

# 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 × 260 cores



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



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



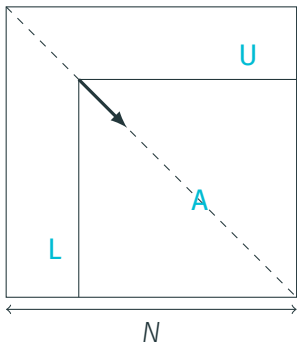
Stampede, United States, #20  
5 Pflops  
Fat tree  
6,400 × (8 cores + 1 Xeon Phi)

# 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

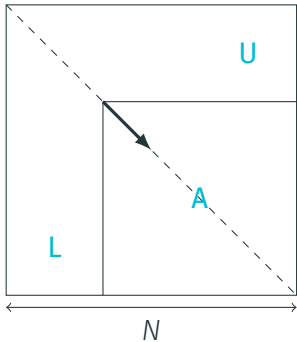
---

# 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

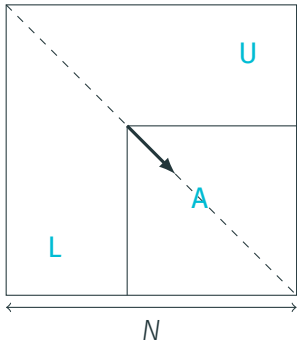
---

# 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

---

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

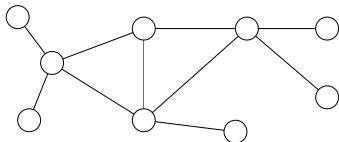
Keywords: capacity planning, co-design

Simulation may help

# SIMULATION OF HPC APPLICATIONS

## Off-line

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

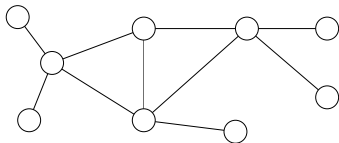




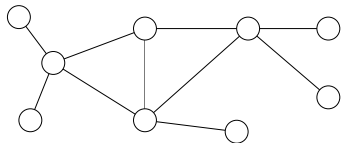
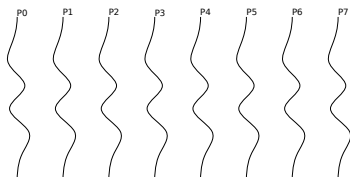
# SIMULATION OF HPC APPLICATIONS

## 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
[...]
```



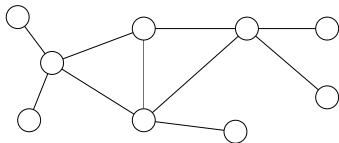
## On-line



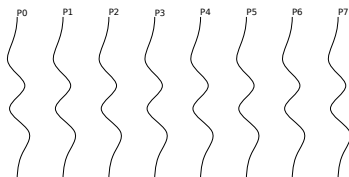
# SIMULATION OF HPC APPLICATIONS

## Off-line

```
- 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=3.526s
- 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



Simgrid: both approaches

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:

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

Very optimistic

# SCALABLE HPL SIMULATION

---

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`

≥ 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 `dt_rsm`

$\geq 90\%$  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
```

---



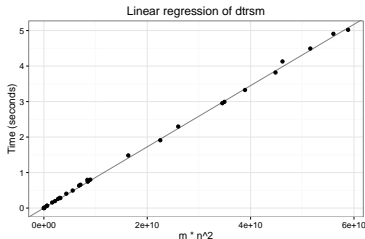
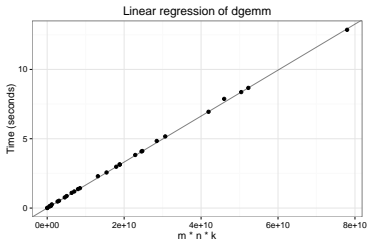
# COMPUTATION KERNEL SAMPLING

`dgemm` and `dtrsm`

≥ 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
```



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

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

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
```

---

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
```

