On the Autotuning of Task-Based Numerical Libraries for Heterogeneous Architectures

Jesús Cámara

Joint work with

E. Agullo, J. Cuenca and D. Giménez

International Conference on Parallel Computing (ParCo2019)

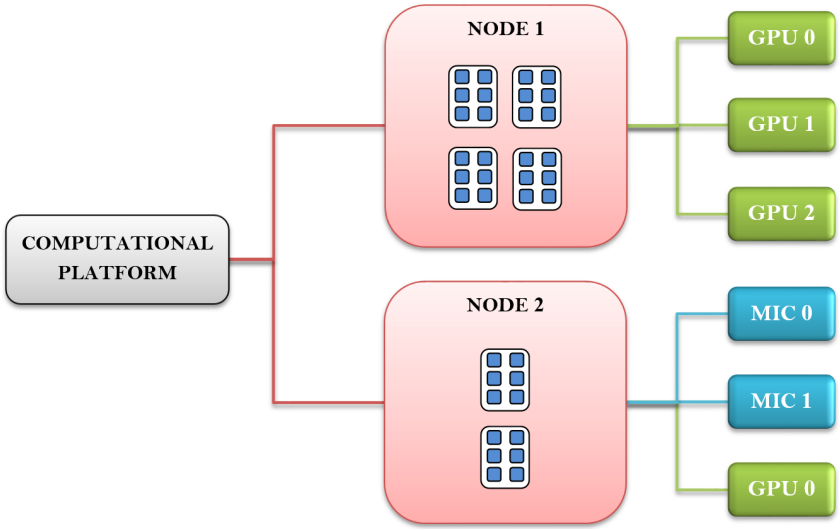
Prague, September 2019

Overview

Overview

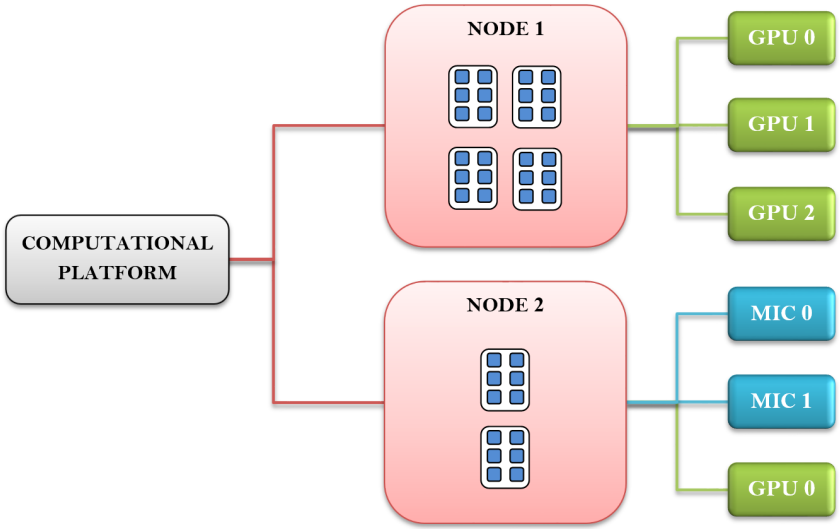
Motivation

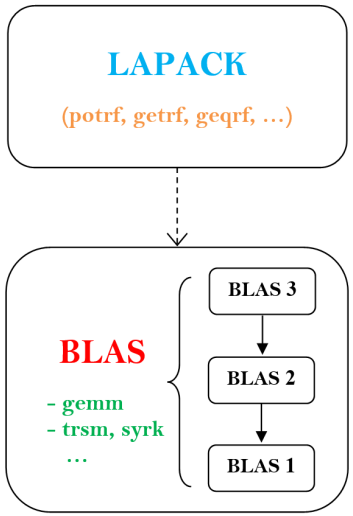
The complexity of modern computational platforms makes the design of high performance numerical libraries extremely challenging.



Motivation

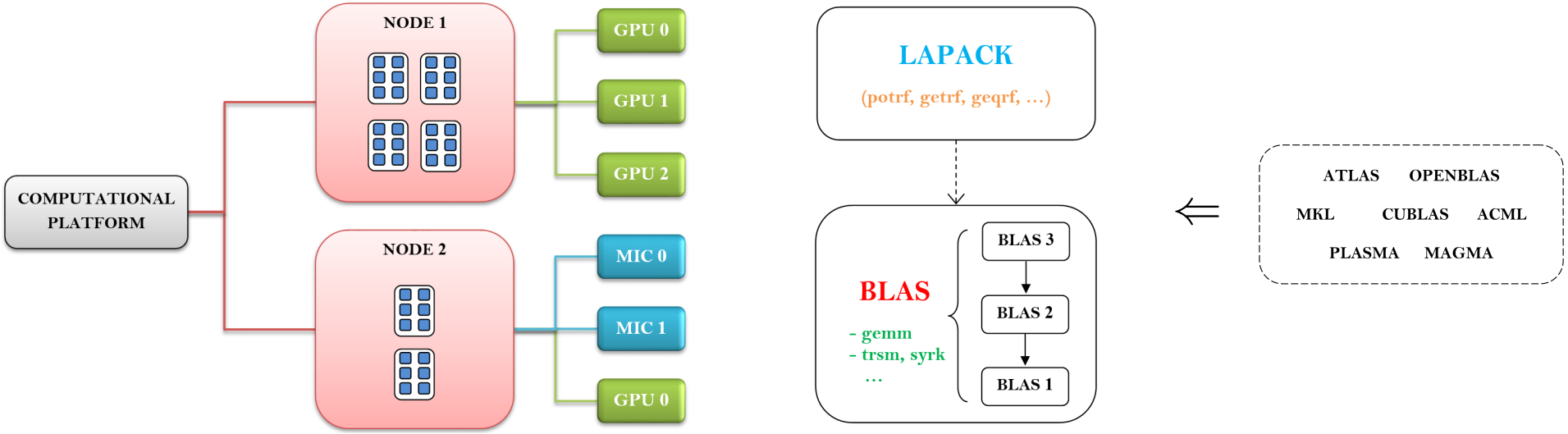
The complexity of modern computational platforms makes the design of high performance numerical libraries extremely challenging.





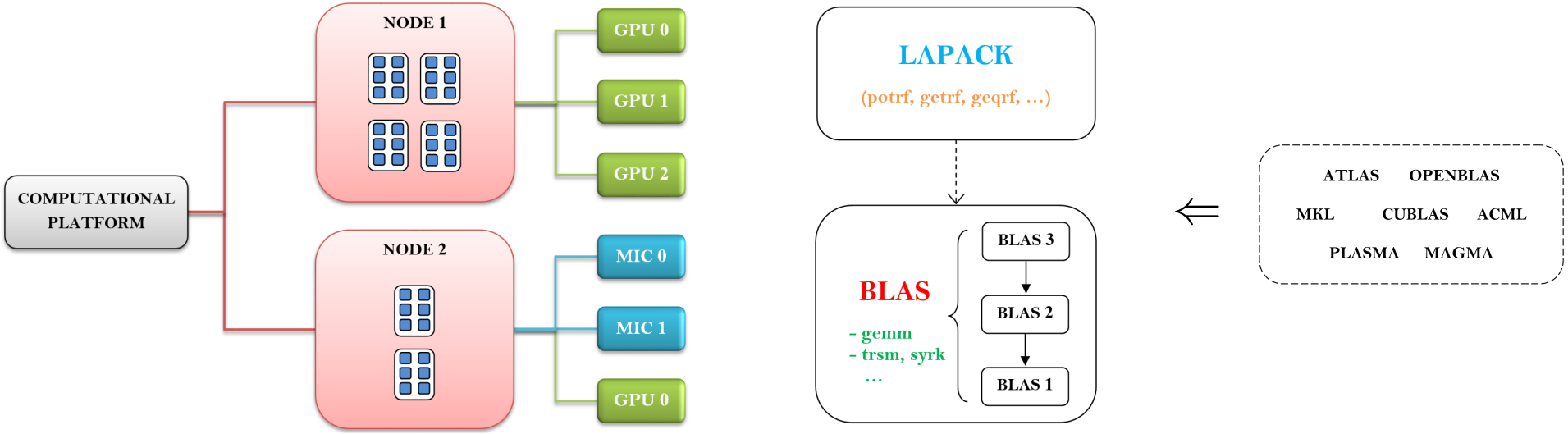
Motivation

The complexity of modern computational platforms makes the design of high performance numerical libraries extremely challenging.



Motivation

The complexity of modern computational platforms makes the design of high performance numerical libraries extremely challenging.



<u>Task-Based Libraries</u> = Sequential Task Flow (STF) design of Linear Algebra Routines on top of Runtime Systems.

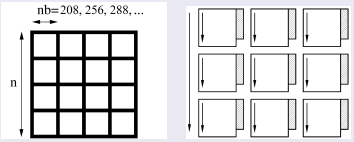
Motivation

Chameleon: Dense Linear Algebra Library derived from PLASMA (tile algorithms) with a STF design on top of Runtime Systems.

Linear Algebra

$AX = B$

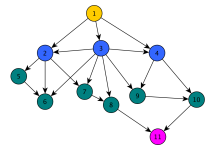
Tile Matrix Layout



Sequential-Task-Flow

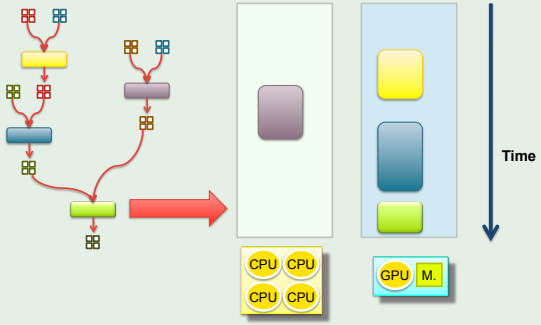
```
for (j = 0; j < N; j++)
    Task(A[j]);
```

Direct Acyclic Graph



$\Rightarrow$

Runtime Systems (Scheduling of Tasks)



StarPU

Optimized Kernels

MKL, cuBLAS, ...

$\Rightarrow$

Heterogeneous Platforms

Optimization Strategies

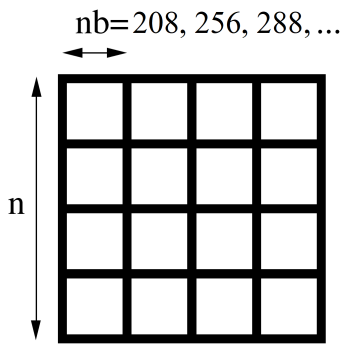
Auto-Tuning

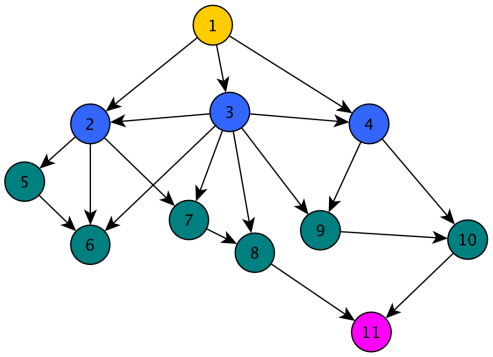
Process to obtain the best value for some algorithmic parameters in order to reduce the execution time of the routines with an efficient use of the heterogeneous platform.

Optimization Strategies

Auto-Tuning

Process to obtain the best value for some algorithmic parameters in order to reduce the execution time of the routines with an efficient use of the heterogeneous platform.

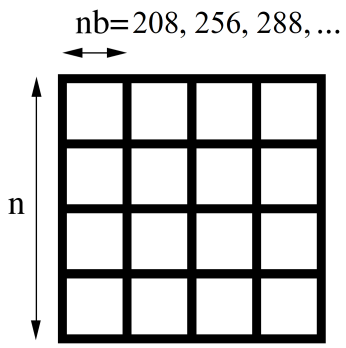


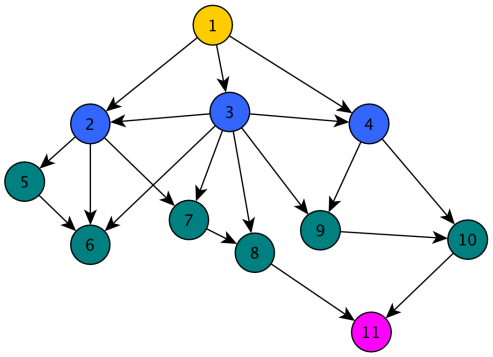


Optimization Strategies

Auto-Tuning

Process to obtain the best value for some algorithmic parameters in order to reduce the execution time of the routines with an efficient use of the heterogeneous platform.





Performance vs. Concurrency

Trade off between the performance of a task (the higher *nb*, the higher the performance) and the concurrency between tasks (the smaller *nb*, the wider the DAG of tasks).

Optimization Strategies

The RS does most of the work:

- Handles data consistency.
- Handles data dependencies.
- Handles scheduling of tasks to the CUs.
- ...but it not takes care about the best value for some APs.

Optimization Strategies

The RS does most of the work:

- Handles data consistency.
- Handles data dependencies.
- Handles scheduling of tasks to the CUs.
- ...but it not takes care about the best value for some APs.

Optimization Strategies

The RS does most of the work:

- Handles data consistency.
- Handles data dependencies.
- Handles scheduling of tasks to the CUs.
- ...but it not takes care about the best value for some APs.

Optimization Strategies

The RS does most of the work:

- Handles data consistency.
- Handles data dependencies.
- Handles scheduling of tasks to the CUs.
- ...but it not takes care about the best value for some APs.

Optimization Strategies

The RS does most of the work:

- Handles data consistency.
- Handles data dependencies.
- Handles scheduling of tasks to the CUs.
- ...but it not takes care about the best value for some APs.

Optimization Strategies

The RS does most of the work:

- Handles data consistency.
- Handles data dependencies.
- Handles scheduling of tasks to the CUs.
- ...but it not takes care about the best value for some APs.

Which parameters to consider?

Optimization Strategies

The RS does most of the work:

- Handles data consistency.
- Handles data dependencies.
- Handles scheduling of tasks to the CUs.
- ...but it not takes care about the best value for some APs.

Which parameters to consider?

- Tile-Size.

Optimization Strategies

The RS does most of the work:

- Handles data consistency.
- Handles data dependencies.
- Handles scheduling of tasks to the CUs.
- ...but it not takes care about the best value for some APs.

Which parameters to consider?

- Tile-Size.
- Number of Computing Units.

Optimization Strategies

The RS does most of the work:

- Handles data consistency.
- Handles data dependencies.
- Handles scheduling of tasks to the CUs.
- ...but it not takes care about the best value for some APs.

Which parameters to consider?

- Tile-Size.
- Number of Computing Units.

Other Parameters?

Scheduling Policies, Threads per CPU kernel, ...

Optimization Strategies

The RS does most of the work:

- Handles data consistency.
- Handles data dependencies.
- Handles scheduling of tasks to the CUs.
- ...but it not takes care about the best value for some APs.

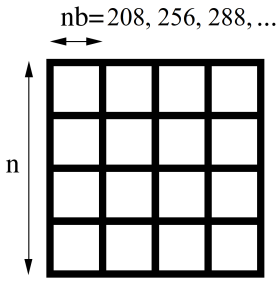
Which parameters to consider?

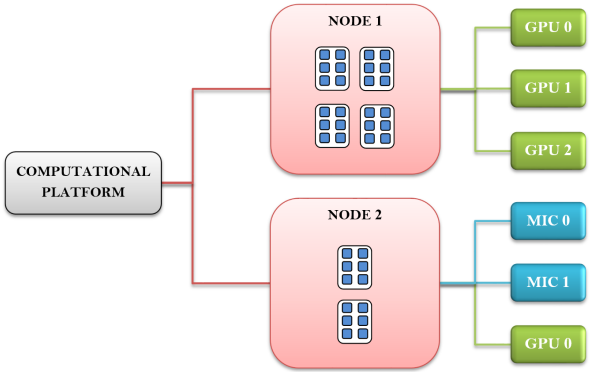
- Tile-Size.
- Number of Computing Units.

Other Parameters? Not adressed here!

Scheduling Policies, Threads per CPU kernel, ...

Optimization Strategies

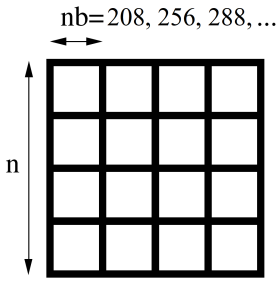


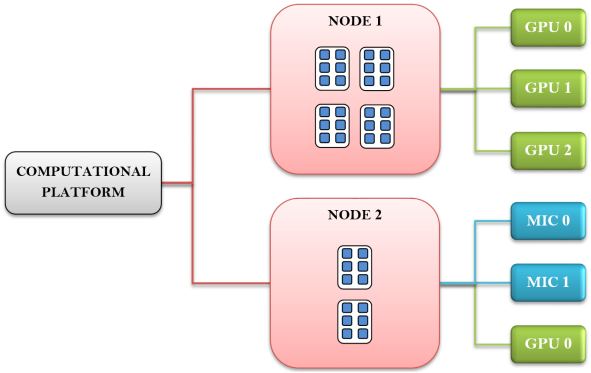


Goal

- Search the best *nb* value for each problem size *n*.

Optimization Strategies

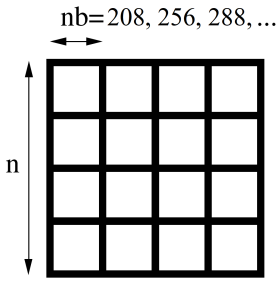


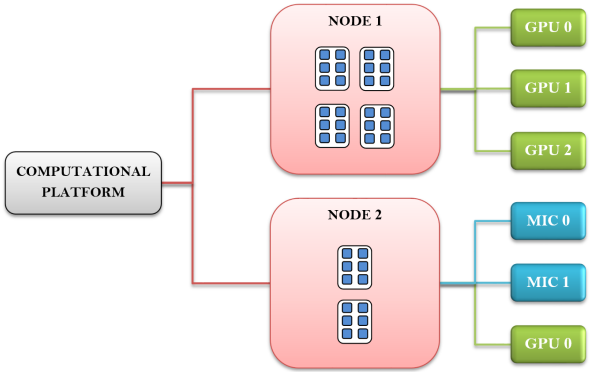


Goal

- Search the best *nb* value for each problem size *n*.
- Select the number of computing units to use.

Optimization Strategies





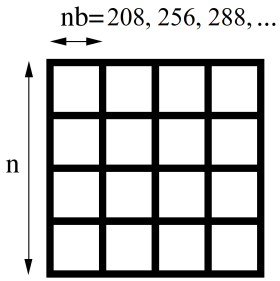
Goal

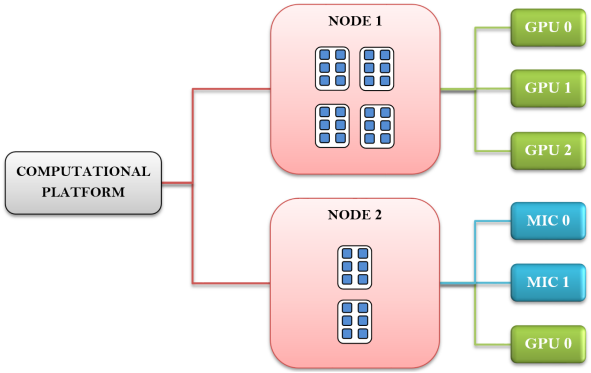
- Search the best *nb* value for each problem size *n*.
- Select the number of computing units to use.

Four Strategies (S1, S2, S3, S4):

- **S1**: Empirical+Exhaustive.
- **S2**: Empirical+Pruned.
- **S3**: Simulated+Exhaustive.
- **S4**: Simulated+Pruned.

Optimization Strategies





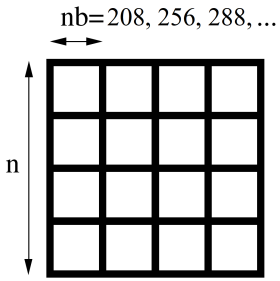
Goal

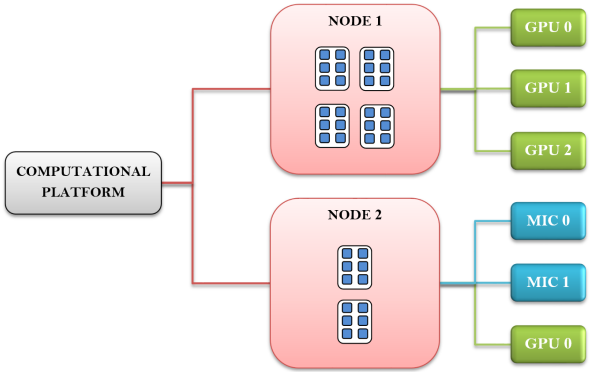
- Search the best *nb* value for each problem size *n*.
- Select the number of computing units to use.

Four Strategies (S1, S2, S3, S4):

- **S1**: Empirical+Exhaustive.
- **S2**: Empirical+Pruned.
- **S3**: Simulated+Exhaustive.
- **S4**: Simulated+Pruned.

Optimization Strategies





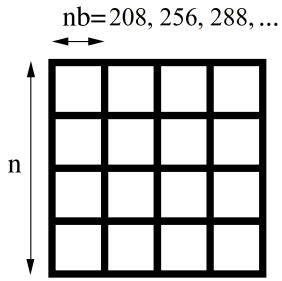
Goal

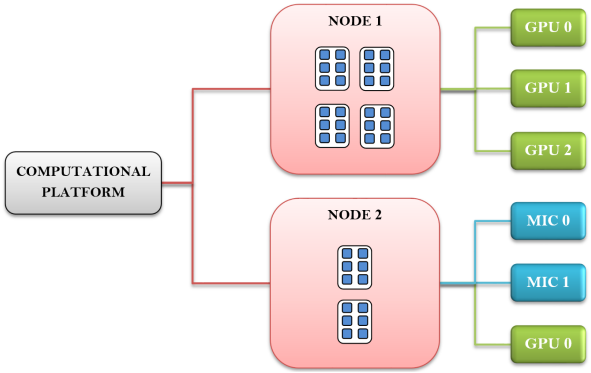
- Search the best *nb* value for each problem size *n*.
- Select the number of computing units to use.

Four Strategies (S1, S2, S3, S4):

- **S1**: Empirical+Exhaustive.
- **S2**: Empirical+Pruned.
- **S3**: Simulated+Exhaustive.
- **S4**: Simulated+Pruned.

Optimization Strategies





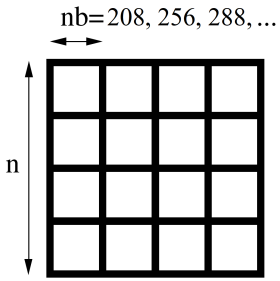
Goal

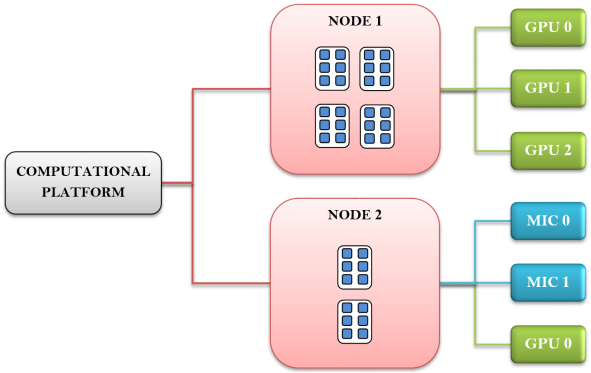
- Search the best *nb* value for each problem size *n*.
- Select the number of computing units to use.

Four Strategies (S1, S2, S3, S4):

- **S1**: Empirical+Exhaustive.
- **S2**: Empirical+Pruned.
- **S3**: Simulated+Exhaustive.
- **S4**: Simulated+Pruned.

