Performance Analysis of Tile Low-Rank Cholesky Factorization Using PaRSEC Instrumentation Tools

Qinglei Cao, Yu Pei, Thomas Herault, Kadir Akbudak, Aleksandr Mikhalev, George Bosilca, Hatem Ltaief, David Keyes, and Jack Dongarra

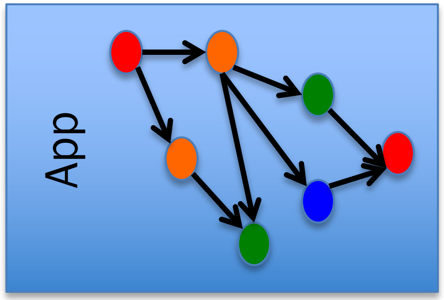
Protools19

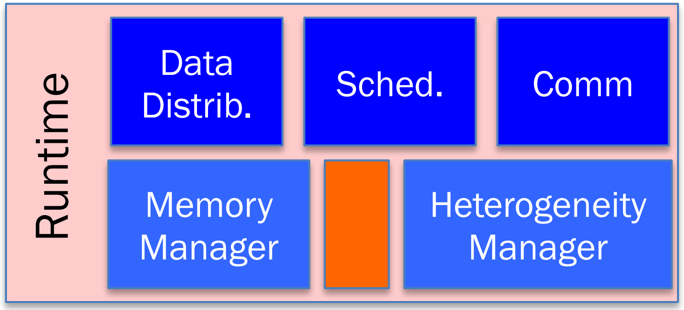
ICL INNOVATIVE COMPUTING LABORATORY

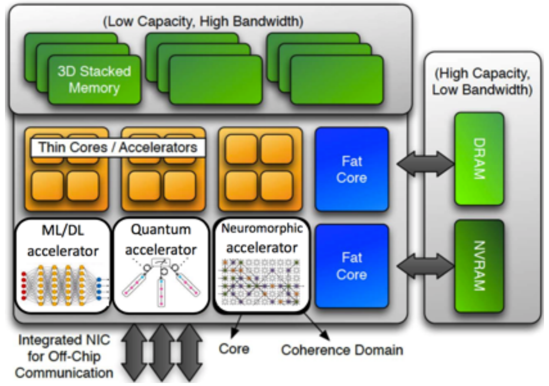
THE UNIVERSITY OF TENNESSEE KNOXVILLE

KAUST

PaRSEC, task-based programming







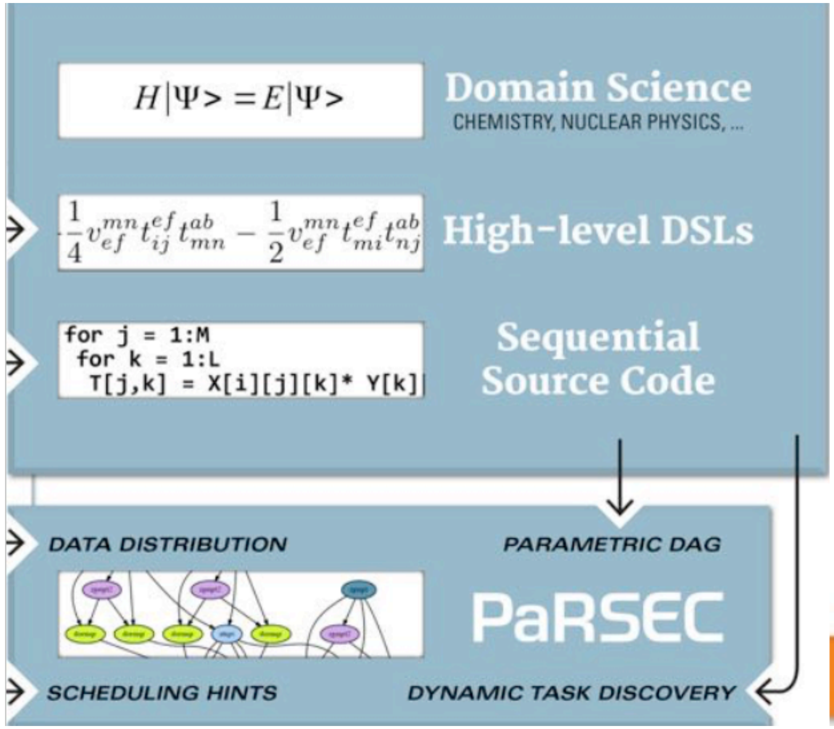
- Focus on data dependencies, data flows, and tasks
- Don’t develop for an architecture but for a portability layer
- Let the runtime deal with the hardware characteristics
  - But provide as much user control as possible
- StarSS, StarPU, Swift, Parallex, Quark, Kaapi, DuctTeip, ..., and PaRSEC

PaRSEC

PaRSEC: a generic runtime system for asynchronous, architecture aware scheduling of fine-grained tasks on distributed many-core heterogeneous architectures.

Concepts

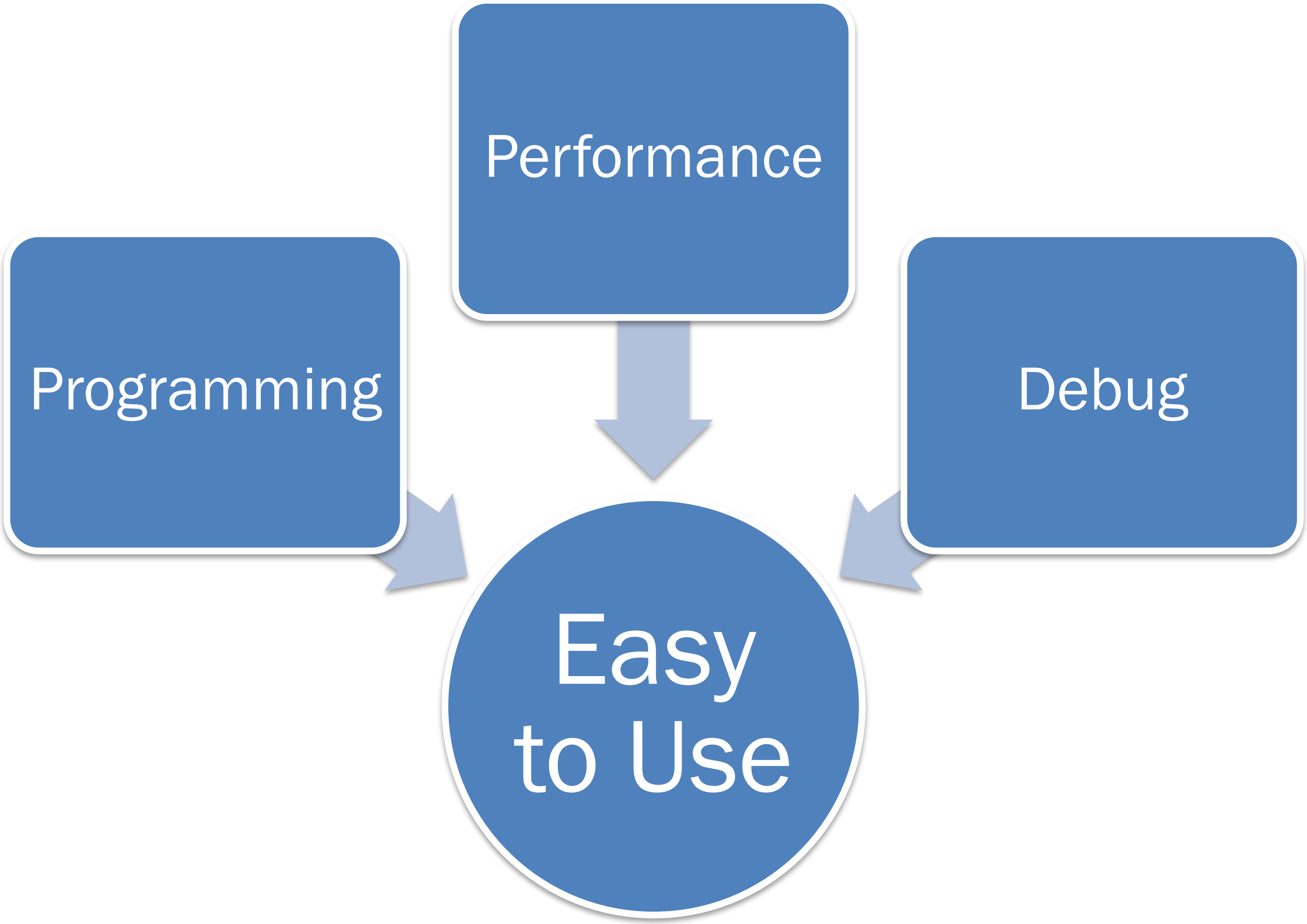
- Clear separation of concerns: compiler optimize each task class, developer describe dependencies between tasks, the runtime orchestrate the dynamic execution
- Interface with the application developers through specialized domain specific languages (PTG/JDF, Python, insert_task, fork/join, …)
- Separate algorithms from data distribution
- Make control flow executions a relic



