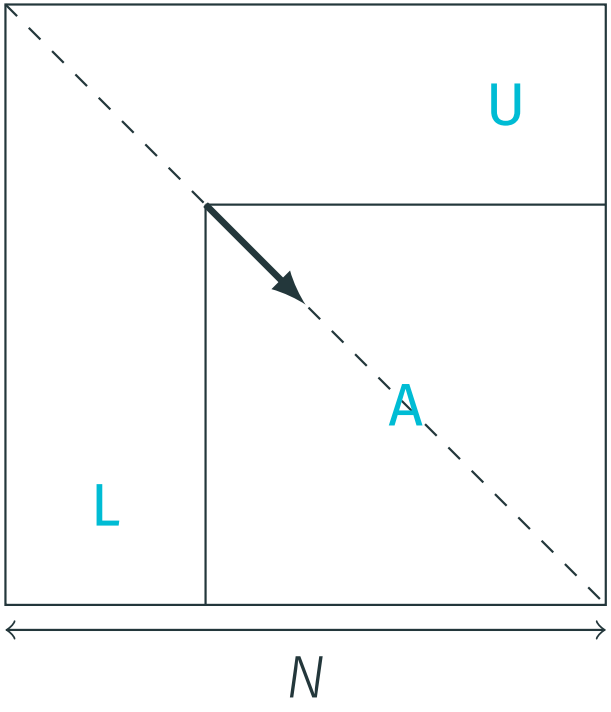
allocate the matrix
for $k = N$ ***to*** 0 **do**
  allocate the panel
  various functions (max, swap,...)
  compute the inverse
  broadcast
  update

High Performance LINPACK (HPL)

Benchmark used to establish the Top500

LU factorization, $A = L \times U$

Complexity: $\mathrm{flop}(N) = \frac{2}{3}N^3 + 2N^2 + O(N)$



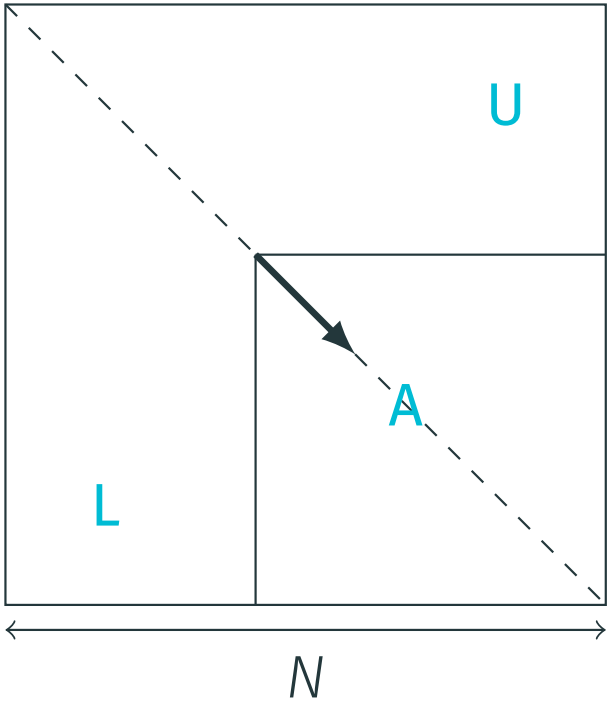
allocate the matrix
for $k = N$ ***to*** 0 **do**
  allocate the panel
  various functions (max, swap,...)
  compute the inverse
  broadcast
  update

High Performance LINPACK (HPL)

Benchmark used to establish the Top500

LU factorization, $A = L \times U$

Complexity: $\mathrm{flop}(N) = \frac{2}{3}N^3 + 2N^2 + O(N)$



allocate the matrix
for $k = N$ ***to*** 0 **do**
  allocate the panel
  various functions (max, swap,...)
  compute the inverse
  broadcast
  update

Open questions in HPC

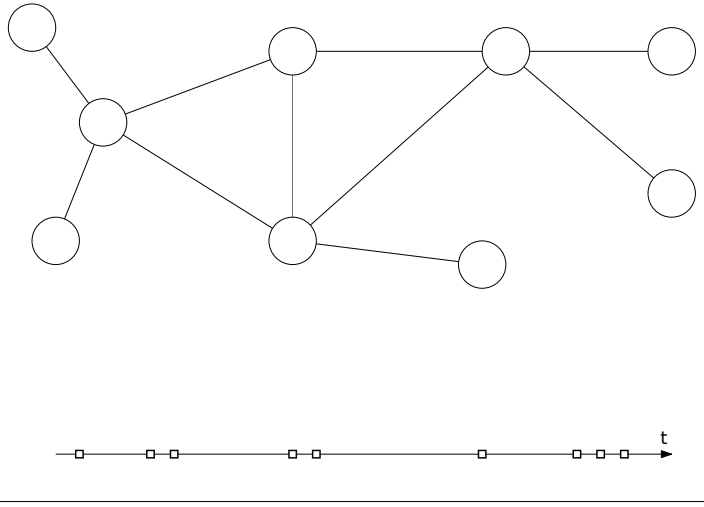
- Topology (torus, fat tree, dragonfly, etc.)
- Routing algorithm
- Scheduling (when? where?)
- Workload (job size, behavior)

Keywords: capacity planning, co-design

Simulation may help

Off-line

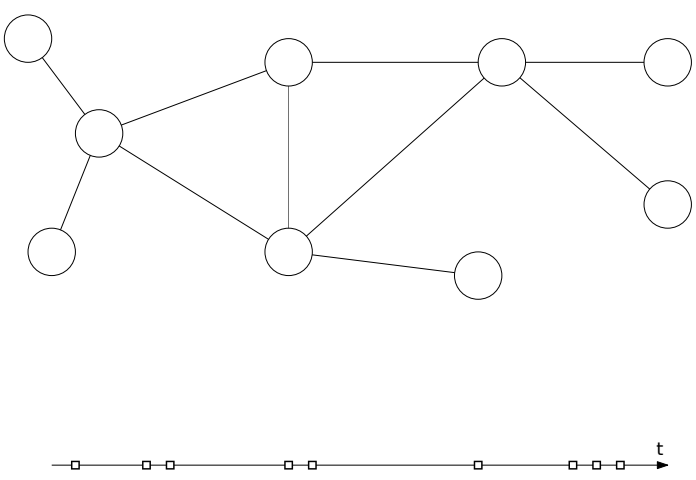
```
- P5: MPI_Recv at t=0.872s
- P3: MPI_Wait at t=0.881s
- P7: MPI_Send at t=1.287s
- P5: MPI_Recv at t=1.568s
- P7: MPI_Send at t=2.221s
- P0: MPI_Recv at t=2.559s
- P3: MPI_Wait at t=2.602s
- P0: MPI_Send at t=3.520s
- P1: MPI_Recv at t=4.257s
- P2: MPI_Recv at t=4.514s
- P6: MPI_Send at t=5.017s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



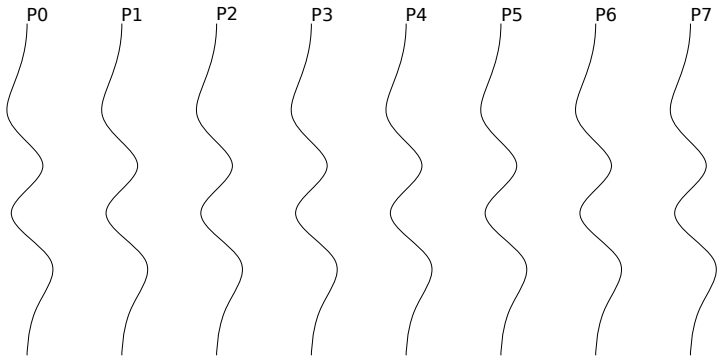
Simulation of HPC applications

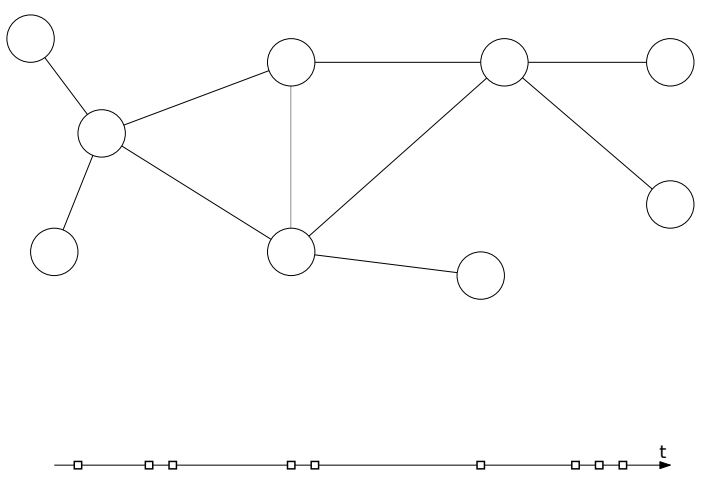
Off-line