Optimization Strategies





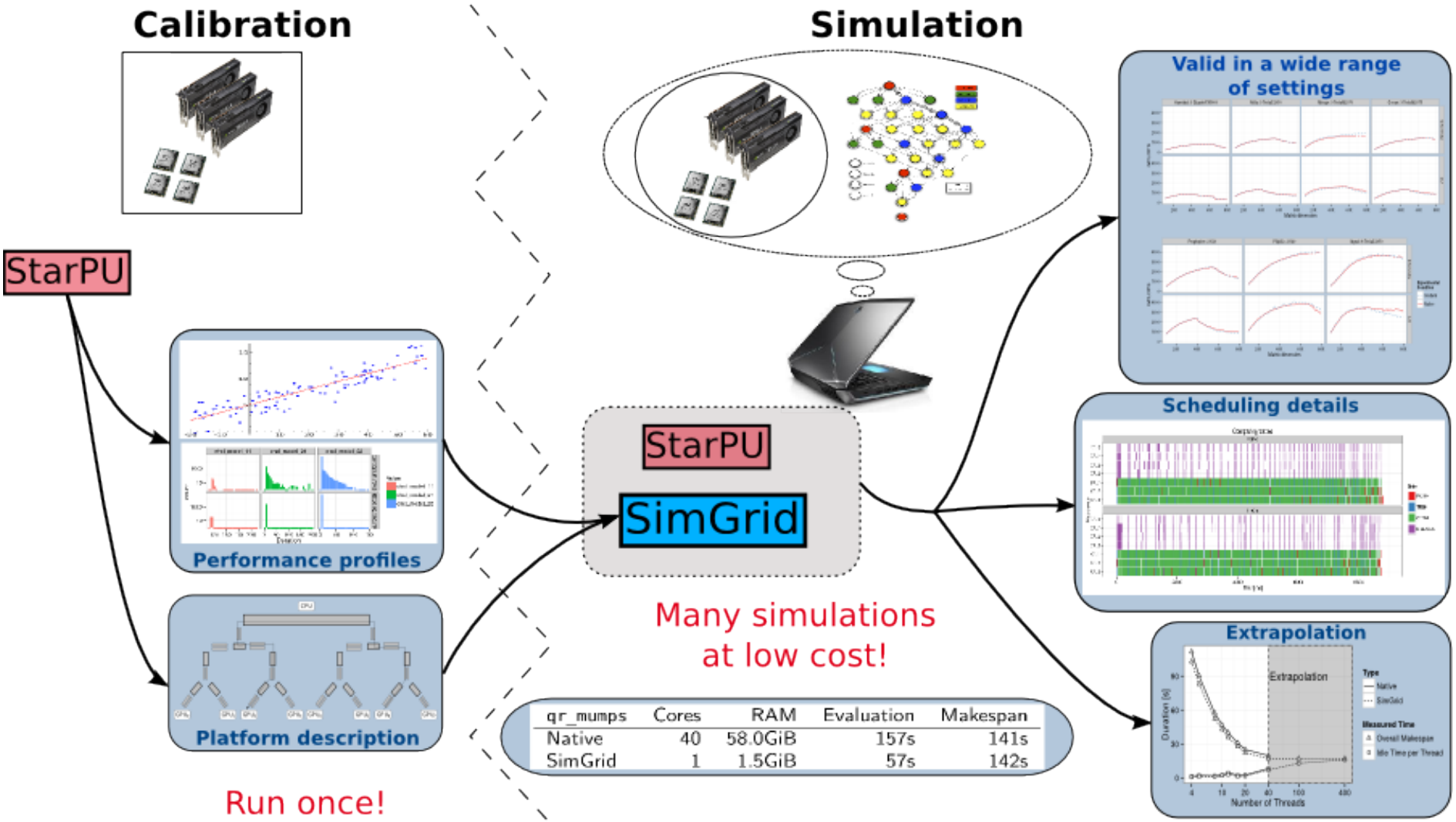
Goal

- Search the best *nb* value for each problem size *n*.
- Select the number of computing units to use.

Four Strategies (S1, S2, S3, S4):

- **S1**: Empirical+Exhaustive.
- **S2**: Empirical+Pruned.
- **S3**: Simulated+Exhaustive.
- **S4**: Simulated+Pruned.

Optimization Strategies

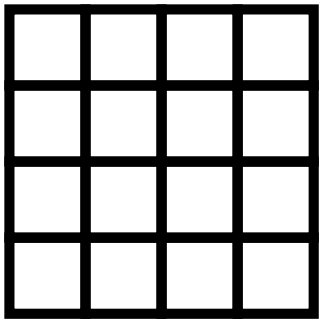


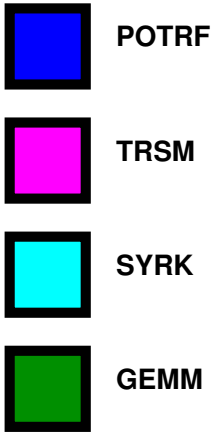
qr_mumps	Cores	RAM	Evaluation	Makespan
Native	40	58.0GiB	157s	141s
SimGrid	1	1.5GiB	57s	142s

Overview

The Cholesky Algorithm

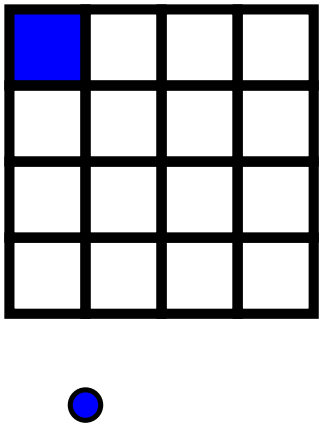
```
for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW,A[i][j], R,A[j][j]);
    }
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++) {
            GEMM (RW,A[i][k],
                  R,A[i][j],
                  R,A[k][j]);
        }
    }
}
```

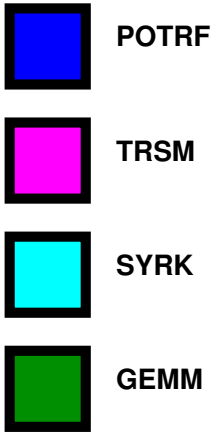




The Cholesky Algorithm

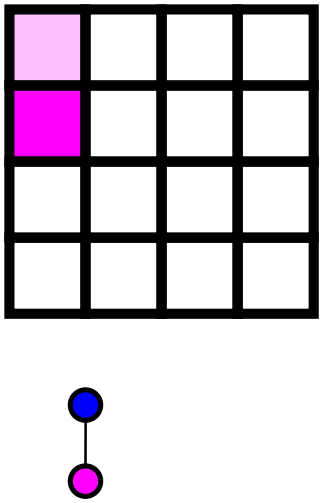
```
for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW,A[i][j], R,A[j][j]);
    }
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++) {
            GEMM (RW,A[i][k],
                  R,A[i][j],
                  R,A[k][j]);
        }
    }
}
```

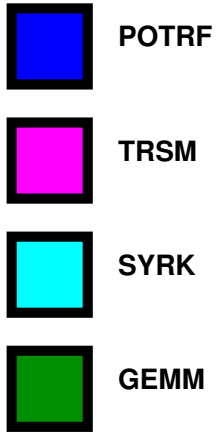




The Cholesky Algorithm

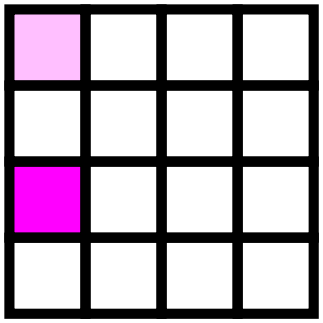
```
for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW,A[i][j], R,A[j][j]);
    }
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++) {
            GEMM (RW,A[i][k],
                  R,A[i][j],
                  R,A[k][j]);
        }
    }
}
```

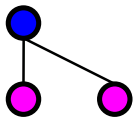


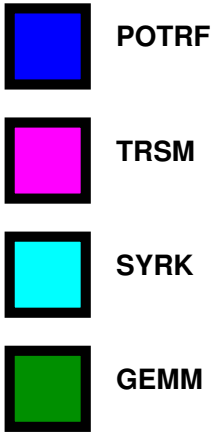


The Cholesky Algorithm

```
for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW,A[i][j], R,A[j][j]);
    }
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++) {
            GEMM (RW,A[i][k],
                  R,A[i][j],
                  R,A[k][j]);
        }
    }
}
```

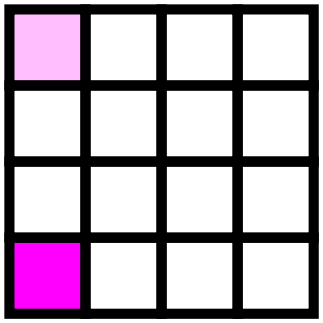


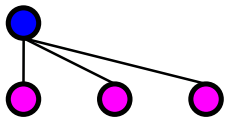


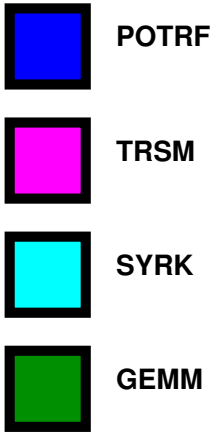


The Cholesky Algorithm

```
for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW,A[i][j], R,A[j][j]);
    }
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++) {
            GEMM (RW,A[i][k],
                  R,A[i][j],
                  R,A[k][j]);
        }
    }
}
```

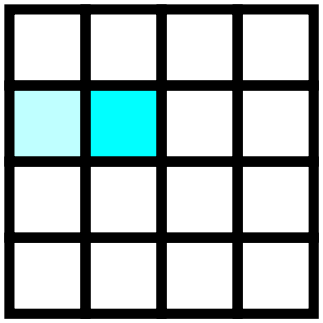


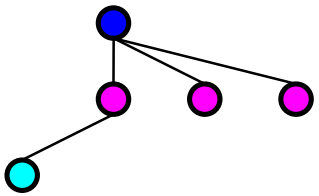


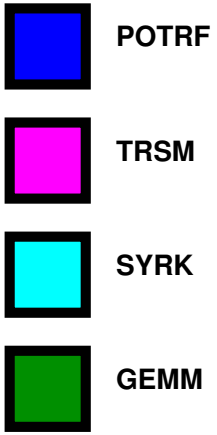


The Cholesky Algorithm

```
for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW,A[i][j], R,A[j][j]);
    }
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++) {
            GEMM (RW,A[i][k],
                  R,A[i][j],
                  R,A[k][j]);
        }
    }
}
```

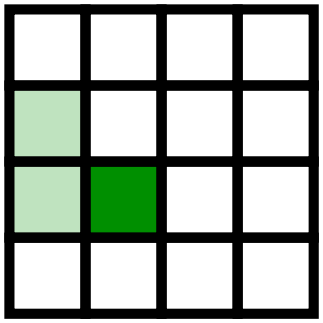


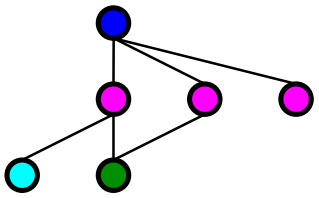


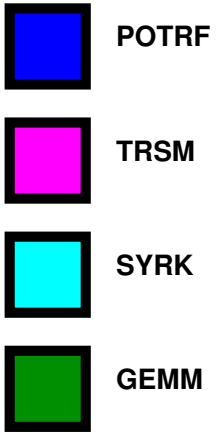


The Cholesky Algorithm

```
for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW,A[i][j], R,A[j][j]);
    }
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++) {
            GEMM (RW,A[i][k],
                  R,A[i][j],
                  R,A[k][j]);
        }
    }
}
```

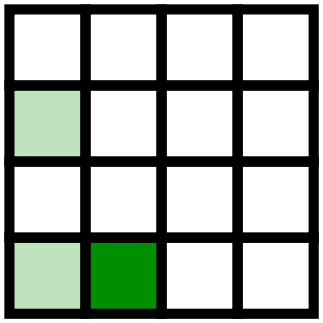


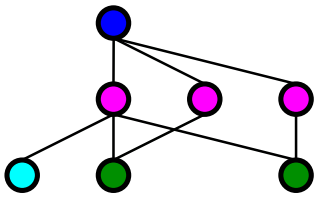


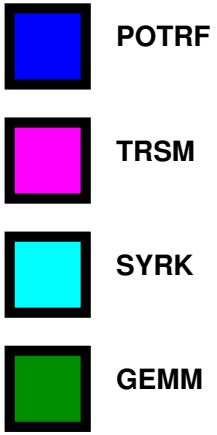


The Cholesky Algorithm

```
for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW,A[i][j], R,A[j][j]);
    }
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++) {
            GEMM (RW,A[i][k],
                  R,A[i][j],
                  R,A[k][j]);
        }
    }
}
```

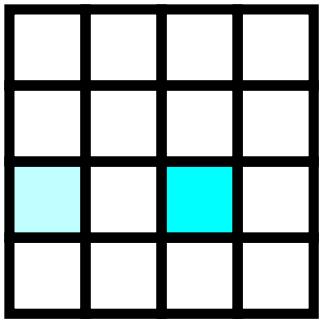


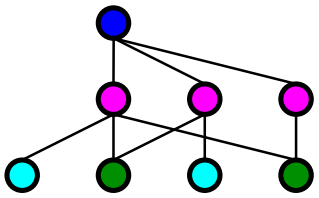


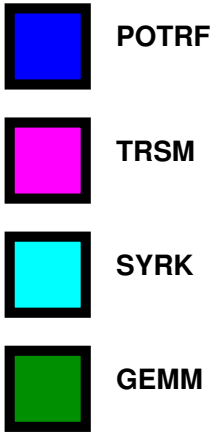


The Cholesky Algorithm

```
for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW,A[i][j], R,A[j][j]);
    }
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++) {
            GEMM (RW,A[i][k],
                  R,A[i][j],
                  R,A[k][j]);
        }
    }
}
```

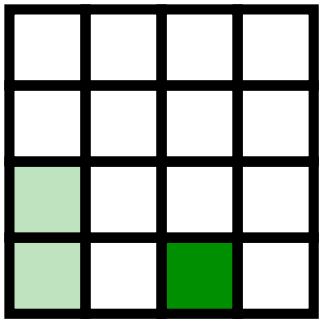


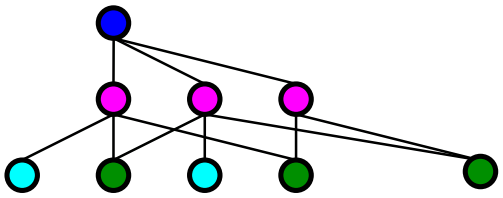


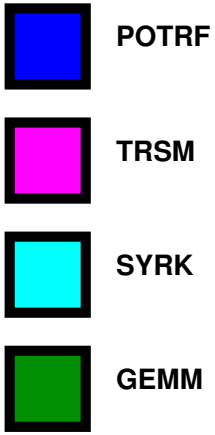


The Cholesky Algorithm

```
for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW,A[i][j], R,A[j][j]);
    }
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++) {
            GEMM (RW,A[i][k],
                  R,A[i][j],
                  R,A[k][j]);
        }
    }
}
```

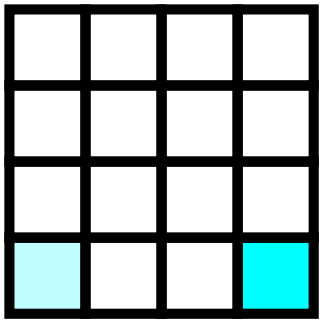


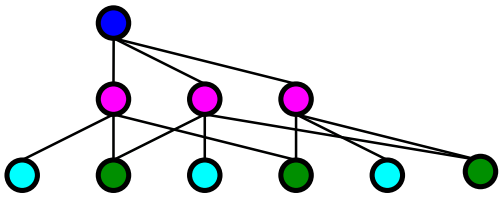


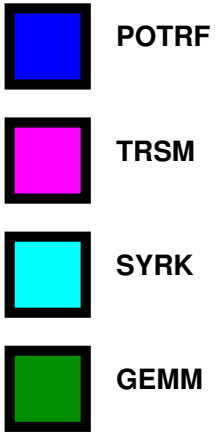


The Cholesky Algorithm

```
for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW,A[i][j], R,A[j][j]);
    }
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++) {
            GEMM (RW,A[i][k],
                  R,A[i][j],
                  R,A[k][j]);
        }
    }
}
```

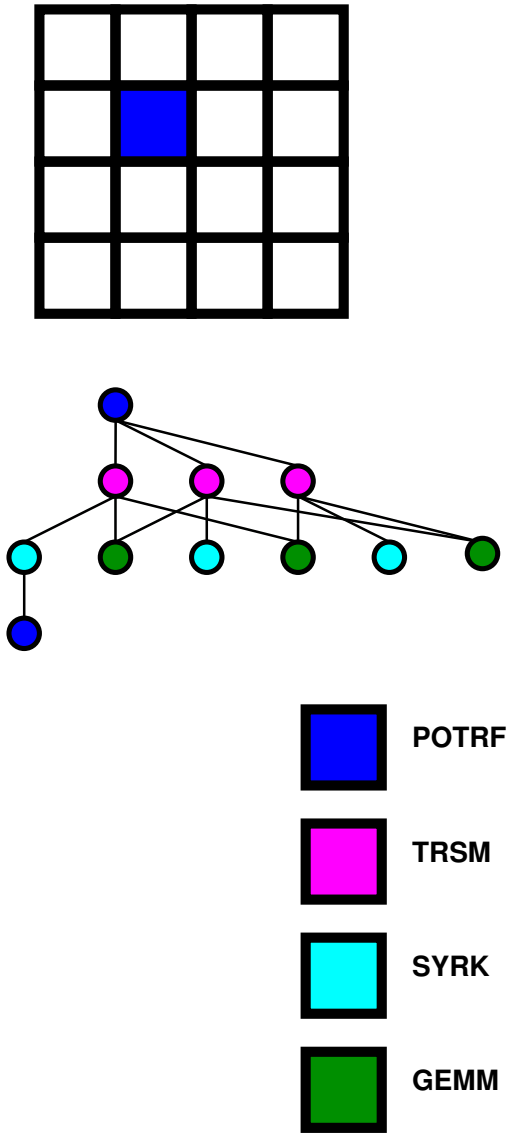






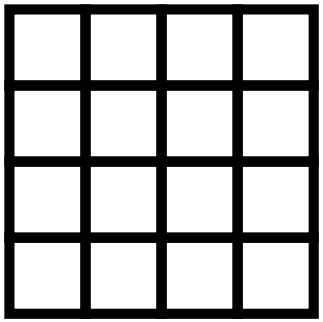
The Cholesky Algorithm

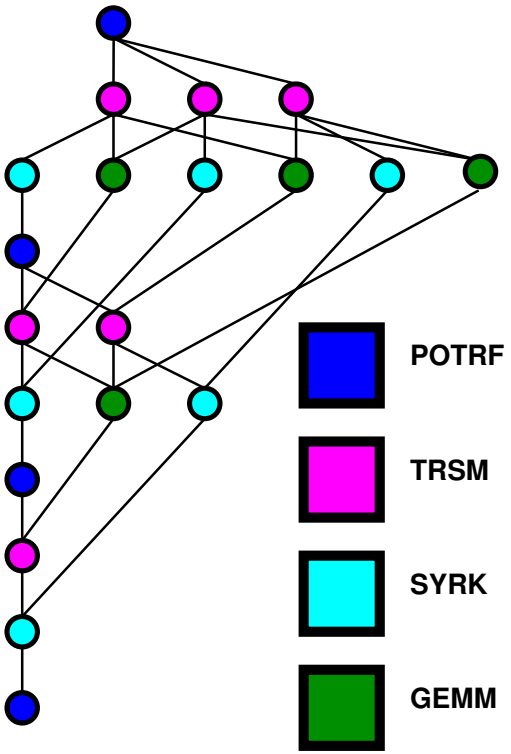
```
for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW,A[i][j], R,A[j][j]);
    }
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++) {
            GEMM (RW,A[i][k],
                  R,A[i][j],
                  R,A[k][j]);
        }
    }
}
```



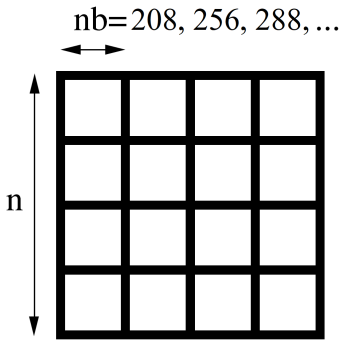
The Cholesky Algorithm

```
for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW,A[i][j], R,A[j][j]);
    }
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++) {
            GEMM (RW,A[i][k],
                  R,A[i][j],
                  R,A[k][j]);
        }
    }
}
```





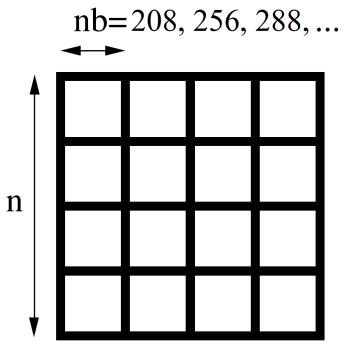
The Cholesky Algorithm



Goal

Search the best *nb* value for a given problem size *n*.

The Cholesky Algorithm

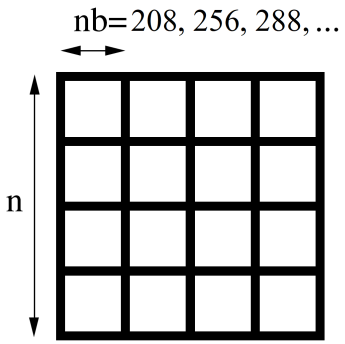


Goal

Search the best *nb* value for a given problem size *n*.

Which *nb* range?

The Cholesky Algorithm



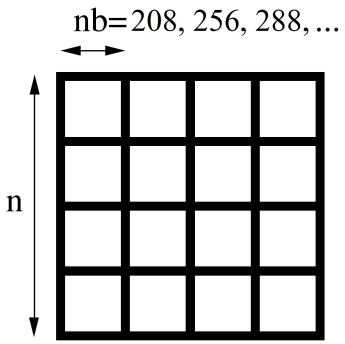
Goal

Search the best *nb* value for a given problem size *n*.

Which *nb* range?

- In principle any.

The Cholesky Algorithm



Goal

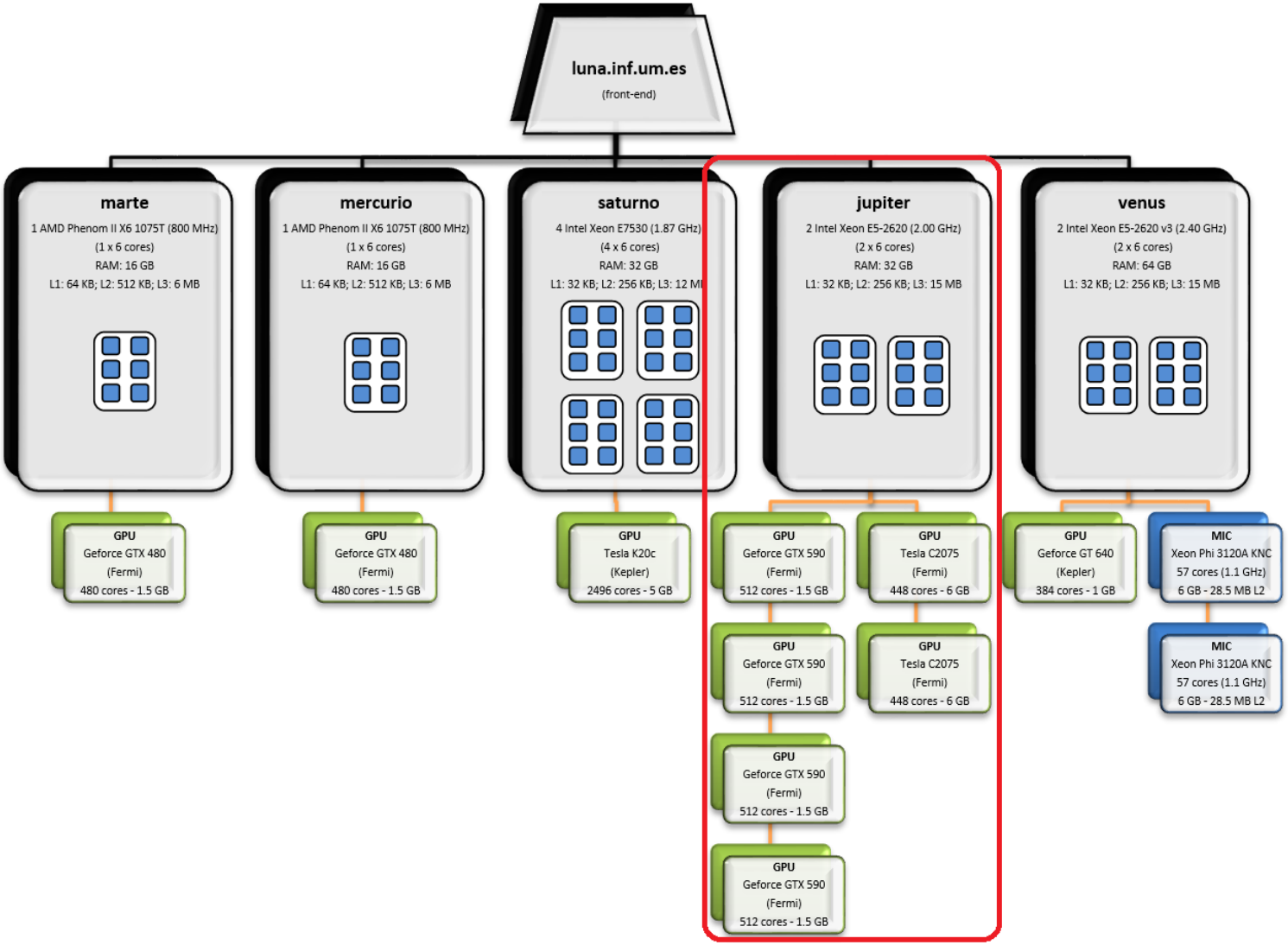
Search the best *nb* value for a given problem size *n*.

Which *nb* range?