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
```

---

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

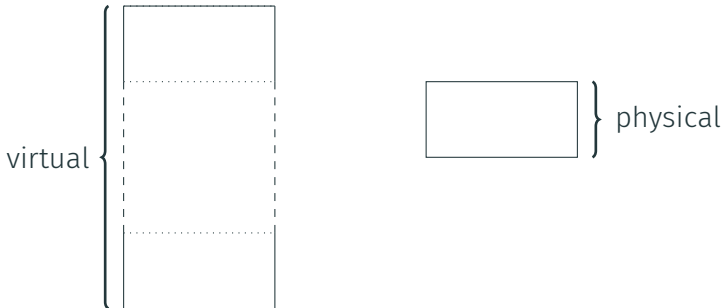
Memory consumption still too large

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

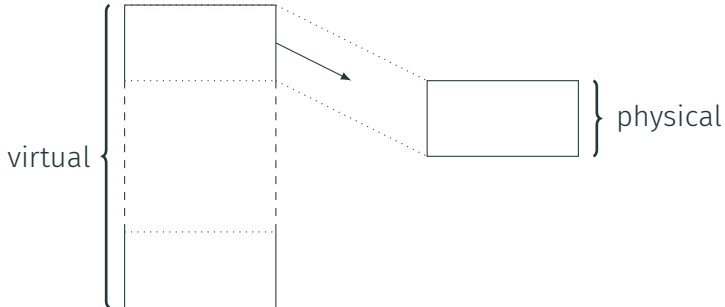
Memory consumption still too large

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

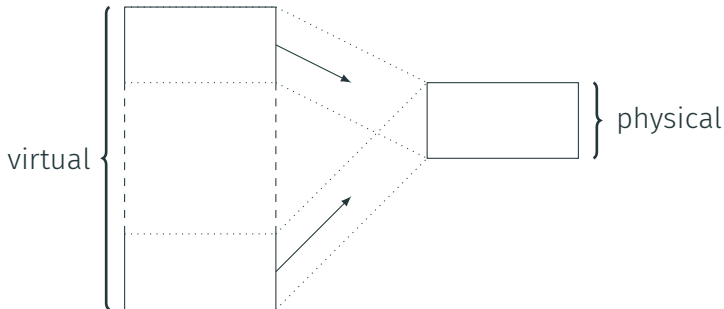
Memory consumption still too large

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
```

---

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

---

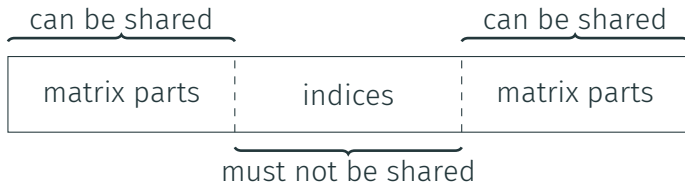
```
allocate the matrix
for  $k = N$  to 0 do
  allocate the panel
  various functions (max, swap,...)
  compute the inverse
  broadcast
  update
```

---



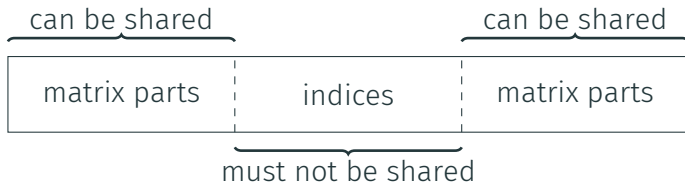
Problem: panel buffers  
Must remain contiguous

```
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
```

---



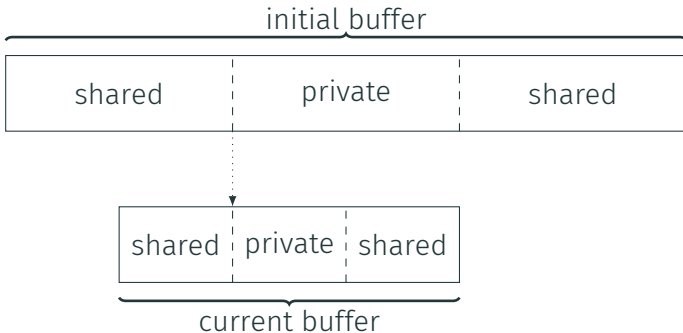
## REUSING THE PANEL BUFFERS

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 matrix
  various functions (max, swap,...)
  compute the inverse
  broadcast
  update
```



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

---

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

---

huge pages

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 page table
  various functions (max, swap,...)
  compute the inverse
  broadcast
  update
```

