Parallel schedulers on dense matrices

Christian Eder
joint work with Jean-Charles Faugère

POLSYS Team, UPMC, Paris, France

June 11, 2013

Preconditions I

- Using **dense matrices** with **unsigned int64** entries.
- Computing in F_p, p some **prime** $< 2^{16}$.
- We compared the following set of parallel schedulers:
  1. **pthread** (or in other words, by hand),
  2. **OpenMP** (sometimes together with pthread),
  3. **Intel TBB** (using lambda expressions),
  4. **XKAAPI** (in particular, the C interface KAAPIC),

Preconditions I

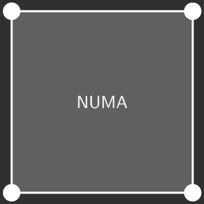
- Using **dense matrices** with **unsigned int64** entries.
- Computing in F_p, p some **prime** $< 2^{16}$.
- We compared the following set of parallel schedulers:
  1. **pthread** (or in other words, by hand),
  2. **OpenMP** (sometimes together with pthread),
  3. **Intel TBB** (using lambda expressions),
  4. **XKAAPI** (in particular, the C interface KAAPIC),

Note

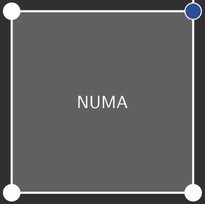
The implemented algorithms are **not optimized** in order to keep the influence on the schedulers as low as possible.

Preconditions II

Results presented computed on the **HPAC compute server**



Results presented computed on the **HPAC compute server**

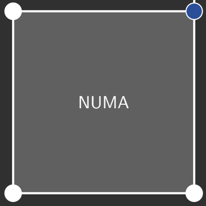


with hyperthreading: 64 **cores**

- 8 Intel Xeon E5-4620 cores @ 2.20 GHz
- L1 cache: 32 KB
- L2 cache: 256 KB
- shared L3 cache: 16 MB
- 96 GB RAM

Preconditions II

Results presented computed on the **HPAC compute server**



with hyperthreading: 64 **cores**

- 8 Intel Xeon E5-4620 cores @ 2.20 GHz
- L1 cache: 32 KB
- L2 cache: 256 KB
- shared L3 cache: 16 MB
- 96 GB RAM

Also tested on: 48-core (real cores) AMD Magny Cours NUMA,
4-core (8 with hyperthreading) Intel Sandy Bridge.

Tested algorithms

1. **Naive Dense Matrix Multiplication**
2. **Dense Gaussian Elimination**:
   (a) **Naive** implementation (with and without pivoting)
   (b) **Cache-oblivious** implementation (GEP by Chowdhury and Ramachandran without pivoting)

Naive dense matrix multiplication

We compared several variants of parallelized FOR loops:

Naive dense matrix multiplication

We compared several variants of parallelized FOR loops:

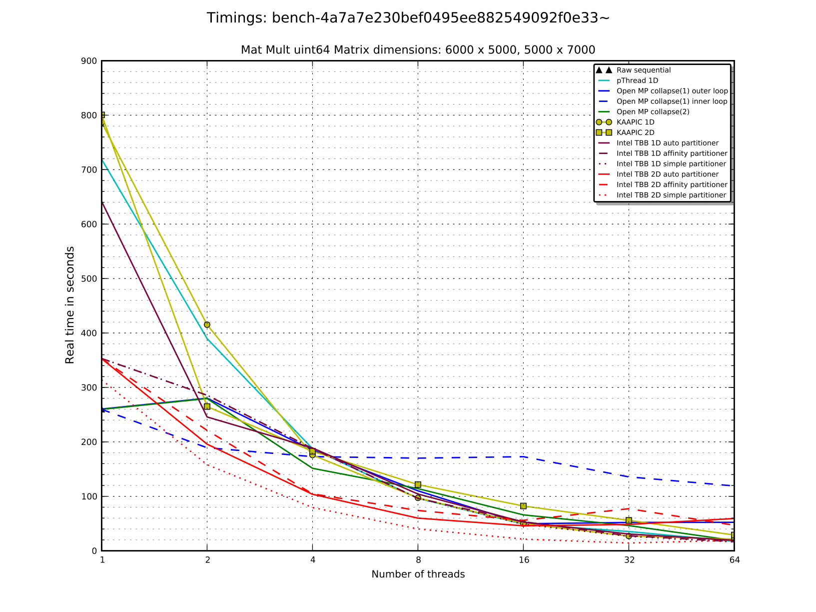
- 1-**dimensional** vs. 2-**dimensional** parallel loops

