SIMULATION OF A TASK-BASED SPARSE DIRECT SOLVER

E. Agullo, A. Buttari, A. Guermouche, A. Legrand, I. Masliah and L. Stanisic

SIAM CSE 2019, 25 February - 1 March 2019, Spokane, WA, USA

Linear systems and direct methods

Sparse linear systems

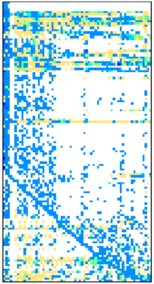
Many applications from physics, engineering, chemistry, geodesy, etc, require the solution of a linear system like

$Ax = b$, with A, rectangular, sparse and potentially large

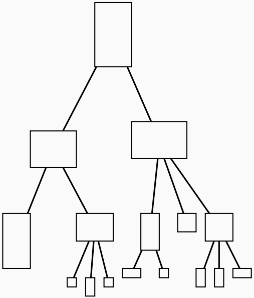
$$
\begin{array}{llcccc}
m \geq n & \min_x \|Ax - b\|_2 & \rightarrow & QR = A, & z = Q^T b, & x = R^{-1} z \\
m < n & min\|x\|_2, \quad Ax = b & \rightarrow & QR = A^T, & z = R^{-T} b, & x = Qz
\end{array}
$$

A sparse matrix is mostly filled with zeros:

- Reduce memory storage.
- Reduce computational costs.
- Generate parallelism.



The multifrontal method[1] in a nutshell



```
forall fronts f in topological order

   ! allocate and initialize front
   call activate(f)

   ! front assembly
   forall children c of f
      call assemble(c, f)
      ! Deactivate child
      call deactivate(c)
   end do

   ! front factorization
   call factorize(f)
end do
```

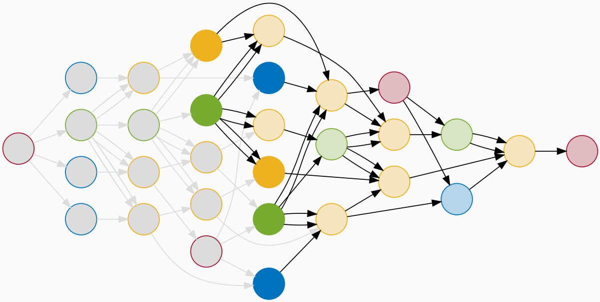
- The elimination tree is pre-computed based on the matrix structure
- Nodes are dense matrices called fronts which can be very small, square or rectangular
- In the actual factorization the tree is traversed bottom up and when a node is visited, the corresponding front is allocated, assembled – using children – and factorized

[1] Duff et al., "The multifrontal solution of indefinite sparse symmetric linear systems".

TASK PARALLELISM AND RUNTIME SYSTEMS

Task parallelism

Workload is expressed as a Directed Acyclic Graph of tasks



- Inherently expresses concurrency and data flow
- Asynchronous execution
- Allows for dynamic scheduling policies

Sequential Task Flow (STF) runtimes

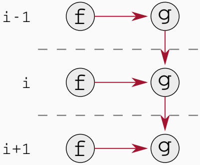
The STF is a portable programming model where tasks are submitted sequentially and their execution is delegated to a runtime

```
for (i = 1; i < N; i++) {
  x[i] = f(x[i])
  y[i] = g(x[i], y[i-1])
}
```

Sequential Task Flow (STF) runtimes

The STF is a portable programming model where tasks are submitted sequentially and their execution is delegated to a runtime

```
for (i = 1; i < N; i++) {
  submit(f, x[i]:RW)
  submit(g, y[i]:R, x[i]:R, y[i-1]:R)
}
```



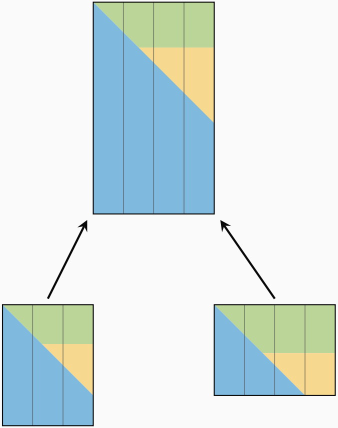
We have chosen StarPU because of its numerous features:

- data management/transfer
- plug&play scheduling policies
- support for accelerators
- support for SimGrid[2], . . .

[2] Stanisic, Thibault, et al., "Modeling and Simulation of a Dynamic Task-Based Runtime System for Heterogeneous Multi-core Architectures".