Runtime

- Permeable portability layer for heterogeneous architectures
- Scheduling policies adapt every execution to the hardware & ongoing system status
- Data movements between producers and consumers are inferred from dependencies. Communications/computations overlap naturally unfold
- Coherency protocols minimize data movements
- Memory hierarchies (including NVRAM and disk) integral part of the scheduling decisions

PaRSEC



PaRSEC Profiling Tools

Trace Collection Framework

- Sits at the core of the performance profiling system
- Events as identifiable entities
- Scalable for many- thread environments
  - One profiling stream for each thread
  - Additional helping threads in charge of I/O, memory allocators, compactors, …
  - Additional buffers allocated in advance

PINS: PaRSEC INStrumentati on

- The Trace Collection Framework is used within the PaRSEC runtime through the **P**aRSEC **INS**trumentation (PINS) interface
- PINS registers callbacks for all the important steps of a task or communication life cycle
- Dynamically configurable to generate only the events pertinent to the run

PaRSEC Profiling Tools

Dependency Analysis

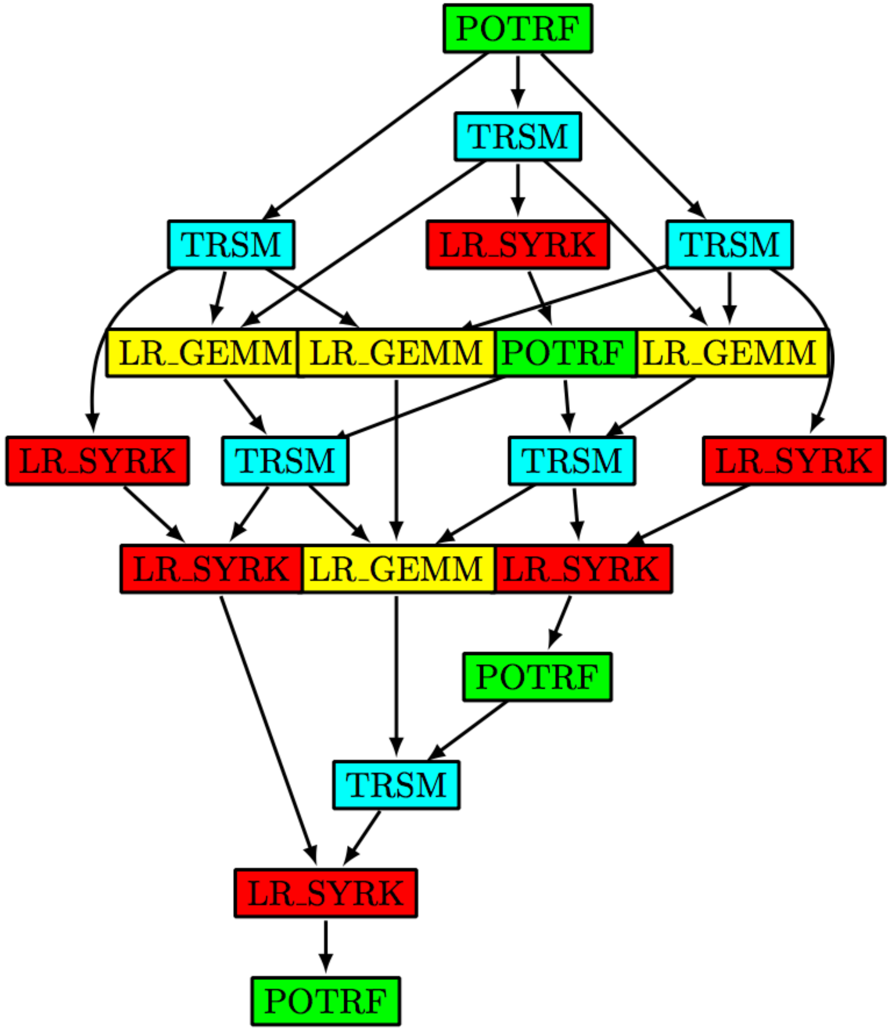
- It is necessary to connect information with the actual DAG of tasks
- Automatically generate DOT file each with a partial view of the DAG
  - Collection of the DAG can be done offline
  - One DOT file per process with tools to concatenate the different DOT files

Trace Conversion Tools

- The binary format of trace files is not exposed to the user, needs to be a portable and exploitable file format
- Hierarchical Data Format (HDF5) following the structure required by the popular Pandas Library
- Tools to take the generated trace and convert it into a Gantt chart
- Provides a library to read the DOT files that are generated into a NetworkX [29] representation

PaRSEC Profiling Tools

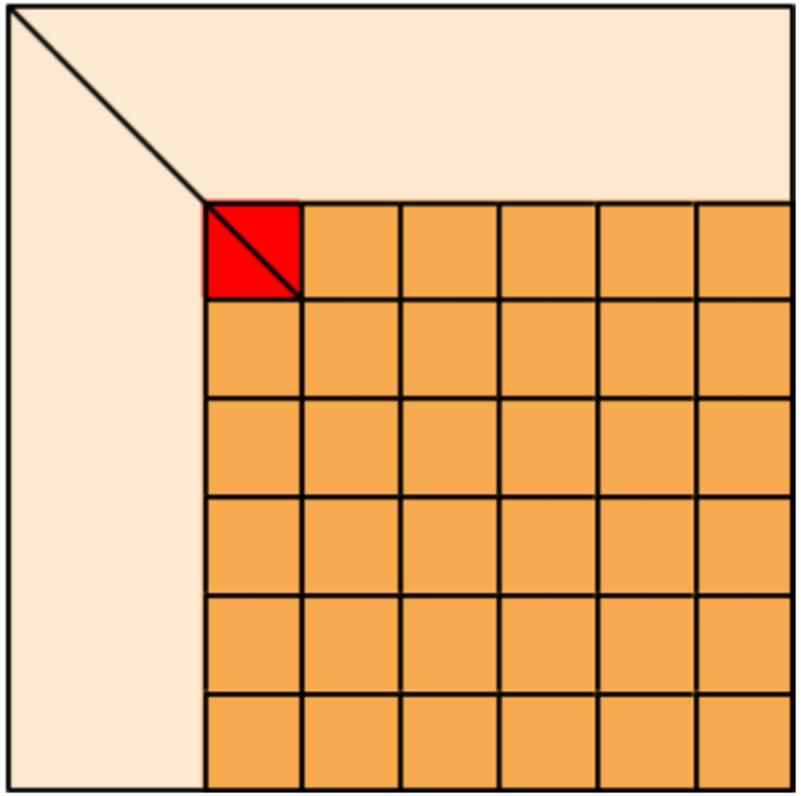
```
$> python
>>> import pandas as pd
>>> t = pd.HDFStore('dpotrf.h5')
>>> t.event_types
ACTIVATE_CB          6
Device delegate      1
MPI_ACTIVATE         2
MPI_DATA_CTL         3
MPI_DATA_PLD_RCV     5
MPI_DATA_PLD_SND     4
 PUT_CB              7
TASK_MEMORY          0
potrf_dgemm          8
potrf_dpotrf        11
 potrf_dsyrk         9
potrf_dtrsm         10
dtype: int64
```



TLR Cholesky Factorization

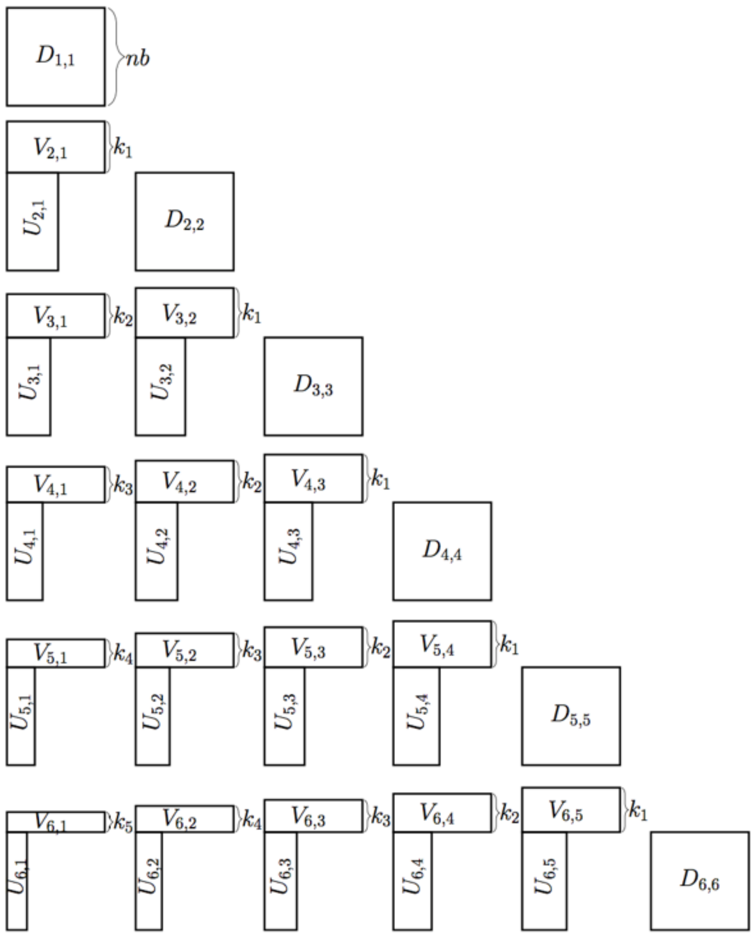
The Cholesky factorization of an N * N real symmetric, positive-definite matrix A has the form: $A = LL^T$, where L is an N → N real lower triangular matrix with positive diagonal elements.