Naive dense matrix multiplication

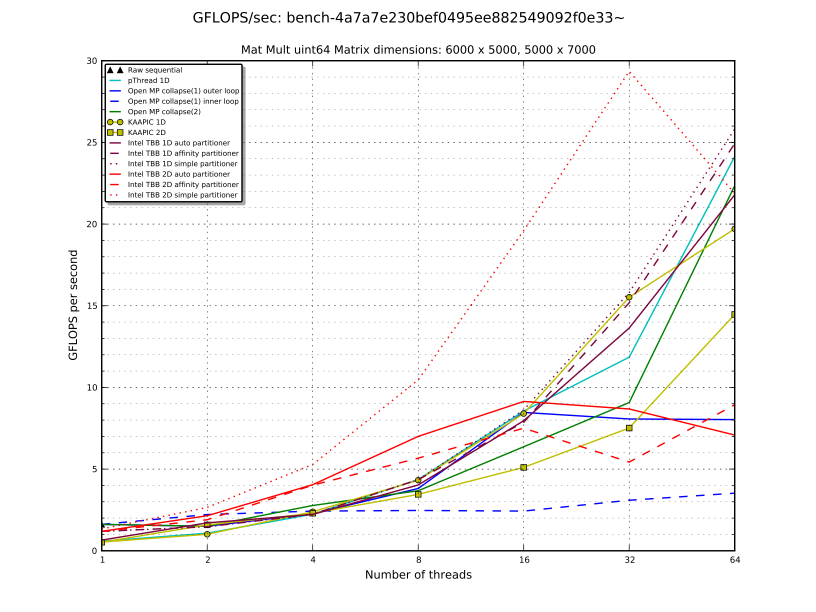
We compared several variants of parallelized FOR loops:

- **1-dimensional** vs. **2-dimensional** parallel loops
- For Intel TBB we compared the different integrated schedulers:
  - **auto partitioner:** Splitting work to balance load
  - **affine partitioner:** Improves choice of CPU affinity
  - **simple partitioner:** Recursively splits a range until it is no longer divisible (grainsize is critical)

Timings


Timings: bench-4a7a7e230bef0495ee882549092f0e33~
Mat Mult uint64 Matrix dimensions: 6000 x 5000, 5000 x 7000
Real time in seconds
0
100
200
300
400
500
600
700
800
900
Number of threads
1
2
4
8
16
32
64
Raw sequential
pThread 1D
Open MP collapse(1) outer loop
Open MP collapse(1) inner loop
Open MP collapse(2)
KAAPIC 1D
KAAPIC 2D
Intel TBB 1D auto partitioner
Intel TBB 1D affinity partitioner
Intel TBB 1D simple partitioner
Intel TBB 2D auto partitioner
Intel TBB 2D affinity partitioner
Intel TBB 2D simple partitioner

GFLOPS/sec


GFLOPS/sec: bench-4a7a7e230bef0495ee882549092f0e33~
Mat Mult uint64 Matrix dimensions: 6000 x 5000, 5000 x 7000
Raw sequential
pThread 1D
Open MP collapse(1) outer loop
Open MP collapse(1) inner loop
Open MP collapse(2)
KAAPIC 1D
KAAPIC 2D
Intel TBB 1D auto partitioner
Intel TBB 1D affinity partitioner
Intel TBB 1D simple partitioner
Intel TBB 2D auto partitioner
Intel TBB 2D affinity partitioner
Intel TBB 2D simple partitioner
GFLOPS per second
0
5
10
15
20
25
30
Number of threads
1
2
4
8
16
32
64

Compared to naive multiplication we saw a different behaviour:

Compared to naive multiplication we saw a different behaviour:

- **KAAPIC**, **Open MP** and **Intel TBB** are in the same range.