1D STF multifrontal QR

The task-based multifrontal QR factorization



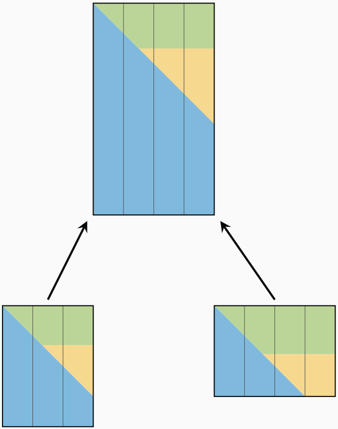
```
do f=1, nfronts ! in postorder
   ! compute front structure
   call activate(f)
   ! allocate and initialize front
   call init(f)

   do c=1, f%nc ! for all the children of f
      do j=1,c%n
         ! assemble column j of c into f
         call assemble(c(j), f)
      end do
      ! Deactivate child
      call deactivate(c)
   end do

   do p=1, f%n
      ! panel reduction of column p
      call _geqrt(f(p))
      do u=p+1, f%n
         ! update of column u with panel p
         call _gemqrt(f(p), f(u))
      end do
   end do
end do
```

Sequential multifrontal *QR* code with 1D block partitioning

The task-based multifrontal QR factorization



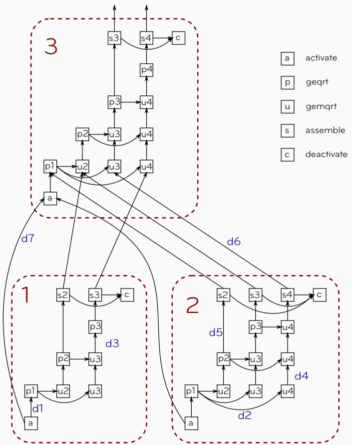
```
do f=1, nfronts ! in postorder
   ! compute structure and register handles
   call activate(f)
   ! allocate and initialize front
   call submit(init, f:RW)

   do c=1, f%nc ! for all the children of f
      do j=1,c%n
         ! assemble column j of c into f
         call submit(assemble, c(j):R, f:RW)
      end do
      ! Deactivate child
      call submit(deactivate, c:RW)
   end do

   do p=1, f%n
      ! panel reduction of column p
      call submit(_geqrt, f(p):RW)
      do u=p+1, f%n
         ! update of column u with panel p
         call submit(_gemqrt, f(p):R, f(u):RW)
      end do
   end do
end do
! wait for the tasks to be executed
call wait_tasks_completion()
```

- STF multifrontal *QR* code with 1D block partitioning
- Elimination tree is transformed into a DAG

The task-based multifrontal QR factorization




```
do f=1, nfronts ! in postorder
   ! compute structure and register handles
   call activate(f)
   ! allocate and initialize front
   call submit(init, f:RW)

   do c=1, f%nc ! for all the children of f
      do j=1,c%n
         ! assemble column j of c into f
         call submit(assemble, c(j):R, f:RW)
      end do
      ! Deactivate child
      call submit(deactivate, c:RW)
   end do

   do p=1, f%n
      ! panel reduction of column p
      call submit(_geqrt, f(p):RW)
      do u=p+1, f%n
         ! update of column u with panel p
         call submit(_gemqrt, f(p):R, f(u):RW)
      end do
   end do
end do
! wait for the tasks to be executed
call wait_tasks_completion()
```

- Seamless exploitation of tree and node parallelism
- Inter-level concurrency (parent-child pipelining)

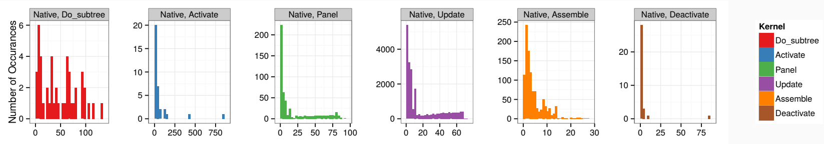
SIMULATION OF QR_MUMPS ON TOP OF STARPU-SIMGRID: 1D CODE

Simulating sparse solvers

Porting qr_mumps on top of SimGrid[3]

- Changing main for the subroutine
- Changing compilation process
- Careful kernel modeling as matrix dimension keeps changing

- Dense kernels during a single experiment are always executed with the same block/tile size $\leadsto$ duration very stable
- Sparse kernels depend on their input parameters $\leadsto$ more variability
- Cannot model sparse kernels with simple mean values



[3] Stanisic, Agullo, et al., "Fast and Accurate Simulation of Multithreaded Sparse Linear Algebra Solvers".