---

huge pages

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 page table
    various functions (max, swap,...)
    compute the inverse
    broadcast
    update
    
```

huge pages

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

allocated virtual memory  $\rightarrow$

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

entry size  $\rightarrow$

page size  $\rightarrow$

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

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 matrix
  various functions (max, swap,...)
  compute the inverse
  broadcast
  update
    
```

huge pages

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

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

allocated virtual memory  $\rightarrow$   $N^2 \times 8$

entry size  $\rightarrow$   $8$

page size  $\rightarrow$   $4,096$

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

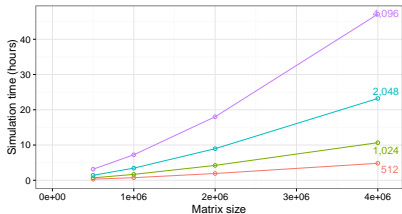
Solution: using huge pages

# SCALABILITY

---

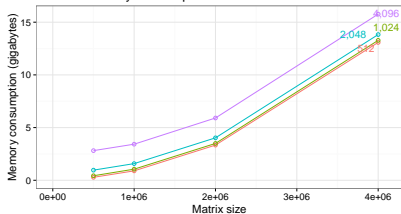
# SCALABILITY

Simulation time for different matrix sizes

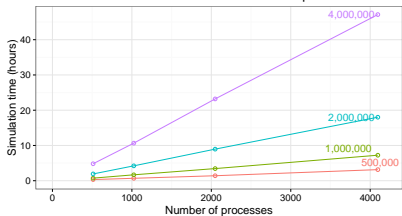


Number of processes 512 1,024 2,048 4,096

Memory consumption for different matrix sizes

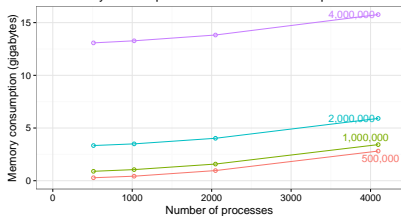


Simulation time for different number of processes



Matrix size 500,000 1,000,000 2,000,000 4,000,000

Memory consumption for different number of processes



# VALIDATION

---



# WHAT?

Let's compare:

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

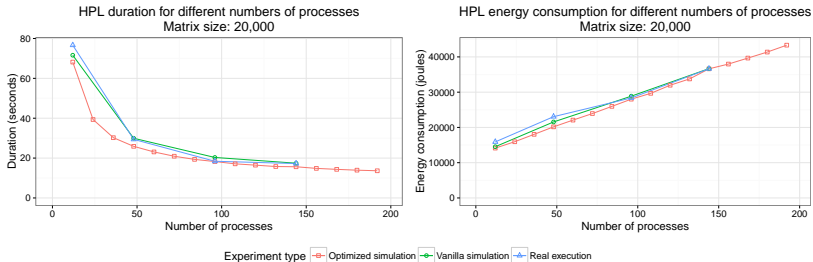
Measuring the duration of HPL and its energy consumption