Naive dense Gaussian Elimination

Compared to naive multiplication we saw a different behaviour:

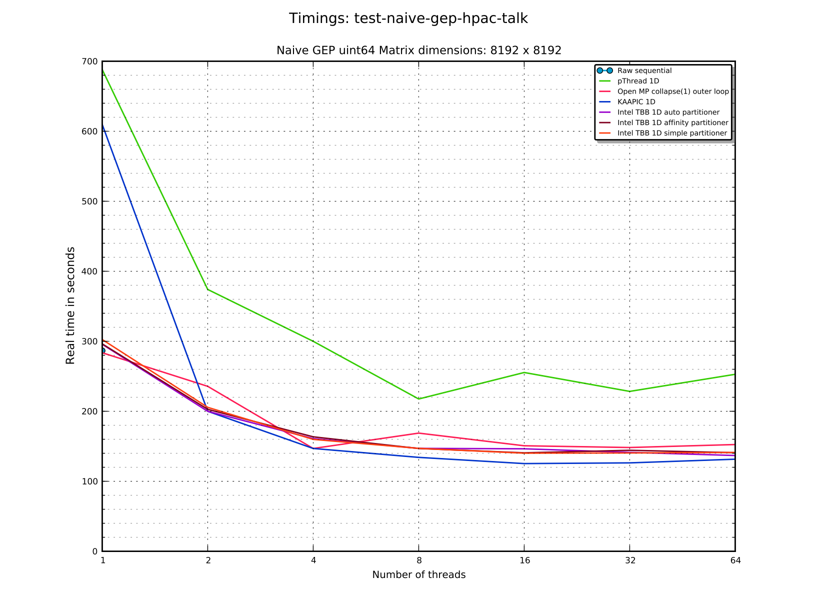
- **KAAPIC**, **Open MP** and **Intel TBB** are in the same range.
- **Open MP** behaves a bit worse when it comes to hyperthreading.

Naive dense Gaussian Elimination

Compared to naive multiplication we saw a different behaviour:

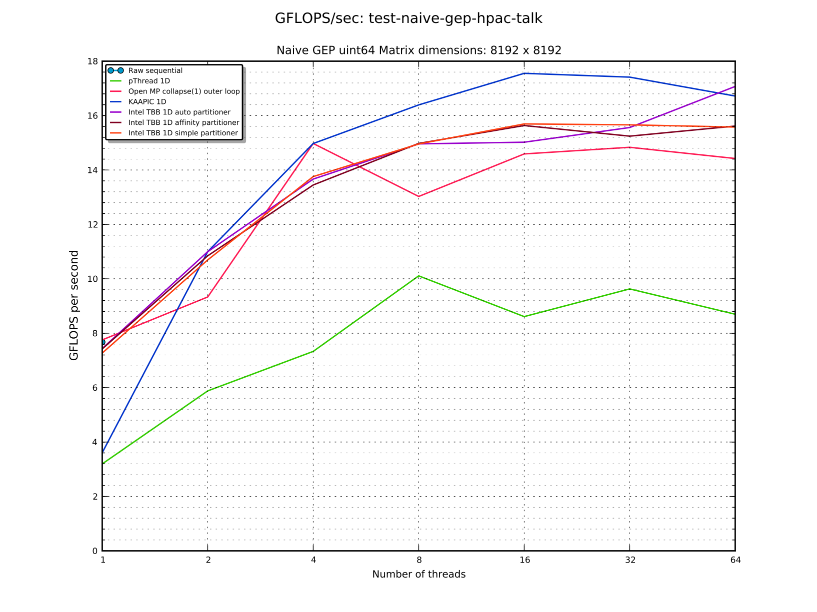
- **KAAPIC**, **Open MP** and **Intel TBB** are in the same range.
- **Open MP** behaves a bit worse when it comes to hyperthreading.
- **pthread** implementation slows down due to lack of real scheduler.