Example for modeling kernels: `Panel`

- Theoretical `Panel` complexity:

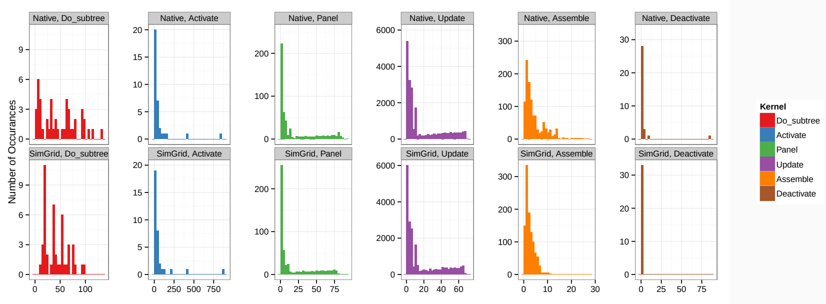
$$T_{\texttt{Panel}} = a + 2b(NB^2 \times MB) - 2c(NB^3 \times BK) + \frac{4d}{3}NB^3$$

- We can do a linear regression based on ad hoc calibration

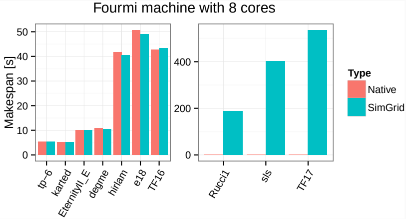
	Panel Duration	
Constant	-2.49×10^{1} $(-2.83 \times 10^{1},\ -2.14 \times 10^{1})$	***
$NB^2 \times MB$	5.49×10^{-7} $(5.46 \times 10^{-7},\ 5.51 \times 10^{-7})$	***
$NB^3 \times BK$	-5.52×10^{-7} $(-5.57 \times 10^{-7},\ -5.48 \times 10^{-7})$	***
NB^3	1.50×10^{-5} $(1.30 \times 10^{-5},\ 1.70 \times 10^{-5})$	***
Observations		493
R^2		0.999
Note:		*p<0.1; **p<0.05; ***p<0.01

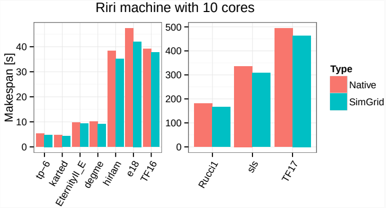
Comparing kernel duration distributions

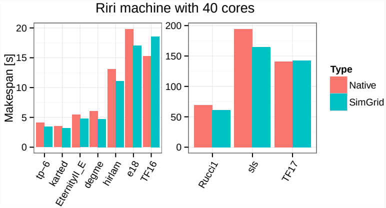
	Do_subtree	activate	Panel	Update	Assemble
1.	#Flops	#Zeros	*NB*	*NB*	#Coeff
2.	#Nodes	#Assemble	*MB*	*MB*	/
3.	/	/	*BK*	*BK*	/
R^2	0.99	0.99	0.99	0.99	0.86



Overview of simulation accuracy



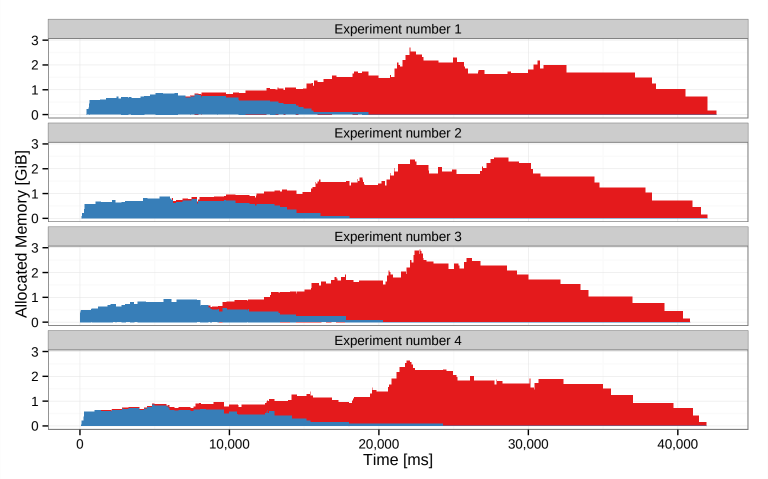




- Most of the time, simulation is slightly optimistic
- With bigger and architecturally more complex machines, error increases