# 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

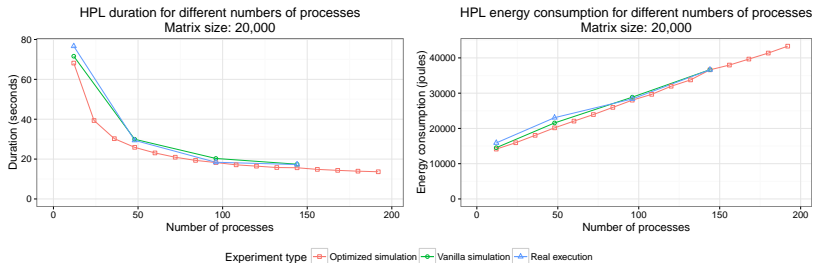
# RESULT



Prediction error:  $\leq 12\%$

Simulation systematically too optimistic

# RESULT



Prediction error:  $\leq 12\%$

Simulation systematically too optimistic

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

## CONCLUSION

---

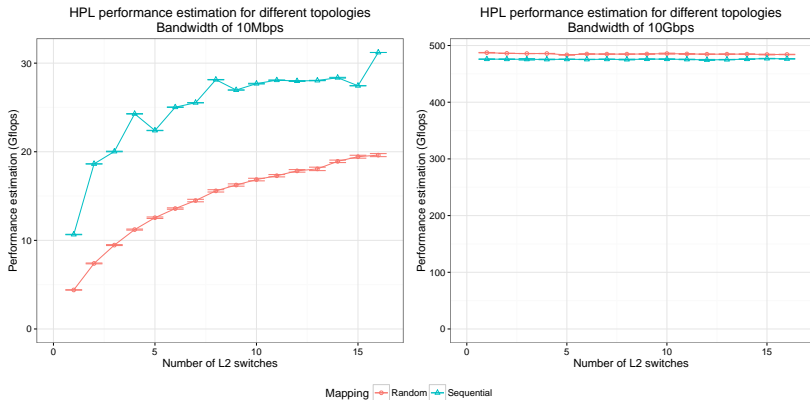
Simulation of HPL, accurate and efficient

Can reach the scales of the largest supercomputers

Small modifications to HPL (300/34k lines)

Several improvements to Simgrid

## Capacity planning of supercomputers



Failures, energy consumption

### Top500

1. IBM System z10 EC12  
 2. Oracle Exalytics  
 3. Oracle Exalytics  
 4. Oracle Exalytics

5. Oracle Exalytics  
 6. Oracle Exalytics  
 7. Oracle Exalytics  
 8. Oracle Exalytics

9. Oracle Exalytics  
 10. Oracle Exalytics

11. Oracle Exalytics  
 12. Oracle Exalytics

13. Oracle Exalytics  
 14. Oracle Exalytics

15. Oracle Exalytics

16. Oracle Exalytics

17. Oracle Exalytics

18. Oracle Exalytics

19. Oracle Exalytics

20. Oracle Exalytics

### Issue Performance L1/INPC (HPL)

Benchmark used to establish the Top500  
 LU factorization,  $N = 1 \times 10^9$   
 Complexity:  $8N^3/3 = 40^9 + 20^9 + O(N)$

allocate the matrix  
 for  $i = 0$  to  $N$  do  
 allocate the panel  
 various functions (solve, swap, ...) compute the inverse  
 deallocate  
 continue

### Open questions in HPC

- Topology (bus, fat tree, dragonfly, etc.)
- Routing algorithm
- Scheduling (what's "what's")
- Bandwidth (pin size, network)

Hypothesis: capacity planning, co-design  
 Simulation may help

### Simulation of HPC applications

single node approach

### Objective: simulation of Stampede's execution of HPL

Real execution:
 

- Matrix of size 2,875,000
- Using 4,000 IBM processors
- About 2 days

 Requirement for the emulation of Stampede's execution:
 

- > 2.875,000<sup>2</sup> = 8.25 bytes in 500 *nodes* of memory
- > 2.5, 000 + 2.5 bytes in 500 *nodes*

 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 network
- Modified HPL
- Scripts
- Modifications to Stampede (integrated in the main project)

### COMPUTATION KERNEL SAMPLING

Higher and  $\alpha$  size  
 > 90% of the simulation time  
 Solution: modeling these functions to ingest their duration

$T_{\text{simulated}}(M, N, K) = M \times N \times K = 1.706 \text{Msd} \times 10^{10}$   
 $T_{\text{actual}}(M, N) = M \times N^2 = 8.624 \text{Msd} \times 10^{10}$

### COMPUTATION PHASING

90% of the simulation time spent in HPL

Graphs:
 

- Initialization and verification functions
- Other BLAS and HPL functions

 Solution: just skip them

### REDUCING THE MEMORY CONSUMPTION

Memory consumption still too large  
 Solution: use `HPL_PARTIAL_SHARED_MALLOC`

### REDUCING THE MEMORY CONSUMPTION

Problem: panel buffers  
 Must remain contiguous

Solution: `HPL_PARTIAL_SHARED_MALLOC`  
 Arbitrary number of shared and private blocks.