Timings



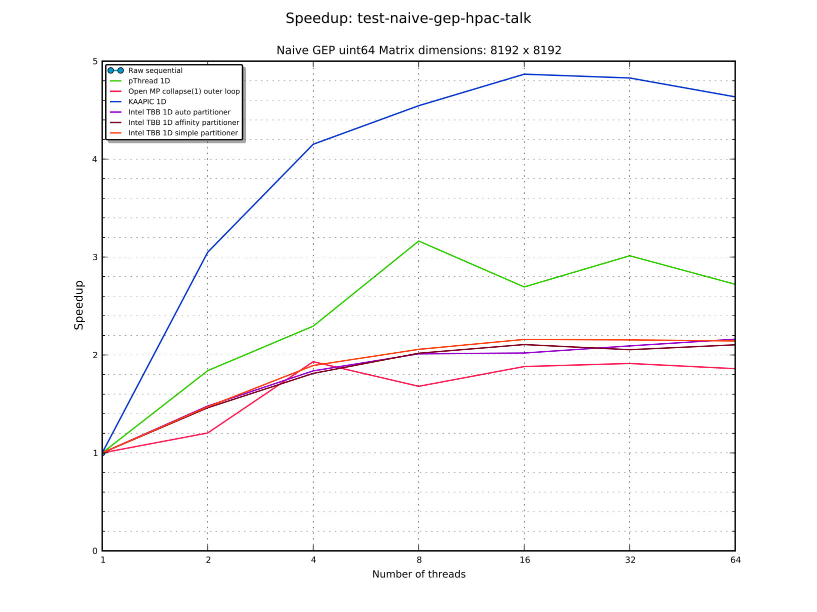
Timings: test-naive-gep-hpac-talk
Naive GEP uint64 Matrix dimensions: 8192 x 8192
Raw sequential
pThread 1D
Open MP collapse(1) outer loop
KAAPIC 1D
Intel TBB 1D auto partitioner
Intel TBB 1D affinity partitioner
Intel TBB 1D simple partitioner
Real time in seconds
Number of threads
0
100
200
300
400
500
600
700
1
2
4
8
16
32
64

GFLOPS/sec



GFLOPS/sec: test-naive-gep-hpac-talk
Naive GEP uint64 Matrix dimensions: 8192 x 8192
Raw sequential
pThread 1D
Open MP collapse(1) outer loop
KAAPIC 1D
Intel TBB 1D auto partitioner
Intel TBB 1D affinity partitioner
Intel TBB 1D simple partitioner
GFLOPS per second
Number of threads
0
2
4
6
8
10
12
14
16
18
1
2
4
8
16
32
64

Speedup



Speedup: test-naive-gep-hpac-talk
Naive GEP uint64 Matrix dimensions: 8192 x 8192
Raw sequential
pThread 1D
Open MP collapse(1) outer loop
KAAPIC 1D
Intel TBB 1D auto partitioner
Intel TBB 1D affinity partitioner
Intel TBB 1D simple partitioner
Speedup
Number of threads
0
1
2
3
4
5
1
2
4
8
16
32
64

- Basics
- Naive dense matrix multiplication
- Naive dense Gaussian Eliminaton
- **Cache-oblivious dense Gaussian Elimination**
- Features of the library

Cache-oblivious dense Gaussian Elimination

Implemented **I-GEP** from [CR10].

Cache-oblivious dense Gaussian Elimination

Implemented **I-GEP** from [CR10].
Basic ideas are:

- Assume matrix of dimensions $2^k \times 2^k$.