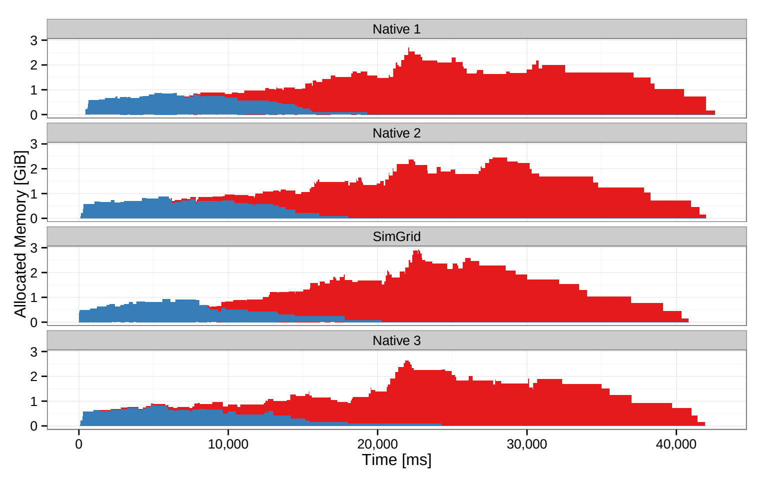
Studying memory consumption

- Minimizing memory footprint is often critical
- Remember scheduling is dynamic so consecutive Native experiments have different output



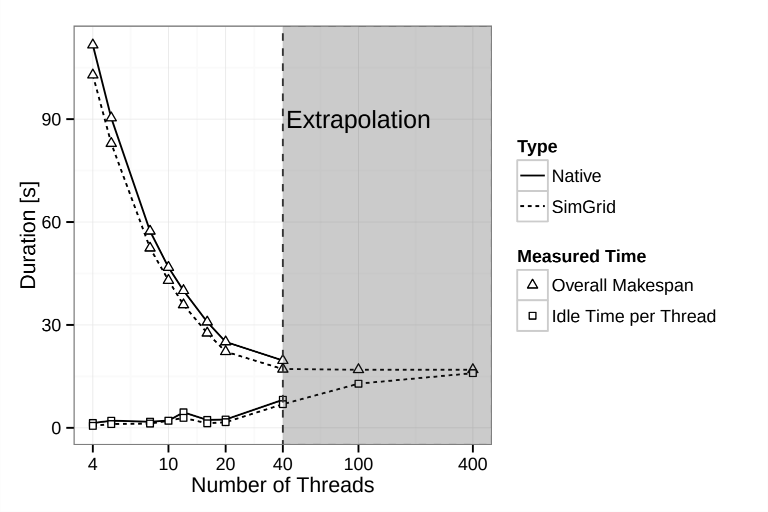
Studying memory consumption

- Minimizing memory footprint is often critical
- Remember scheduling is dynamic so consecutive Native experiments have different output



Extrapolating to larger machines

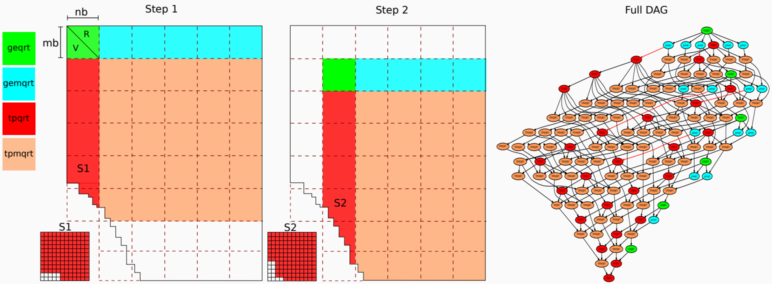
- Predicting performance in idealized context
- Studying the parallelization limits of the problem



SIMULATION OF QR_MUMPS ON TOP OF STARPU-SIMGRID: 2D FULLY-FEATURED CODE

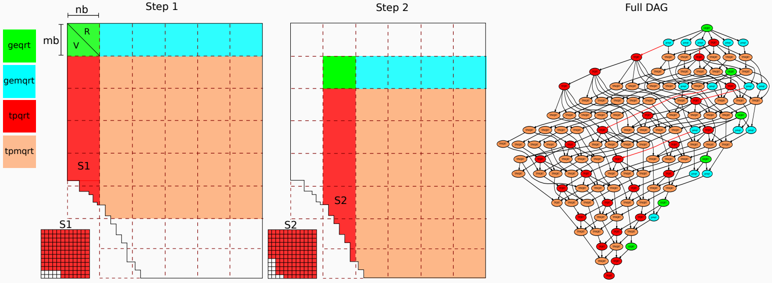
2D fully-featured code

- Increase concurrency through 2D ("tile") algorithms
- Exploit sparsity of the staircase structure within fronts