```
- 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
[...]
```



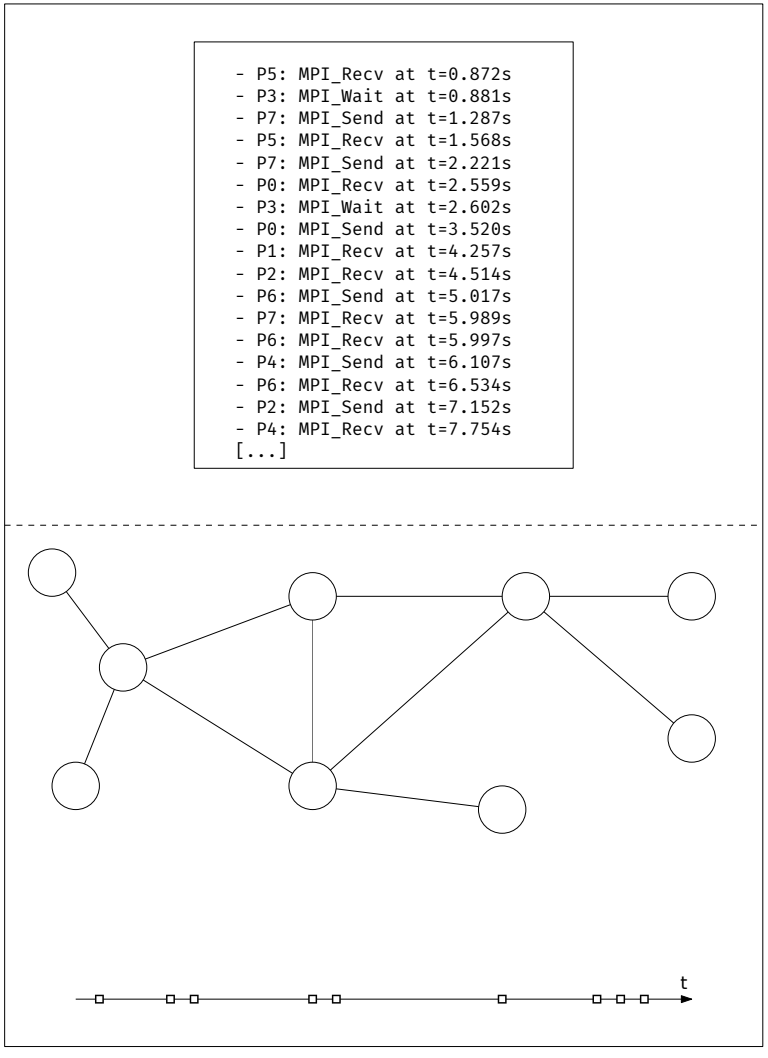
On-line



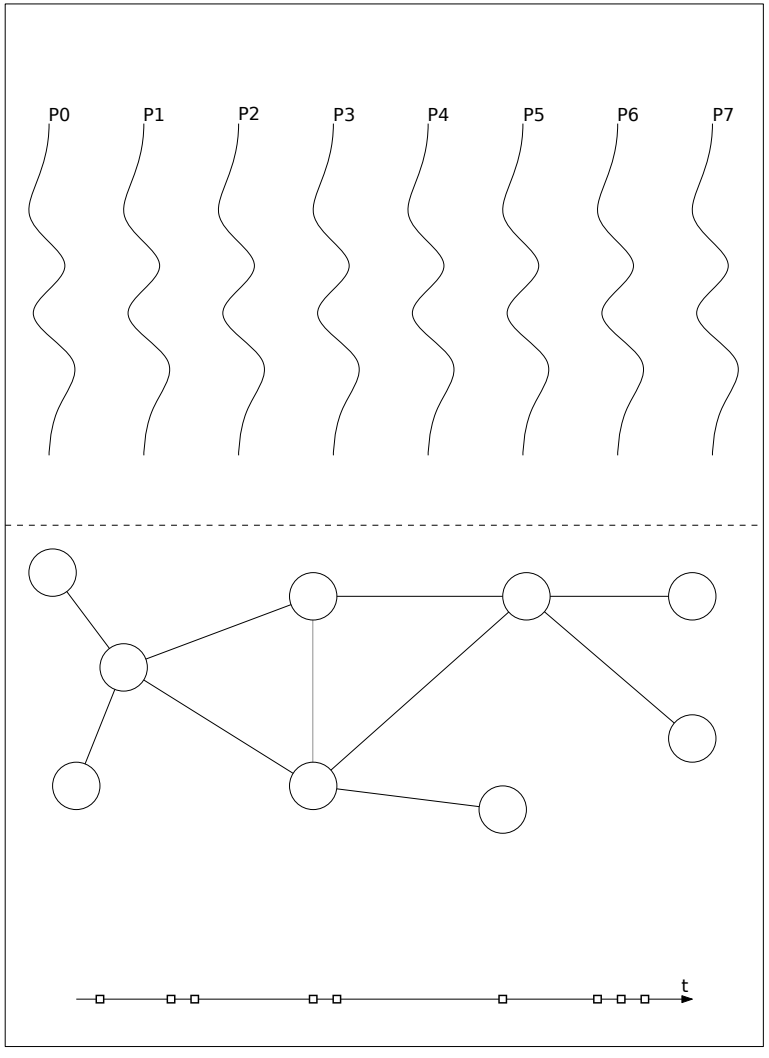


Simulation of HPC applications

Off-line



On-line



Simgrid: both approaches

Objective: simulation of Stampede's execution of HPL

Real execution:

- Matrix of size 3,875,000
- Using 6,006 MPI processes
- About 2 hours

Objective: simulation of Stampede's execution of HPL

Real execution:

- Matrix of size 3,875,000
- Using 6,006 MPI processes
- About 2 hours

Requirement for the emulation of Stampede's execution:

- $\geq 3,875,000^2 \times 8$ bytes $\approx$ 120 terabytes of memory
- $\geq 6,006 \times 2$ hours $\approx$ 500 days

Very optimistic

Scalable HPL simulation

Methodology

Several optimizations. For each of them:

- Evaluate the (possible) loss of prediction accuracy
- Evaluate the (possible) gain of performance

Publicly available:

- Laboratory notebook
- Modified HPL
- Scripts
- Modifications to Simgrid (integrated in the main project)

Computation kernel sampling

`dgemm` and `dtrsm`

$\geq 90\%$ of the simulation time

allocate the matrix
for $k = N$ ***to*** 0 **do**
  allocate the panel
  various functions (max, swap,...)
  ~~compute the inverse~~
  broadcast
  ~~update~~

Computation kernel sampling

`dgemm` and `dtrsm`

$\geq 90\%$ of the simulation time

Solution: modeling these functions to inject their duration

allocate the matrix
for $k = N$ ***to*** 0 **do**
  allocate the panel
  various functions (max, swap,...)
  ~~compute the inverse~~
  broadcast
  ~~update~~

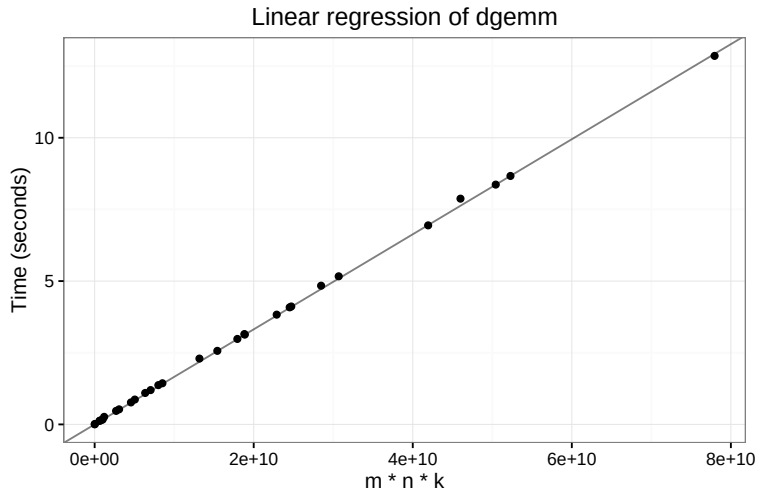
Computation kernel sampling

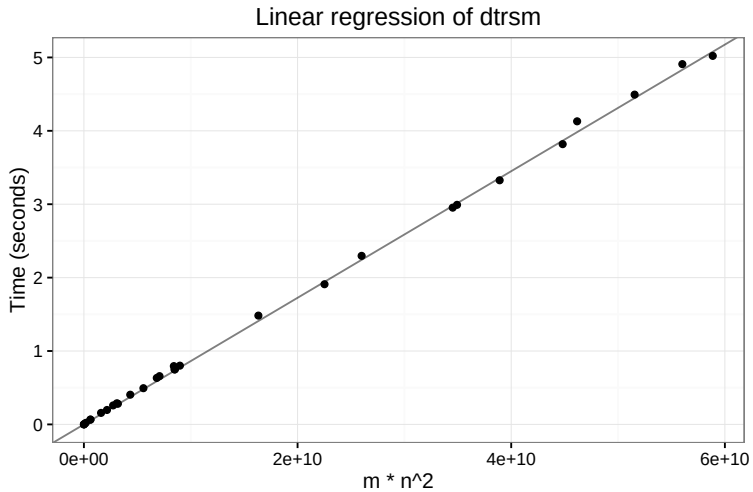
`dgemm` and `dtrsm`

$\geq 90\%$ of the simulation time

Solution: modeling these functions to inject their duration

allocate the matrix
for $k = N$ ***to*** 0 **do**
- allocate the panel
- various functions (max, swap,...)
- ~~compute the inverse~~
- broadcast
- ~~update~~





$$T_{dgemm}(M, N, K) = M \times N \times K \times 1.706348 \times 10^{-10}$$

$$T_{dtrsm}(M, N) = M \times N^2 \times 8.624970 \times 10^{-11}$$

Computation pruning

68% of the simulation time spent in HPL

allocate the matrix
for $k = N$ ***to*** 0 **do**
- allocate the panel
- ~~various functions (max, swap,...)~~
- ~~compute the inverse~~
- broadcast
- ~~update~~

Computation pruning

68% of the simulation time spent in HPL

Culprits:

- Initialization and verification functions
- Other BLAS and HPL functions

allocate the matrix
for $k = N$ ***to*** 0 **do**
  allocate the panel
  ~~various functions (max, swap,...)~~
  ~~compute the inverse~~
  broadcast
  ~~update~~

Computation pruning

68% of the simulation time spent in HPL

Culprits:

- Initialization and verification functions
- Other BLAS and HPL functions

Solution: just skip them