Cache-oblivious dense Gaussian Elimination

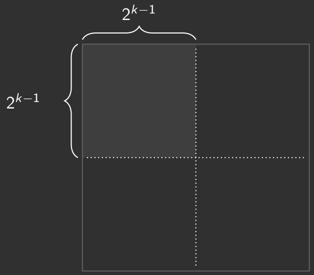
Implemented **I-GEP** from [CR10].
Basic ideas are:

- Assume matrix of dimensions $2^k \times 2^k$.
- Do not consider pivoting.

Cache-oblivious dense Gaussian Elimination

Implemented **I-GEP** from [CR10].
Basic ideas are:

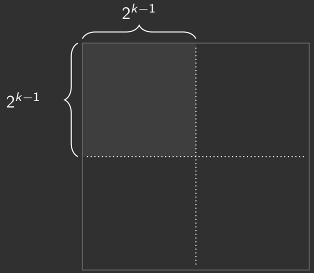
- Assume matrix of dimensions $2^k \times 2^k$.
- Do not consider pivoting.
- Recursively split matrix in 4 same-sized parts.



Cache-oblivious dense Gaussian Elimination

Implemented **I-GEP** from [CR10].
Basic ideas are:

- Assume matrix of dimensions $2^k \times 2^k$.
- Do not consider pivoting.
- Recursively split matrix in 4 same-sized parts.

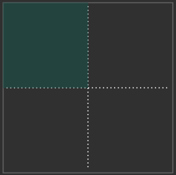


- Stop recursion once parts fit in cache.

Cache-oblivious dense Gaussian Elimination

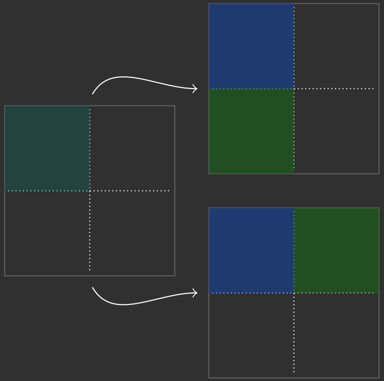
Needs a bit of globally bookkeeping (inverse pivots, etc.)

Cache-oblivious dense Gaussian Elimination



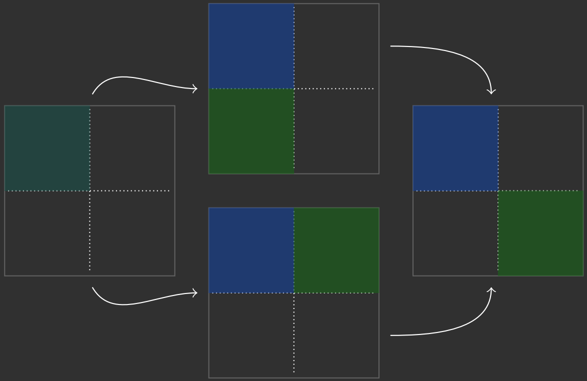
Needs a bit of globally bookkeeping (inverse pivots, etc.)

Cache-oblivious dense Gaussian Elimination



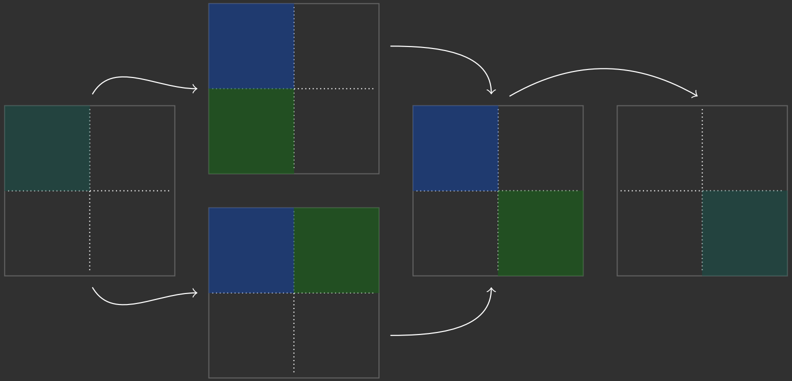
Needs a bit of globally bookkeeping (inverse pivots, etc.)