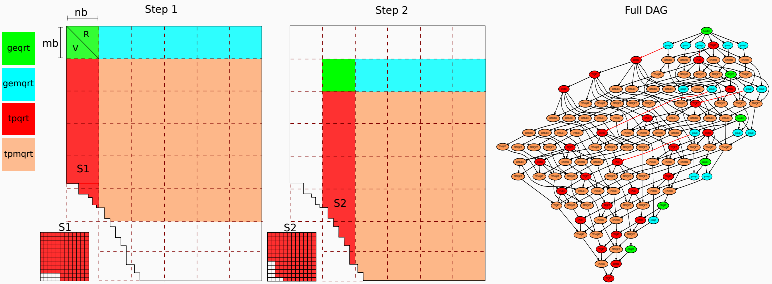
2D fully-featured code

- Increase concurrency through 2D ("tile") algorithms
- Exploit sparsity of the staircase structure within fronts
- Focus on the update (tpmqrt) kernel



2D fully-featured code

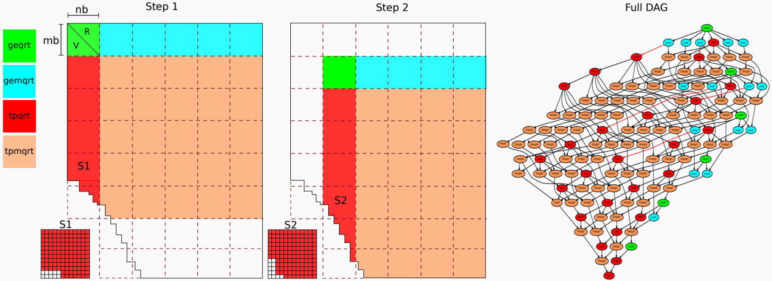
- Increase concurrency through 2D ("tile") algorithms
- Exploit sparsity of the staircase structure within fronts



How to handle the staircase structure?

2D fully-featured code

- Increase concurrency through 2D ("tile") algorithms
- Exploit sparsity of the staircase structure within fronts

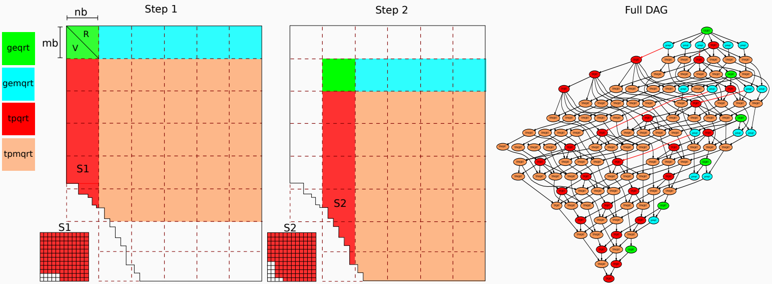


How to handle the staircase structure?

- S1=$\{10,10,10,10,10,12,12,12,12,12,12,12\}$

2D fully-featured code

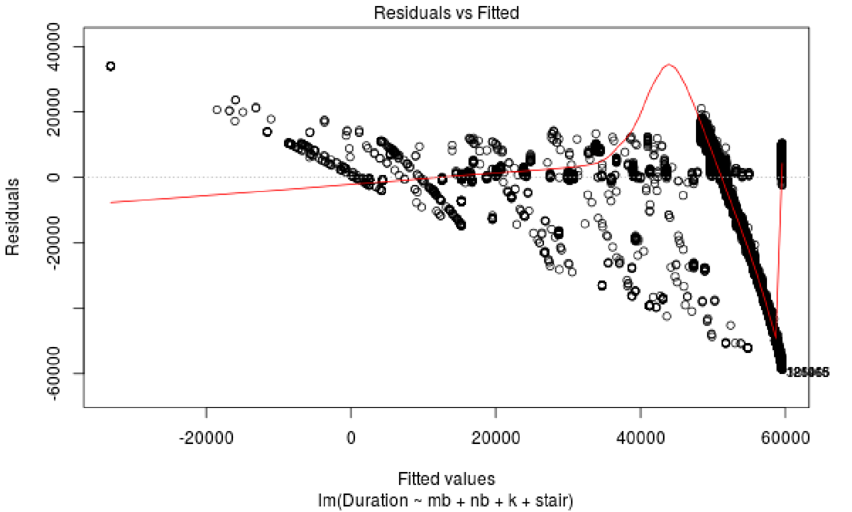
- Increase concurrency through 2D ("tile") algorithms
- Exploit sparsity of the staircase structure within fronts



How to handle the staircase structure?