### REDUCING THE PANEL BUFFERS

At each iteration, one allocation and deallocation by all processes  
 Solution: reuse the buffers (once strictly decreasing) needs to be done carefully

### Using huge pages

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:
 
$$PT_{\text{size}}(N) = \frac{N^2}{\text{page size}} \times \text{entry size}$$

$PT_{\text{size}}(200,000) \approx 63 \text{B}$   
 Solution: using huge pages

### SCALABILITY

### What?

Let's compare:
 

- real experiment
- vanilla simulation
- an optimized simulation

 Measuring the duration of HPL and its energy consumption

### How?

Using Grid5000:
 

- Cluster Taurus, in Lyon
- 56 nodes
- 2 intel Xeon E5-2630, 6 cores/CPU, 2.5GHz
- 32GB RAM
- 4 network, 10Gbps links
- Hyperthreading and turbo-mode disabled

### Result

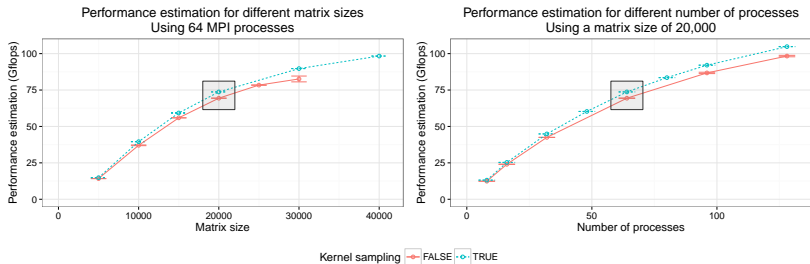
Prediction error: < 1%  
 Simulation systematically too optimistic:
 

- No outliers in *duration* and *dtm* per iteration
- Functions Hopped
- Optimistic network model

Thanks for your attention!  
 Any questions?



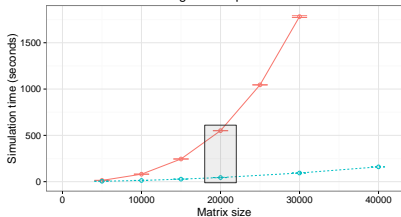
# COMPUTATION KERNEL SAMPLING



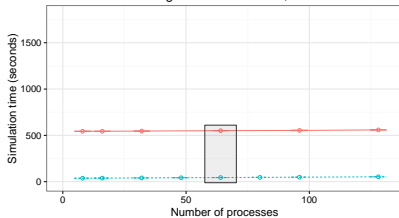
Prediction error:  $\leq 10\%$

# COMPUTATION KERNEL SAMPLING

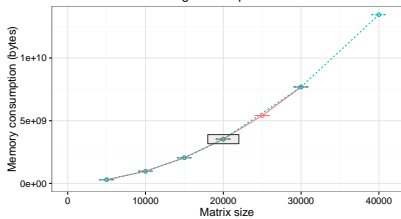
Simulation time for different matrix sizes  
Using 64 MPI processes



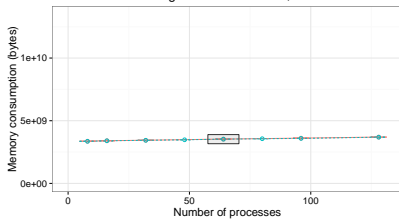
Simulation time for different number of processes  
Using a matrix size of 20,000



Memory consumption for different matrix sizes  
Using 64 MPI processes

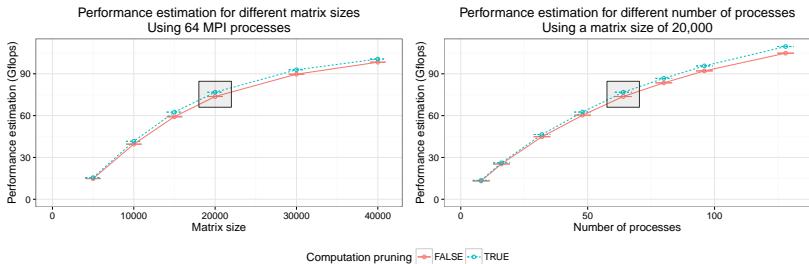


Memory consumption for different number of processes  
Using a matrix size of 20,000