- ❖ Apparently dense matrices arising in scientific applications, such as climate/weather forecasting in computational statistics, seismic imaging in earth science, structural and vibrational analysis in material science.
- ❖ Common properties:
  - Symmetric, positive-definite matrix
  - (Apparently) dense matrices
  - Often data-sparse, Decay of parameter correlations with distance



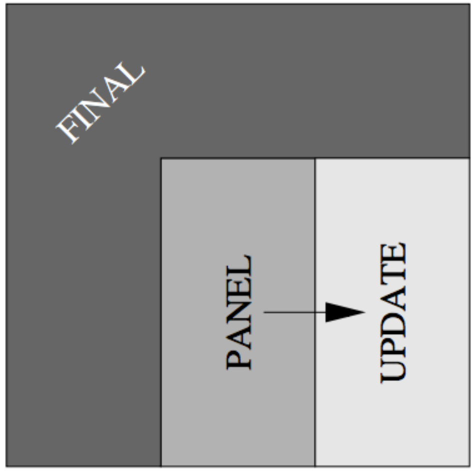
TLR Cholesky Factorization

- Dense matrices might be compressed:
  - Cholesky factorization (for distributed-memory architectures)
  - Tile low rank (TLR) matrix format
  - Significantly less memory
  - Preserving the accuracy requirements of the scientific application
  - Huge performance improvement via cutting down flops



TLR Cholesky Factorization

```
for p = 1 to NT do
    POTRF(D(p,p))
    for i = p+1 to NT do
      TRSM(V(i,p), D(p,p))
       for j = p+1 to NT
          LR_SYRK(D(j,j), U(j,p), V(j,p))
       for i = j+1 to NT do
          LR_GEMM(U(i,p), V(i,p), U(j,p), V(j,p), U(i,j), V(i,j), acc)
```

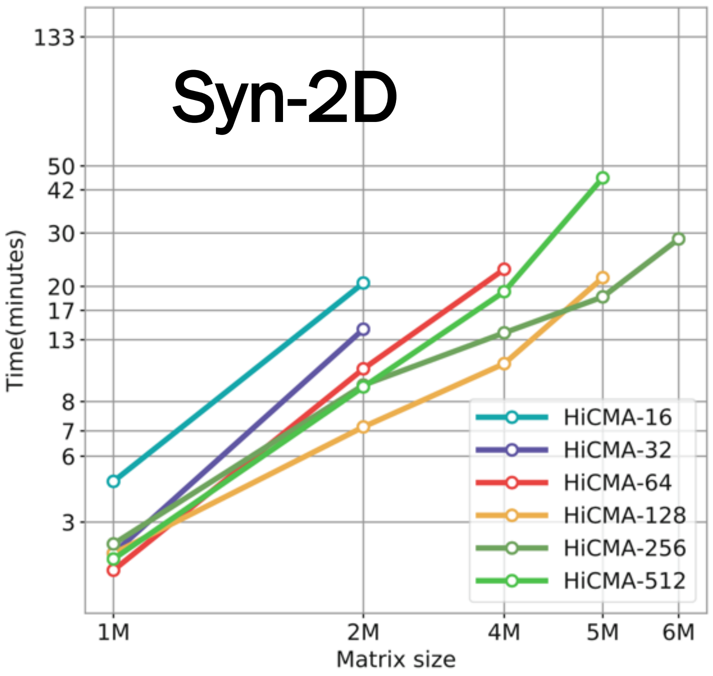


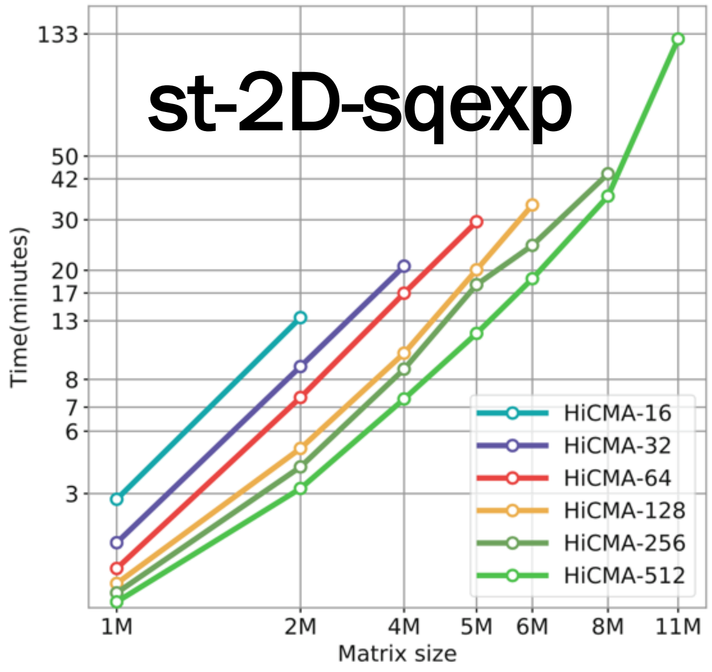
A serial and incompressible critical path of TLR Cholesky: (NT -1) * (POTRF + TRSM + SYRK) + POTRF.

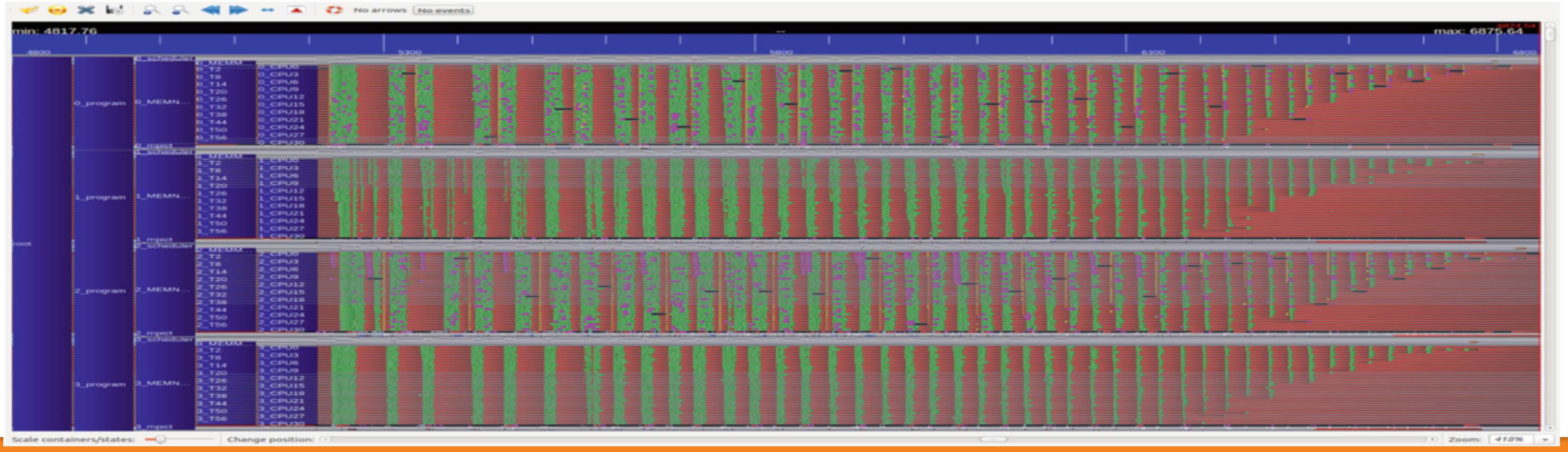
Kernel	Dense Cholesky	TLR Cholesky
POTRF	1/3 * nb^3	1/3 * nb^3
TRSM	nb^3	nb^2 * rank
SYRK/LR_SYRK	nb^3	2 * nb^2 * rank + 4 * nb * rank^2
GEMM/LR_GEMM	2 * nb^3	36 * nb * rank^2
Total	O(N^3)	O(N^2 * rank)