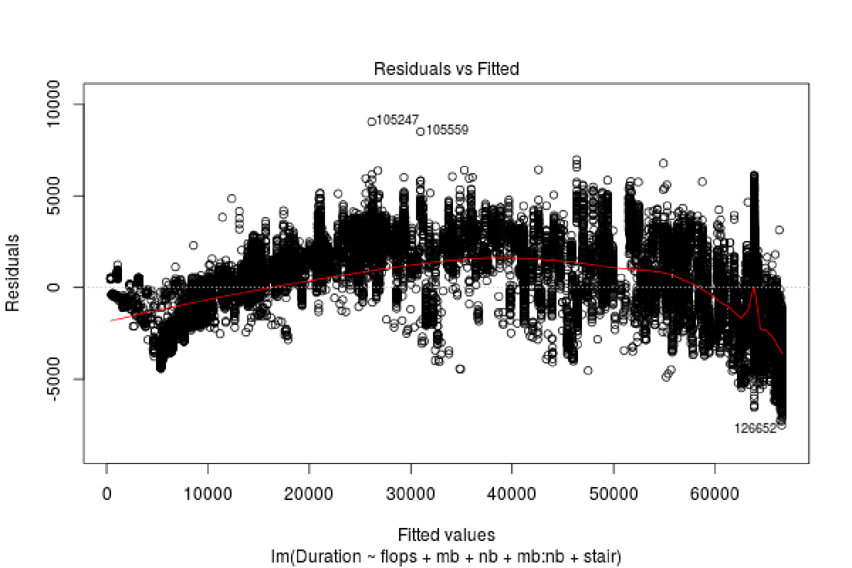
- S1={10,10,10,10,10,12,12,12,12,12,12,12}
- stair $= 10 * 5 + 12 * 7 = 134$

Model m1: $lm(Duration \sim stair + mb + nb + k)$



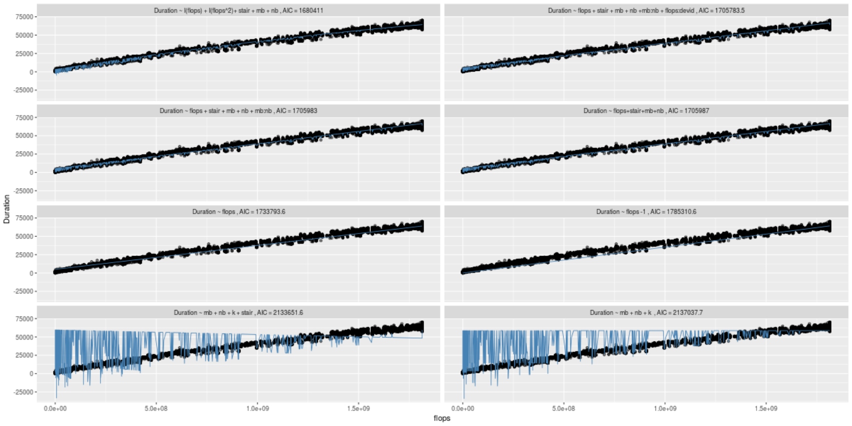
Adjusted R^2: 0.26

Model m2: $lm(Duration \sim\ flop + stair + mb + nb + mb : nb)$



Adjusted R^2: 0.997

Models for tpmqrt kernel ordered wrt AIC



SIMULATION OF QR_MUMPS ON TOP OF STARPU-SIMGRID: GPU-BASED SYSTEMS

GPU-based systems in a nutshell

- Very high computing power ($O(1)$ Tflop/s)
- Very high memory bandwidth ($O(100)$ GB/s)
- Very convenient Gflops/s/Watt ratio ($O(10)$)

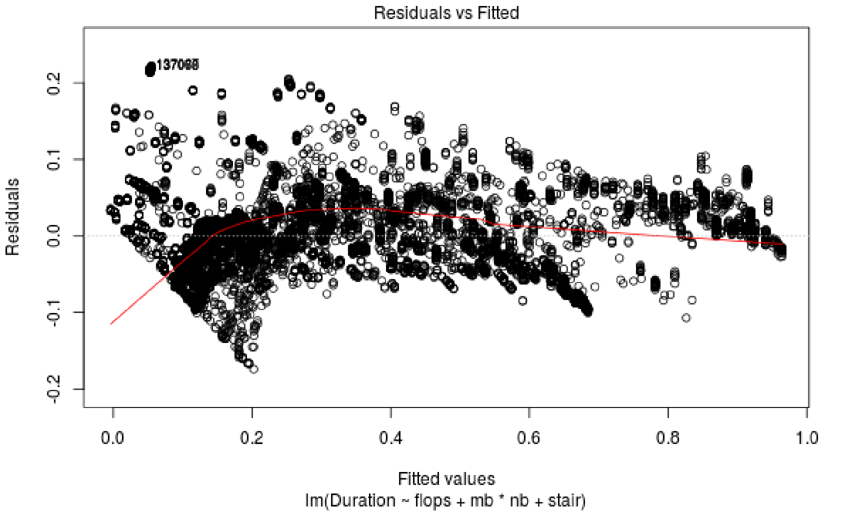
Objective

Exploit heterogeneity (i.e. take advantage of the diversity of resources) to accelerate the multifrontal QR factorization.