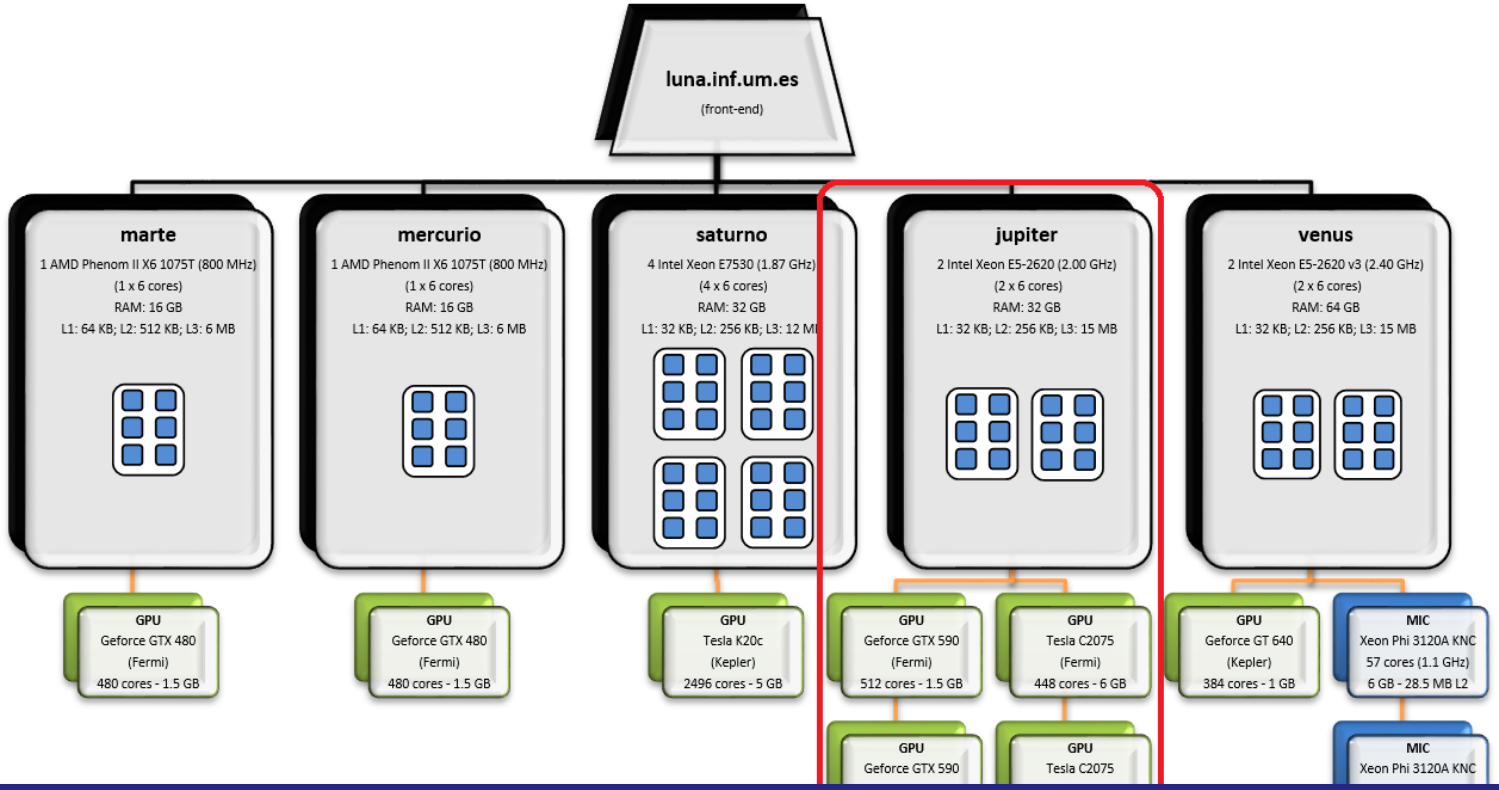
- In principle any.
- Only 8 values considered in this work:
  $\{208, 256, 288, 320, 384, 448, 512, 576\}$

Overview

Heterogeneous Platform



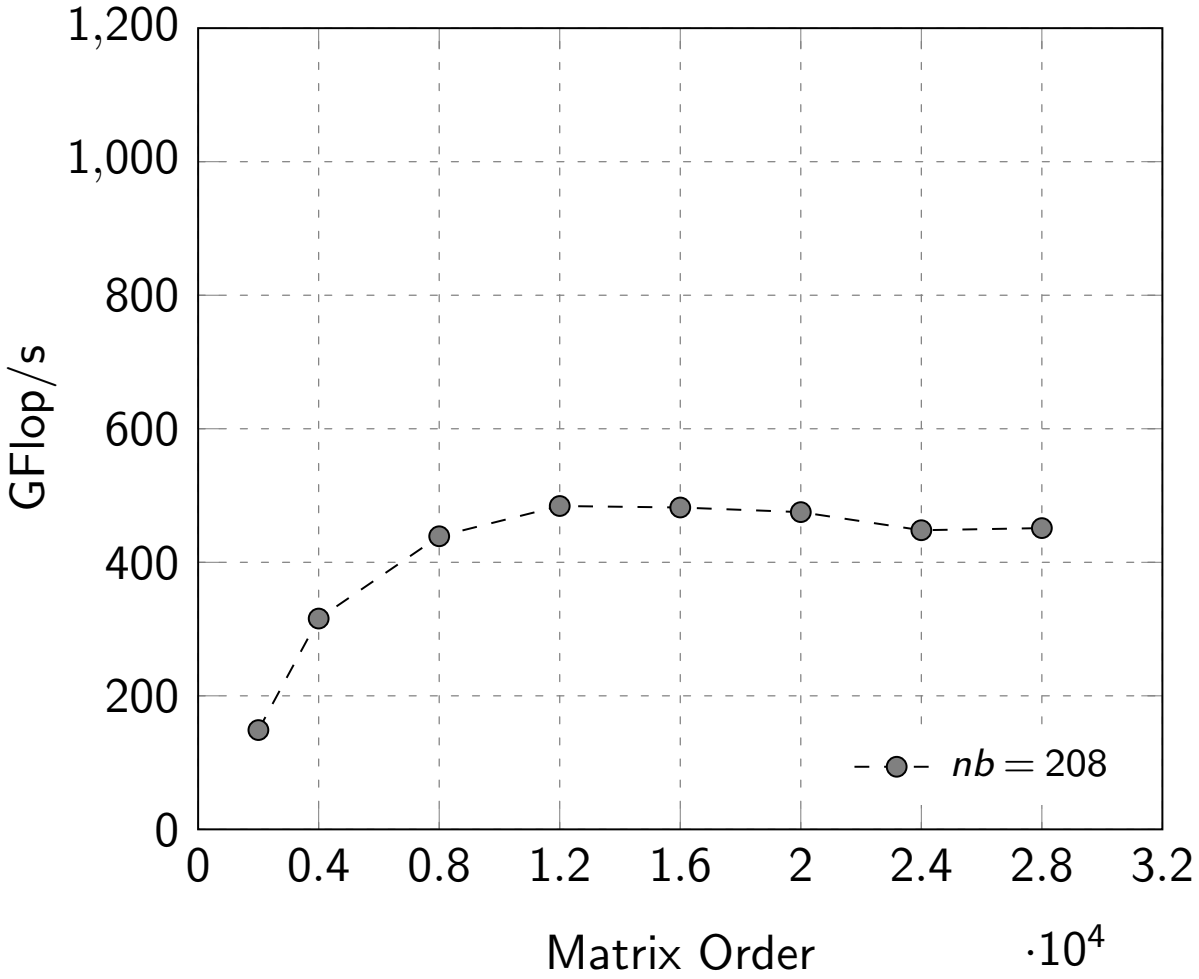
Heterogeneous Platform



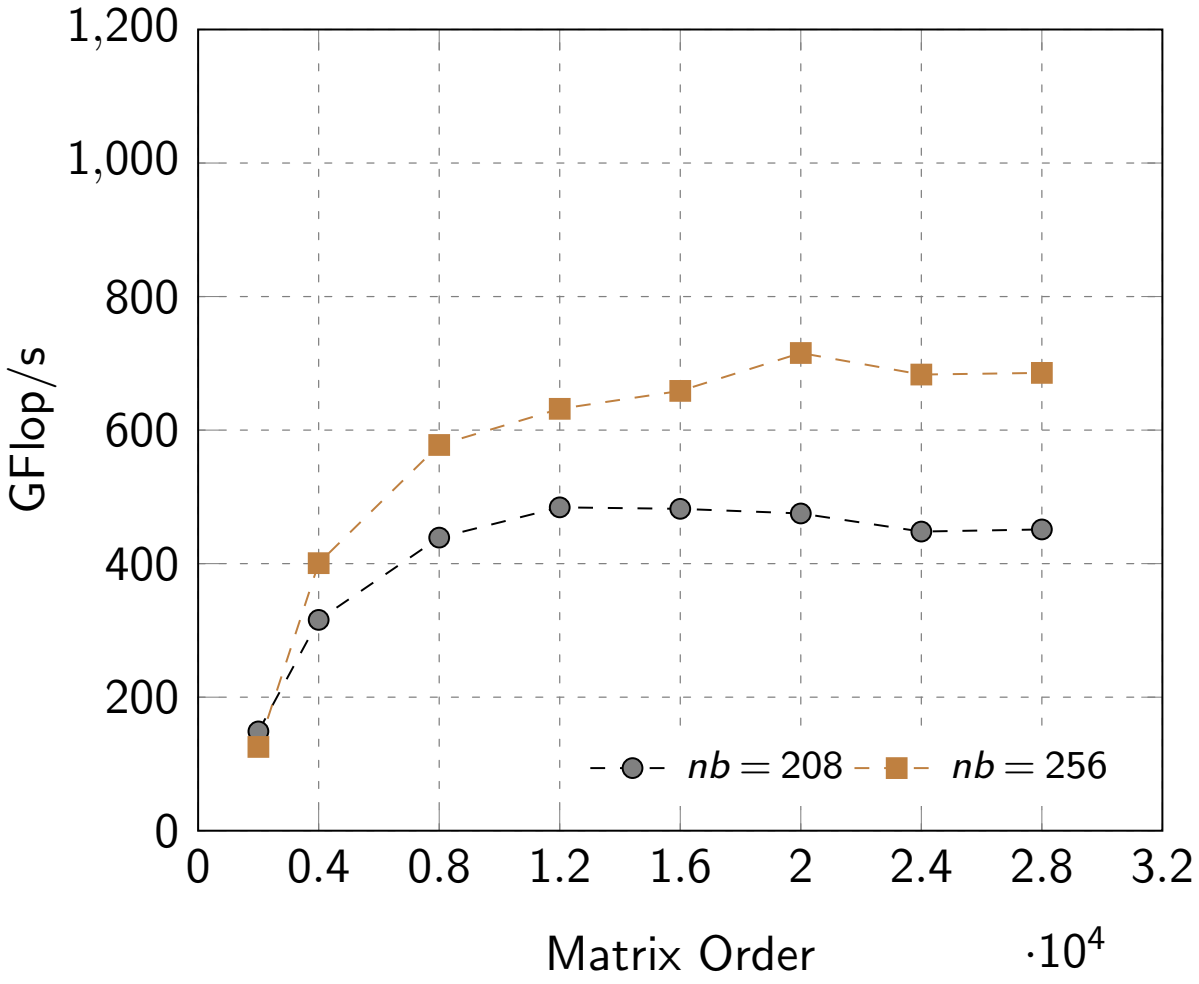
Jupiter

- 2 Intel Xeon E5-2620 Hexa-Core (12 CPU Cores).
- 4 GPU NVIDIA GeForce GTX590.
- 2 GPU NVIDIA Tesla C2075.

S1: Empirical+Exhaustive

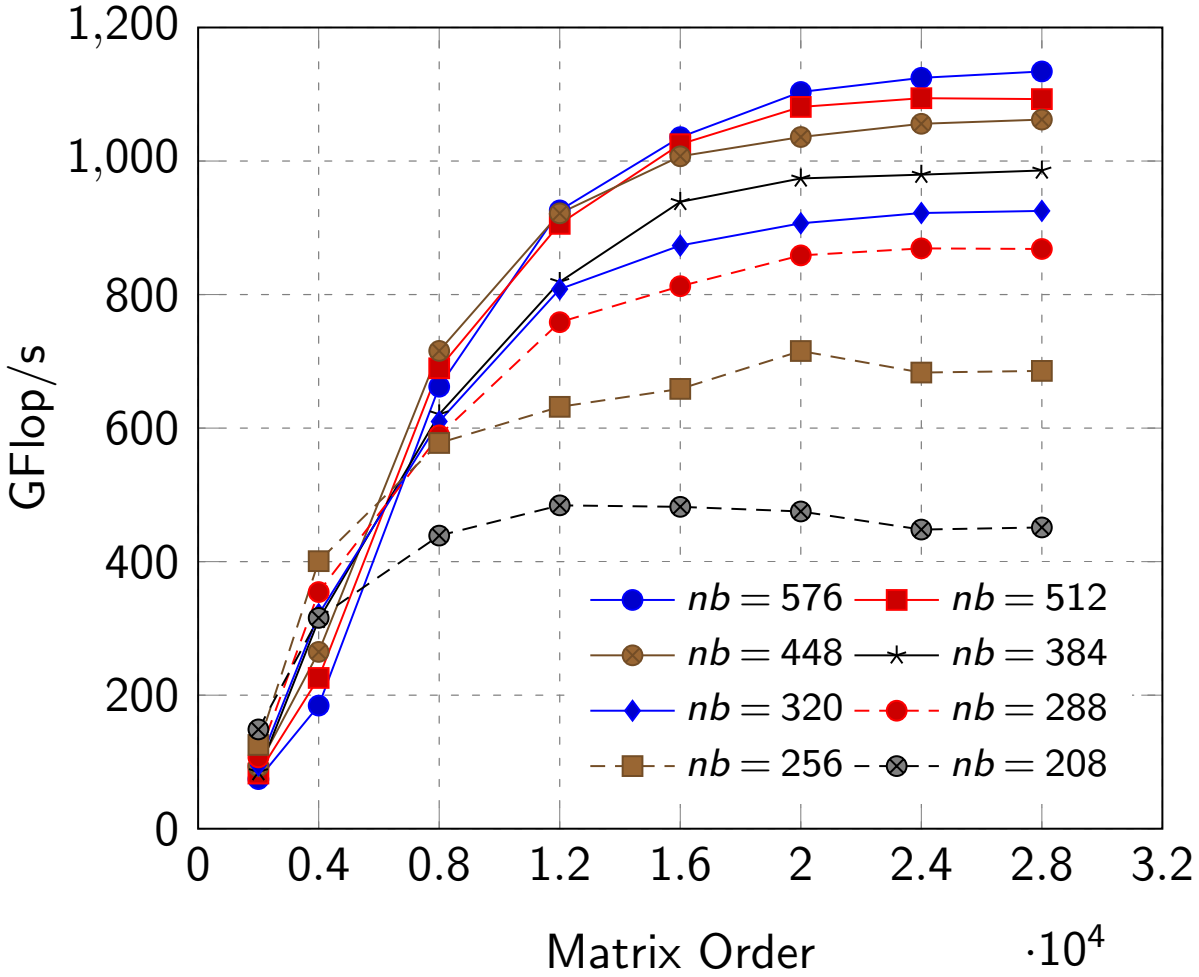


S1: Empirical+Exhaustive



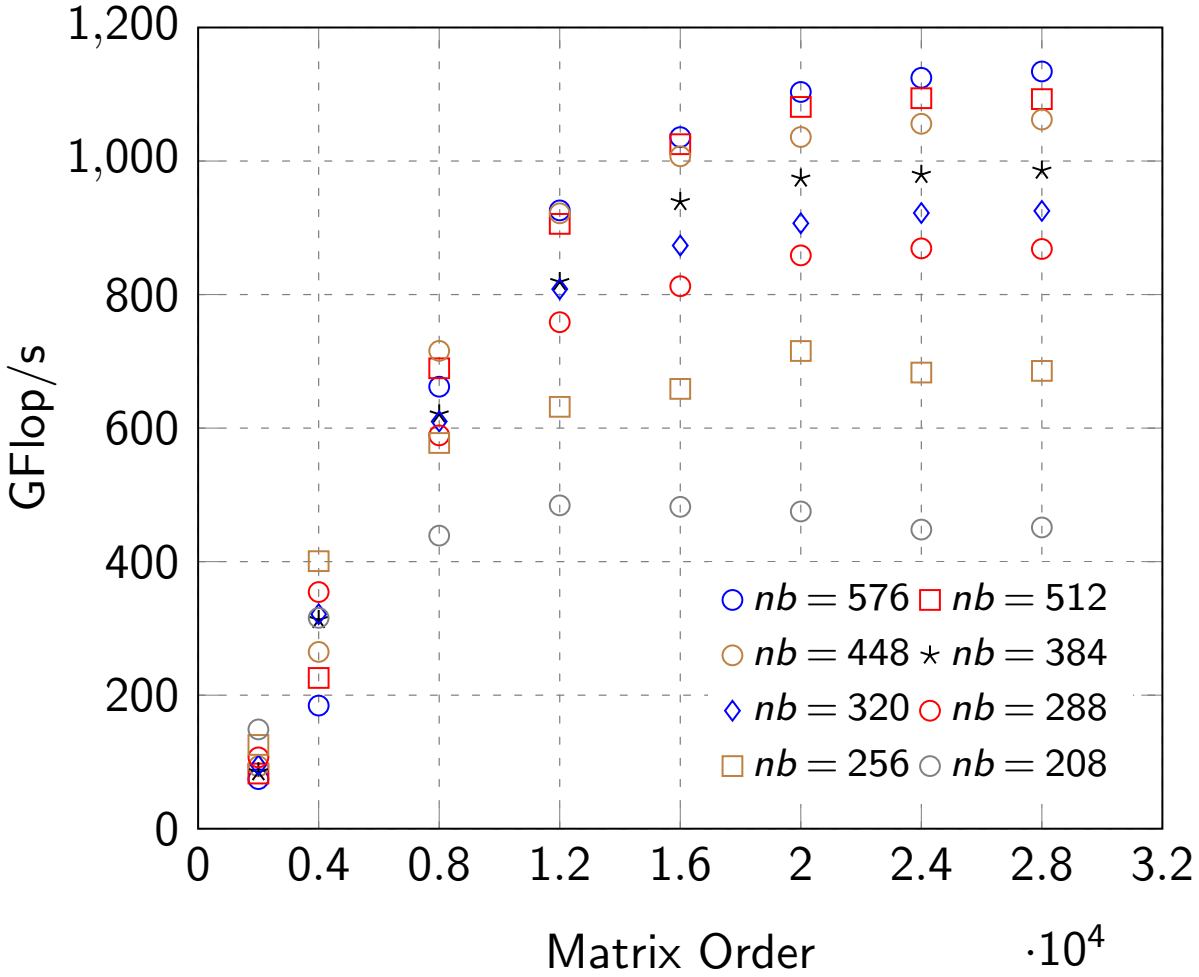
Performance depends on the value of the Tile Size

S1: Empirical+Exhaustive



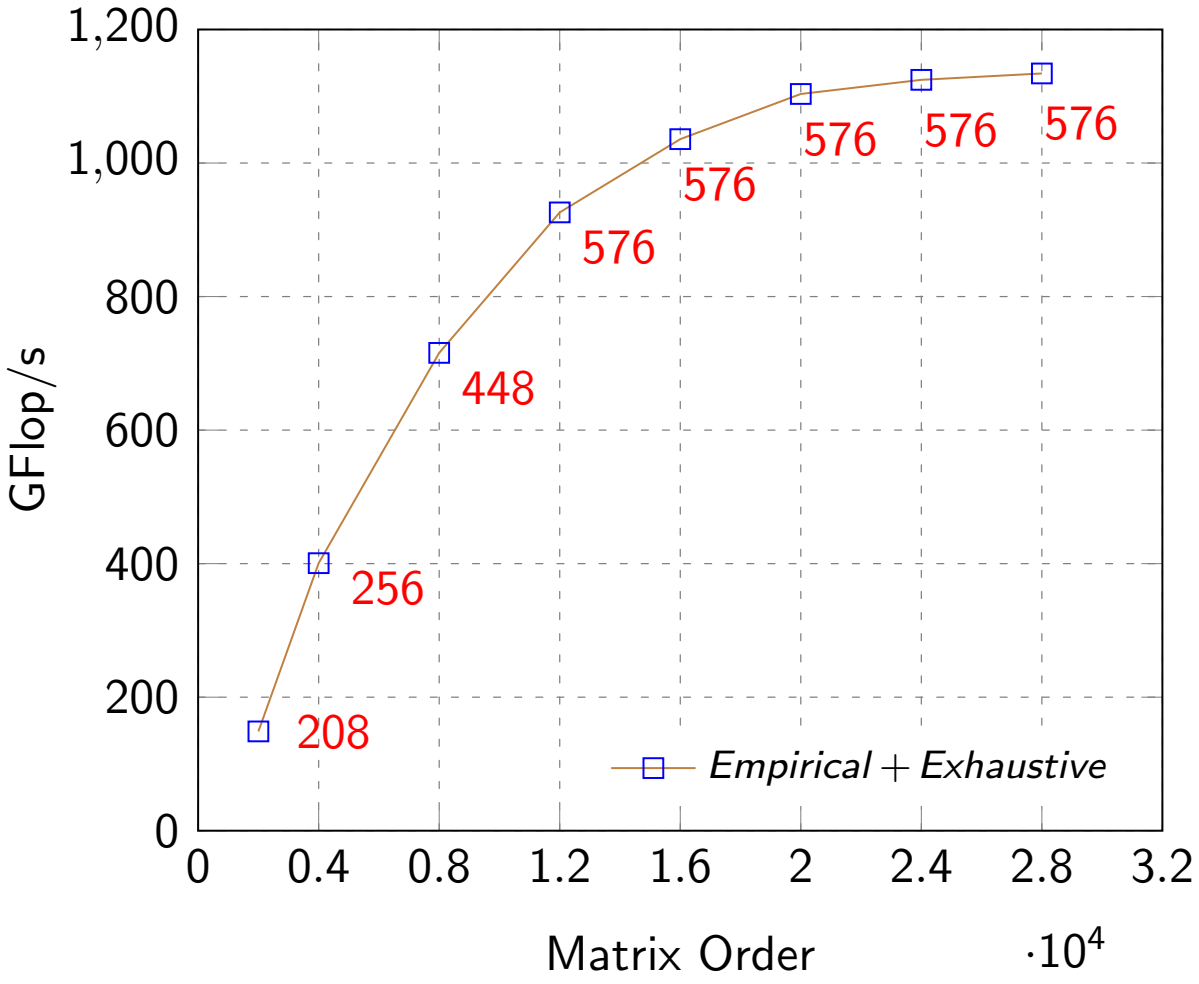
No. Experiments: 8 problem sizes * 8 tile sizes = 64

S1: Empirical+Exhaustive



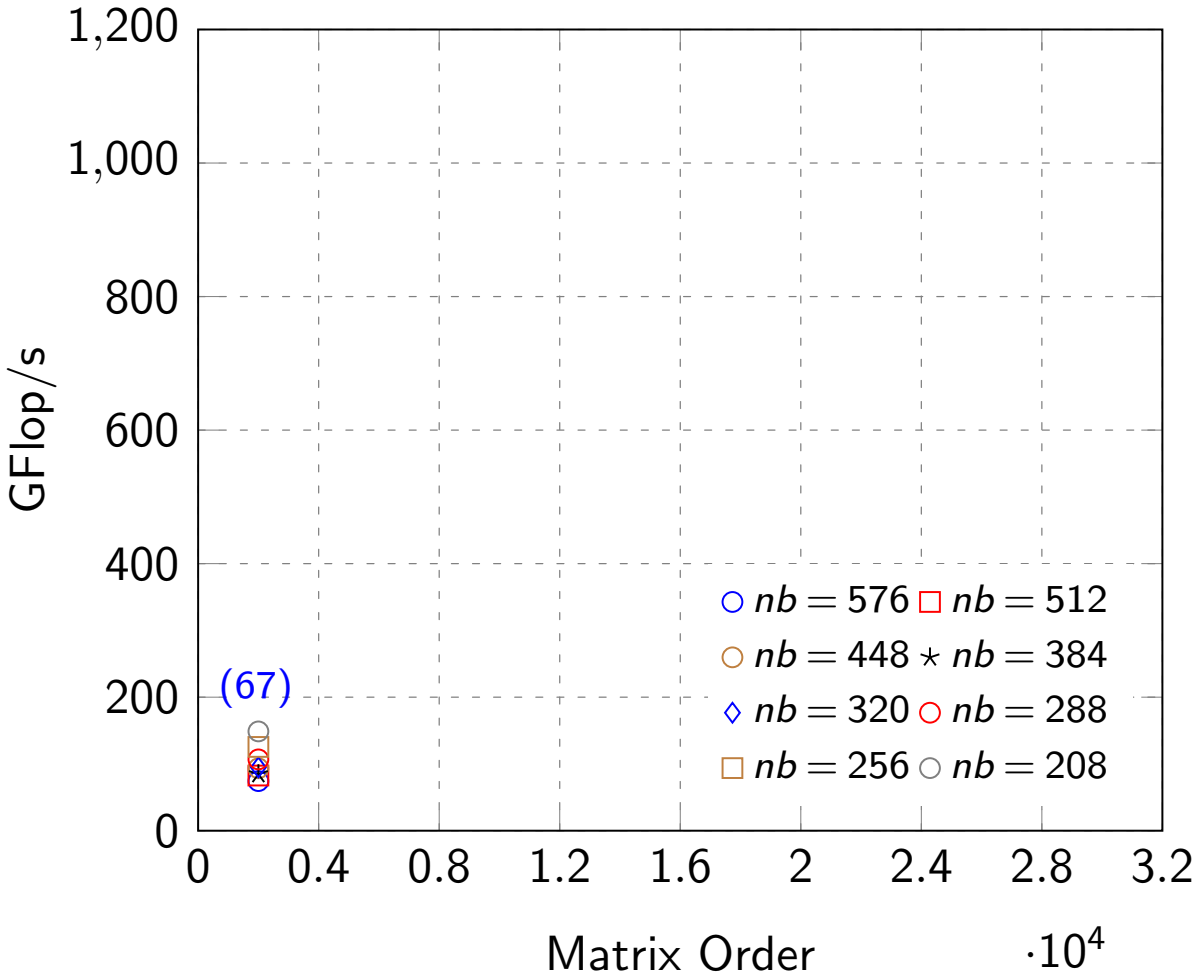
Which *nb* among these 8 values for a *n* given at runtime?