```
allocate the matrix
for k = N to 0 do
    allocate the panel
    various functions (max, swap,...)
    compute the inverse
    broadcast
    update
```

Reducing the memory consumption

Memory consumption still too large

~~allocate the matrix~~
for $k = N$ ***to*** 0 **do**
- allocate the panel
- ~~various functions (max, swap,...)~~
- ~~compute the inverse~~
- broadcast
- ~~update~~

Reducing the memory consumption

Memory consumption still too large

Solution: use SMPI_SHARED_MALLOC

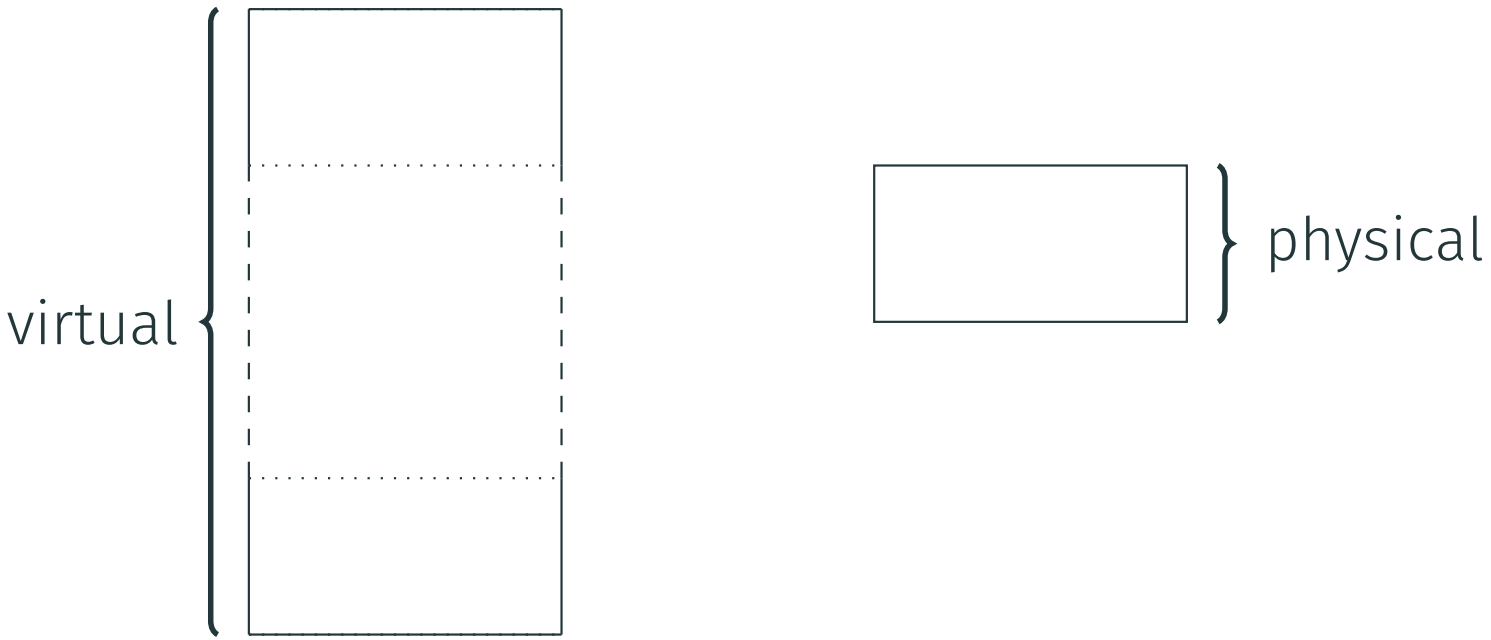
~~allocate the matrix~~
for $k = N$ *to* 0 **do**
- allocate the panel
- ~~various functions (max, swap,...)~~
- ~~compute the inverse~~
- broadcast
- ~~update~~

Reducing the memory consumption

Memory consumption still too large

Solution: use SMPI_SHARED_MALLOC

~~allocate the matrix~~
for $k = N$ ***to*** 0 **do**
- allocate the panel
- ~~various functions (max, swap,...)~~
- ~~compute the inverse~~
- broadcast
- ~~update~~

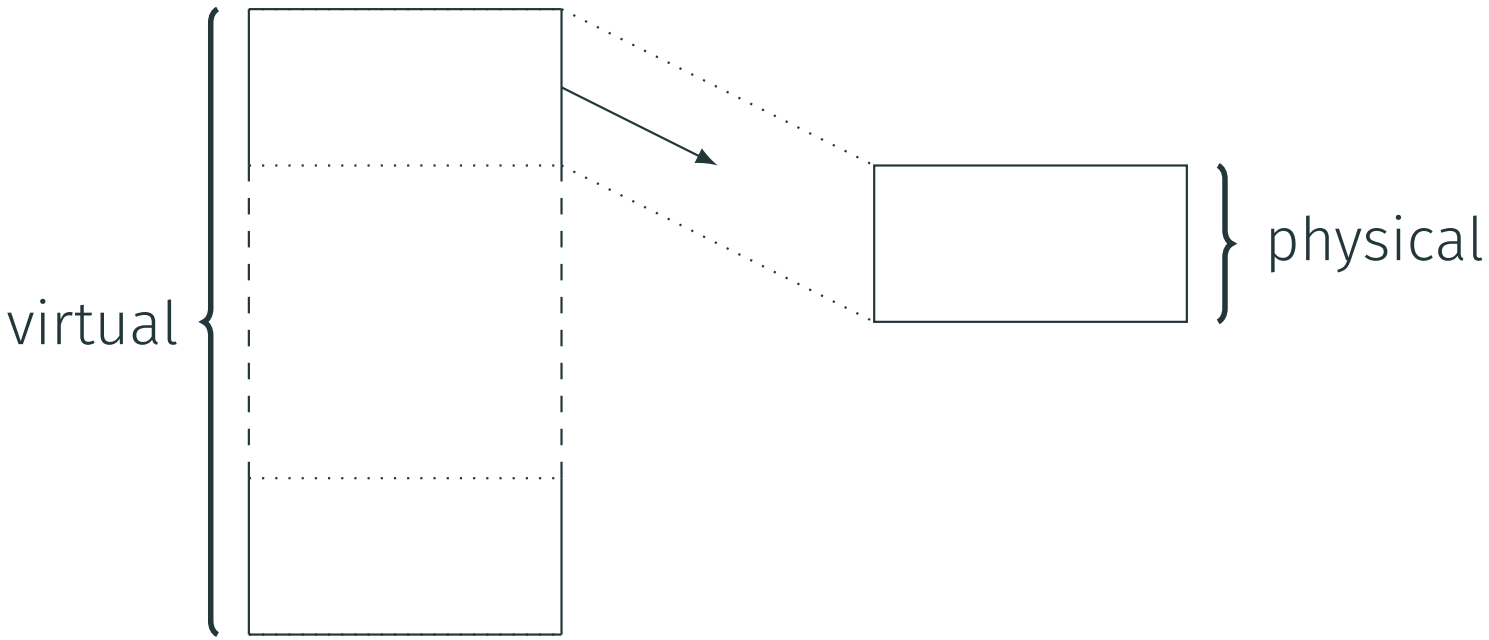


Reducing the memory consumption

Memory consumption still too large

Solution: use `SMPI_SHARED_MALLOC`

~~allocate the matrix~~
for $k = N$ ***to*** 0 **do**
- allocate the panel
- ~~various functions (max, swap,...)~~
- ~~compute the inverse~~
- broadcast
- ~~update~~

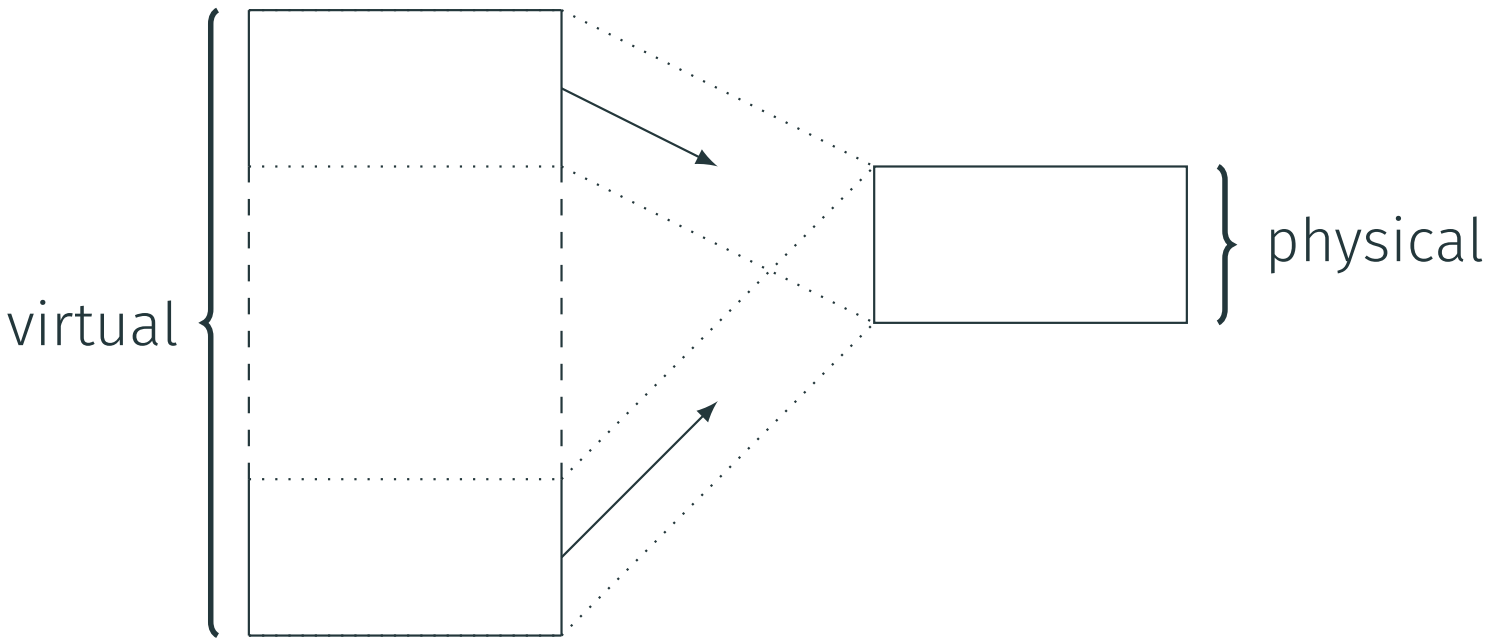


Reducing the memory consumption

Memory consumption still too large

Solution: use SMPI_SHARED_MALLOC

~~allocate the matrix~~
for $k = N$ ***to*** 0 **do**
- allocate the panel
- ~~various functions (max, swap,...)~~
- ~~compute the inverse~~
- broadcast
- ~~update~~



Problem: panel buffers

~~allocate the matrix~~
for $k = N$ *to* 0 do
- ~~allocate the panel~~
- ~~various functions (max, swap,...)~~
- ~~compute the inverse~~
- broadcast
- ~~update~~