Issues:

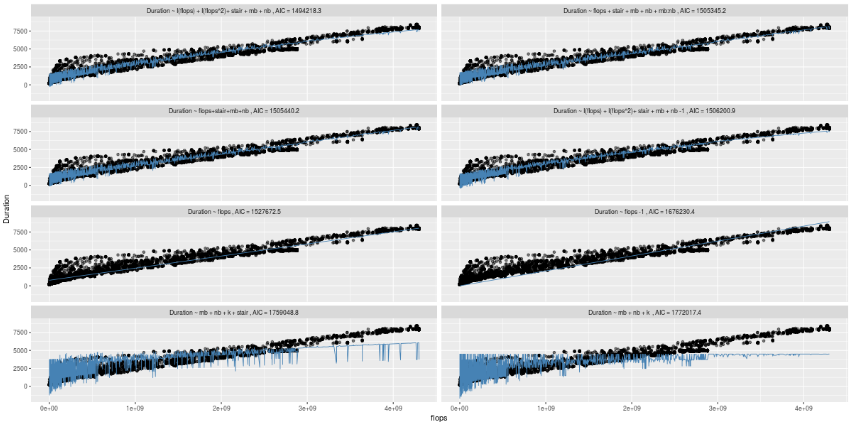
- Granularity: GPUs require coarser grained tasks to achieve full speed;
- Scheduling: account for different computing capabilities and different tasks characteristics while maximizing concurrency;
- Communications: minimize the cost of host-to-device data transfers.

Model m2: $lm(Duration \sim flop + stair + mb + nb + mb : nb)$

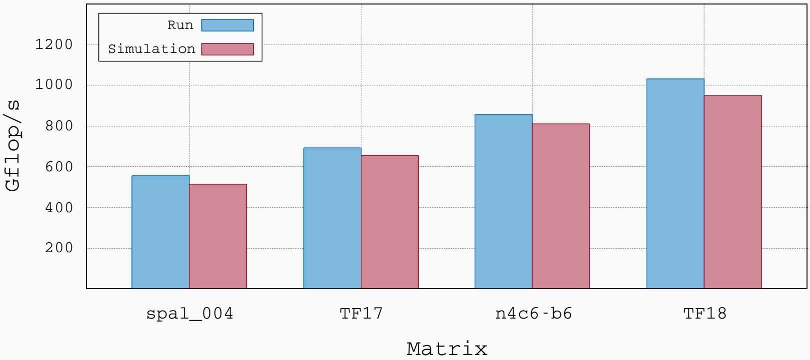


Adjusted R^2: 0.92

Models for tpmqrt kernel on GPU K40 ordered wrt AIC



Overall simulation of an 24 cores + 1 GPU



CONCLUDING REMARKS

Conclusion and perspectives

In a nustshell

The abstraction provided by task-based programming together with modern runtime systems and simuation tools allow for fast and accurate simulation of complex irregular fully-featured codes

To be consolidated

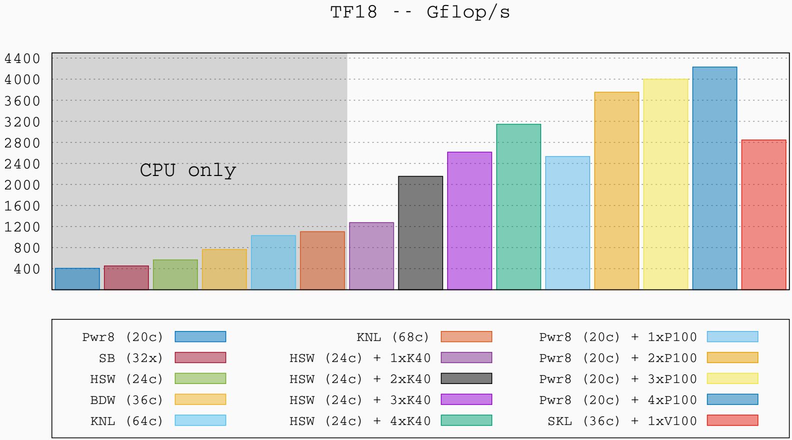
- Consolidate our comparitive study of models for the fully-featured 2D code
- How to model GPU streams?