S1: Empirical+Exhaustive



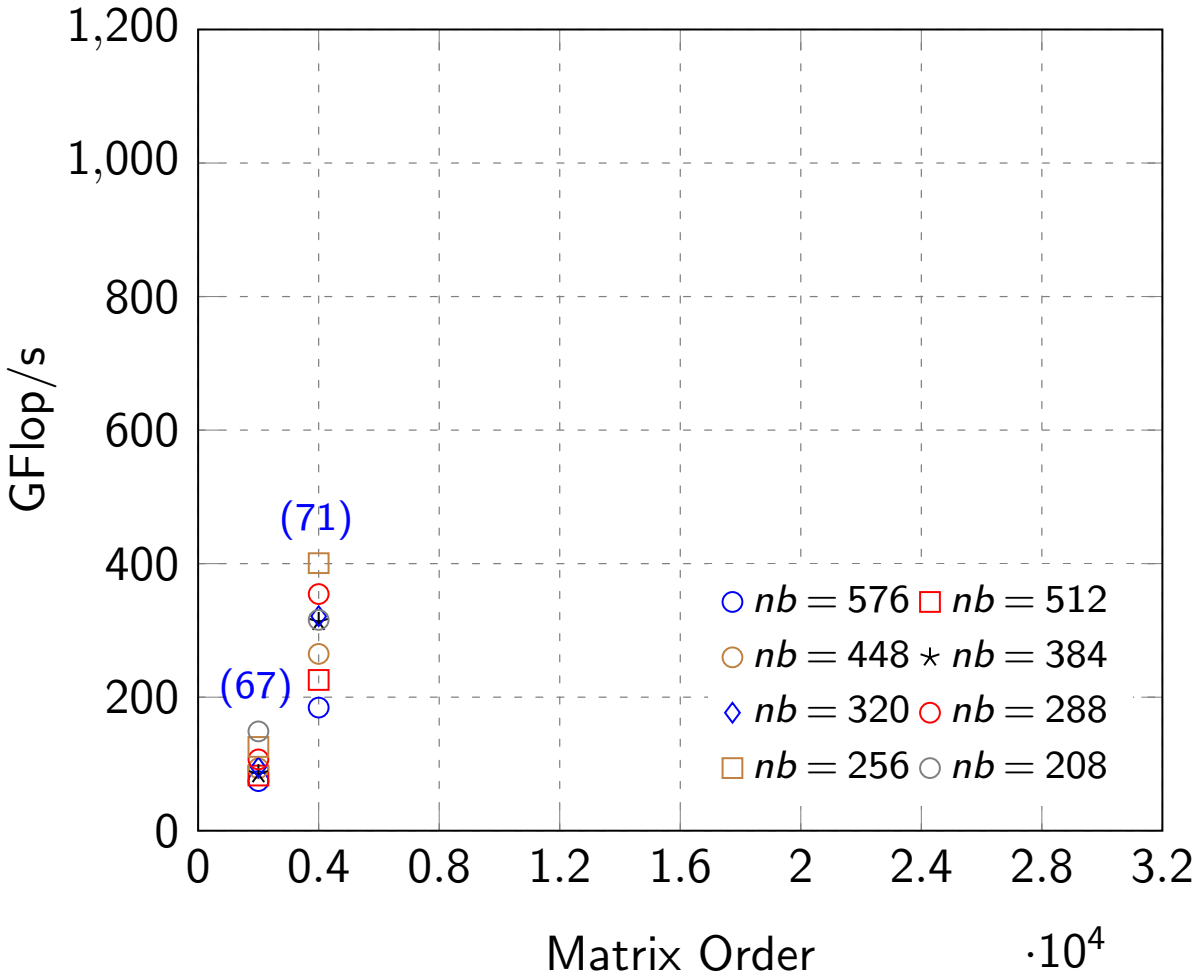
nb with a higher performance for each problem size *n*