Kernel sampling —■— FALSE -●- TRUE

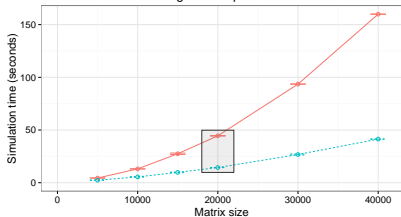
# COMPUTATION PRUNING



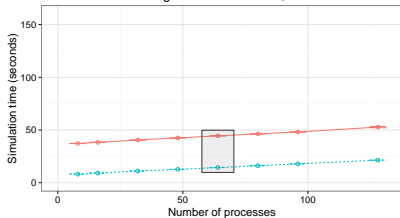
Prediction error:  $\leq 5\%$

# COMPUTATION PRUNING

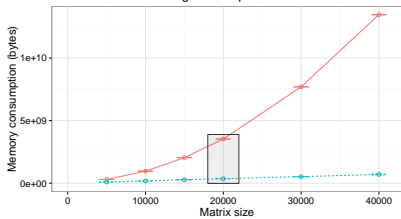
Simulation time for different matrix sizes  
Using 64 MPI processes



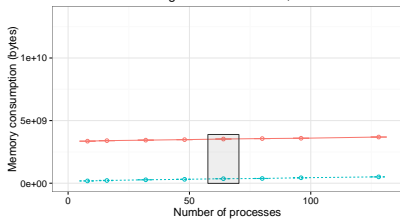
Simulation time for different number of processes  
Using a matrix size of 20,000



Memory consumption for different matrix sizes  
Using 64 MPI processes

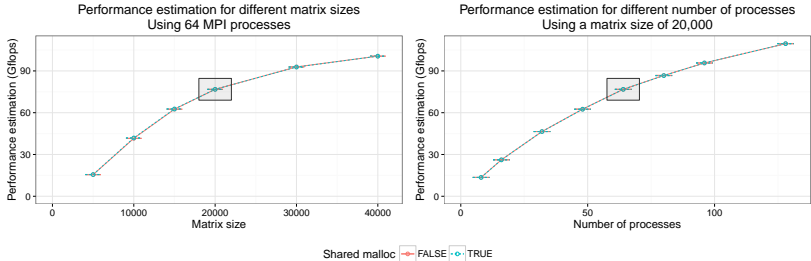


Memory consumption for different number of processes  
Using a matrix size of 20,000



Computation pruning —●— FALSE - - -○- - - TRUE

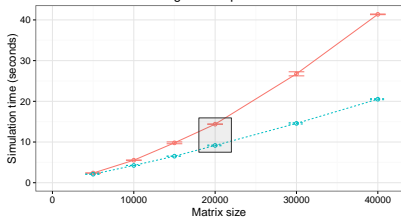
# REDUCING THE MEMORY CONSUMPTION



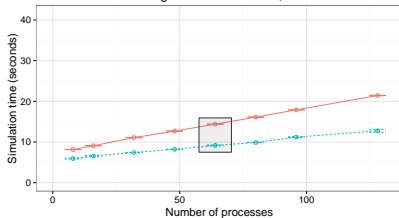
Prediction error:  $\leq 1\%$

# REDUCING THE MEMORY CONSUMPTION

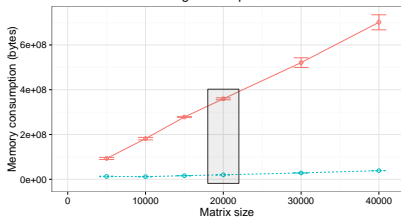
Simulation time for different matrix sizes  
Using 64 MPI processes



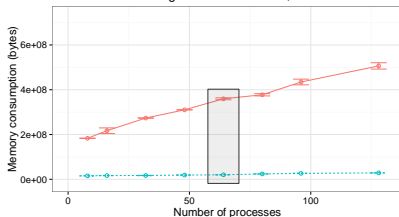
Simulation time for different number of processes  
Using a matrix size of 20,000