Cache-oblivious dense Gaussian Elimination



Needs a bit of globally bookkeeping (inverse pivots, etc.)

Cache-oblivious dense Gaussian Elimination



Needs a bit of globally bookkeeping (inverse pivots, etc.)

Cache-oblivious dense Gaussian Elimination

Differences to the naive approach:

- The base cases are **not parallelized**.

Cache-oblivious dense Gaussian Elimination

Differences to the naive approach:

- The base cases are **not parallelized**.
- There are **no parallel FOR loops**.

Cache-oblivious dense Gaussian Elimination

Differences to the naive approach:

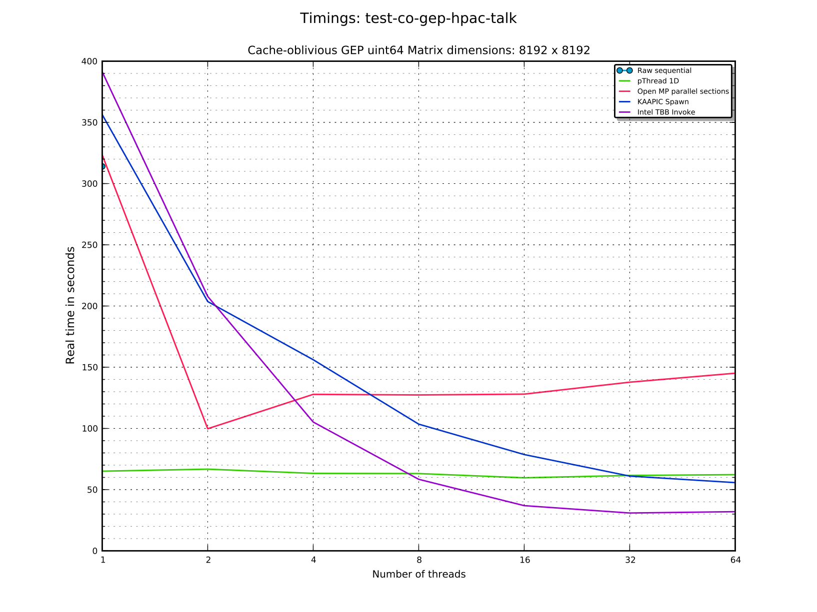
- The base cases are **not parallelized**.
- There are **no parallel FOR loops**.
- Instead we need to use a **recursive task scheduling**:

Cache-oblivious dense Gaussian Elimination

Differences to the naive approach:

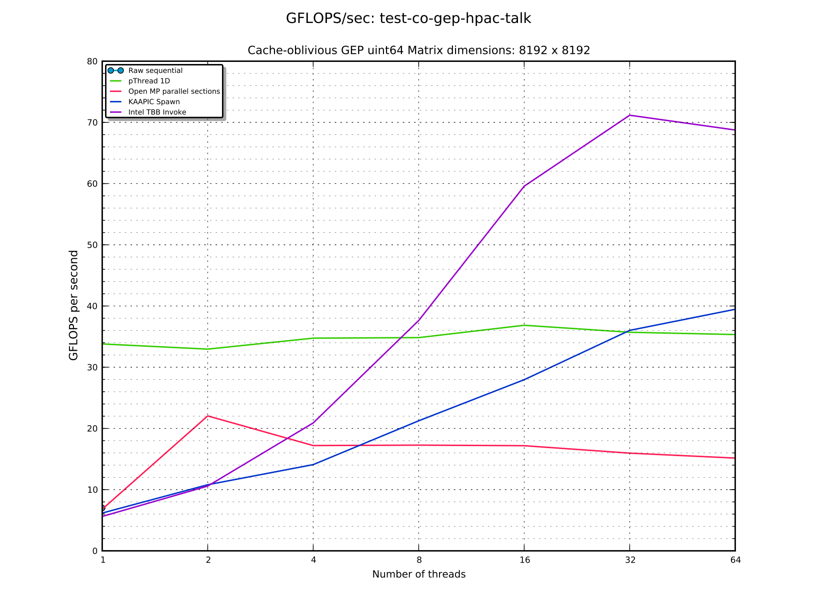
- The base cases are **not parallelized**.
- There are **no parallel FOR loops**.
- Instead we need to use a **recursive task scheduling**:
  - **pthread**: no scheduling, left unbound.
  - **Open MP**: PARALLEL SECTIONS (real tasks should be available in Open MP 4.0)
  - **KAAPIC**: KAAPIC_SPAWN
  - **Intel TBB**: INVOKE