State-of-the-art

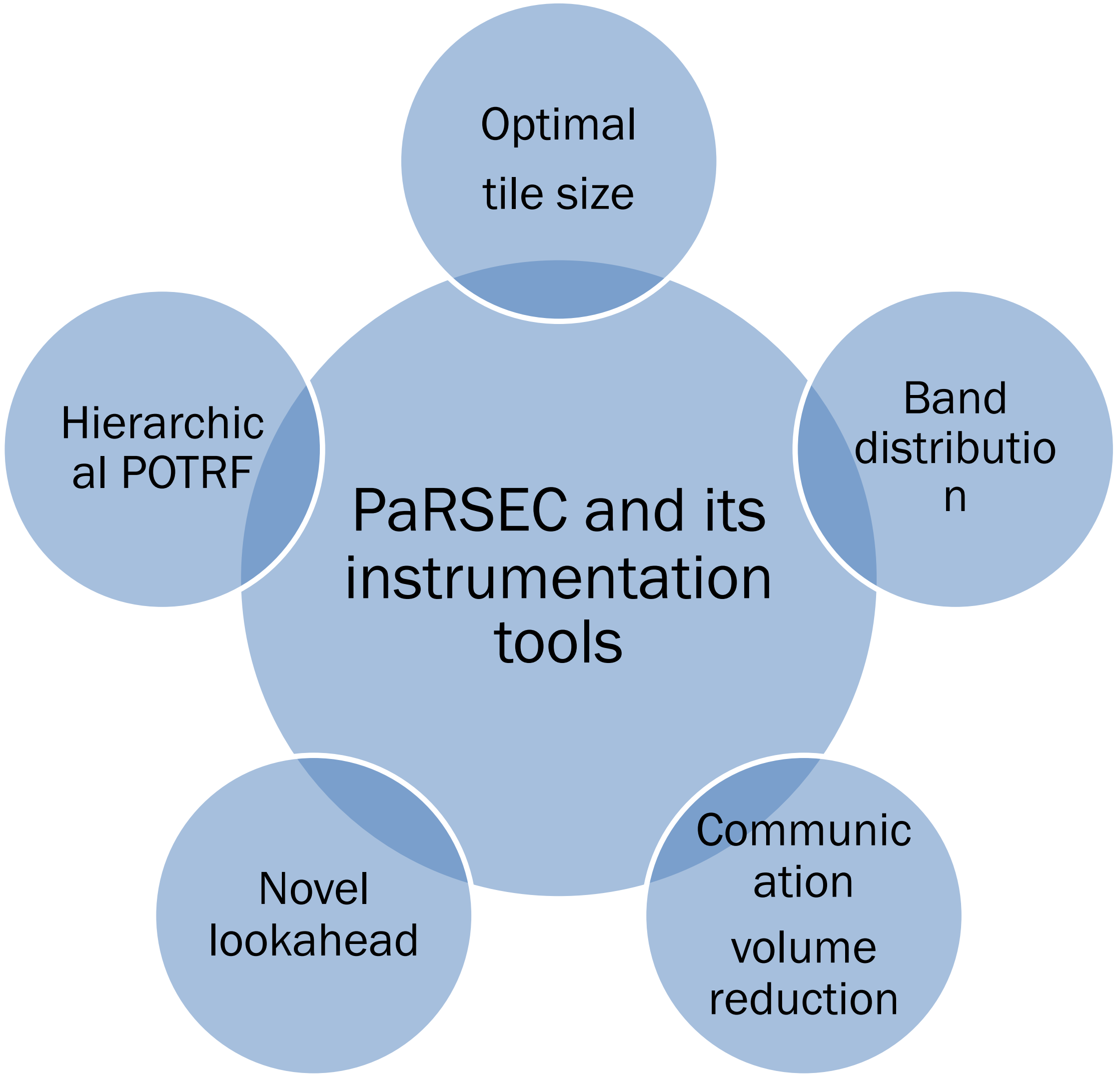
- Shaheen II, a Cray XC40 system, which has 6,174 compute nodes;
- The accuracy threshold of 10−8, which ultimately yields absolute numerical error of order 10−9;





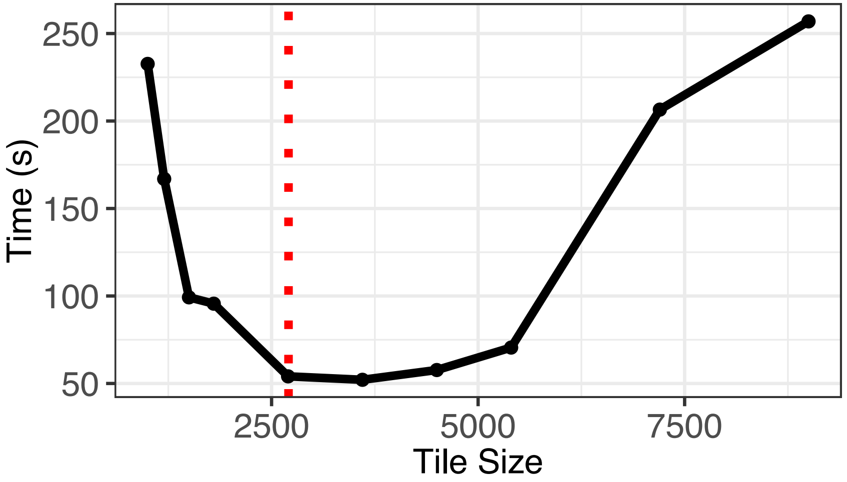


Experiments



Optimization 1: Optimal Tile Size

- Tile size plays a significant role in TLR Cholesky



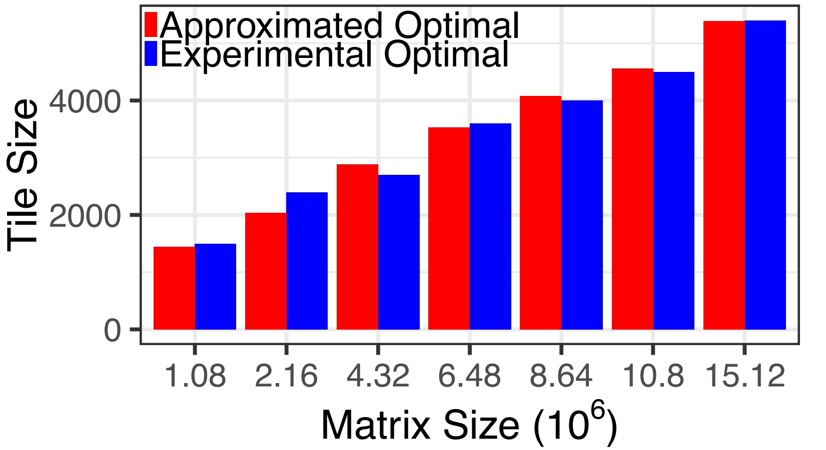
Syn-2D, 16 nodes, 2M

- The profiling tools in PaRSEC gets in:
  - kernel execution time varies for each task, in terms of the number of operations
  - Set special event "ops_count" to gather the operaions count for tasks in the critical path and tasks off the critical path

- **Operation balance** between tiles **on and off critical path.**
- Assume N is the matrix size, node is the number of nodes, k is the average rank of tiles off diagonal, then the best tile size nb can be approximated:

$$nb \approx \sqrt{\frac{3 \times N \times k \times (3 + \sqrt{9 + 32 \times node})}{4 \times node}}$$

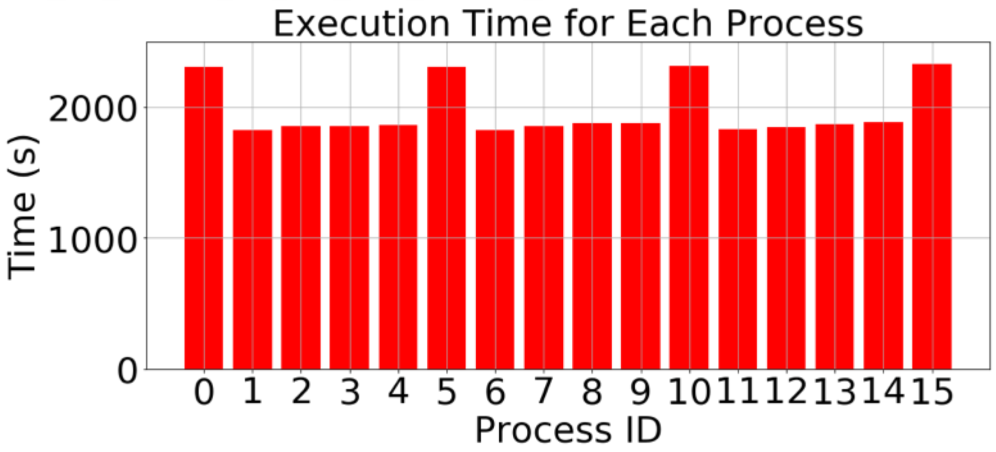
$$nb \propto \sqrt{\frac{N \times k}{\sqrt{node}}}$$



st-2D-sqexp, 256 nodes

Evaluation: Hybrid Data Distributions

- Imbalance: **memory** and **computation**
- PaRSEC's profiling system could provide the execution time for each process, as well as each thread, from which we extract the workload for each process to show load balancing.