Memory consumption for different matrix sizes  
Using 64 MPI processes

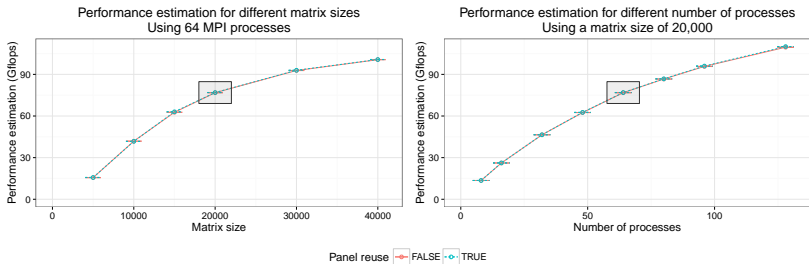


Memory consumption for different number of processes  
Using a matrix size of 20,000



Shared malloc ■ FALSE ● TRUE

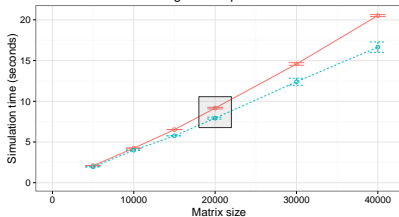
# REUSING THE PANEL BUFFERS



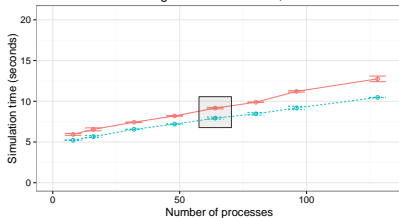
Prediction error:  $\leq 1\%$

# REUSING THE PANEL BUFFERS

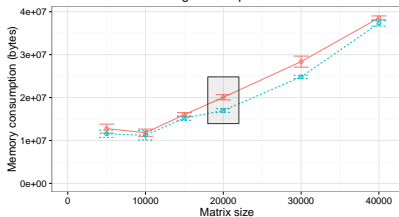
Simulation time for different matrix sizes  
Using 64 MPI processes



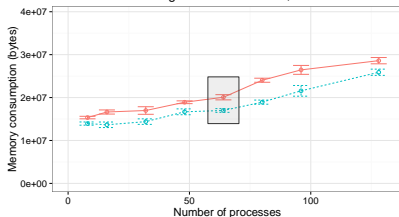
Simulation time for different number of processes  
Using a matrix size of 20,000



Memory consumption for different matrix sizes  
Using 64 MPI processes



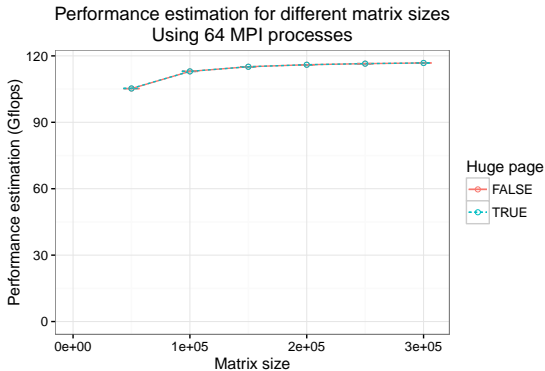
Memory consumption for different number of processes  
Using a matrix size of 20,000



Panel reuse FALSE TRUE



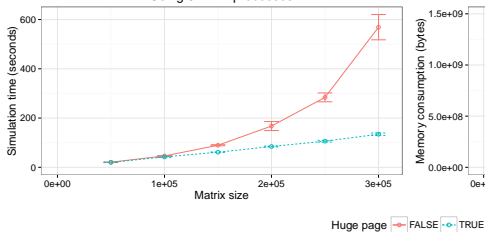
# USING HUGE PAGES



Prediction error:  $\leq 0.1\%$

# USING HUGE PAGES

Simulation time for different matrix sizes  
Using 64 MPI processes



Memory consumption for different matrix sizes  
Using 64 MPI processes