Reducing the memory consumption

Problem: panel buffers

Must remain contiguous

~~allocate the matrix~~
for $k = N$ *to* 0 do
- ~~allocate the panel~~
- ~~various functions (max, swap,...)~~
- ~~compute the inverse~~
- broadcast
- ~~update~~

Reducing the memory consumption

Problem: panel buffers

Must remain contiguous

~~allocate the matrix~~
for $k = N$ *to* 0 do
- ~~allocate the panel~~
- ~~various functions (max, swap,...)~~
- ~~compute the inverse~~
- broadcast
- ~~update~~

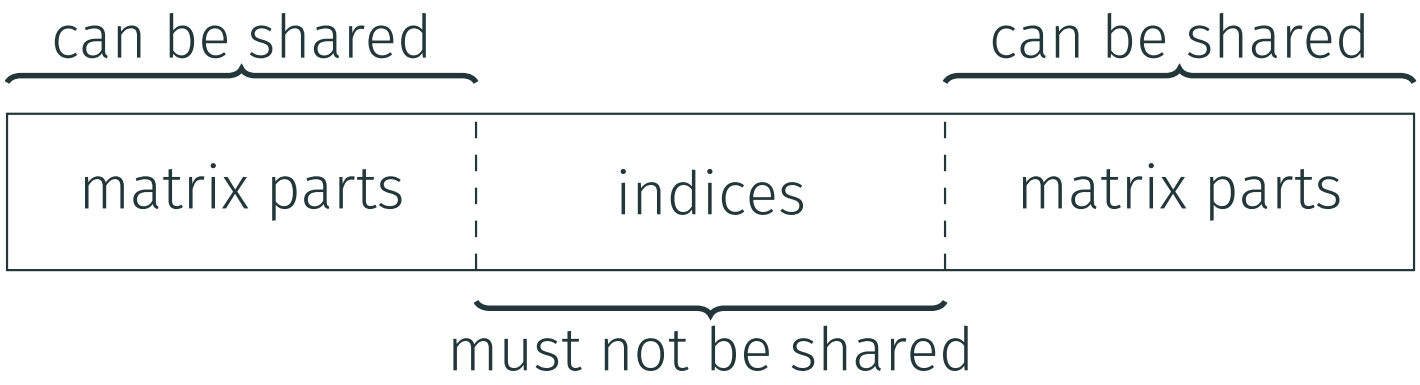
Reducing the memory consumption

#154

Problem: panel buffers

Must remain contiguous

~~allocate the matrix~~
for $k = N$ *to* 0 **do**
- ~~allocate the panel~~
- ~~various functions (max, swap,...)~~
- ~~compute the inverse~~
- broadcast
- ~~update~~



Problem: panel buffers

Must remain contiguous

~~allocate the matrix~~
for $k = N$ *to* 0 **do**
- ~~allocate the panel~~
- ~~various functions (max, swap,...)~~
- ~~compute the inverse~~
- broadcast
- ~~update~~

Solution: `SMPI_PARTIAL_SHARED_MALLOC`

Arbitrary number of shared and private blocks.

Reusing the panel buffers

At each iteration, new allocation and deallocation by all processes

~~allocate the matrix~~
for $k = N$ ***to*** 0 **do**
- ~~allocate the panel~~
- ~~various functions (max, swap,...)~~
- ~~compute the inverse~~
- broadcast
- ~~update~~

Reusing the panel buffers

At each iteration, new allocation and deallocation by all processes

Solution: reuse the buffers (sizes strictly decreasing)

Needs to be done carefully

~~allocate the matrix~~
for $k = N$ *to* 0 do
- ~~allocate the panel~~
- ~~various functions (max, swap,...)~~
- ~~compute the inverse~~
- broadcast
- ~~update~~

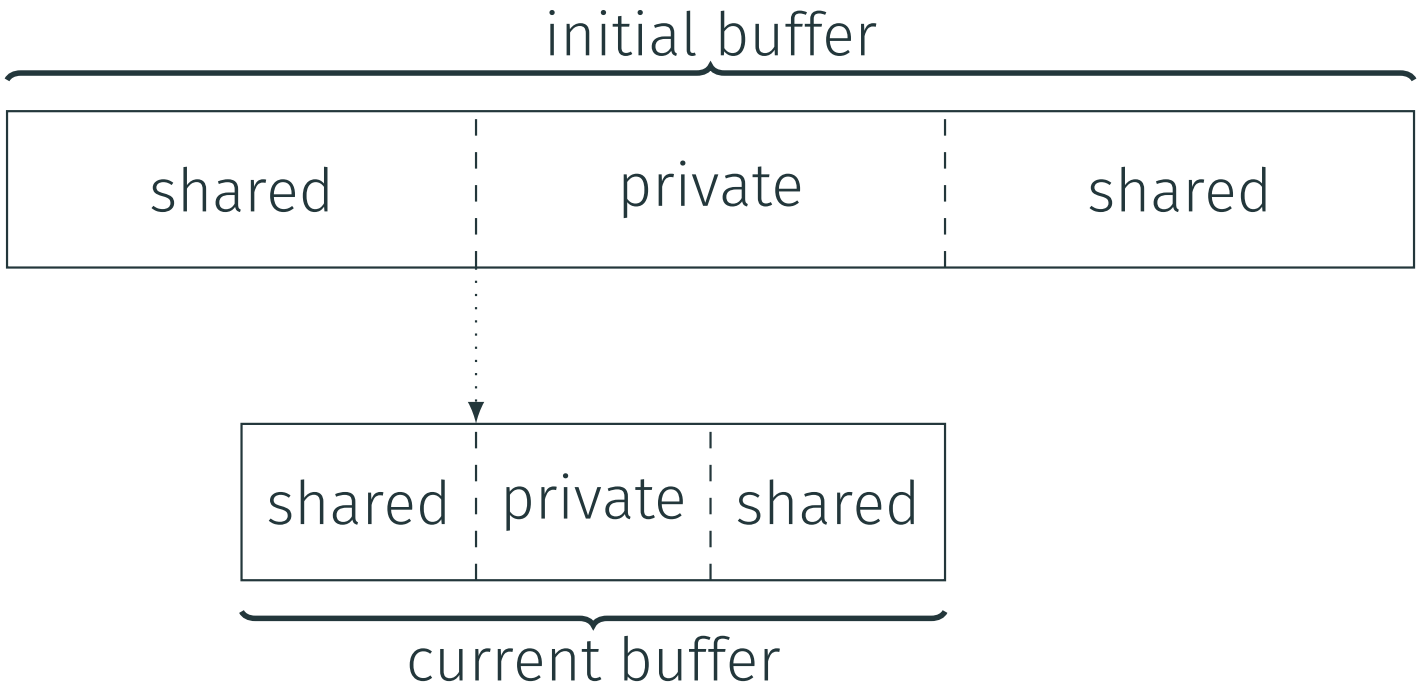
Reusing the panel buffers