S1: Empirical+Exhaustive



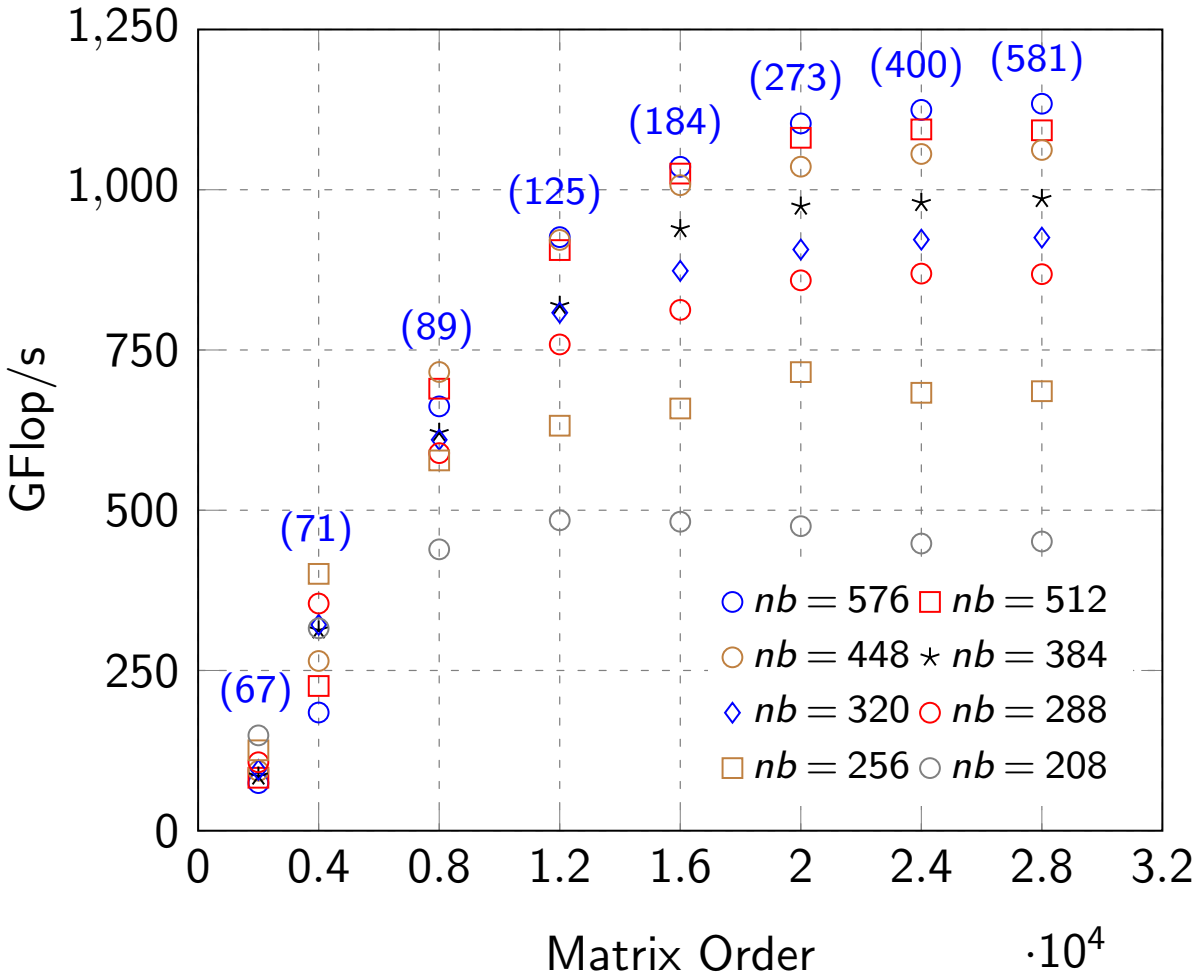
Experimental Time: 67 seconds

S1: Empirical+Exhaustive



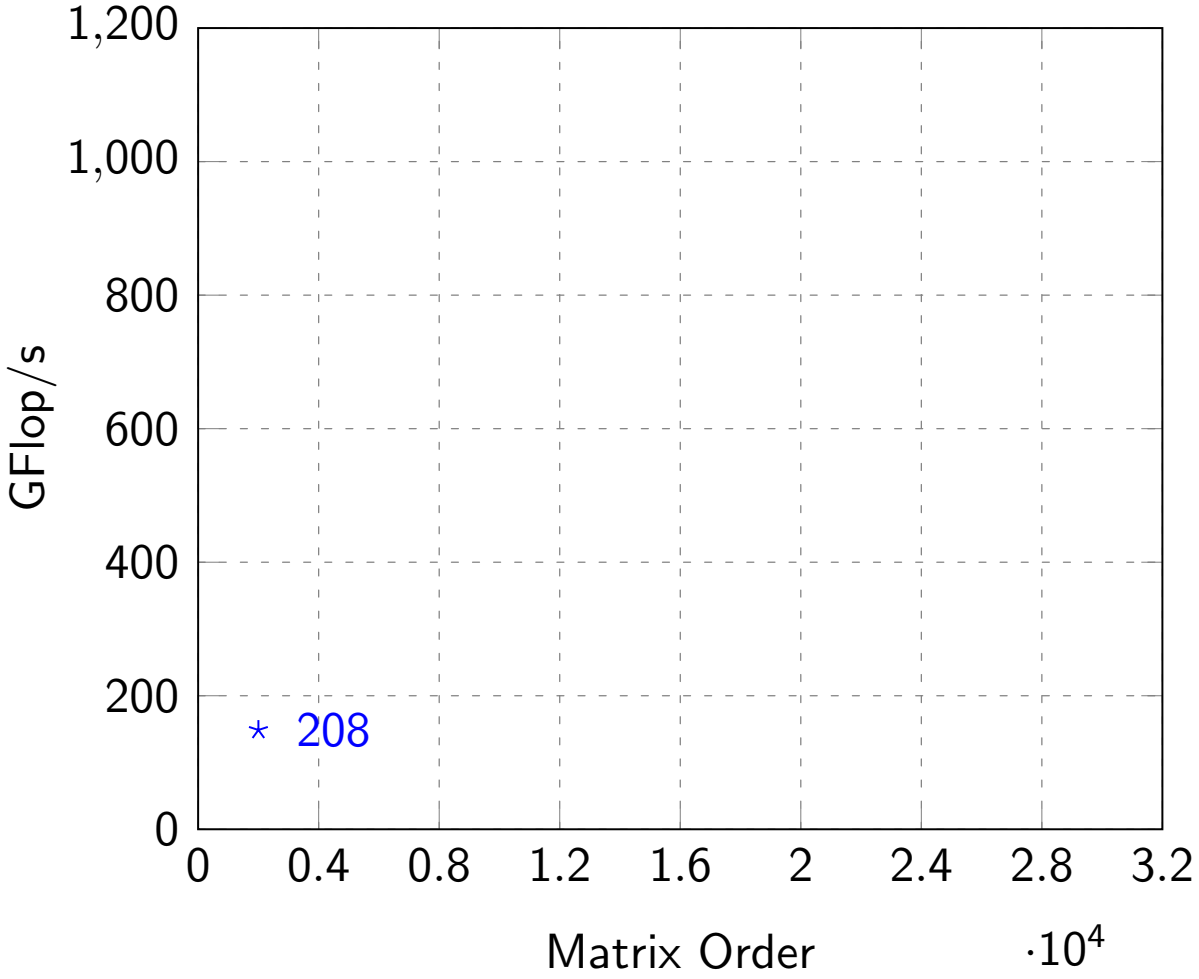
Experimental Time: 138 seconds

S1: Empirical+Exhaustive

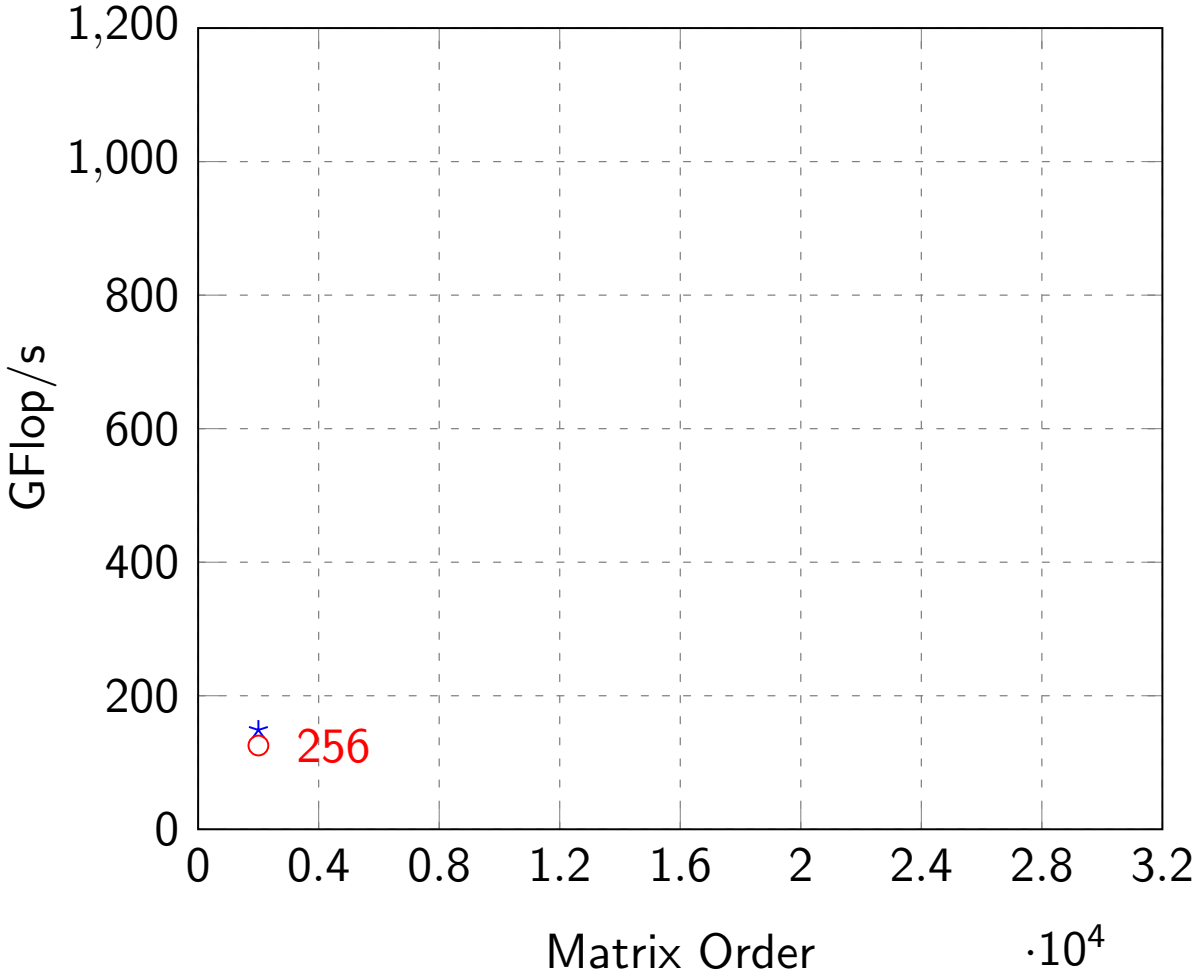


Experimental Time: 30 minutes

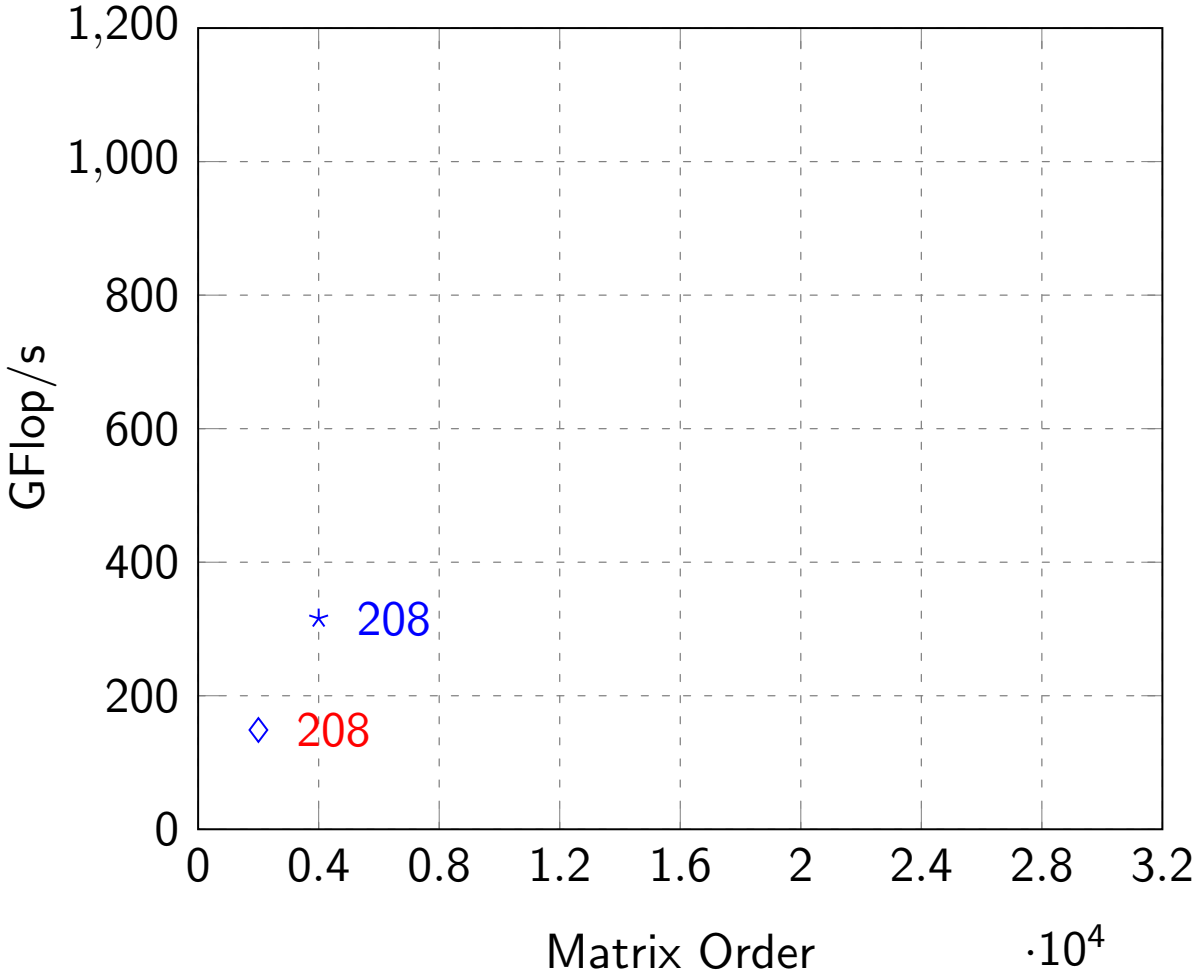
S2: Empirical+Pruned



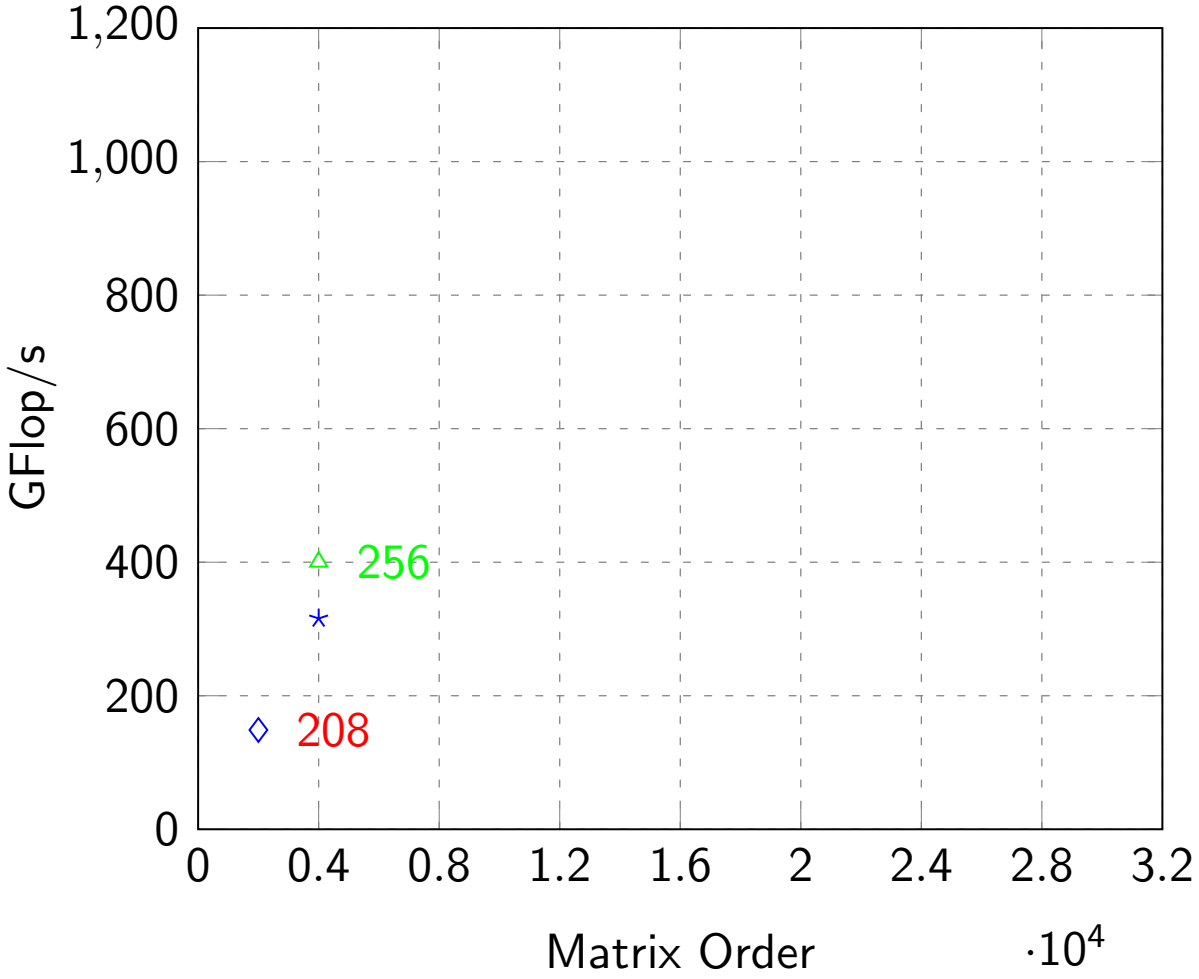
S2: Empirical+Pruned



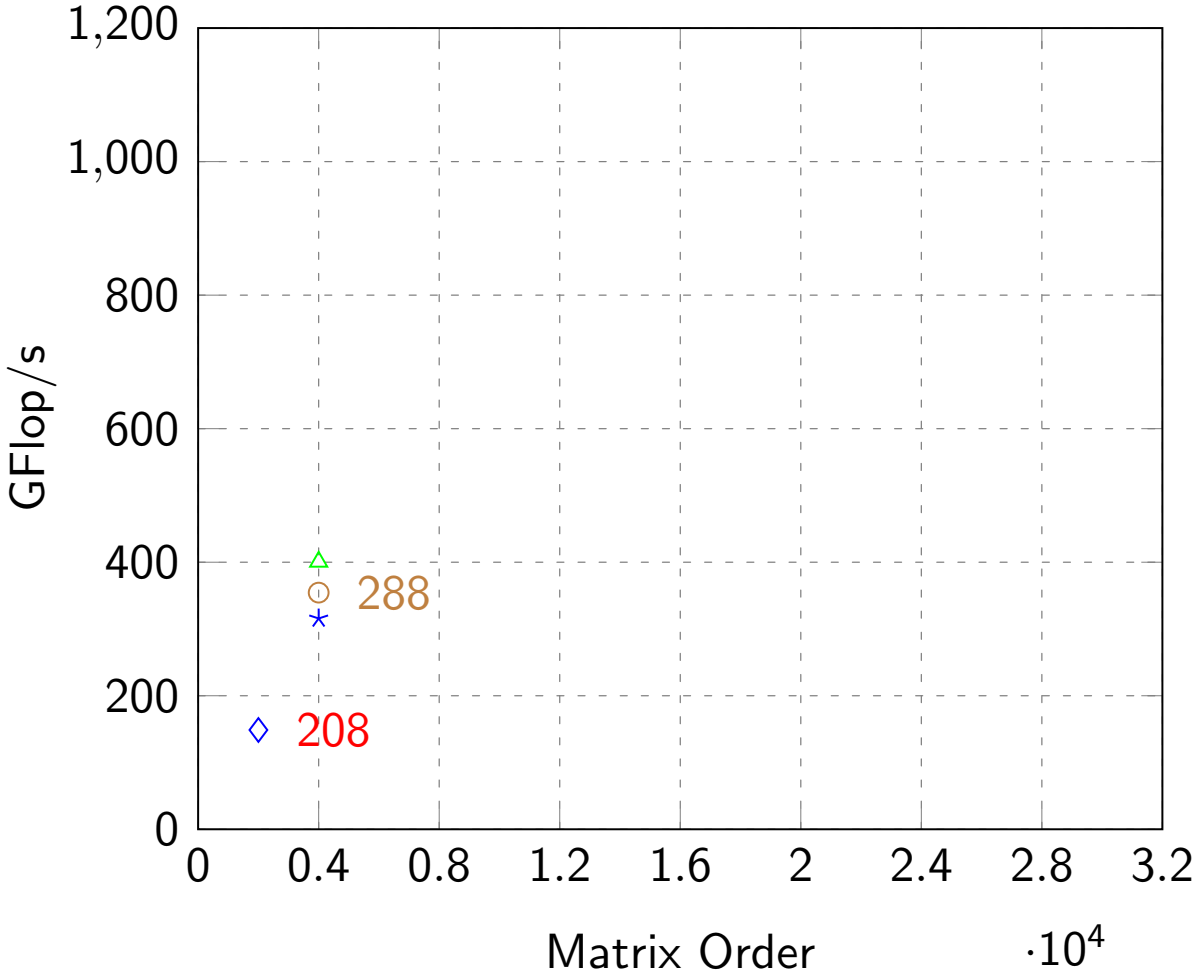
S2: Empirical+Pruned



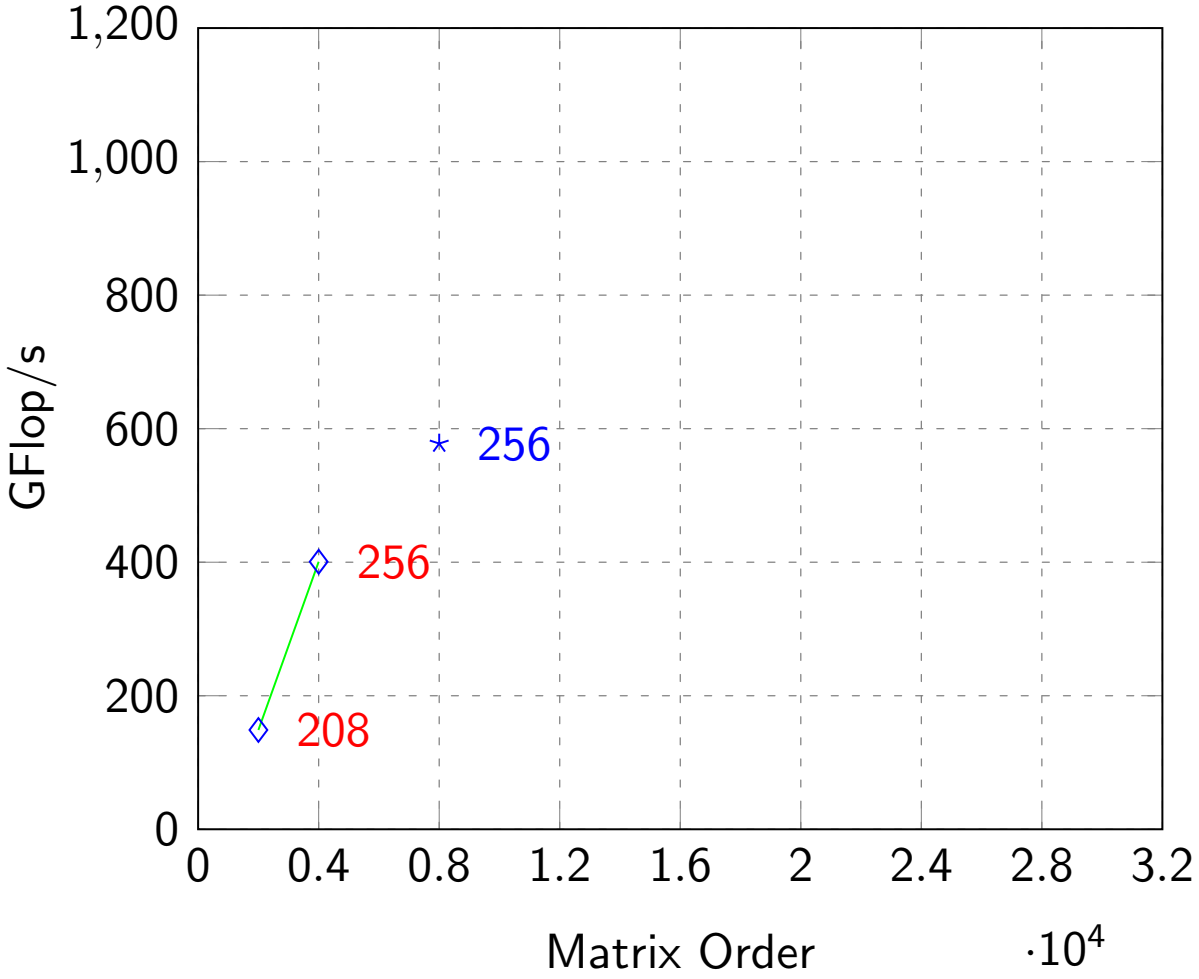
S2: Empirical+Pruned



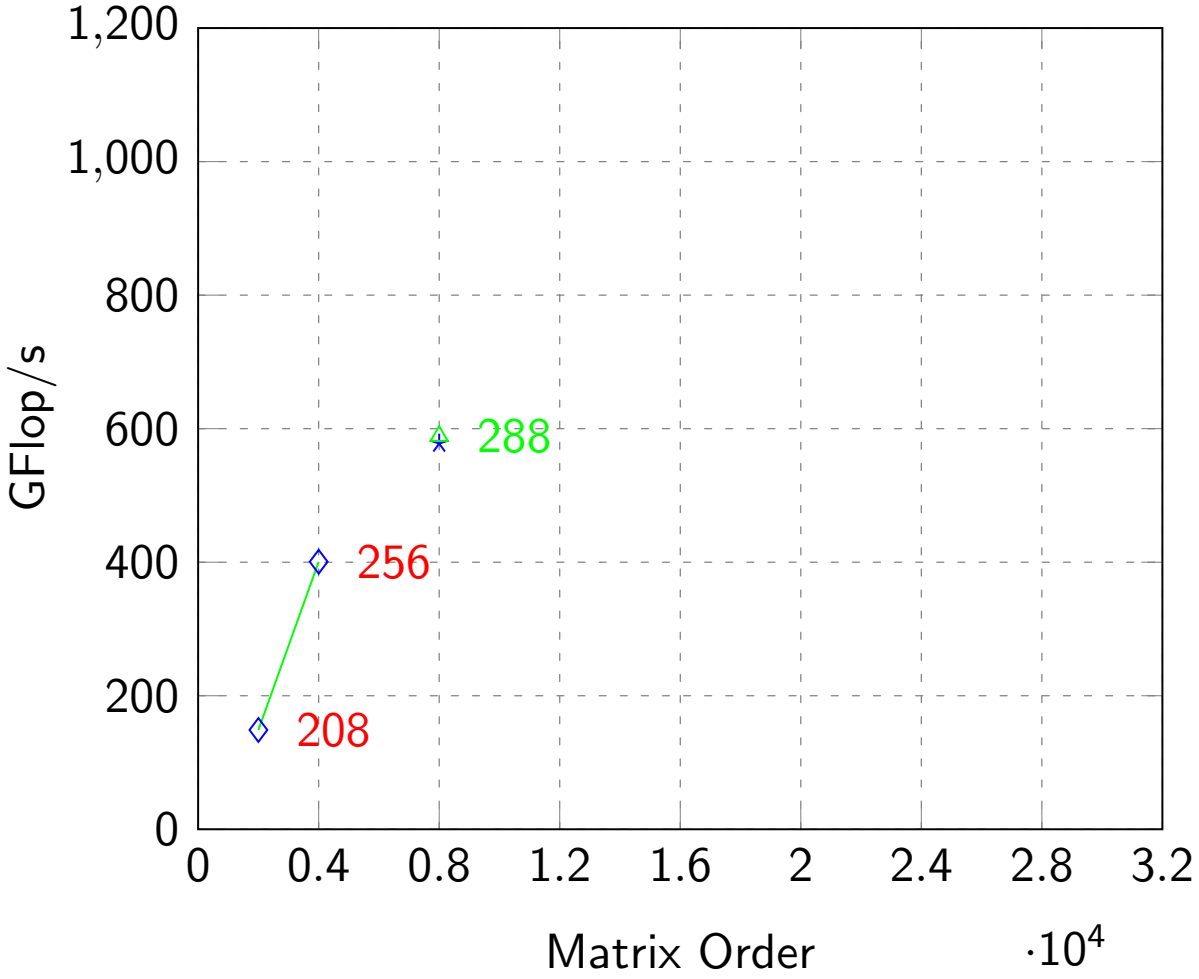
S2: Empirical+Pruned



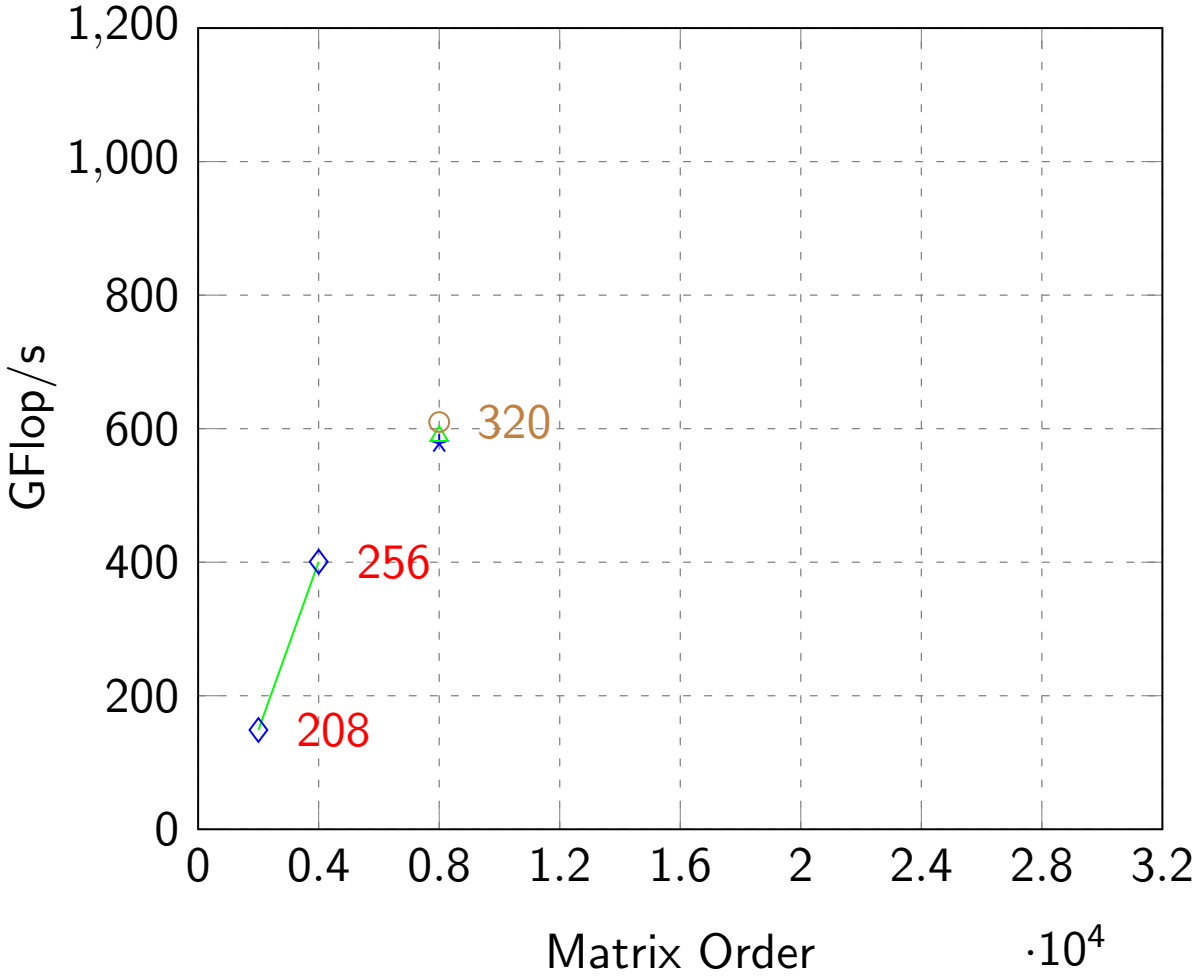
S2: Empirical+Pruned



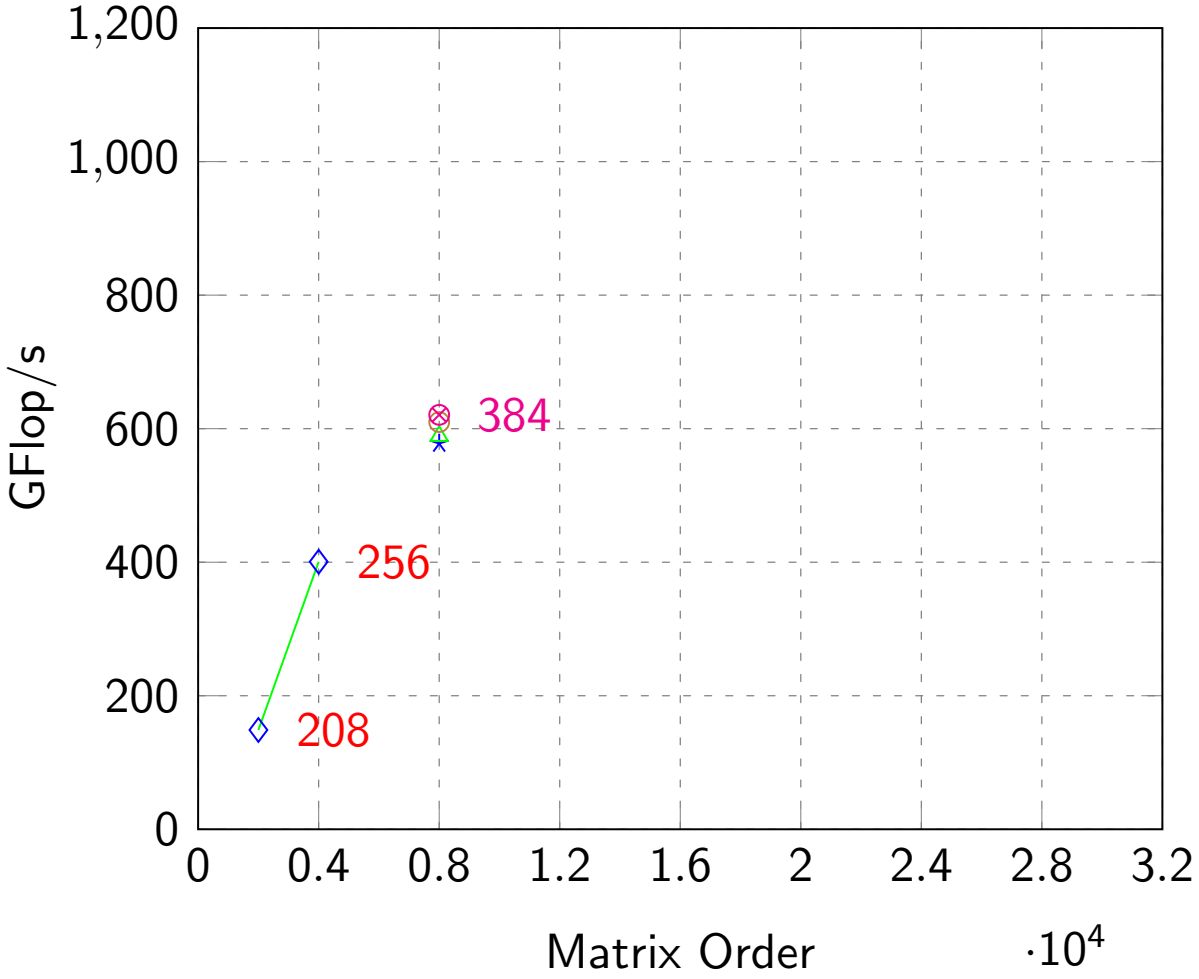