0	1	2	0	1	2	0	1	2
3	**4**	5	3	4	5	3	4	5
6	7	**8**	6	7	8	6	7	8
0	1	2	**0**	1	2	0	1	2
3	4	5	3	**4**	5	3	4	5
6	7	8	6	7	**8**	6	7	8
0	1	2	0	1	2	**0**	1	2
3	4	5	3	4	5	3	**4**	5
6	7	8	6	7	8	6	7	**8**

2DBCDD

0	1	2	0	1	2	0	1	2
3	**1**	5	3	4	5	3	4	5
6	7	**2**	6	7	8	6	7	8
0	1	2	**3**	1	2	0	1	2
3	4	5	3	**4**	5	3	4	5
6	7	8	6	7	**5**	6	7	8
0	1	2	0	1	2	**6**	1	2
3	4	5	3	4	5	3	**7**	5
6	7	8	6	7	8	6	7	**8**

Band distribution,
band_size = 1

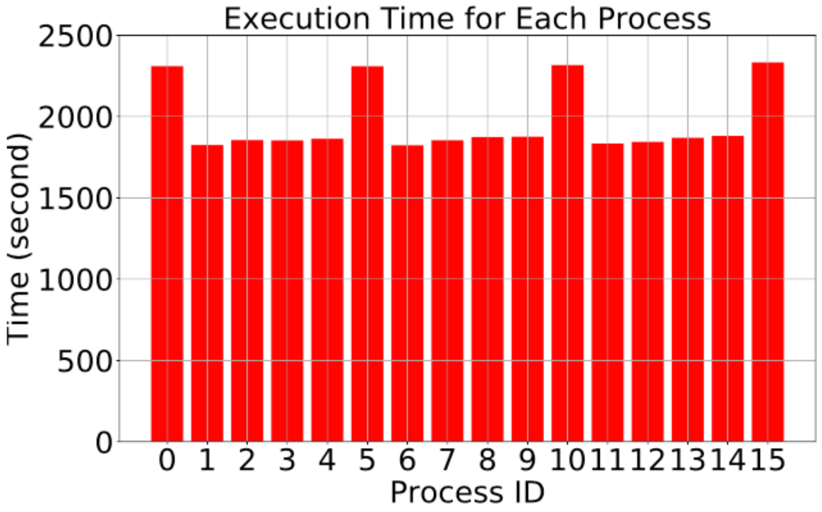
0	**1**	2	0	1	2	0	1	2
0	**1**	**2**	3	4	5	3	4	5
6	**1**	**2**	**3**	7	8	6	7	8
0	1	**2**	**3**	**4**	2	0	1	2
3	4	5	**3**	**4**	**5**	3	4	5
6	7	8	6	**4**	**5**	**6**	7	8
0	1	2	0	1	**5**	**6**	**7**	2
3	4	5	3	4	5	**6**	**7**	**8**
6	7	8	6	7	8	6	**7**	**8**

Band distribution,
band_size = 2

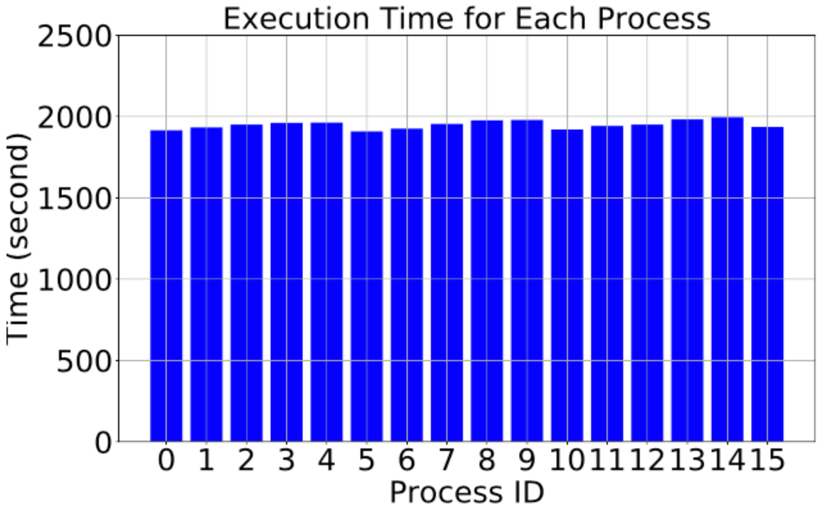
Evaluation: Hybrid Data Distributions

- We use the event *memory* in the profiling system to detail memory usage of both static matrix allocation and dynamic temporary buffers.
- PaRSEC's profiling system also provides the execution time for each process, as well as each thread, from which we extract the workload for each process to show load balancing.

No. of Nodes	Matrix Size	Memory Reduced (GB)
16	1080000	4.374
16	2160000	8.748
16	4320000	17.496
64	2160000	5.103
64	4320000	10.206
64	6480000	20.412