At each iteration, new allocation and deallocation by all processes

Solution: reuse the buffers (sizes strictly decreasing)

Needs to be done carefully

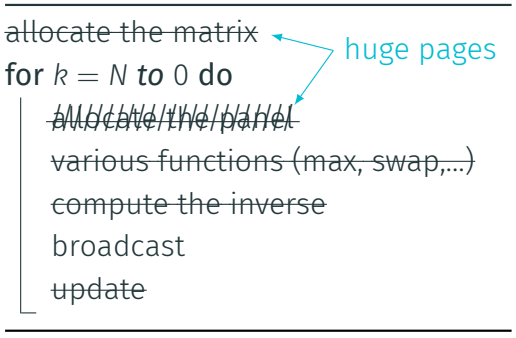
~~allocate the matrix~~
for $k = N$ *to* 0 **do**
- ~~allocate the panel~~
- ~~various functions (max, swap,...)~~
- ~~compute the inverse~~
- broadcast
- ~~update~~



Using huge pages

#168

Problem: at large scales, CPU utilization drops and simulation time explodes

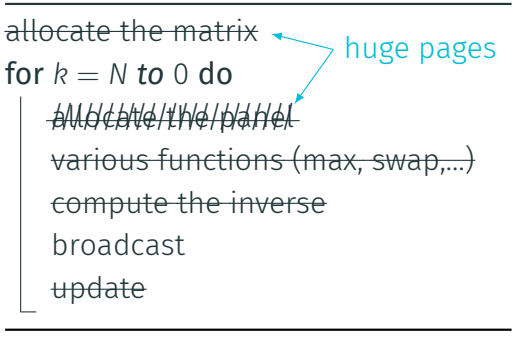


Using huge pages

#168

Problem: at large scales, CPU utilization drops and simulation time explodes

Reason: the page table becomes very large



Problem: at large scales, CPU utilization drops and simulation time explodes
Reason: the page table becomes very large

~~allocate the matrix~~
for $k = N$ ***to*** 0 **do**
  ~~allocate the panel~~
  ~~various functions (max, swap,...)~~
  ~~compute the inverse~~
  broadcast
  ~~update~~

huge pages

Matrix of size $N \Rightarrow$ page table of size:

allocated virtual memory

$$PT_{size}(N) = \frac{N^2 \times 8}{4,096} \times 8$$

entry size

page size

$PT_{size}(600,000) \approx 5\text{GB}$

Using huge pages

#168

Problem: at large scales, CPU utilization drops and simulation time explodes

Reason: the page table becomes very large

~~allocate the matrix~~ ← huge pages

for $k = N$ ***to*** 0 **do**

- ~~allocate the panel~~
- ~~various functions (max, swap,...)~~
- ~~compute the inverse~~
- broadcast
- ~~update~~

Matrix of size $N \Rightarrow$ page table of size:

allocated virtual memory

$$PT_{size}(N) = \frac{N^2 \times 8}{4,096} \times 8$$

entry size

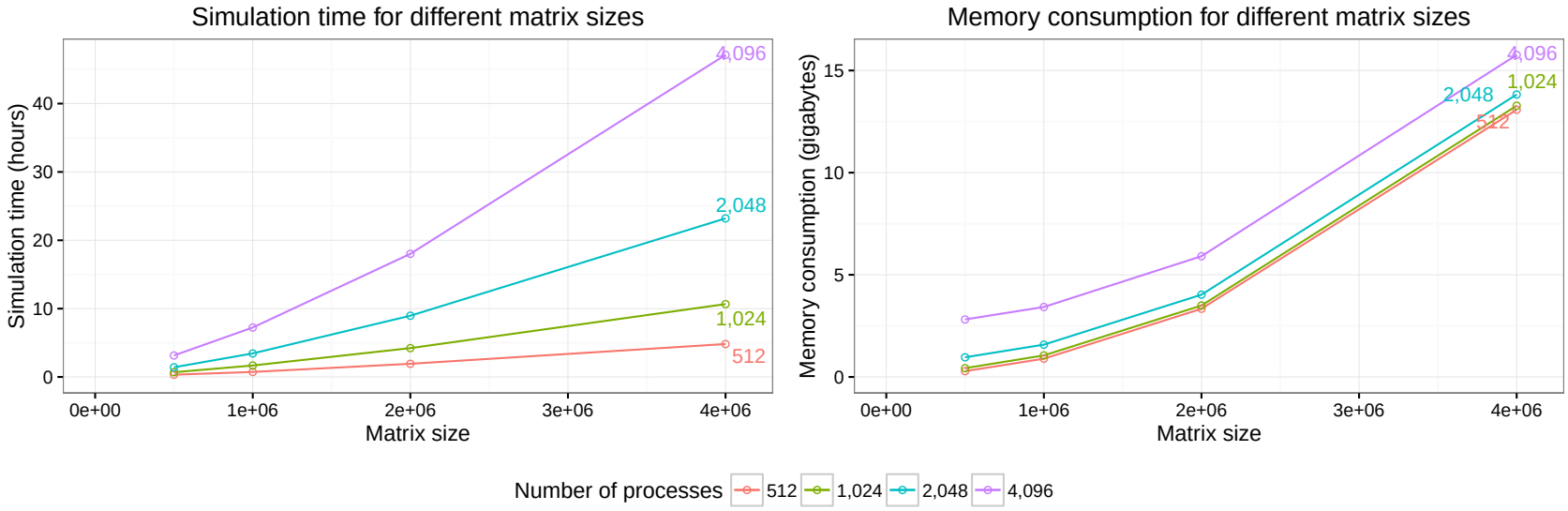
page size

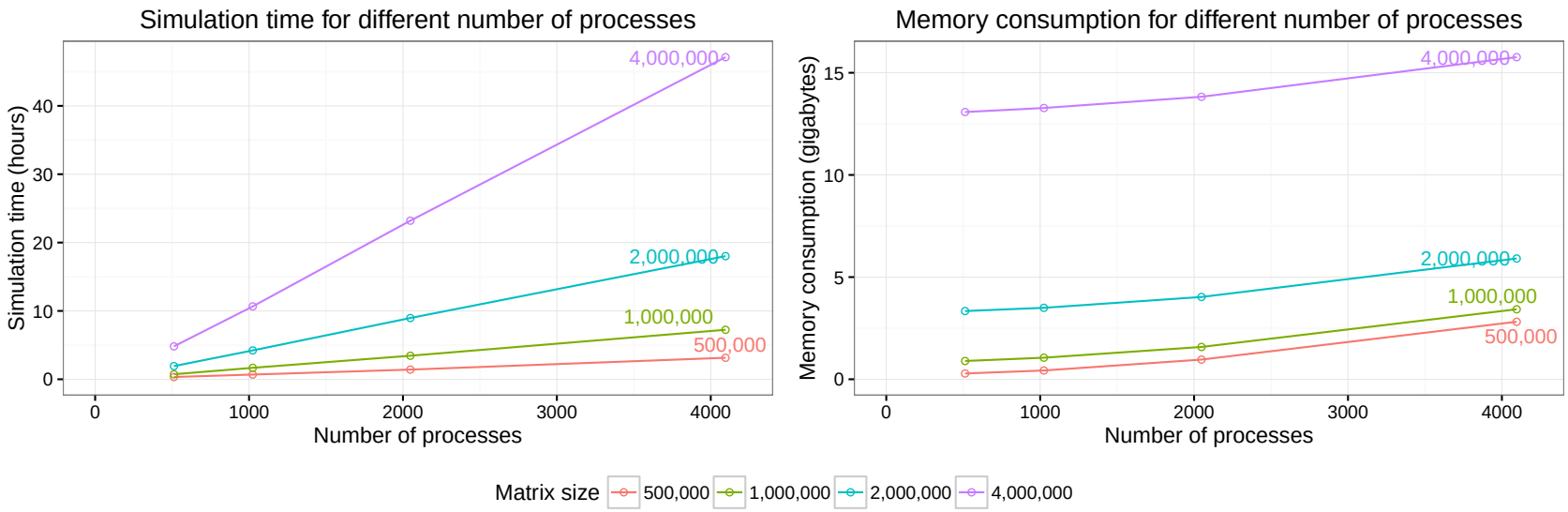
$PT_{size}(600,000) \approx 5\text{GB}$

Solution: using huge pages

Scalability