S2: Empirical+Pruned



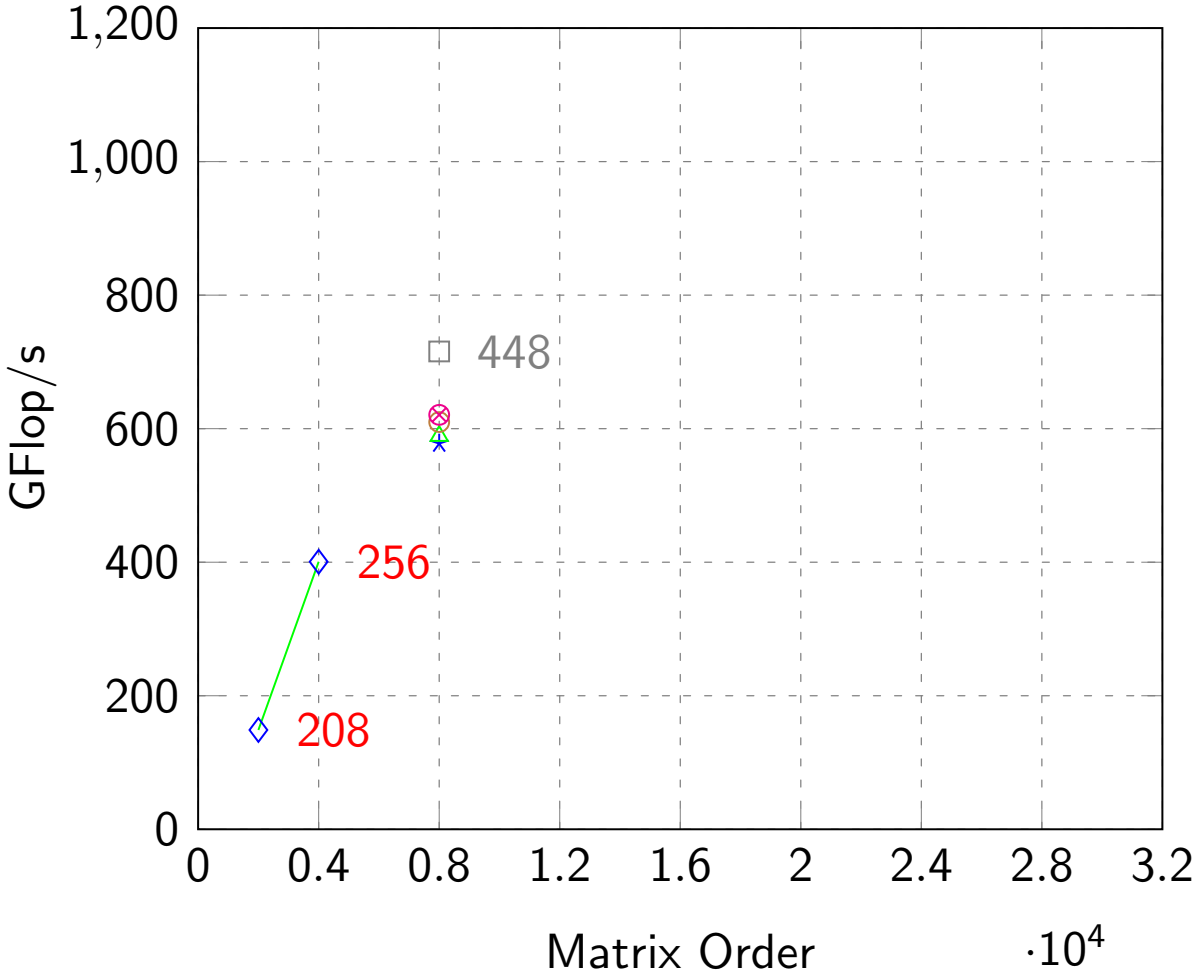
S2: Empirical+Pruned



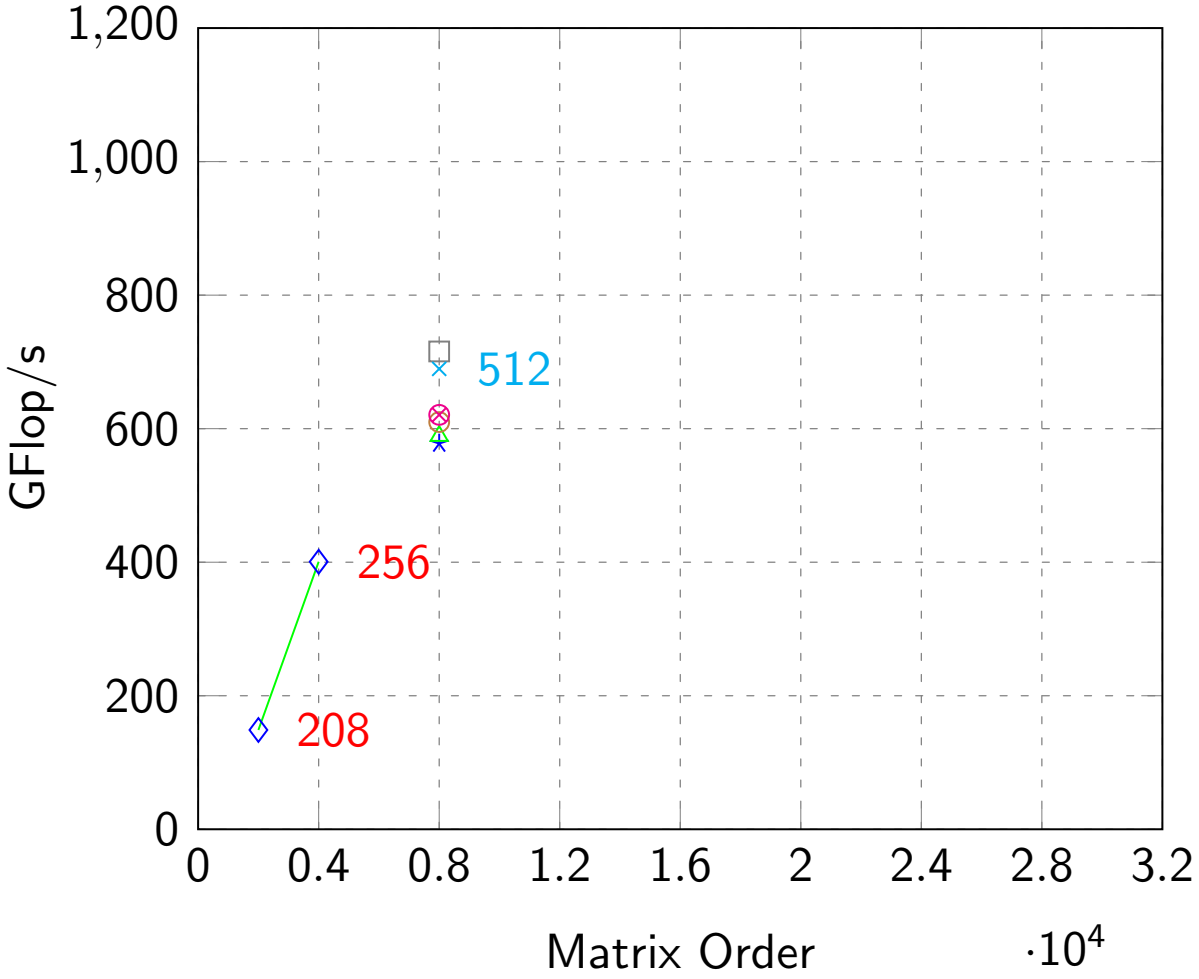
S2: Empirical+Pruned



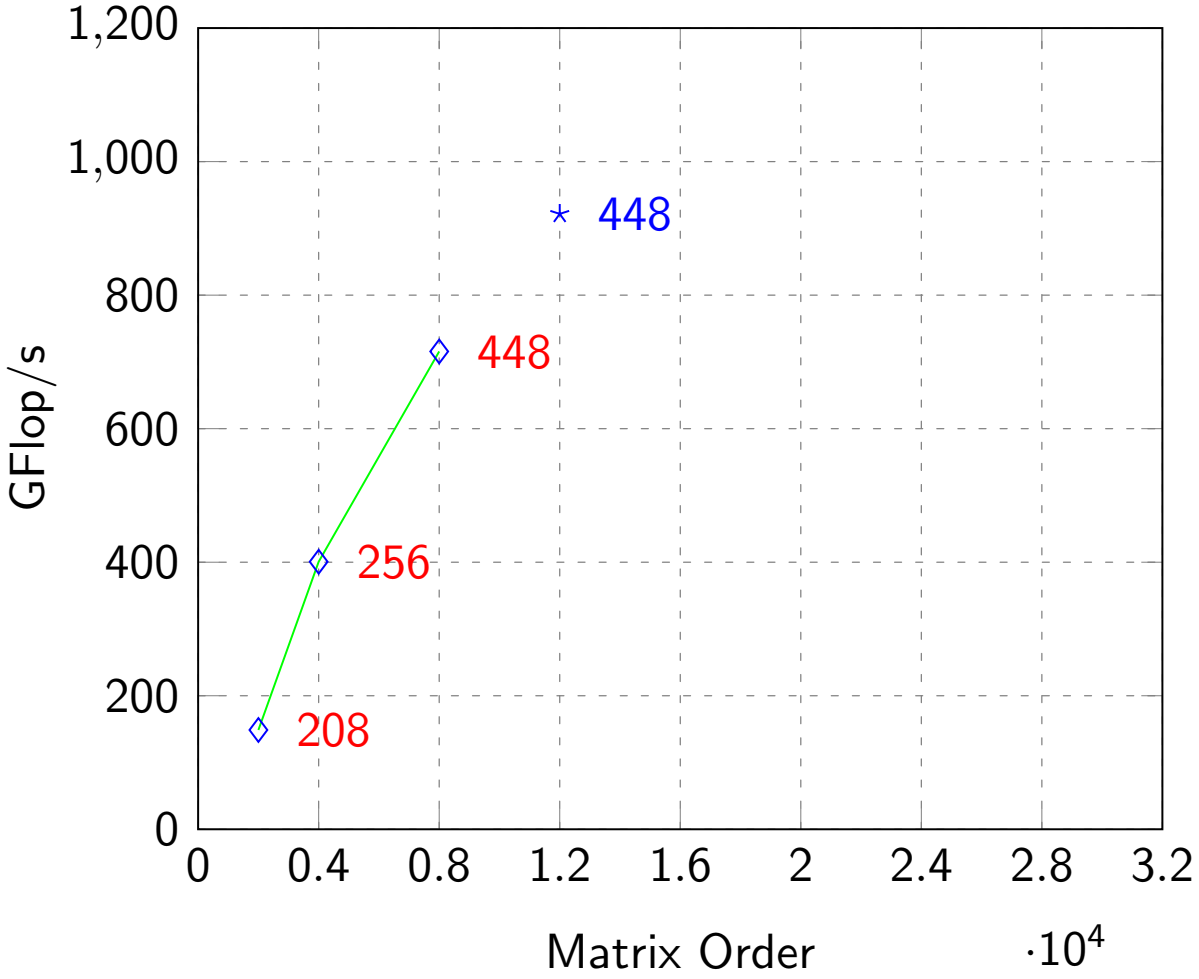
S2: Empirical+Pruned



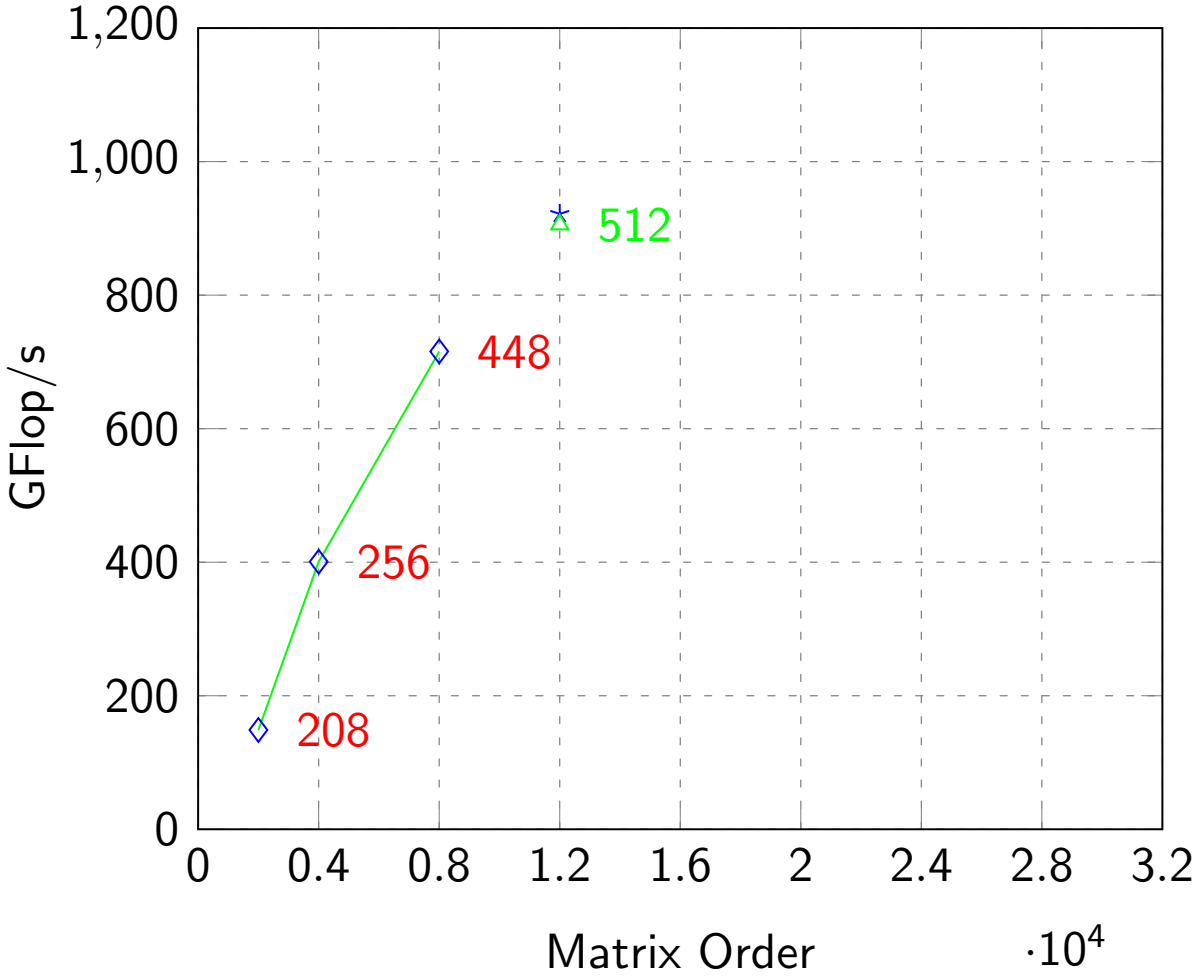
S2: Empirical+Pruned



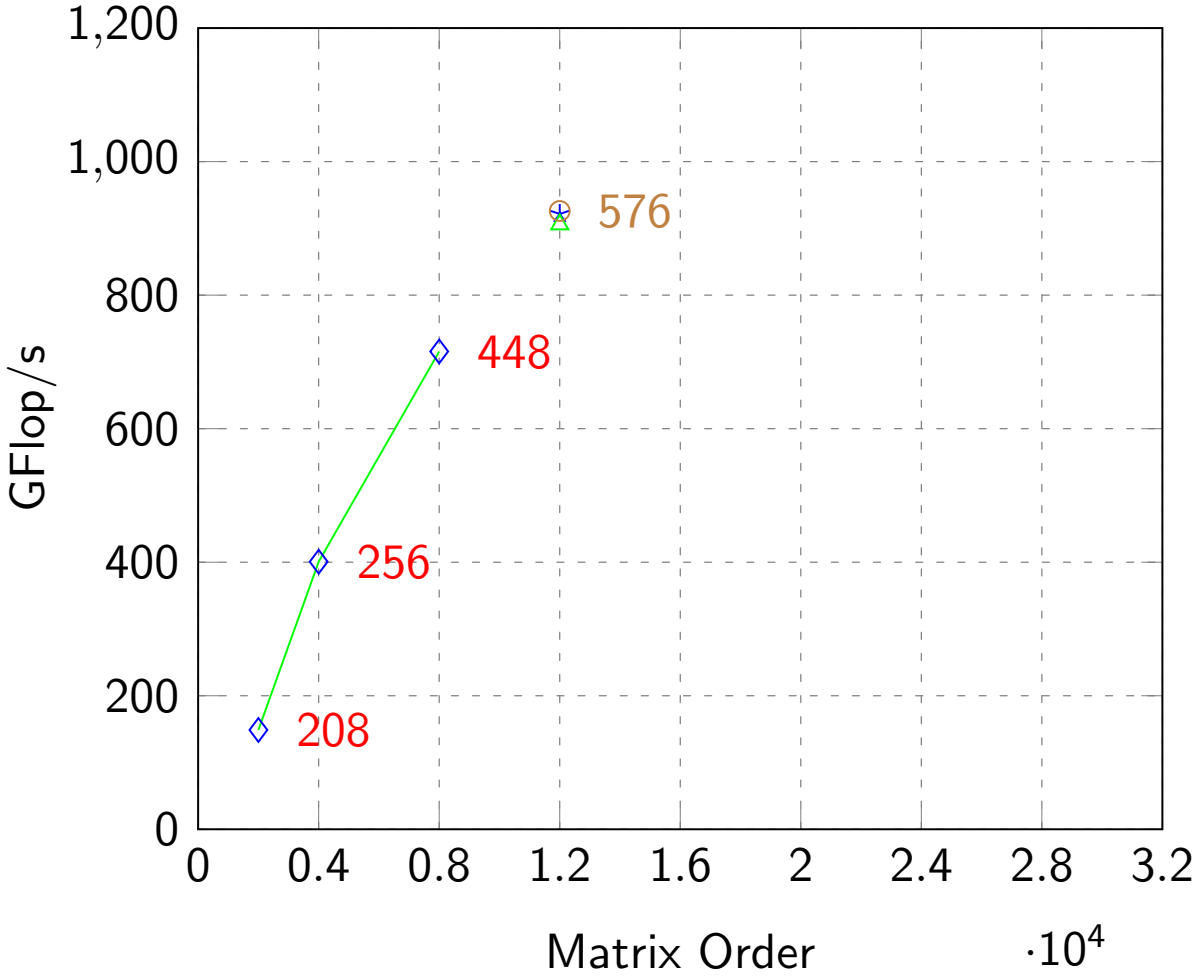
S2: Empirical+Pruned



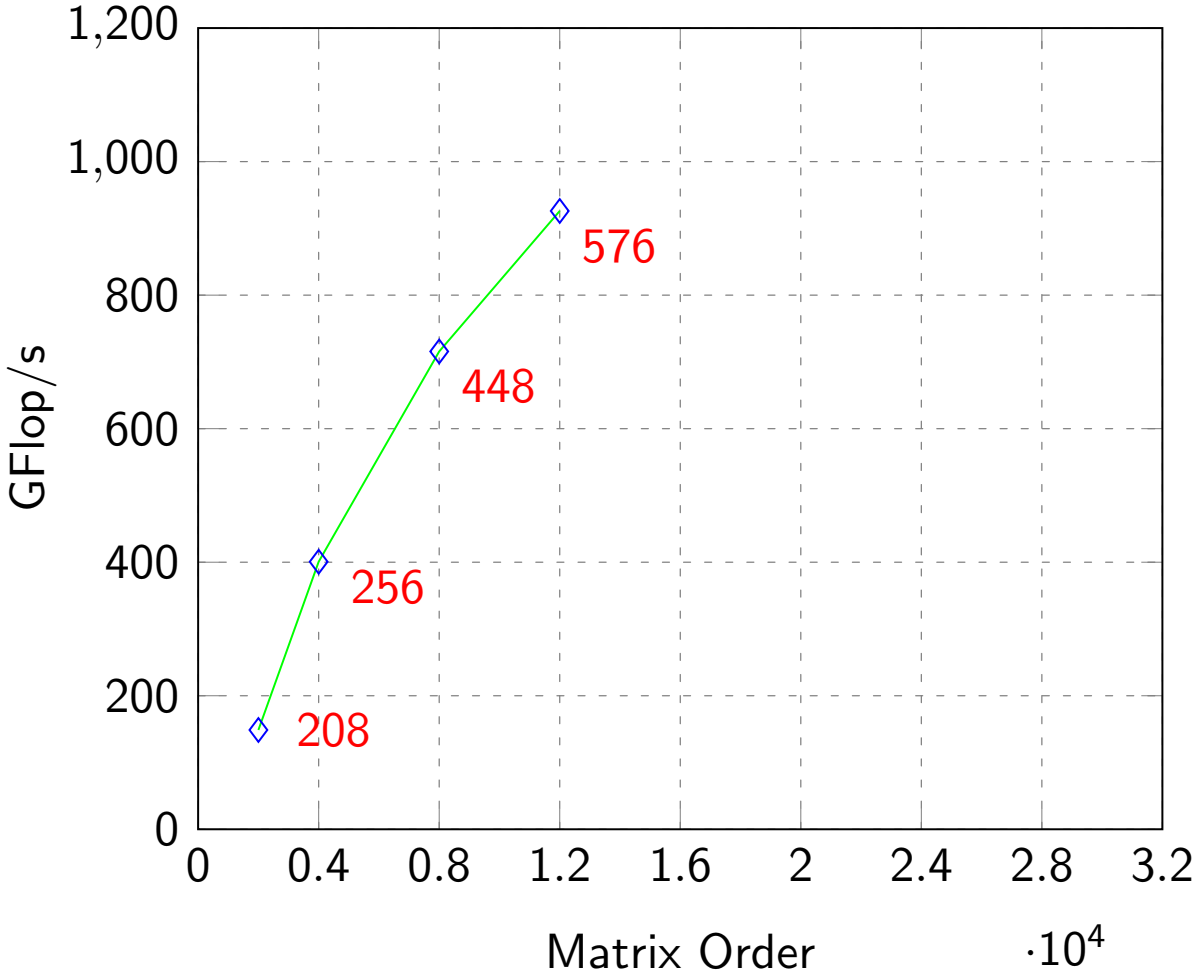
S2: Empirical+Pruned



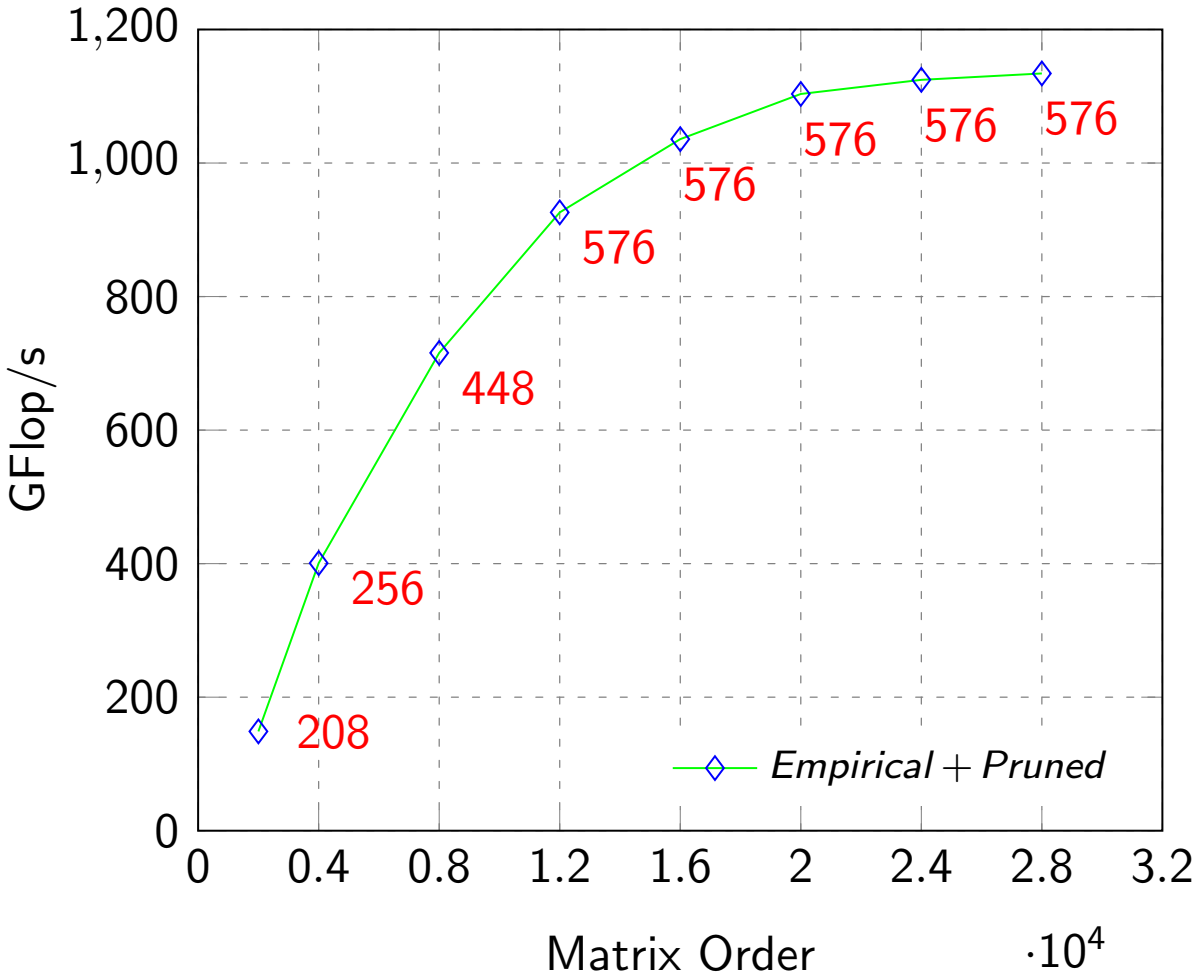
S2: Empirical+Pruned



S2: Empirical+Pruned

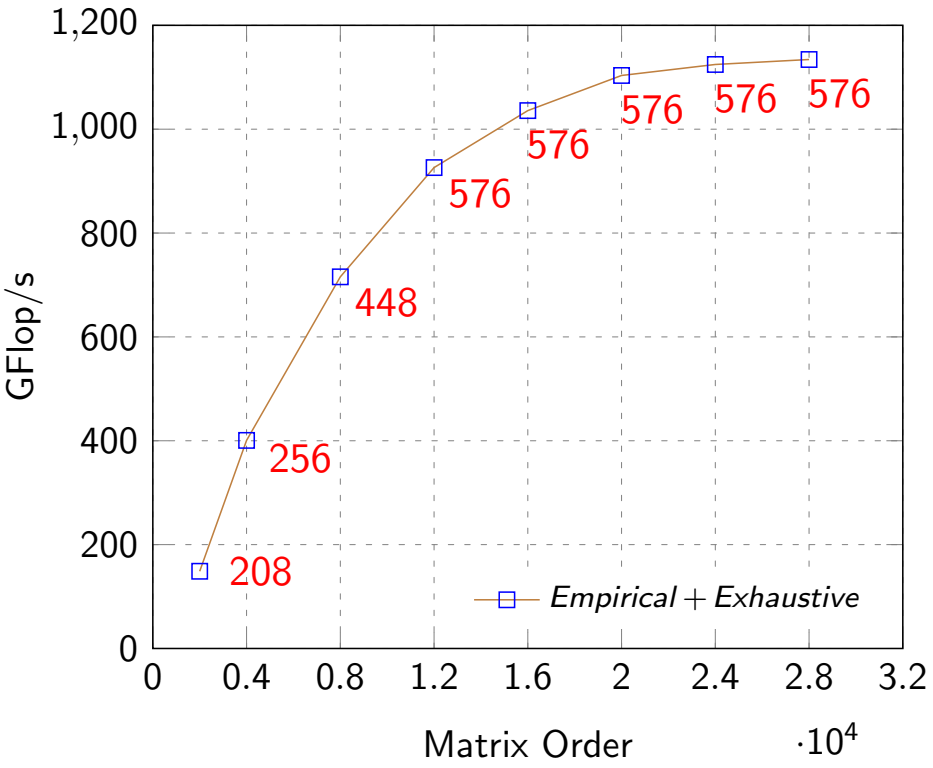


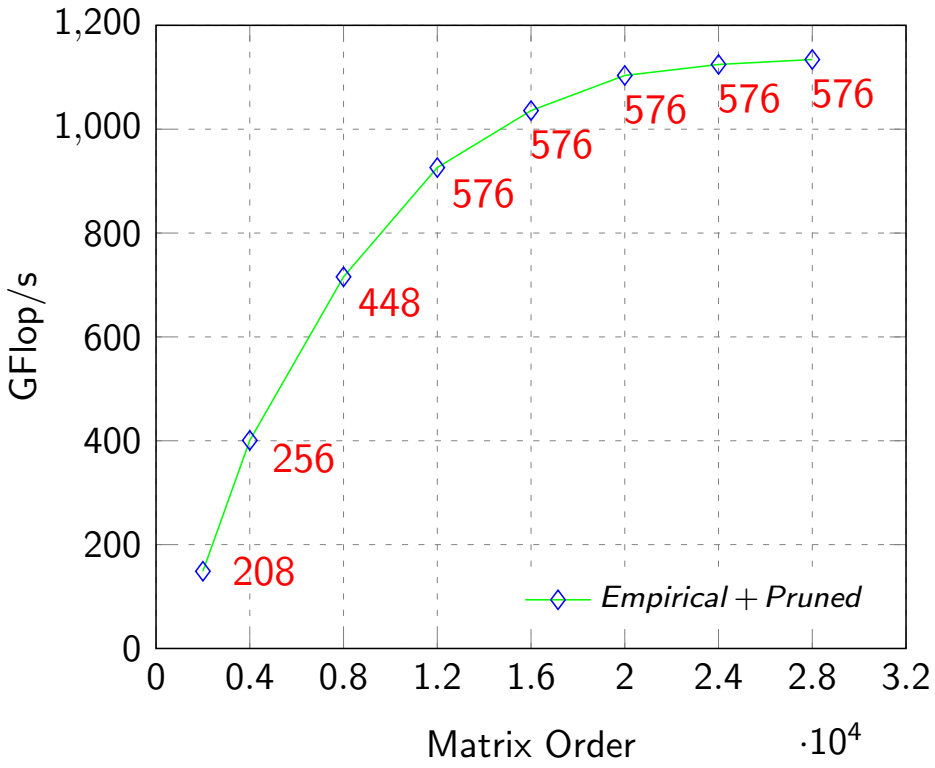
S2: Empirical+Pruned



Experimental Time: 154 seconds

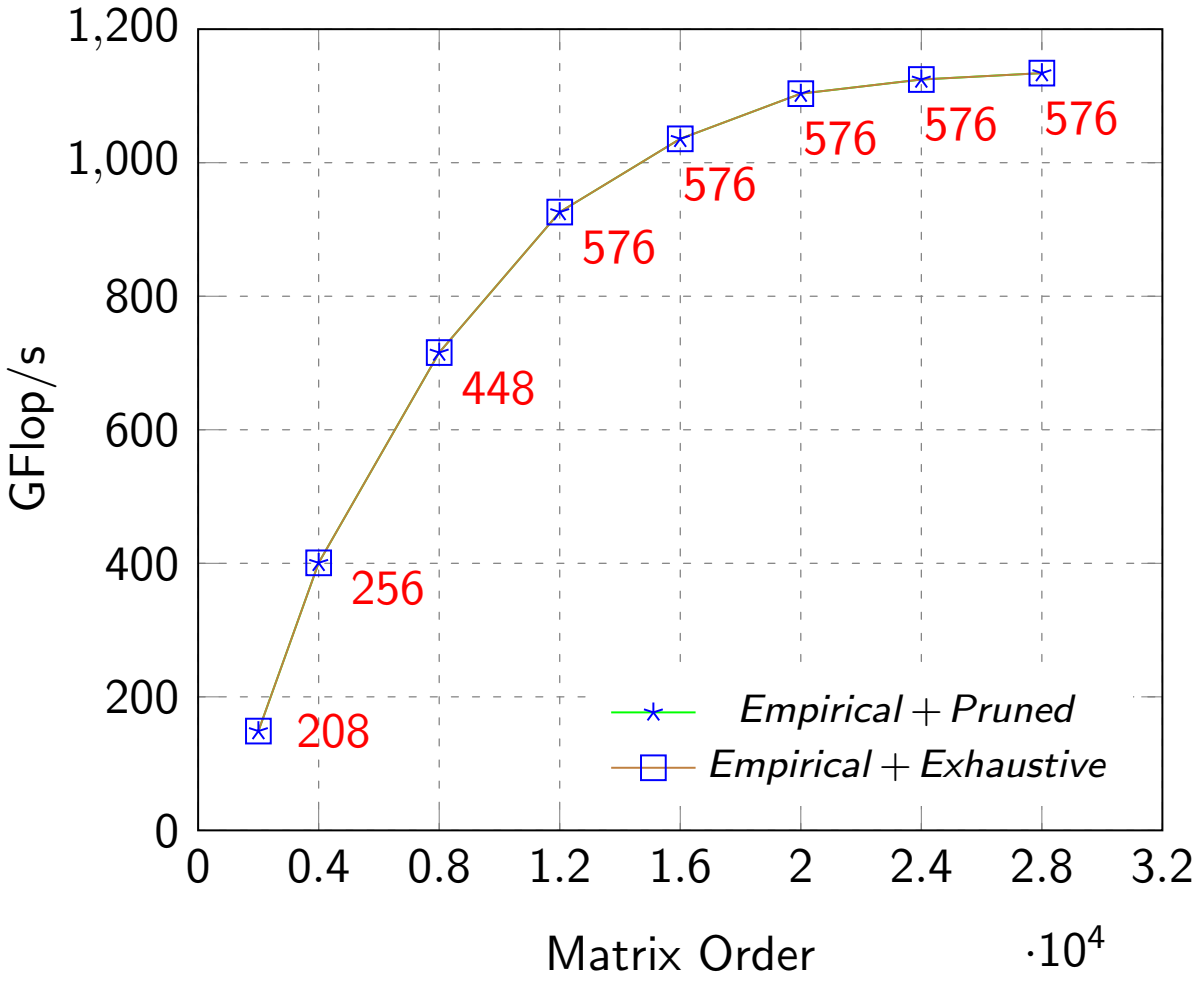
Exhaustive (S1) vs Pruned (S2)