Perspectives

- Conduct reproducible studies of the impact of scheduling policies for the multi-GPUs case
- Exploit simulation as a substitute for fast auto-tuning

COMMERCIALS

Experimental results



Thanks to IDRIS, Plafrim, CALMIP and GENCI for providing access to the resources

References I

[1] E. Agullo, G. Bosilca, A. Buttari, A. Guermouche, and F. Lopez. "Exploiting a Parametrized Task Graph Model for the Parallelization of a Sparse Direct Multifrontal Solver". In: *Euro-Par 2016: Parallel Processing Workshops: Euro-Par 2016 International Workshops, Grenoble, France, August 24-26, 2016, Revised Selected Papers*. Ed. by F. Desprez et al. Cham: Springer International Publishing, 2017, pp. 175–186. ISBN: 978-3-319-58943-5.

[2] E. Agullo, A. Buttari, M. Byckling, A. Guermouche, and I. Masliah. *Achieving high-performance with a sparse direct solver on Intel KNL*. Research Report RR-9035. Inria Bordeaux Sud-Ouest ; CNRS-IRIT ; Intel corporation ; Université Bordeaux, Feb. 2017, p. 15.

[3] E. Agullo, A. Buttari, A. Guermouche, and F. Lopez. "Implementing Multifrontal Sparse Solvers for Multicore Architectures with Sequential Task Flow Runtime Systems". In: *ACM Trans. Math. Softw.* 43.2 (Aug. 2016), 13:1–13:22. ISSN: 0098-3500.

[4] E. Agullo, A. Buttari, A. Guermouche, and F. Lopez. "Multifrontal QR Factorization for Multicore Architectures over Runtime Systems". In: *Euro-Par 2013 Parallel Processing*. Springer Berlin Heidelberg, 2013, pp. 521–532. ISBN: 978-3-642-40046-9.

[5] E. Agullo, A. Buttari, A. Guermouche, and F. Lopez. "Task-Based Multifrontal QR Solver for GPU-Accelerated Multicore Architectures.". In: *HiPC*. IEEE Computer Society, 2015, pp. 54–63. ISBN: 978-1-4673-8488-9.

[6] A. Buttari. "Fine-Grained Multithreading for the Multifrontal QR Factorization of Sparse Matrices". In: *SIAM Journal on Scientific Computing* 35.4 (2013), pp. C323–C345. eprint: http://epubs.siam.org/doi/pdf/10.1137/110846427.

[7] I. S. Duff and J. K. Reid. "The multifrontal solution of indefinite sparse symmetric linear systems". In: *ACM Transactions On Mathematical Software* 9 (1983), pp. 302–325.

[8] X. S. Li, P. Cicotti, and S. B. Baden. *LUsim: A Framework for Simulation-Based Performance Modeling and Prediction of Parallel Sparse LU Factorization*. Tech. rep. Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States), 2008.

[9] L. Stanisic, E. Agullo, A. Buttari, A. Guermouche, A. Legrand, F. Lopez, and B. Videau. "Fast and Accurate Simulation of Multithreaded Sparse Linear Algebra Solvers". In: *Parallel and Distributed Systems (ICPADS), 2015 IEEE 21st International Conference on*. Dec. 2015, pp. 481–490.

References II

[10] L. Stanisic, S. Thibault, A. Legrand, B. Videau, and J. Méhaut. "Modeling and Simulation of a Dynamic Task-Based Runtime System for Heterogeneous Multi-core Architectures". In: *Euro-Par 2014 Parallel Processing - 20th International Conference, Porto, Portugal, August 25-29, 2014. Proceedings*. 2014, pp. 50–62.

?

Thanks!

Questions?

Matrices from the Suite Sparse Matrix Collection

#	Mat. name	m	n	nz	op. count
12	hirlam	1385K	452K	2713K	1384G
13	flower_8_4	55K	125K	375K	2851G
14	Rucci1	1977K	109K	7791K	5671G
15	ch8-8-b3	117K	18K	470K	10709G
16	GL7d24	21K	105K	593K	16467G
17	neos2	132K	134K	685K	20170G
18	spal_004	10K	321K	46168K	30335G
19	n4c6-b6	104K	51K	728K	62245G
20	sls	1748K	62K	6804K	65607G
21	TF18	95K	123K	1597K	194472G
22	lp_nug30	95K	123K	1597K	221644G
23	mk13-b5	135K	270K	810K	259751G