Scalability





Validation

What?

Let's compare:

- a real experiment
- a vanilla simulation
- an optimized simulation

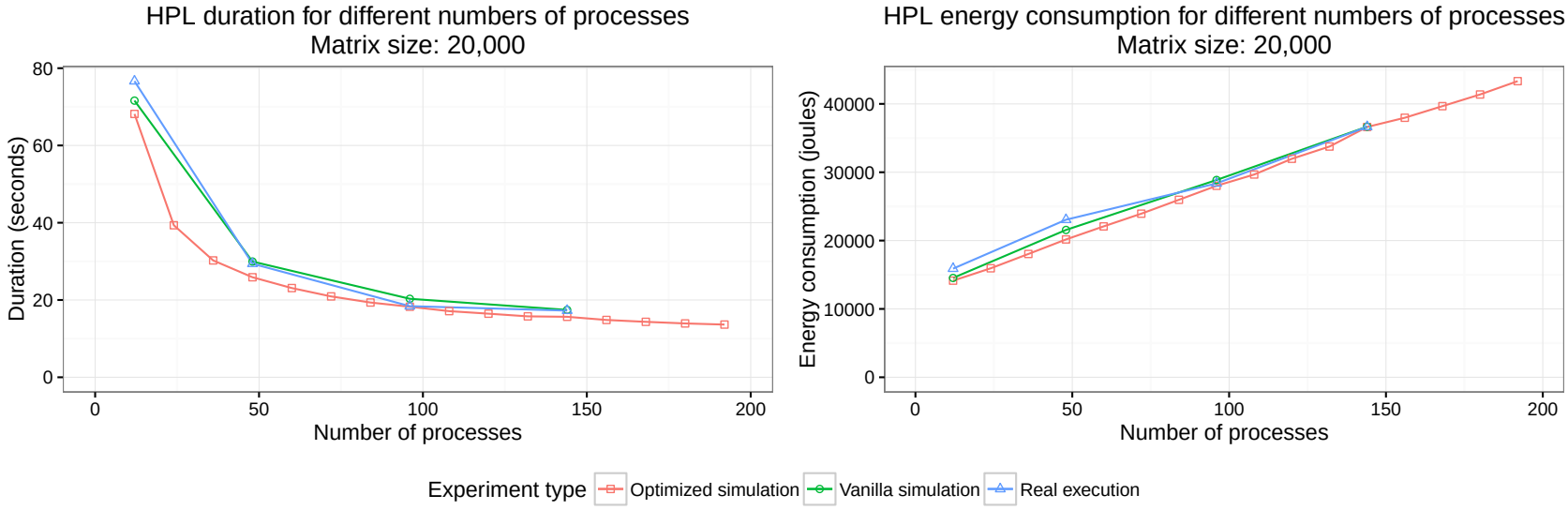
Measuring the duration of HPL and its energy consumption

How?

Using Grid'5000:

- Cluster Taurus, in Lyon
- 16 nodes
- 2 Intel Xeon E5-2630, 6 cores/CPU, 2.3GHz
- 32GB RAM
- 1 switch, 10Gbps links
- Hyperthreading and turbo-mode disabled

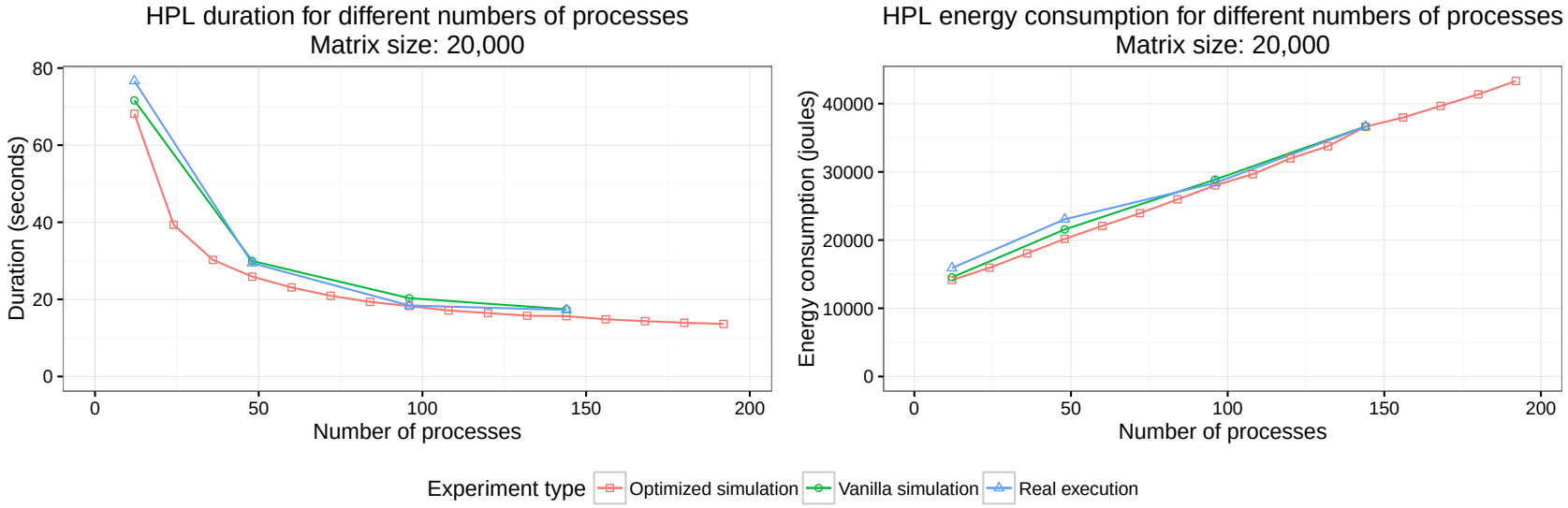
Result



Prediction error: $\leq 12\%$

Simulation systematically too optimistic

Result



Prediction error: $\leq 12\%$

Simulation systematically too optimistic

- No outliers in `dgemm` and `dtrsm` duration
- Functions skipped
- Optimistic network model

Conclusion

To sum up

Simulation of HPL, accurate and efficient

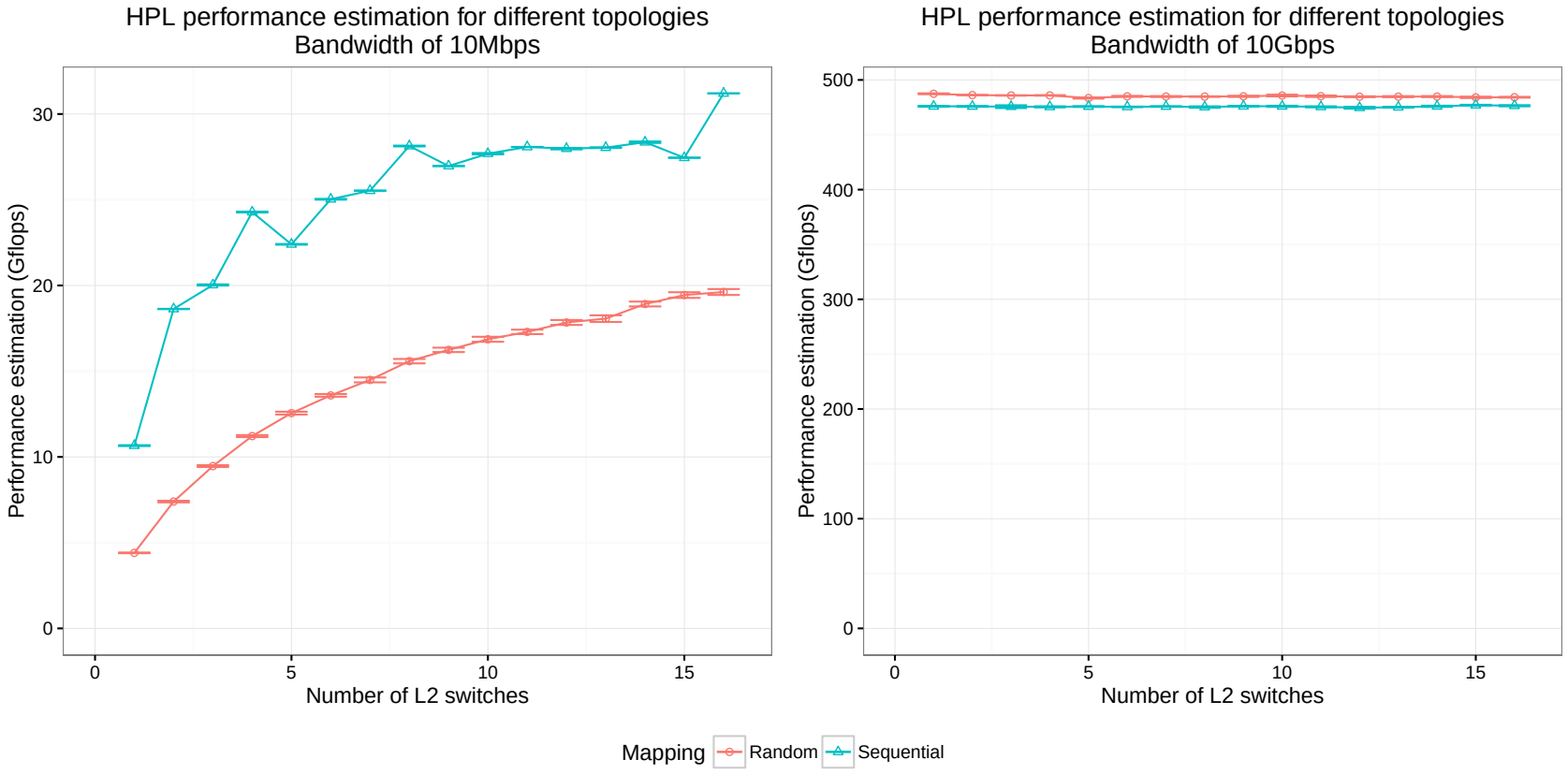
Can reach the scales of the largest supercomputers

Small modifications to HPL (300/34k lines)

Several improvements to Simgrid

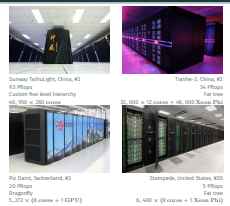
Coming soon...

Capacity planning of supercomputers



Failures, energy consumption

Top500

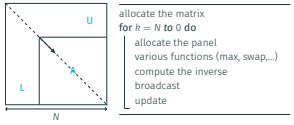


High Performance LINPACK (HPL)

Benchmark used to establish the Top500

LU factorization, $A = L \times U$

Complexity: $\text{flop}(N) = \frac{2}{3}N^3 + 2N^2 + O(N)$



allocate the matrix
for $k = N$ **to** 0 **do**
  allocate the panel
  various functions (max, swap,...)
  compute the inverse
  broadcast
  update

Open questions in HPC

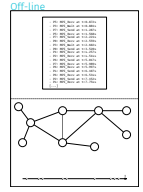
- Topology (torus, fat tree, dragonfly, etc.)
- Routing algorithm
- Scheduling (when? where?)
- Workload (job size, behavior)

Keywords: capacity planning, co-design

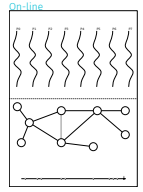
Simulation may help

Simulation of HPC applications

Off-line



On-line