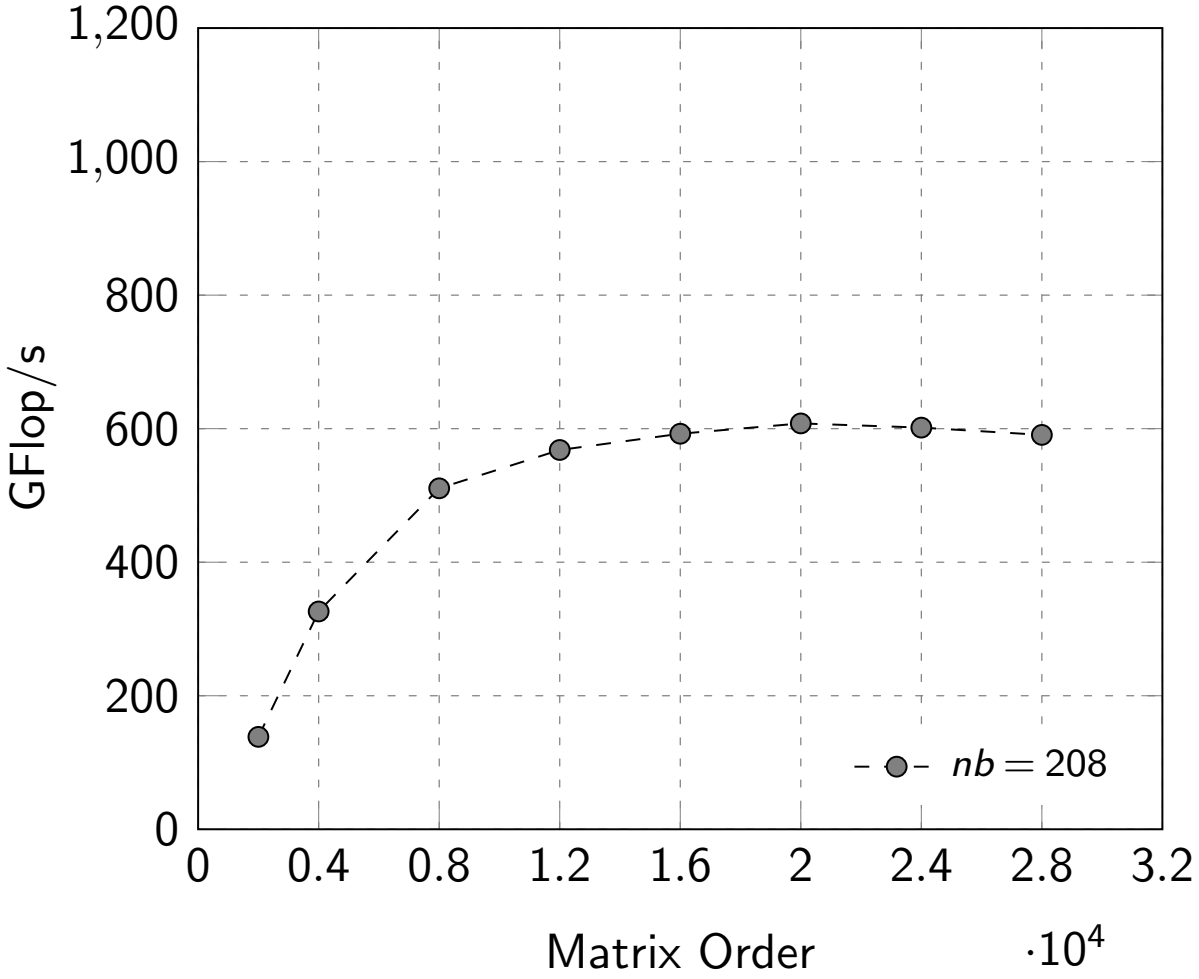
Same *nb* value for each problem size *n*

S1 & S2

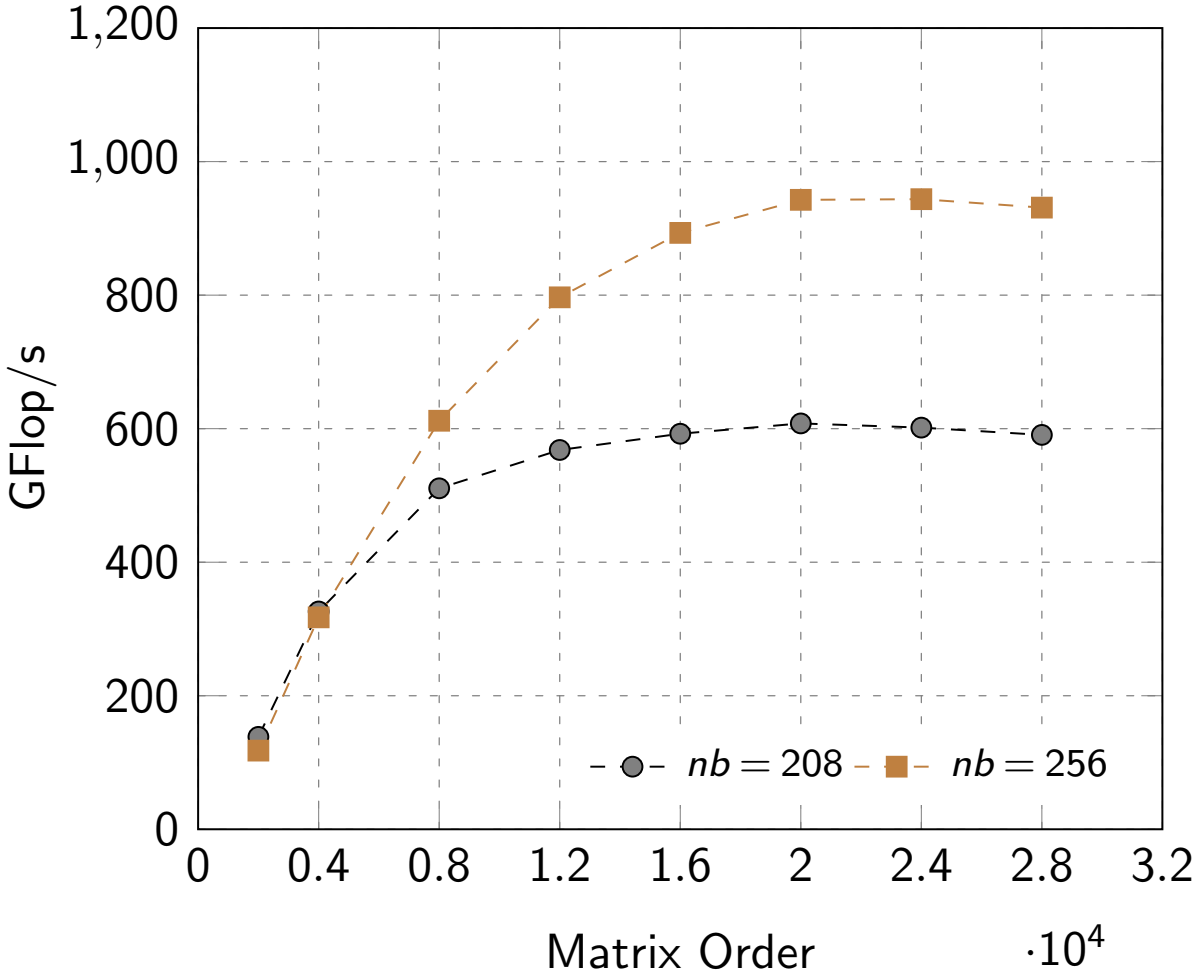


S1: 30 min; **S2**: 154 seconds

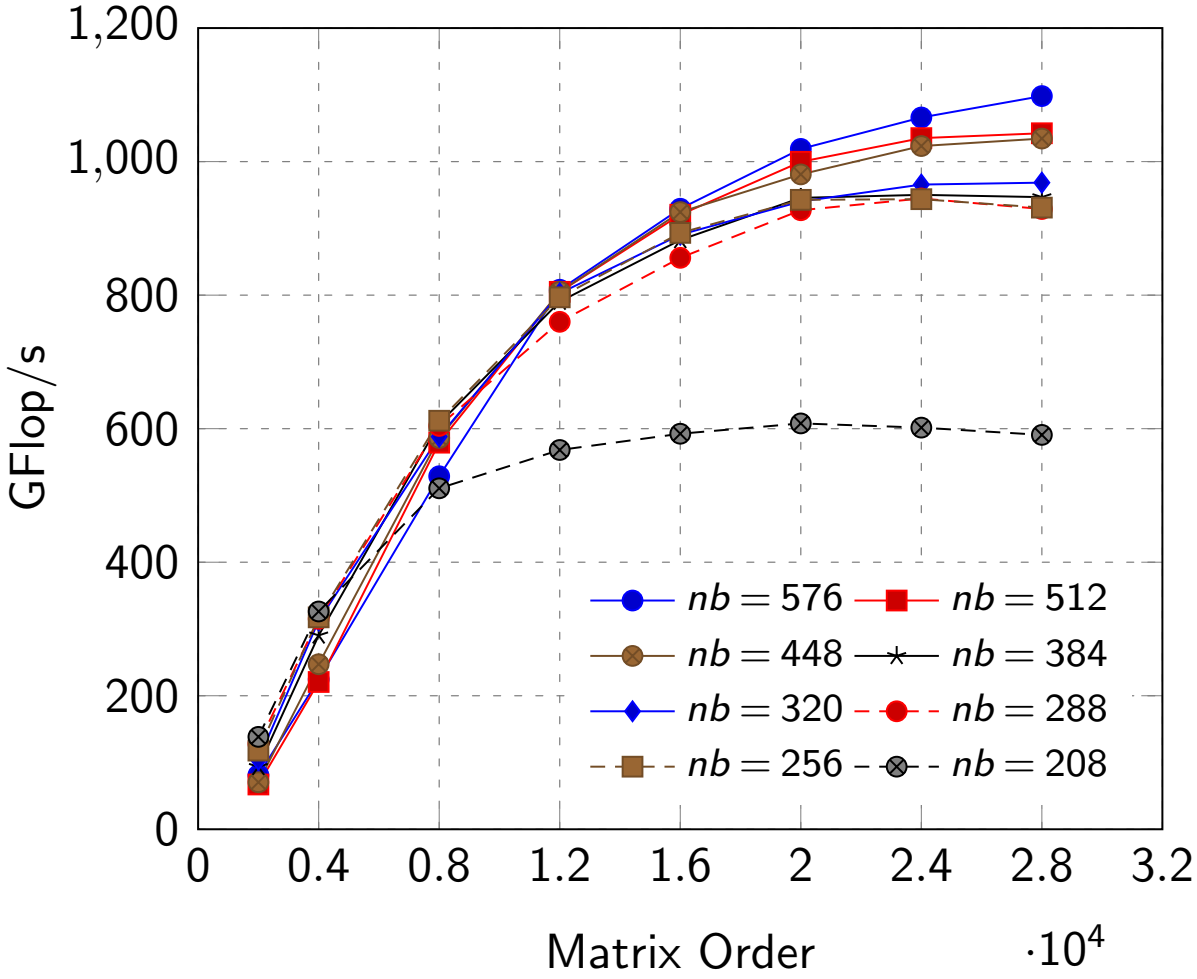
S3: Simulated+Exhaustive



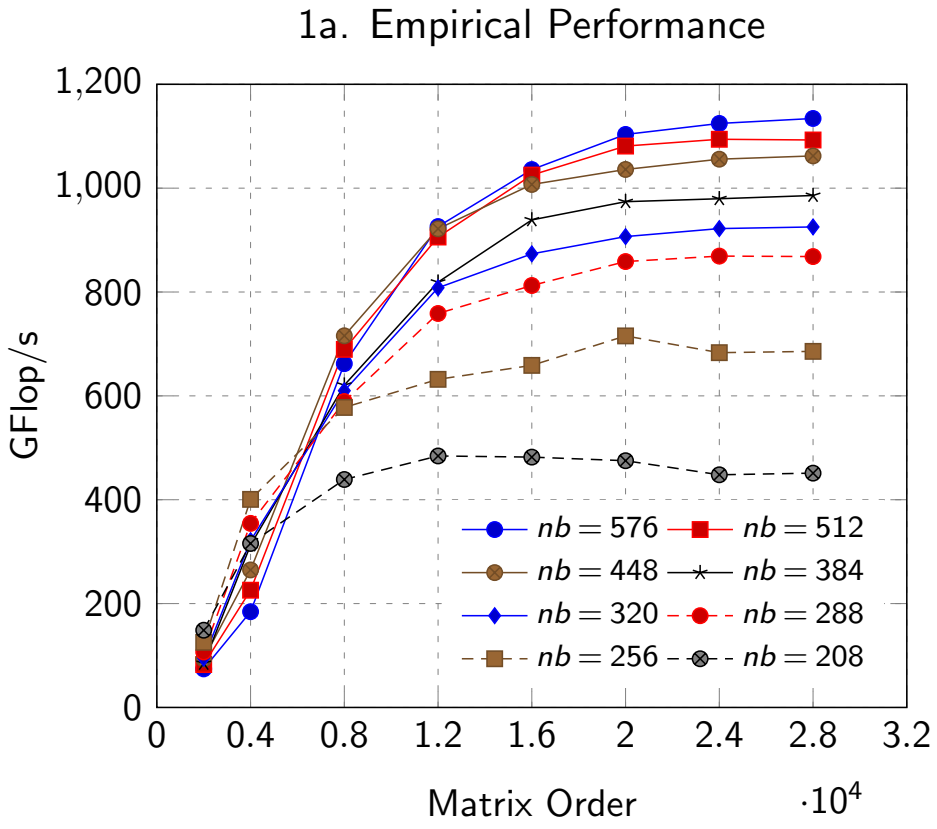
S3: Simulated+Exhaustive

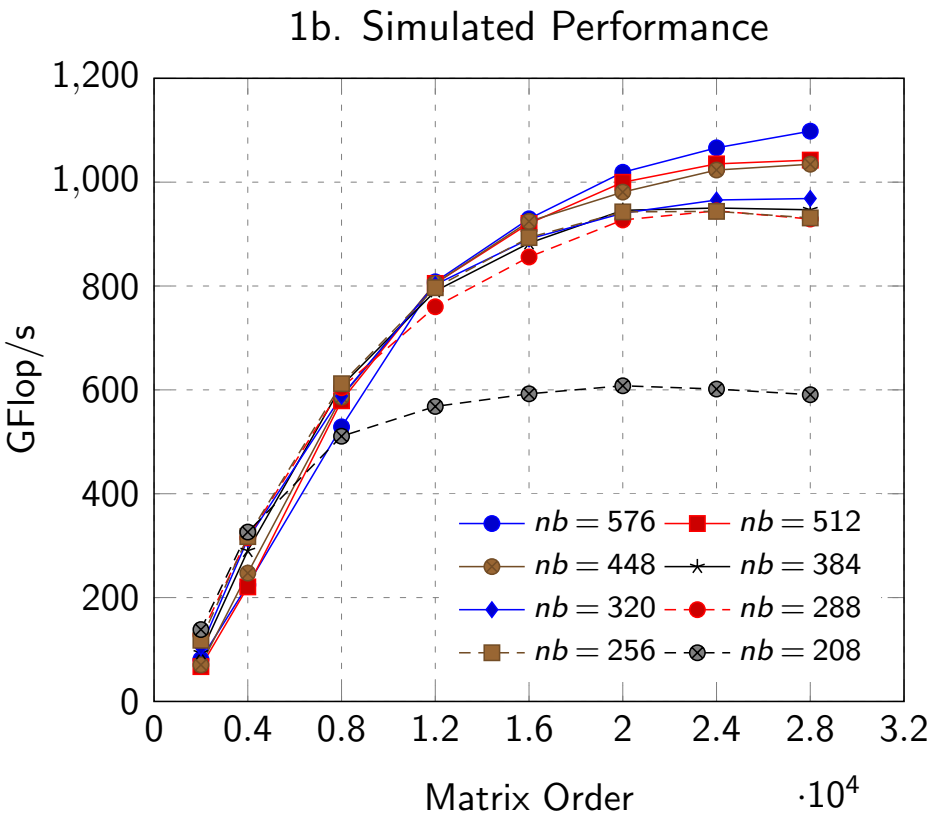


S3: Simulated+Exhaustive

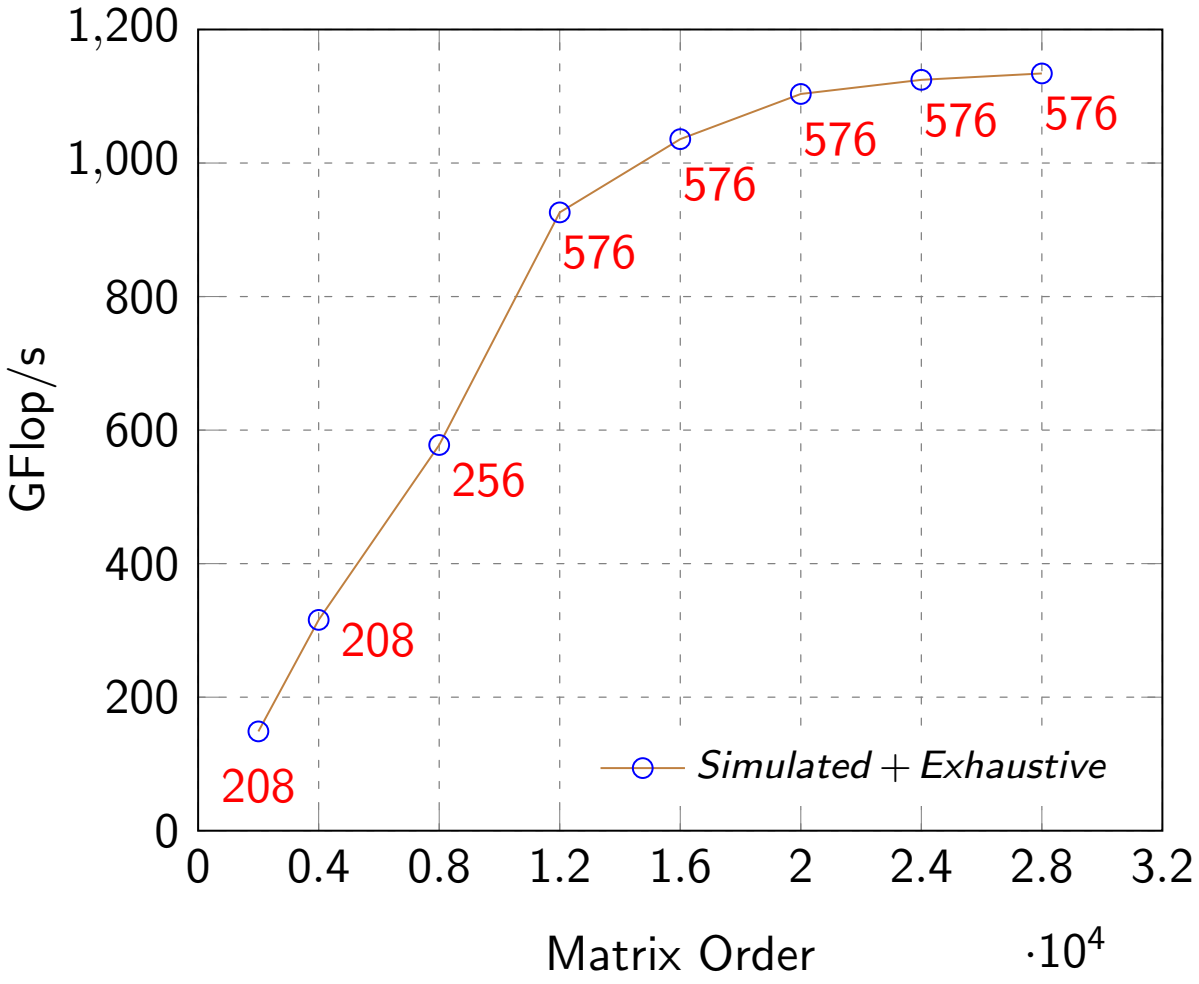


Empirical (S1) vs Simulated (S3)



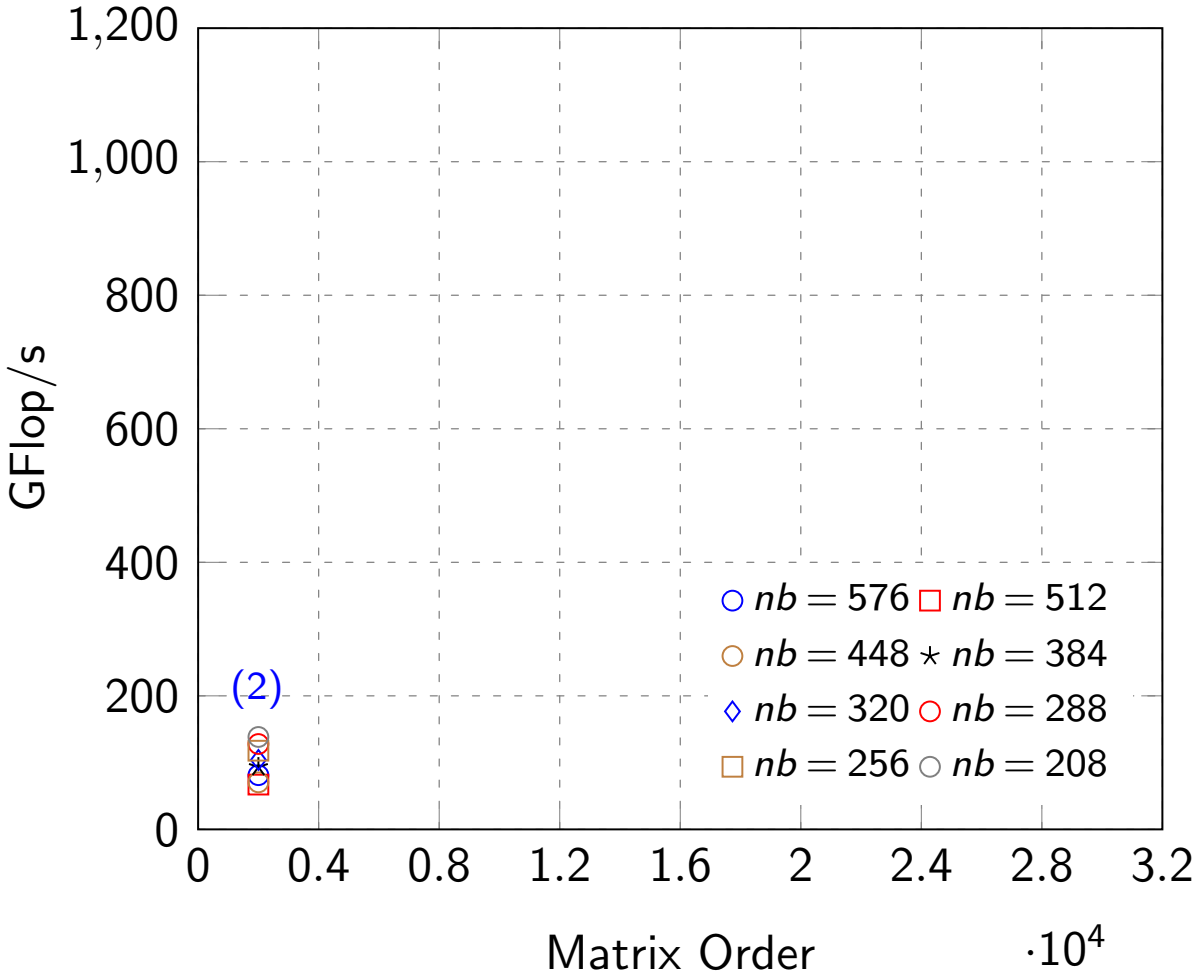


S3: Simulated+Exhaustive



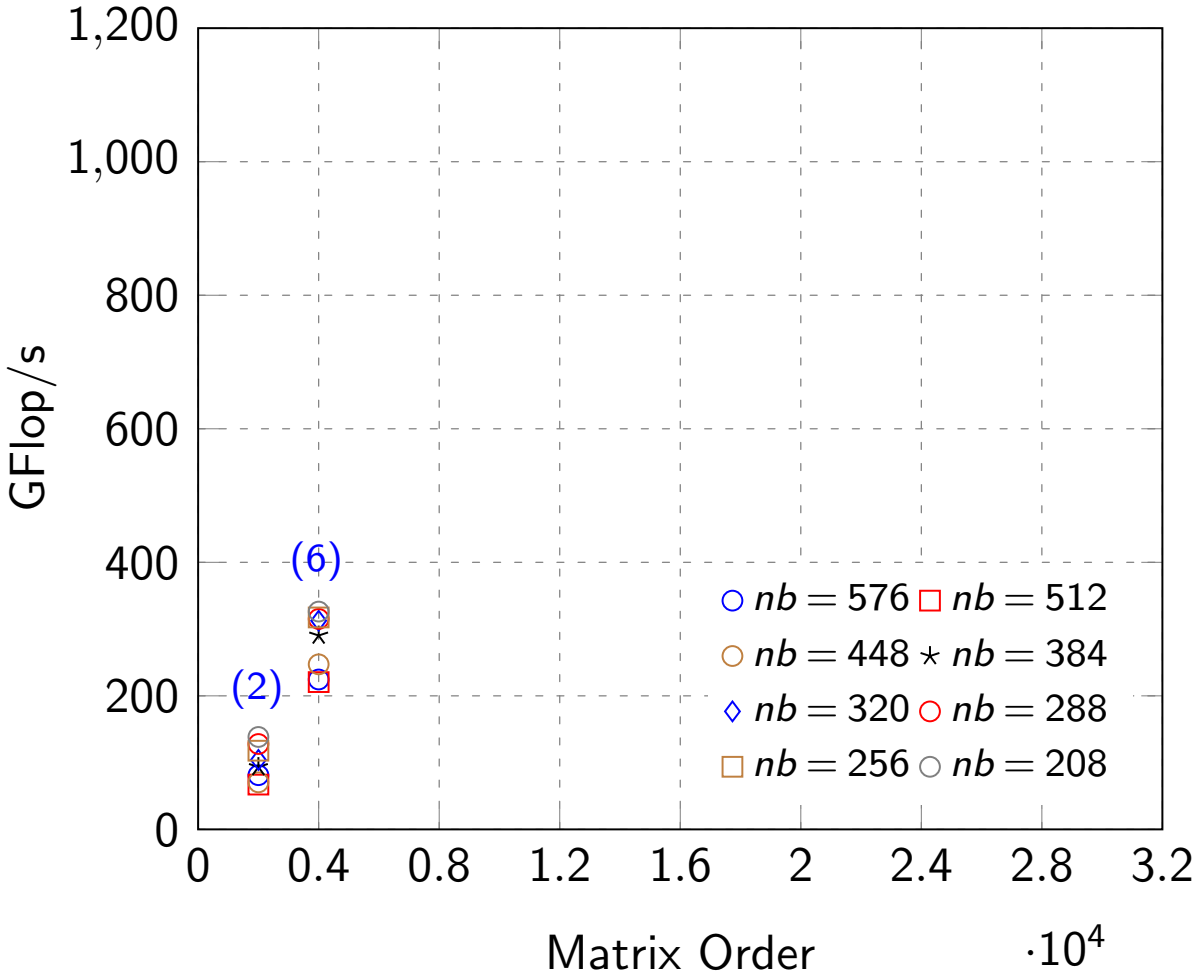
nb with a higher performance for each problem size *n*