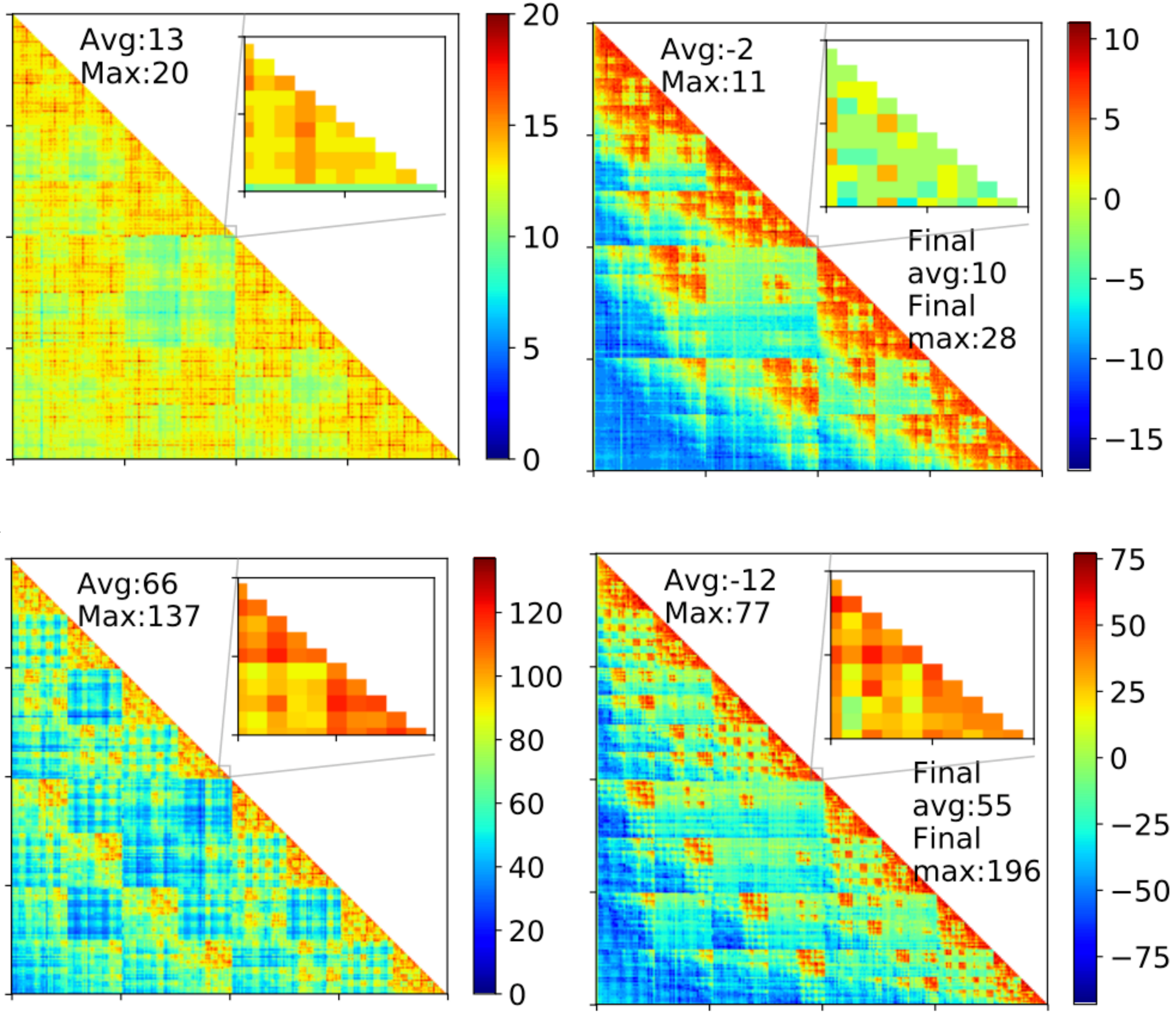
Without the hybrid data distributions



With the hybrid data distributions

Evaluation: Reduce Communication Volume

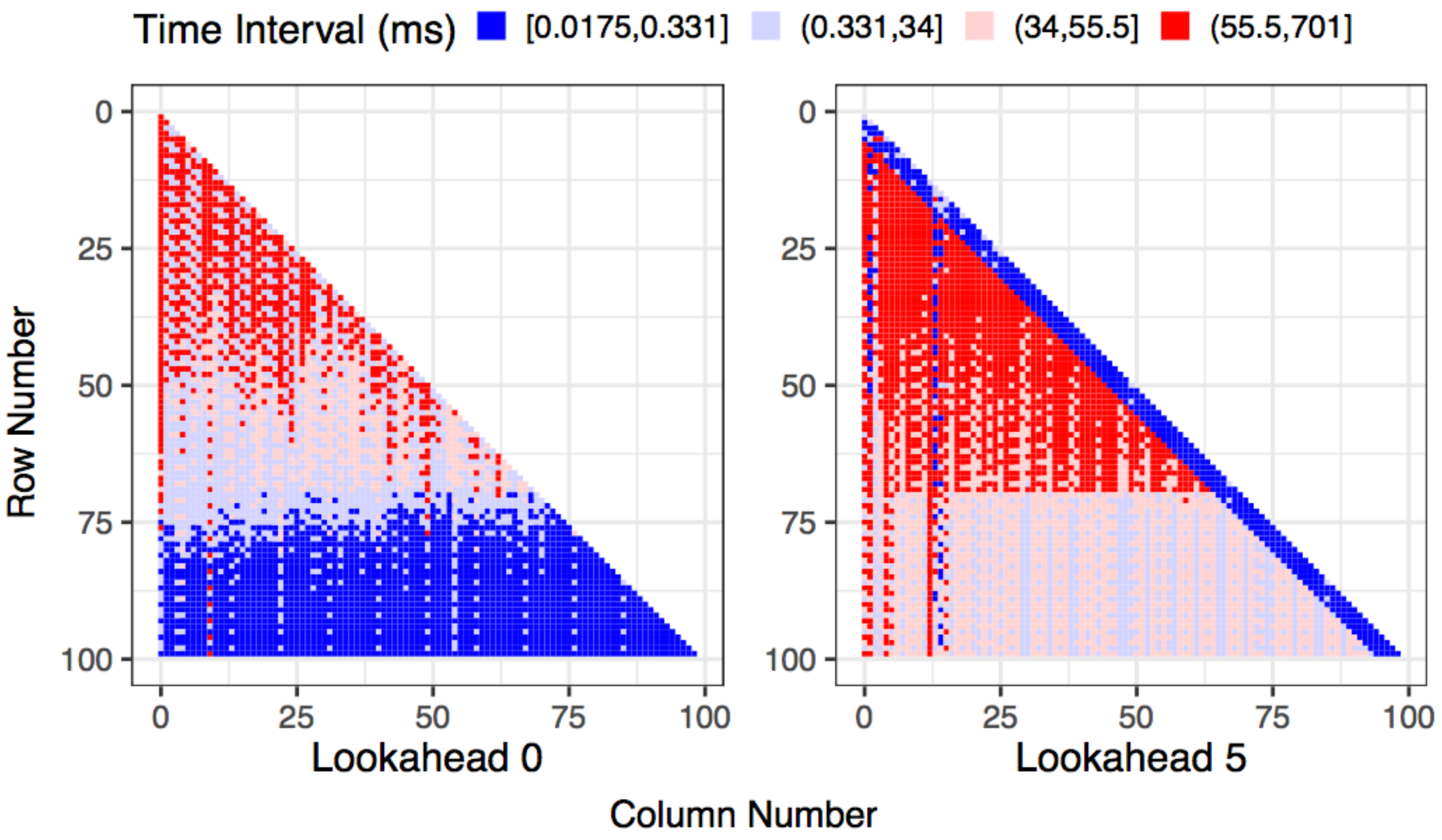
- We used the PaRSEC tracing framework API to register a new, application-specific type of event, and at the execution of each task, we logged the rank of the tile on which the task was working.
- Once the trace was converted, we then wrote application-specific scripts to analyze the HDF5 file, and produce the figures.



Initial rank distributions (i.e., before factorization) on the left and the difference between initial and final ranks (i.e., after factorization) on the right; the matrix size is 1080K × 1080K, and the tile size is 2,700; up for 2D problem, blew for 3D problem

Evaluation: Novel Lookahead

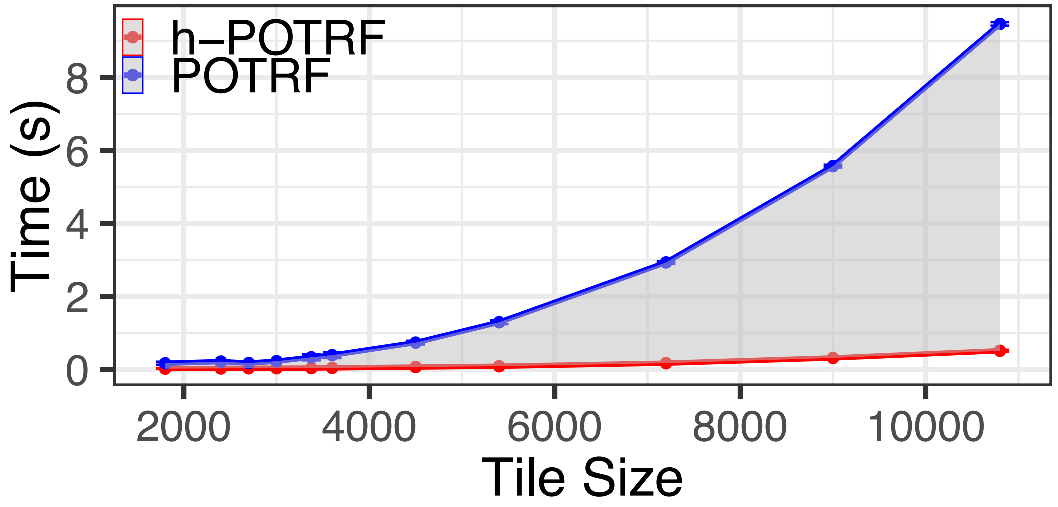
- We profiled the execution to ensure the critical path is respected, i.e. as soon as the data is read PaRSEC enables the critical tasks first.
- To be able to compute the average time it takes for data to be produced on one node and consumed on another, we need to connect the task termination, network activation, payload emission, and remote task execution events.
- This is provided by the PaRSEC profiling system through a combination of the trace information and the DOT file.

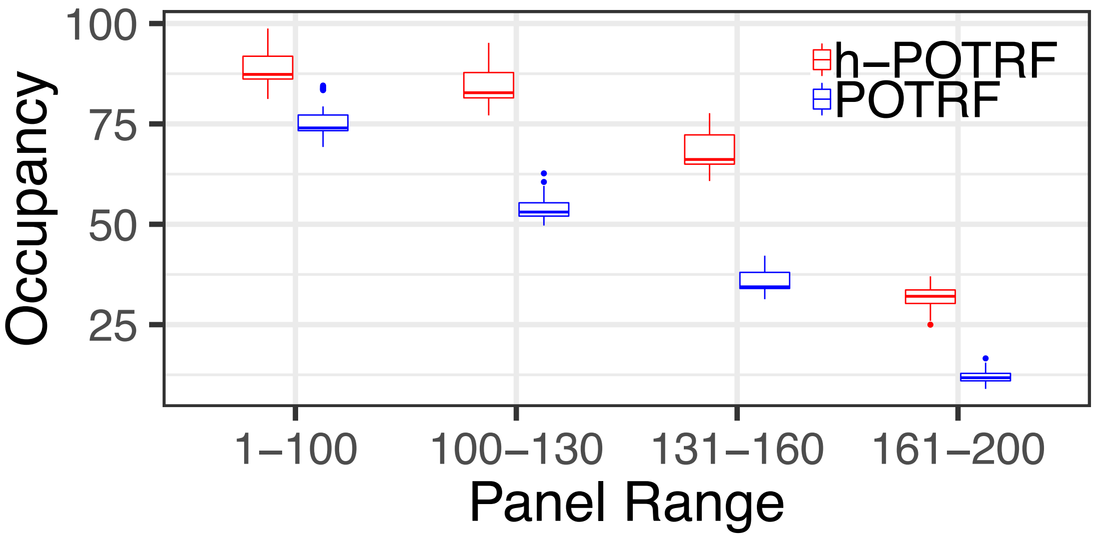


Time between data is ready and TRSM starts for st-2D-sqexp. Left, without lookahead; right, with lookahead of 5; each point represents one TRSM; matrix has 100 × 100 tiles