Simgrid: both approaches

Objective: simulation of Stampede's execution of HPL

Real execution:

- Matrix of size 3,875,000
- Using 6,006 MPI processes
- About 2 hours

Requirement for the emulation of Stampede's execution:

- $\geq 3,875,000^2 \times 8$ bytes $\approx$ 120 terabytes of memory
- $\geq 6,006 \times 2$ hours $\approx$ 500 days

Very optimistic

Methodology

Several optimizations. For each of them:

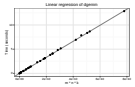
- Evaluate the (possible) loss of prediction accuracy
- Evaluate the (possible) gain of performance

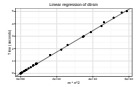
Publicly available:

- Laboratory notebook
- Modified HPL
- Scripts
- Modifications to Simgrid (integrated in the main project)

Computation kernel sampling

`dgemm` and `dtrsm`
$\geq$ 90% of the simulation time
Solution: modeling these functions to inject their duration





$T_{dgemm}(M, N, K) = M \times N \times K \times 1.706348 \times 10^{-10}$

$T_{dtrsm}(M, N) = M \times N^2 \times 8.624970 \times 10^{-11}$

Computation pruning

68% of the simulation time spent in HPL

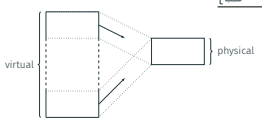
Culprits:

- Initialization and verification functions
- Other BLAS and HPL functions

Solution: just skip them

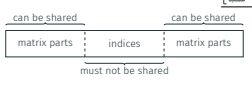
Reducing the memory consumption

Memory consumption still too large
Solution: use `SMPI_SHARED_MALLOC`



Reducing the memory consumption #154

Problem: panel buffers
Must remain contiguous

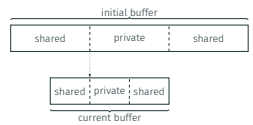


Solution: `SMPI_PARTIAL_SHARED_MALLOC`

Arbitrary number of shared and private blocks.

Reusing the panel buffers

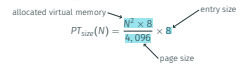
At each iteration, new allocation and deallocation by all processes
Solution: reuse the buffers (sizes strictly decreasing)