S3: Simulated+Exhaustive



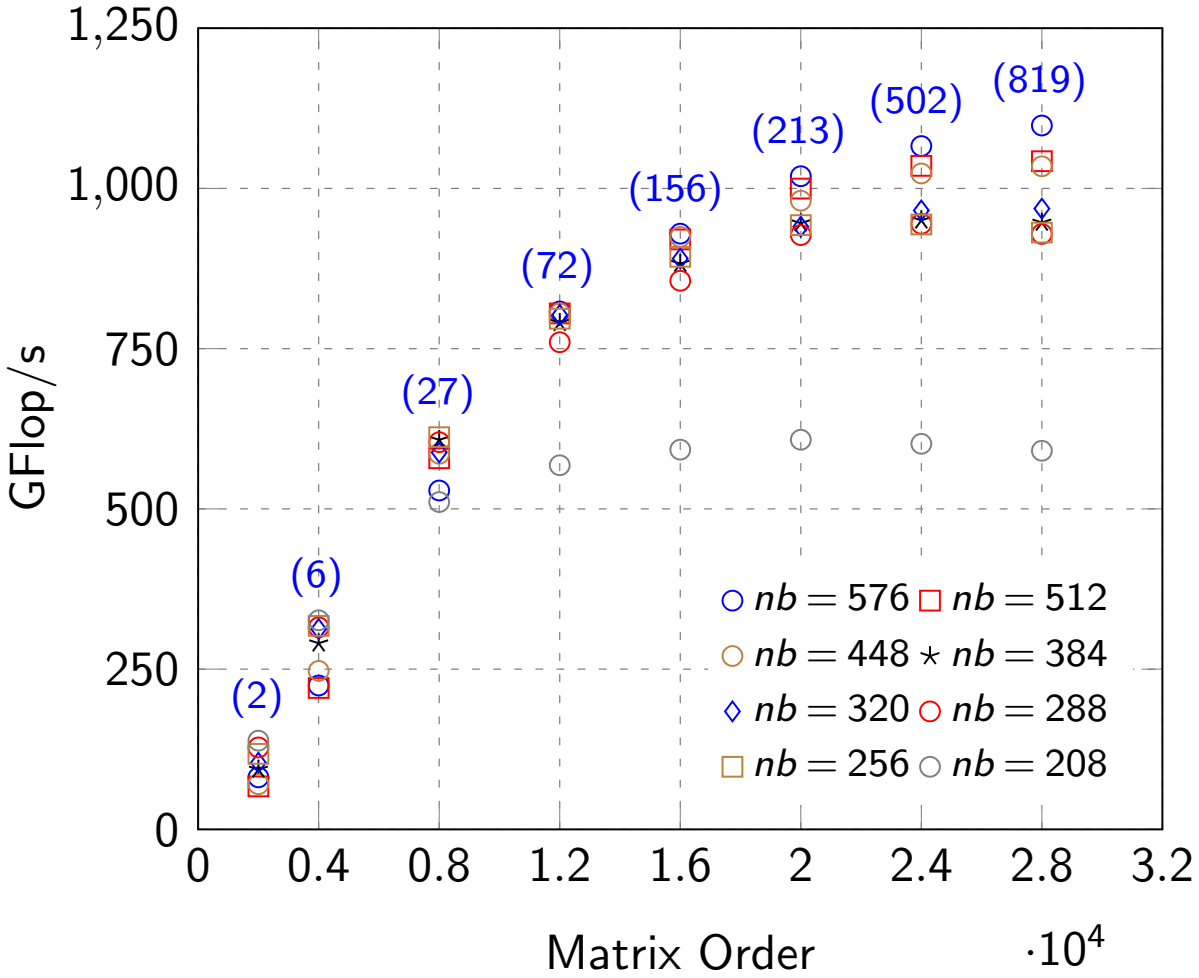
Simulation Time: 2 seconds

S3: Simulated+Exhaustive



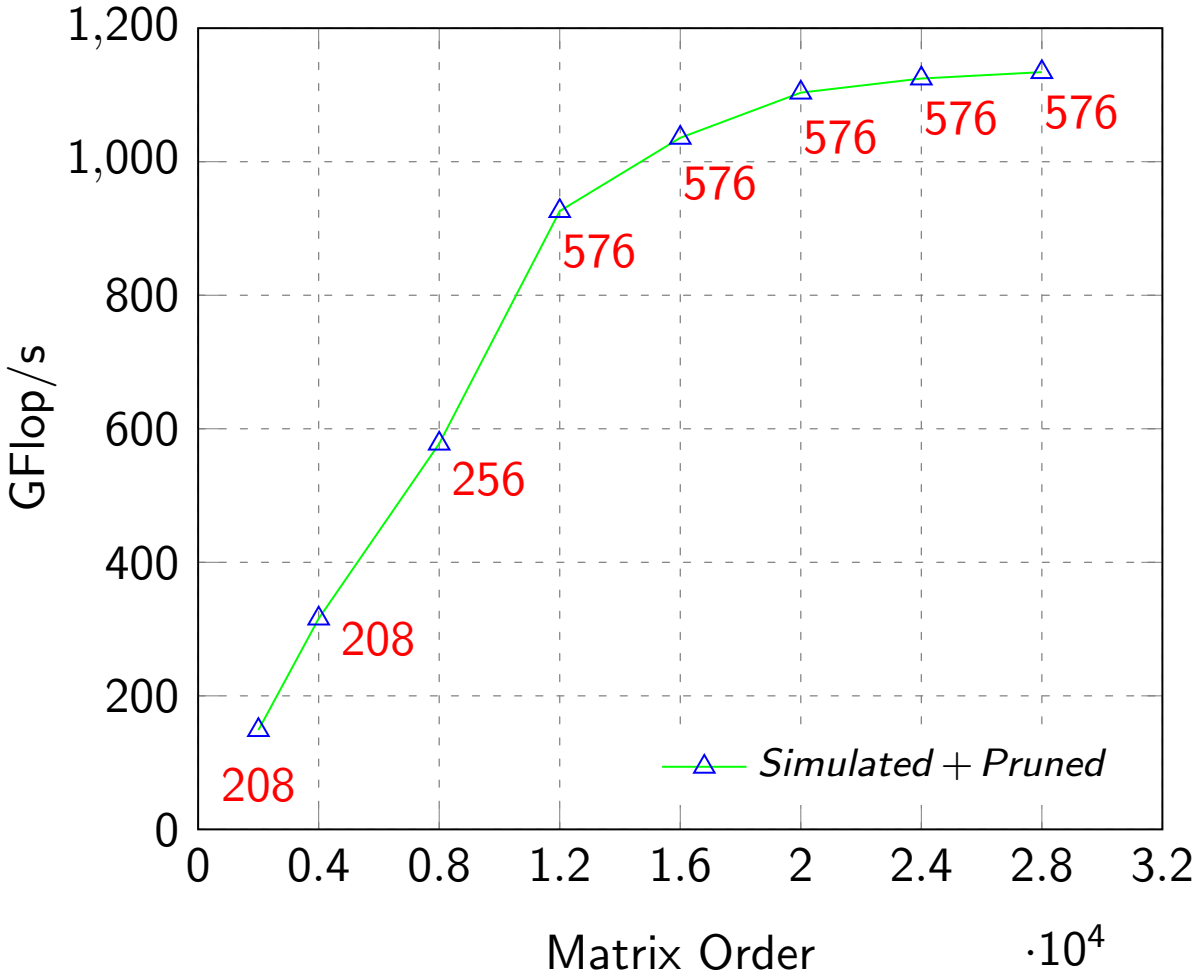
Simulation Time: 8 seconds

S3: Simulated+Exhaustive



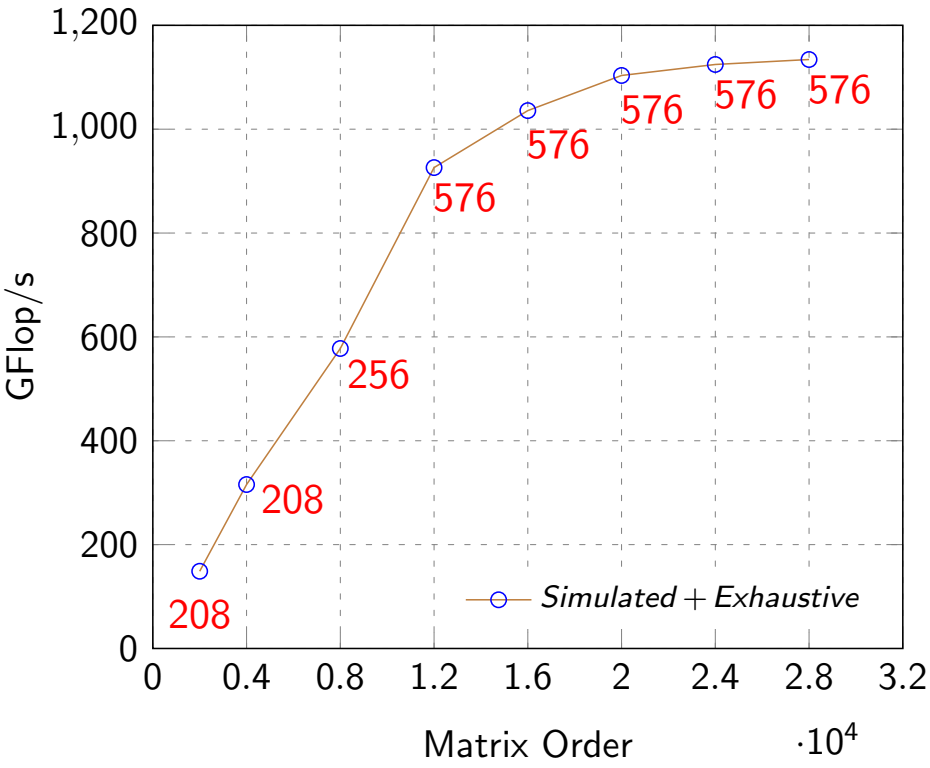
Simulation Time: 30 minutes

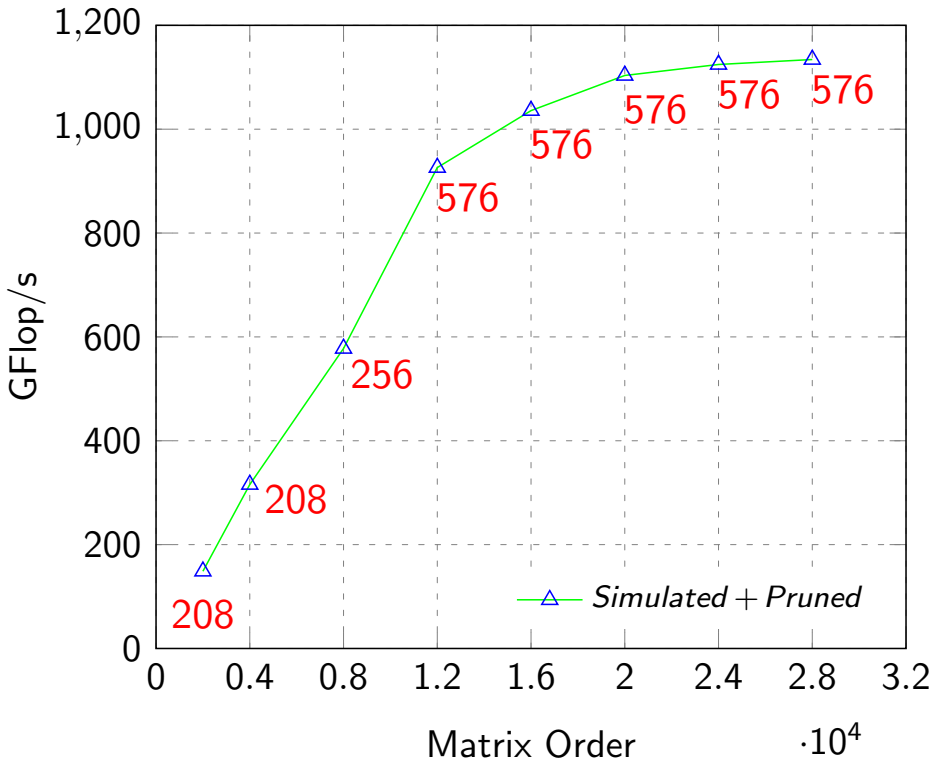
S4: Simulated+Pruned



Simulation Time: 84 seconds

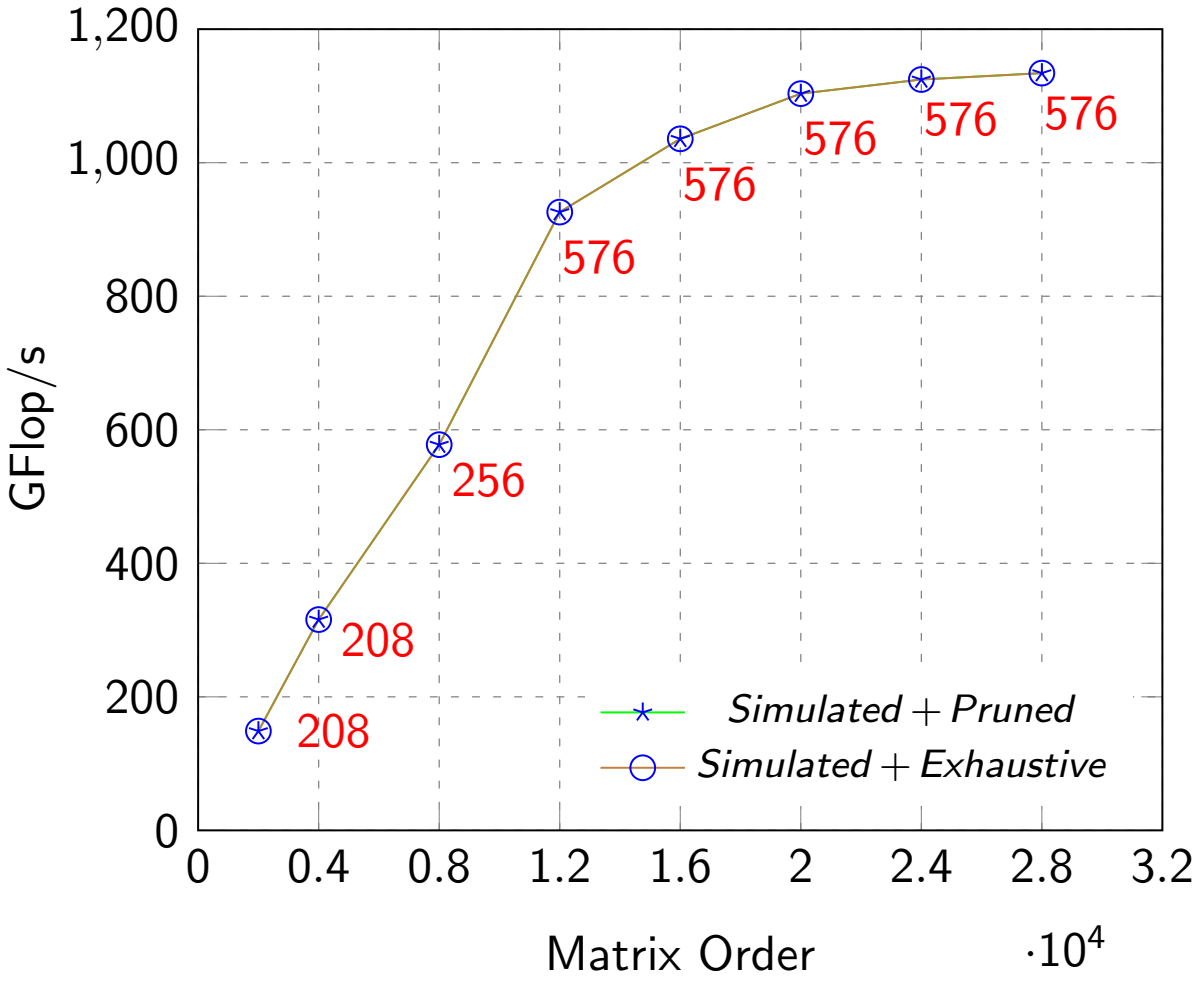
Exhaustive (S3) vs. Pruned (S4)





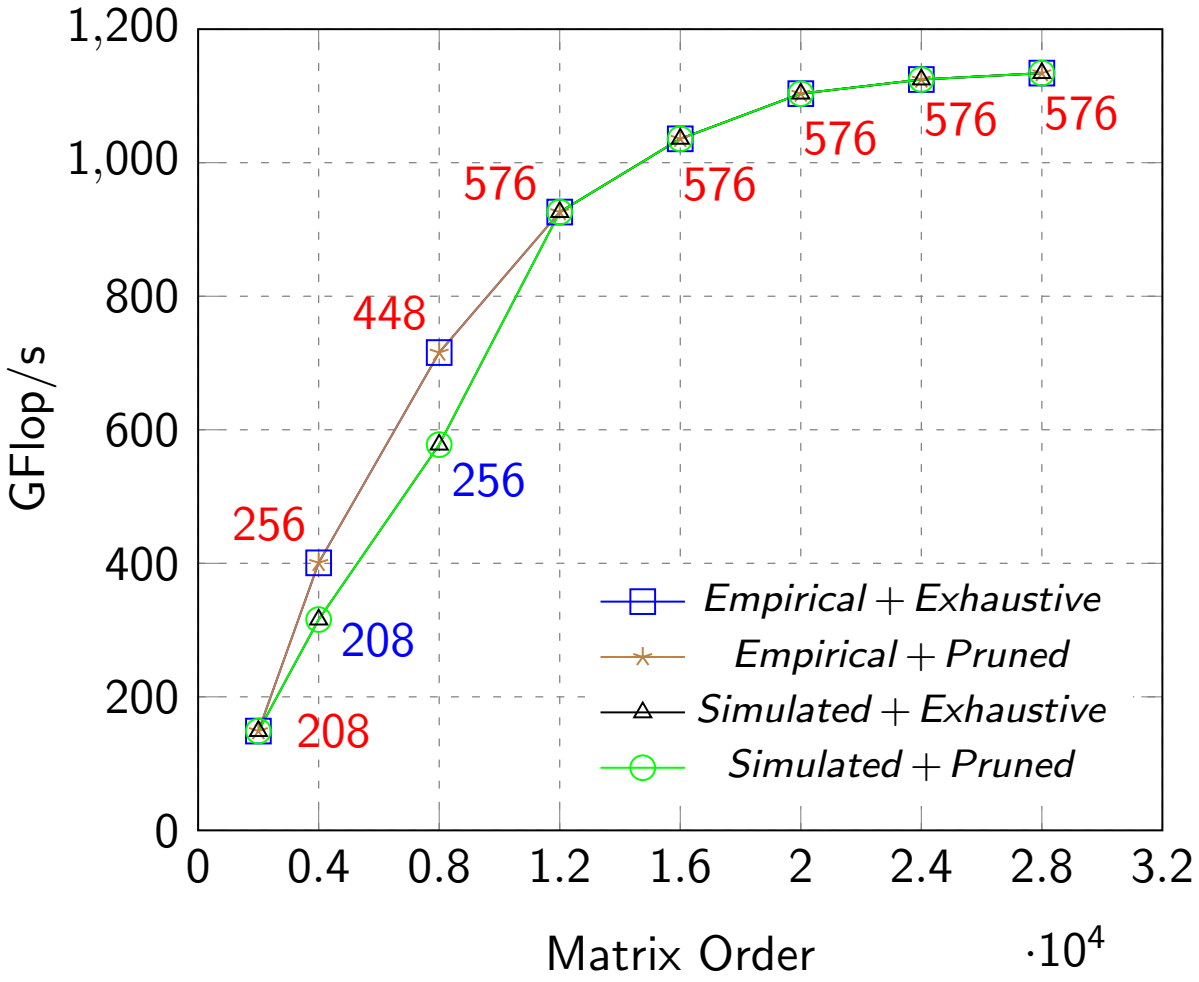
Same *nb* value for each problem size *n*

S3 & S4



S3: 30 min; **S4**: 84 sec

S1 & S2 & S3 & S4



S1: 30 min; **S2**: 154 sec; **S3**: 30 min; **S4**: 84 sec

Overview

Selection of Computing Units

StarPU

Tasks are assigned dinamically to the CUs using a scheduling policy, but it does not take into account the computational power of the CUs.

Selection of Computing Units

StarPU

Tasks are assigned dinamically to the CUs using a scheduling policy, but it does not take into account the computational power of the CUs.

Goal

Decide the best number of CUs (number of CPU cores and GPUs) to use for a given problem size n.

Selection of Computing Units

StarPU

Tasks are assigned dinamically to the CUs using a scheduling policy, but it does not take into account the computational power of the CUs.

Goal

Decide the best number of CUs (number of CPU cores and GPUs) to use for a given problem size n.

Tuning Technique

Selective Searching: successively adds CUs in increasing powerful order using as starting point the best value for nb for the previous problem size n.

Selection of Computing Units

				Performance (GFlops)		
n	*nb*	*CPU_Cores*	*GPU_IDs*	*Tuned*	*Chameleon*	%
1000	112	12	$\{-\}$	76	46	40
2000	192	9	$\{1,5,0\}$	164	117	29
3000	192	8	$\{1,5,0,2\}$	285	196	31
4000	240	7	$\{1,5,0,2,3\}$	412	352	15
5000	256	6	$\{1,5,0,2,3,4\}$	545	465	15
6000	256	6	$\{1,5,0,2,3,4\}$	626	554	12
7000	320	6	$\{1,5,0,2,3,4\}$	687	608	12
8000	320	6	$\{1,5,0,2,3,4\}$	753	682	10
9000	304	6	$\{1,5,0,2,3,4\}$	791	713	10
10000	304	6	$\{1,5,0,2,3,4\}$	834	758	9
11000	512	6	$\{1,5,0,2,3,4\}$	880	803	9
12000	576	6	$\{1,5,0,2,3,4\}$	909	896	1

Overview

Conclusions

- The *pruned* strategy allows us to achieve good results both in the *empirical* and in the *simulated* approaches.
- Results obtained with the *simulated* approach are very similar in terms of performance to that achieved by the *empirical* one.
- The combination *Simulated+Pruned* is a decent tuning strategy to obtain performances close to the *empirical* without (almost) requiring to use the computational platform.
- An appropriate selection of the CUs allows to improve the performance with an efficient exploitation of the heterogeneous platform.

Future Work

- Apply the empirical and simulated tuning strategies to other routines of Chameleon, such as LU or QR.
- Consider additional parameters: inner block (*ib*) and scheduling policies (*eager, lws, dmda, . . .*)
- Integrate the autotuning techniques inside Chameleon.

Thanks for your Attention!