Evaluation: Hierarchical POTRF

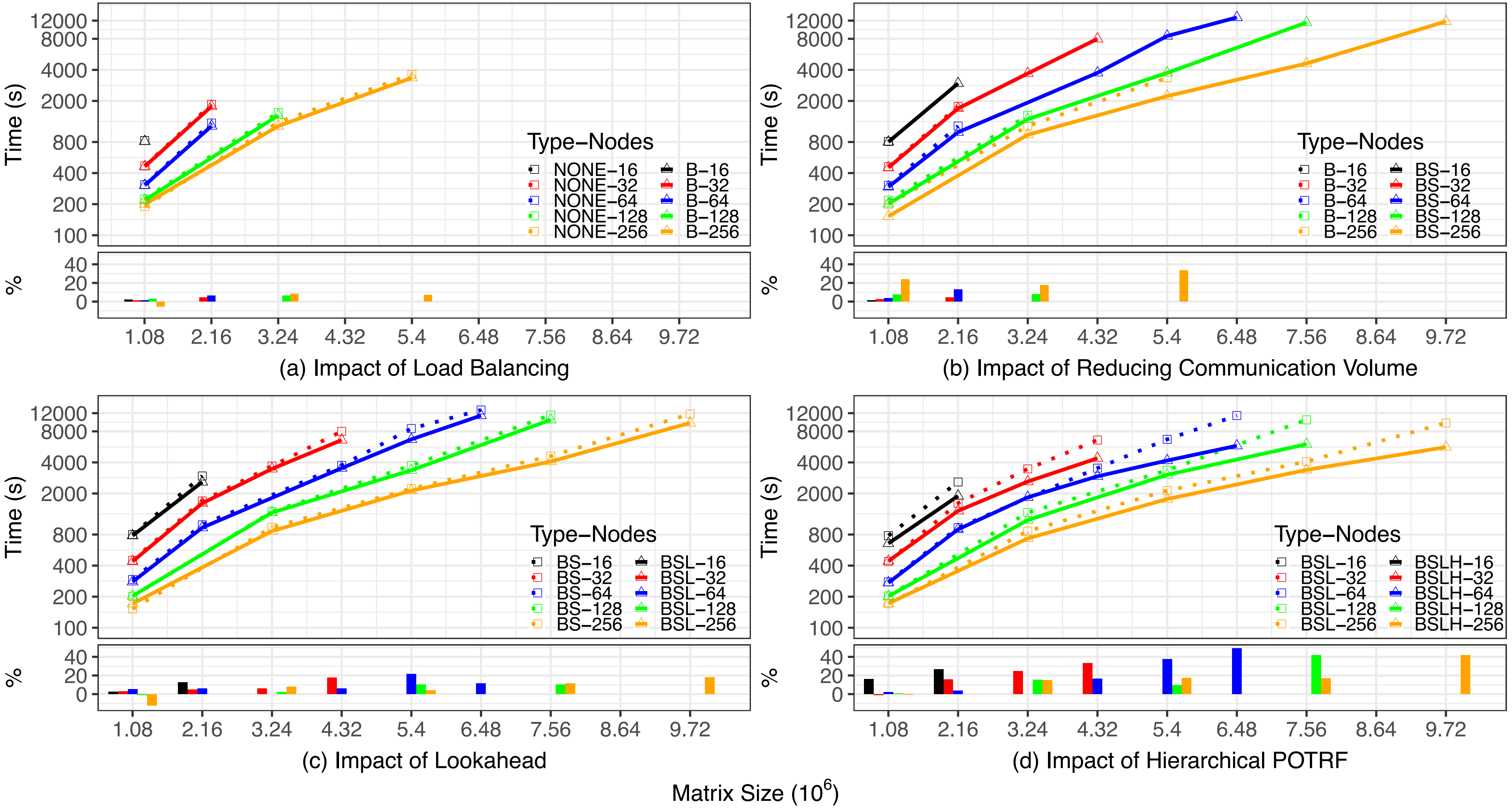
- We exploited the basic timing information produced by the tracing system
- Plus statistical packages provided by pandas and NumPy to compute our metrics: we compute the occupancy of the computational resources during the original run and then during the hierarchical POTRF run.





Impact of Hierarchical POTRF: top, execution time on a single node; bottom, resource occupancy of 540K × 540K matrix on a 3 × 3 process grid with a tile size of 2,700

Incremental Effects



(a) Impact of Load Balancing

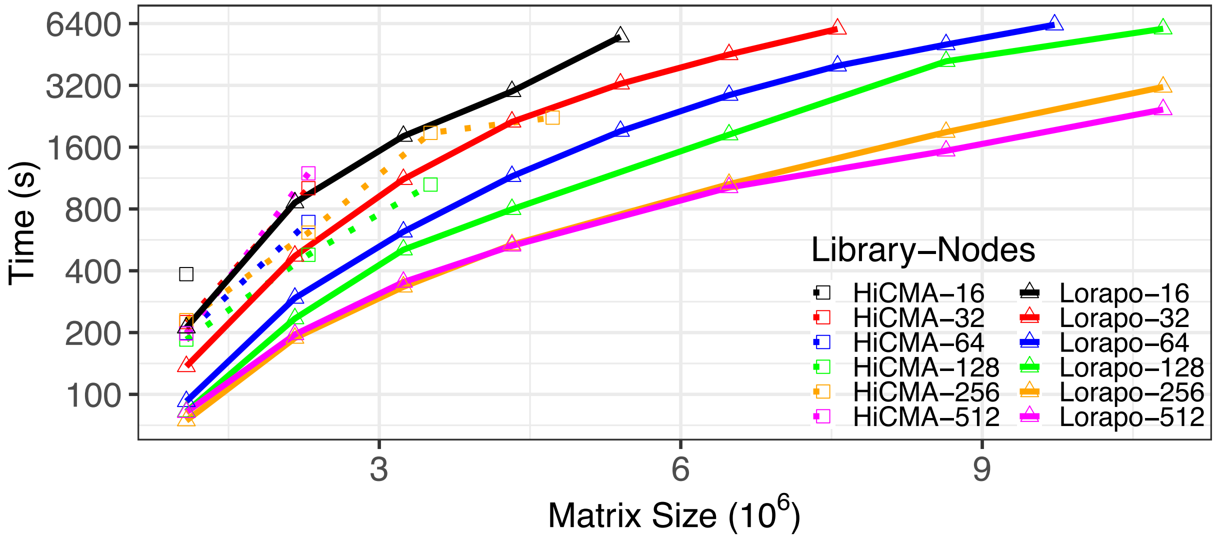
(b) Impact of Reducing Communication Volume