Needs to be done carefully



Using huge pages #168

Problem: at large scales, CPU utilization drops and simulation time explodes
Reason: the page table becomes very large

Matrix of size $N \Rightarrow$ page table of size:

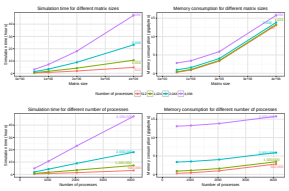


$PT_{size}(N) = \frac{N^2 \times 8}{4,096} \times 8$

$PT_{size}(600,000) \approx 5\text{GB}$

Solution: using huge pages

Scalability



What?

Let's compare:

- a real experiment
- a vanilla simulation
- an optimized simulation

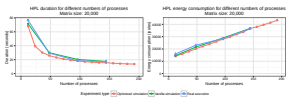
Measuring the duration of HPL and its energy consumption

How?

Using Grid'5000:

- Cluster Taurus, in Lyon
- 16 nodes
- 2 Intel Xeon E5-2630, 6 cores/CPU, 2.3GHz
- 32GB RAM
- 1 switch, 10Gbps links
- Hyperthreading and turbo-mode disabled

Result



Prediction error: $\leq$ 12%

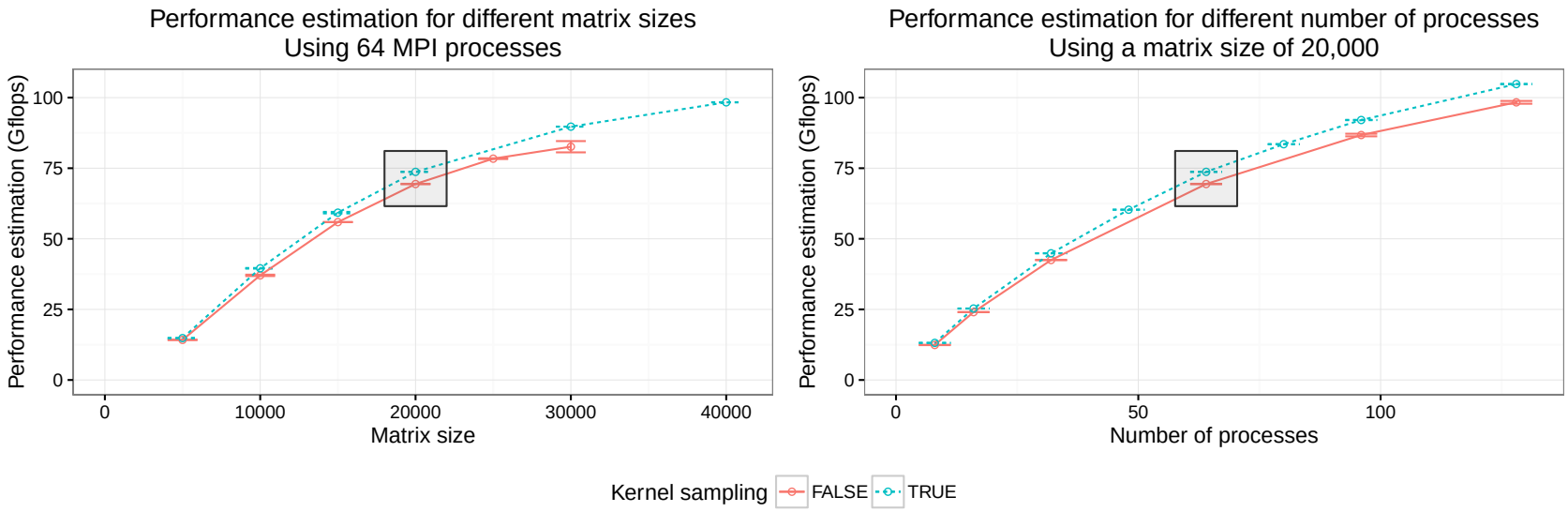
Simulation systematically too optimistic

- No outliers in `dgemm` and `dtrsm` duration
- Functions skipped
- Optimistic network model

Thanks for your attention!

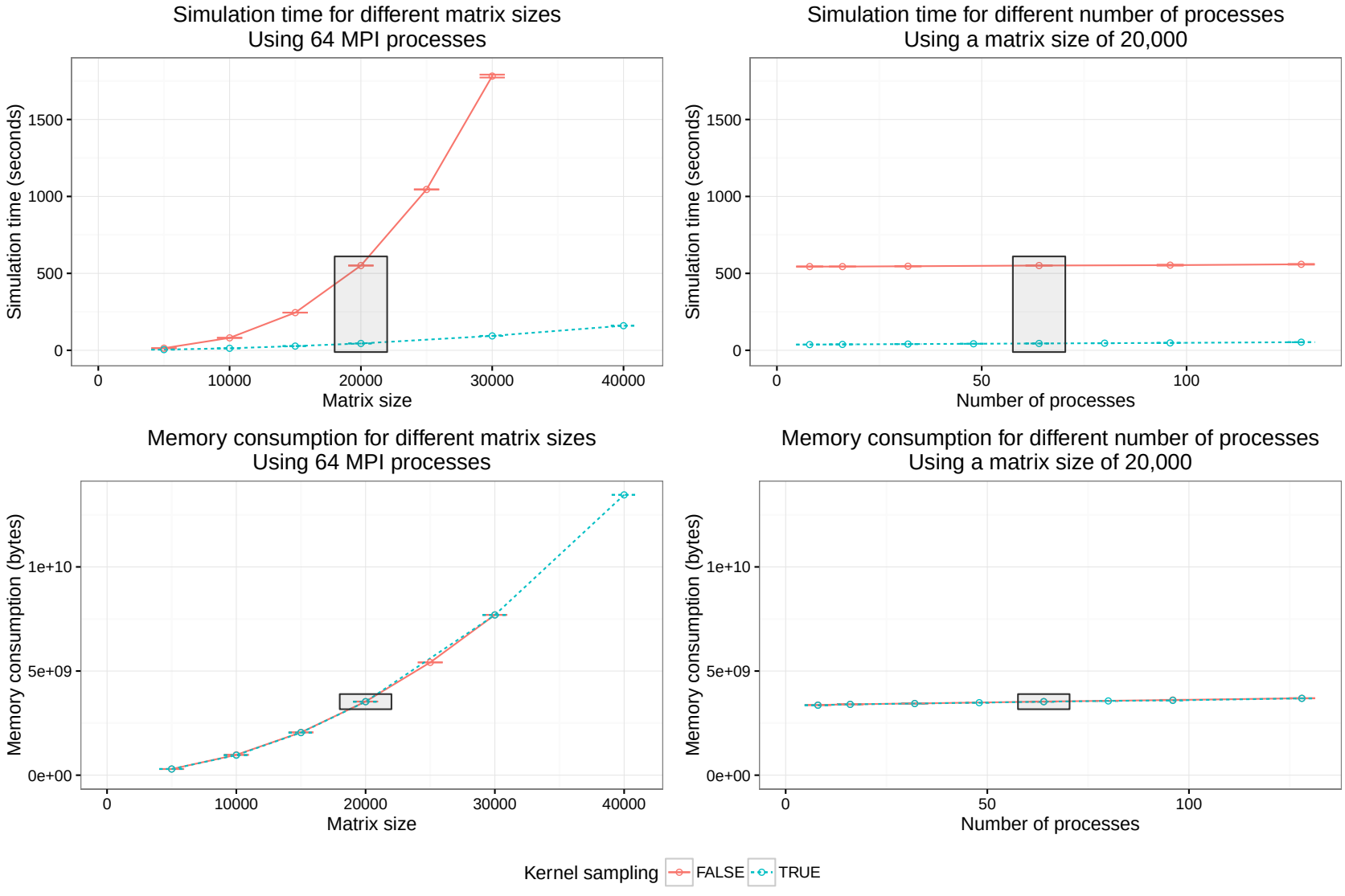
Any questions?

Computation kernel sampling

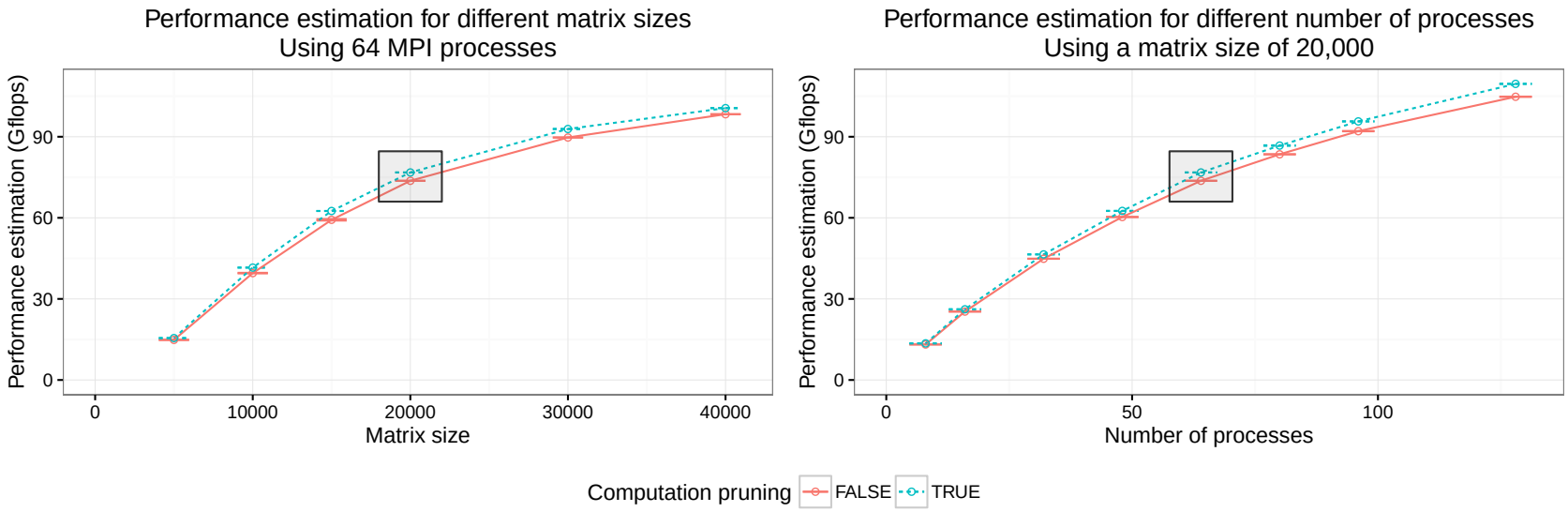


Prediction error: $\leq 10\%$

Computation kernel sampling

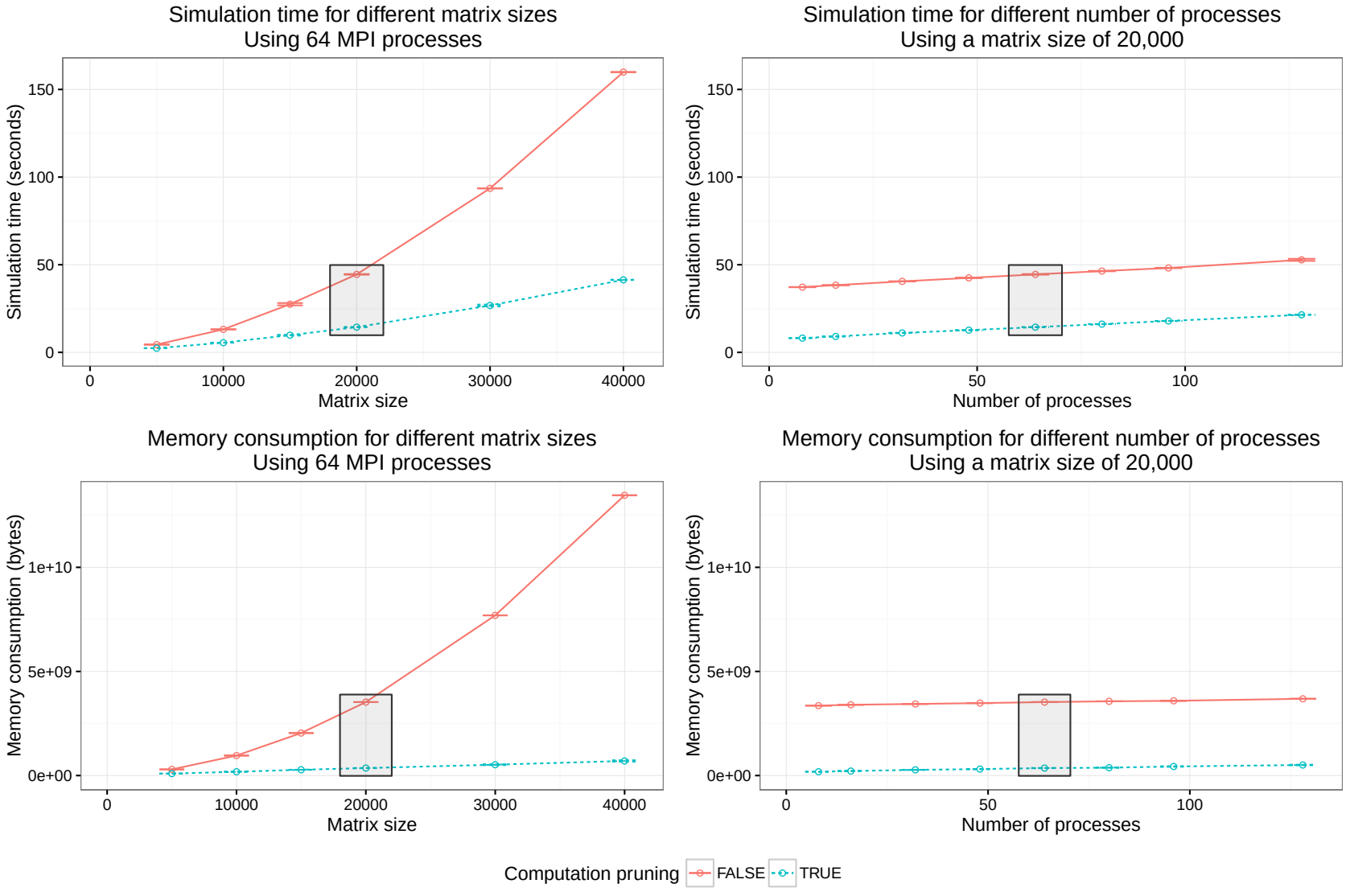


Computation pruning

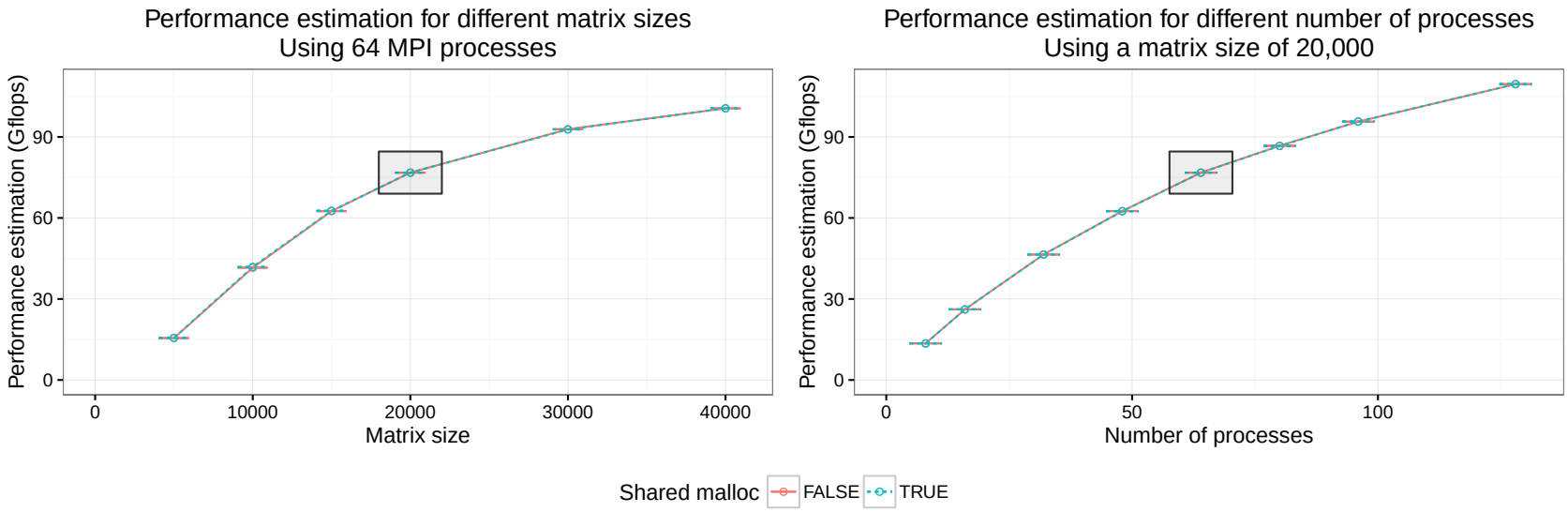


Prediction error: $\leq 5\%$

Computation pruning

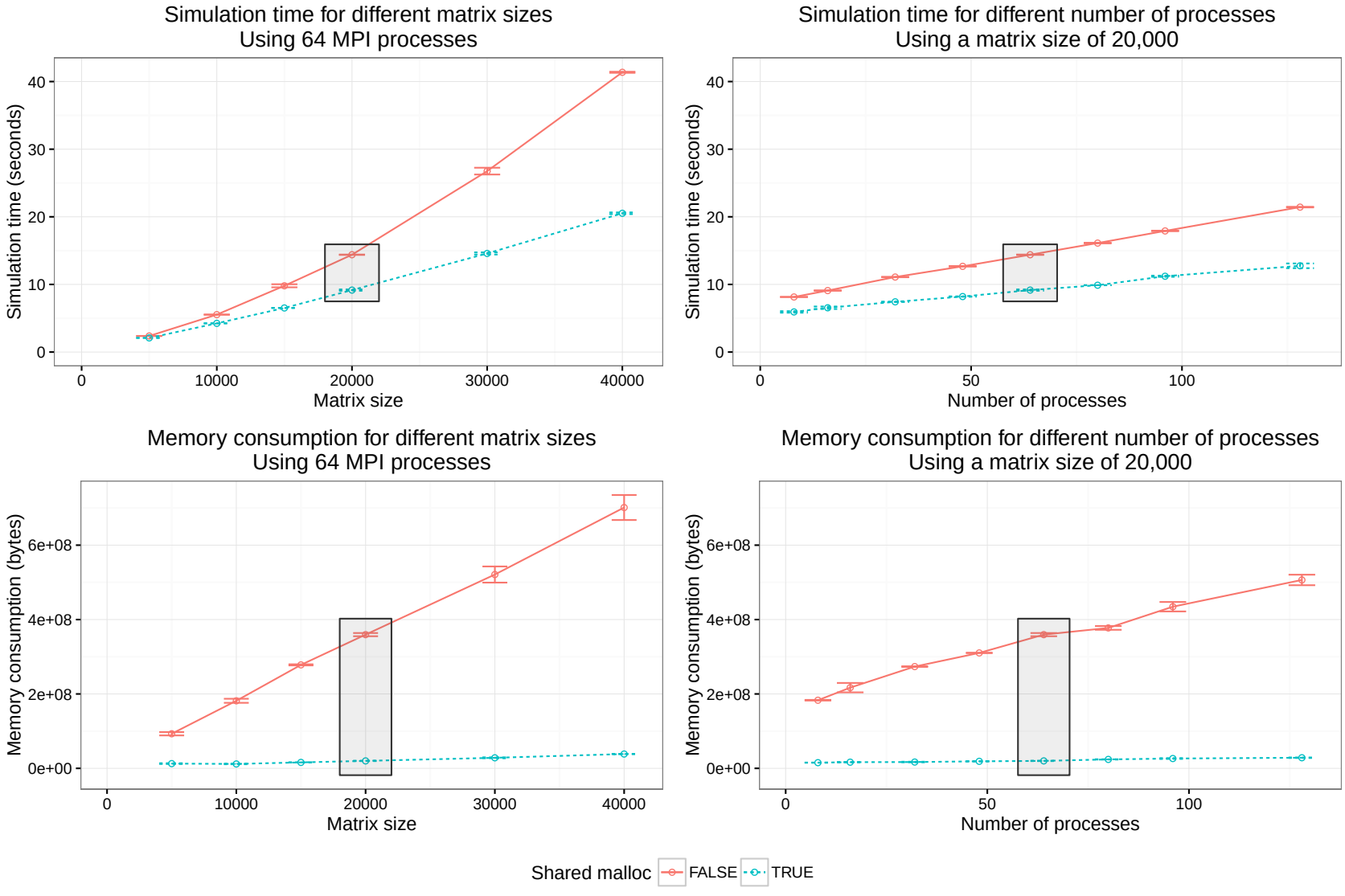


Reducing the memory consumption

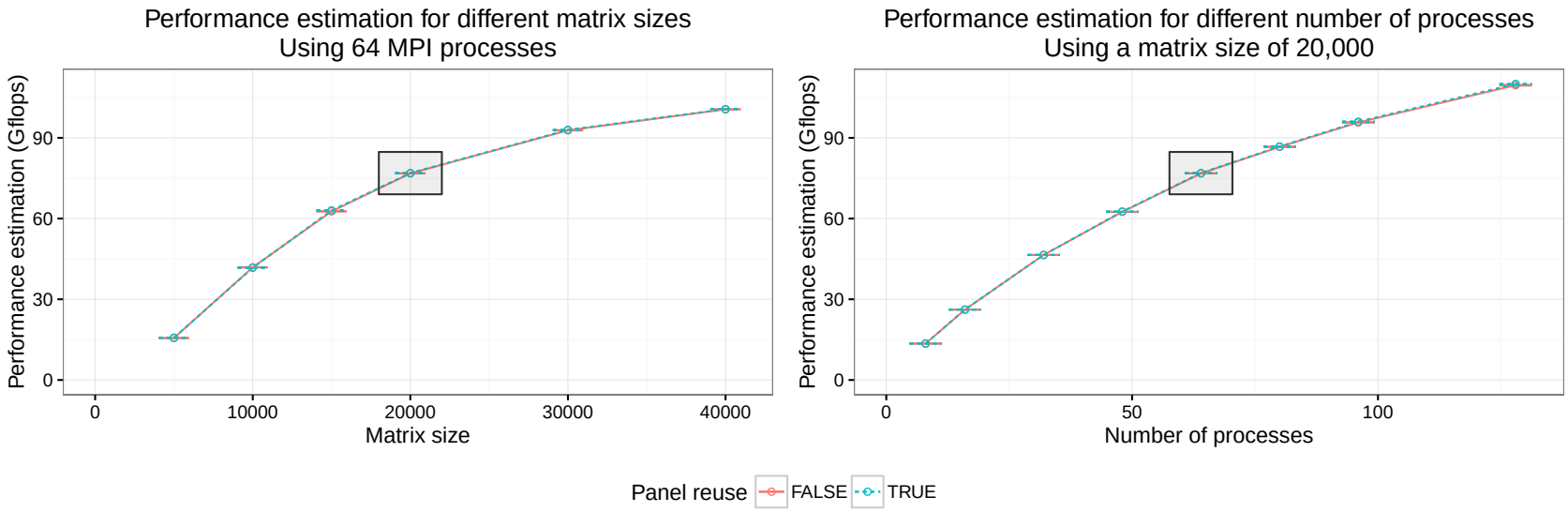


Prediction error: $\leq 1\%$

Reducing the memory consumption

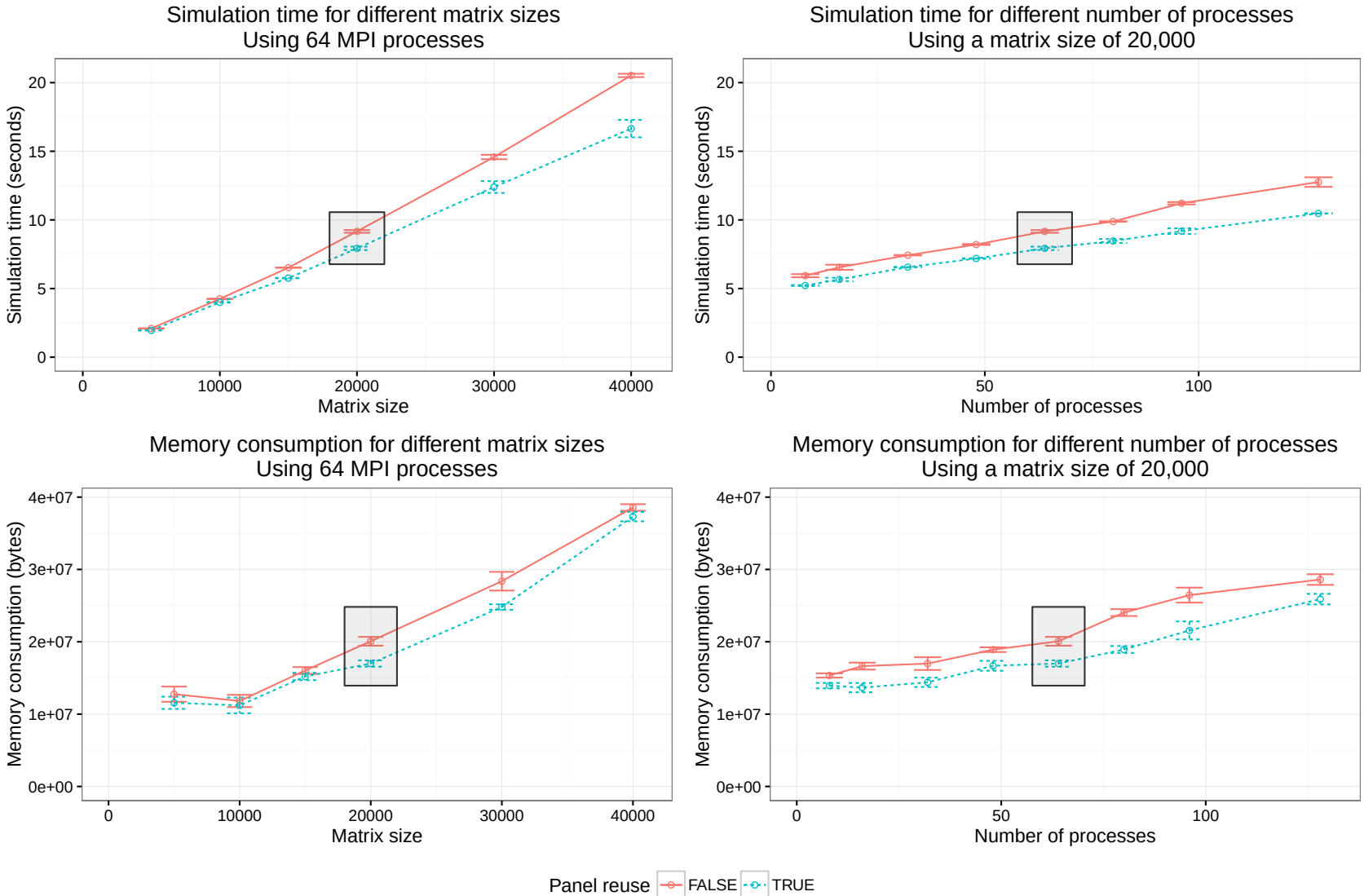


Reusing the panel buffers

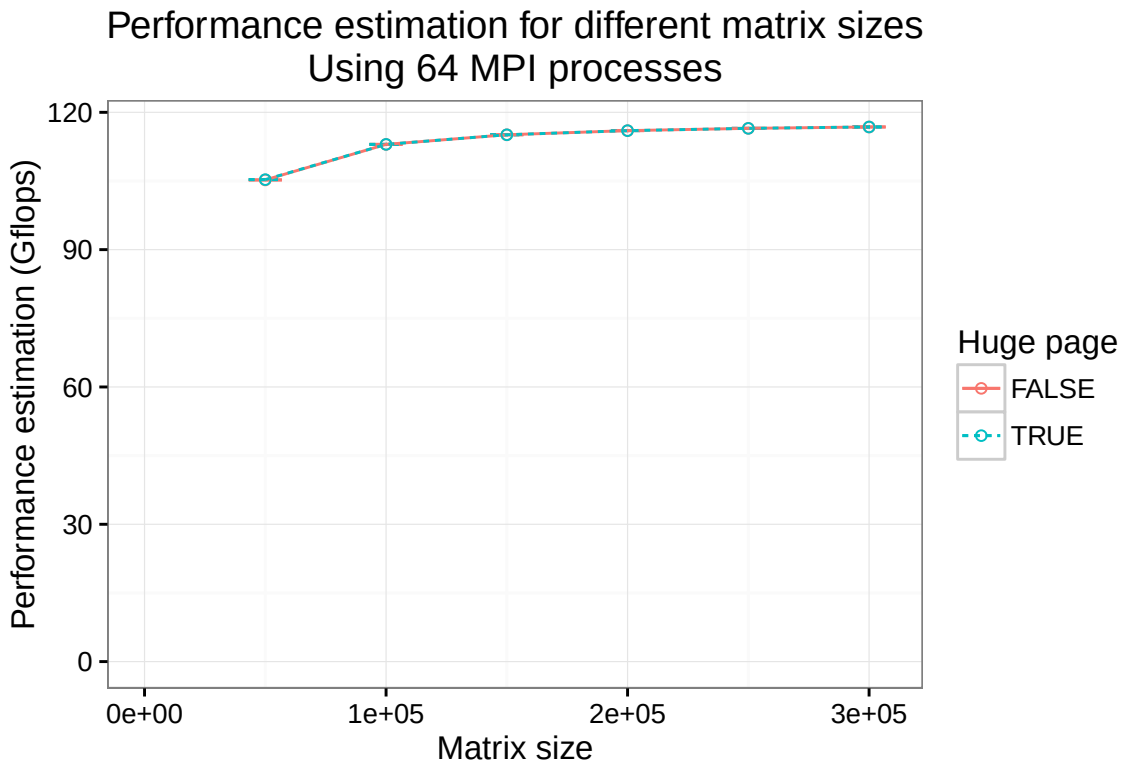


Prediction error: $\leq 1\%$

Reusing the panel buffers



Using huge pages



Prediction error: $\leq 0.1\%$

Using huge pages