(c) Impact of Lookahead

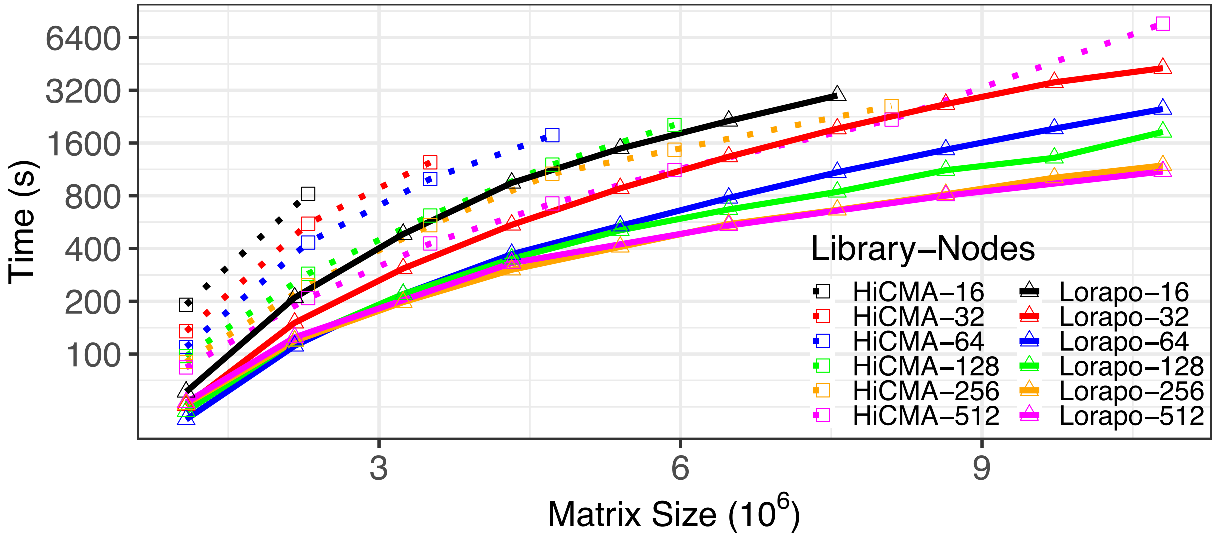
(d) Impact of Hierarchical POTRF

St-3D-sqexp

Comparison with State-of-the-art

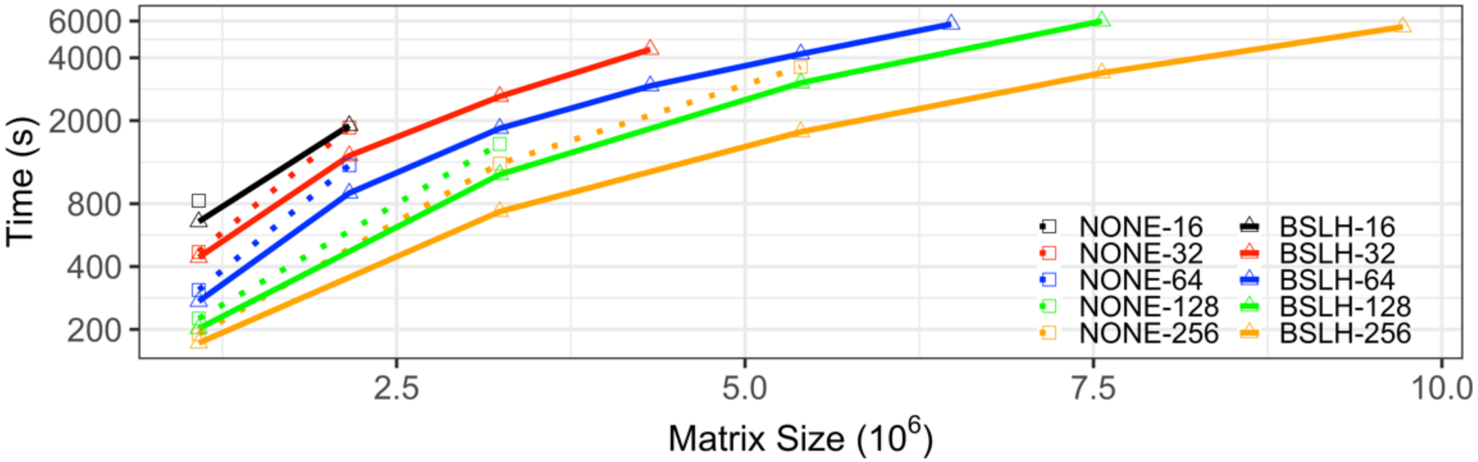


Syn-2D

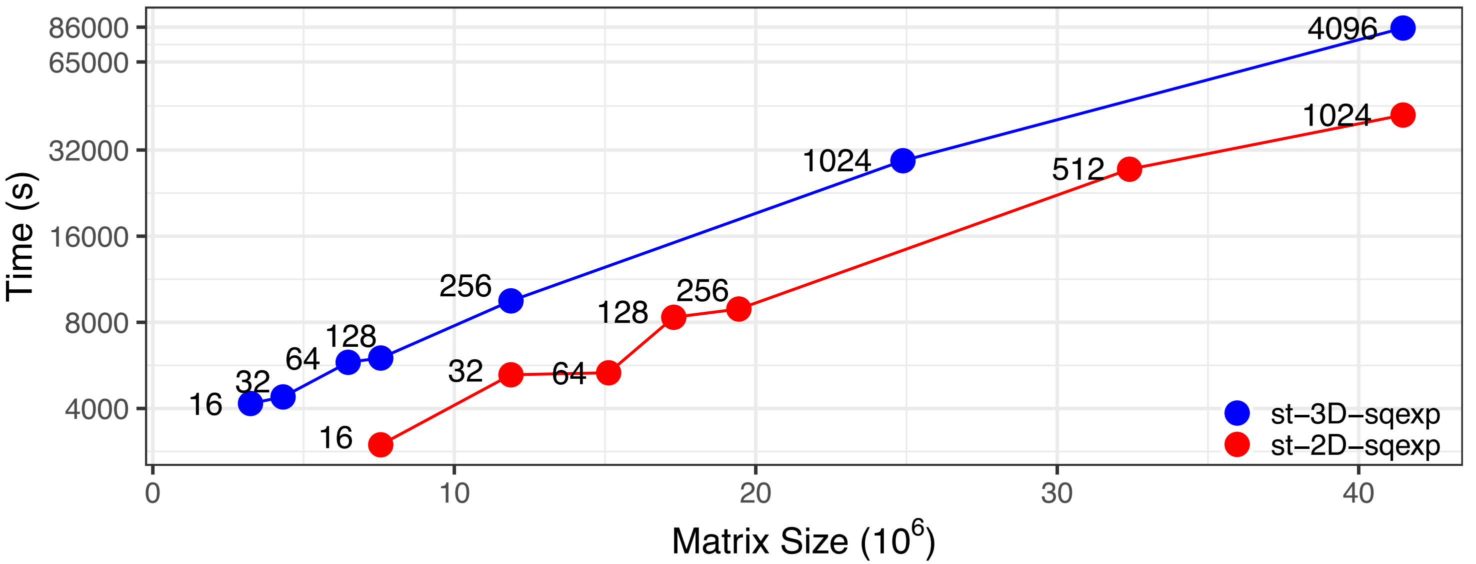


st-2D-sqexp

3D Application and extreme-scale runs



st-3D-sqexp

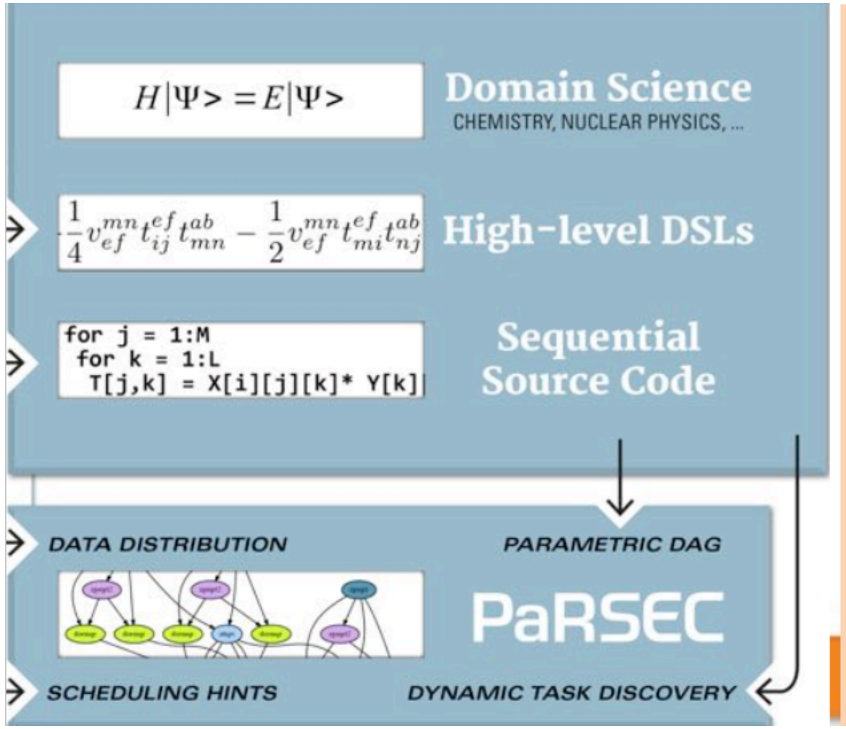


The largest matrices that fit in memory up to 4096 nodes for st-3D-sqexp and 1024 nodes for st-2D-sqexp

Conclusion

- Present the profiling system of PaRSEC: trace collection framework, PINS, Dependency Analysis and Trace;
- Demonstrate the performance analysis using profiling system in PaRSEC to show optimization footprints of TLR Cholesky factorization from data distribution, communication-reducing and synchronization-reducing perspectives;
- Thanks to the optimizations hinted by the profiling system, the new TLR Cholesky factorization achieves an 8X performance speedup over existing state-of-the art implementations on massively parallel supercomputers, solves 3D problem in climate and weather prediction applications and up to 42M geospatial locations on 130,000 cores.

Questions?



Runtime

- Permeable portability layer for heterogeneous architectures
- Scheduling policies adapt every execution to the hardware & ongoing system status
- Data movements between producers and consumers are inferred from dependencies. Communications/computations overlap naturally unfold
- Coherency protocols minimize data movements
- Memory hierarchies (including NVRAM and disk) integral part of the scheduling decisions