Timings



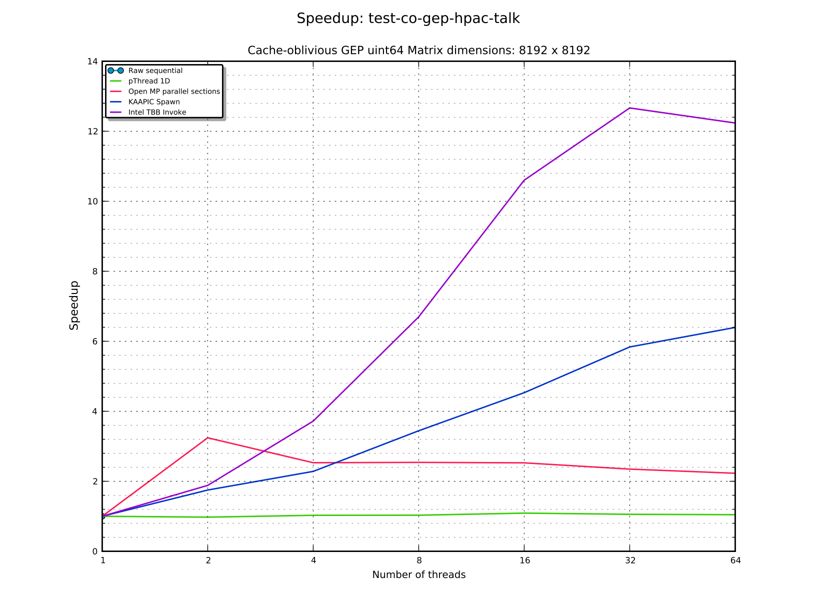
Timings: test-co-gep-hpac-talk
Cache-oblivious GEP uint64 Matrix dimensions: 8192 x 8192
Raw sequential
pThread 1D
Open MP parallel sections
KAAPIC Spawn
Intel TBB Invoke
Real time in seconds
Number of threads
0
50
100
150
200
250
300
350
400
1
2
4
8
16
32
64

GFLOPS/sec



GFLOPS/sec: test-co-gep-hpac-talk
Cache-oblivious GEP uint64 Matrix dimensions: 8192 x 8192
Raw sequential
pThread 1D
Open MP parallel sections
KAAPIC Spawn
Intel TBB Invoke
GFLOPS per second
Number of threads
0
10
20
30
40
50
60
70
80
1
2
4
8
16
32
64

Speedup



Speedup: test-co-gep-hpac-talk
Cache-oblivious GEP uint64 Matrix dimensions: 8192 x 8192
Raw sequential
pThread 1D
Open MP parallel sections
KAAPIC Spawn
Intel TBB Invoke
Speedup
Number of threads
0
2
4
6
8
10
12
14
1
2
4
8
16
32
64

- Basics
- Naive dense matrix multiplication
- Naive dense Gaussian Eliminaton
- Cache-oblivious dense Gaussian Elimination
- **Features of the library**

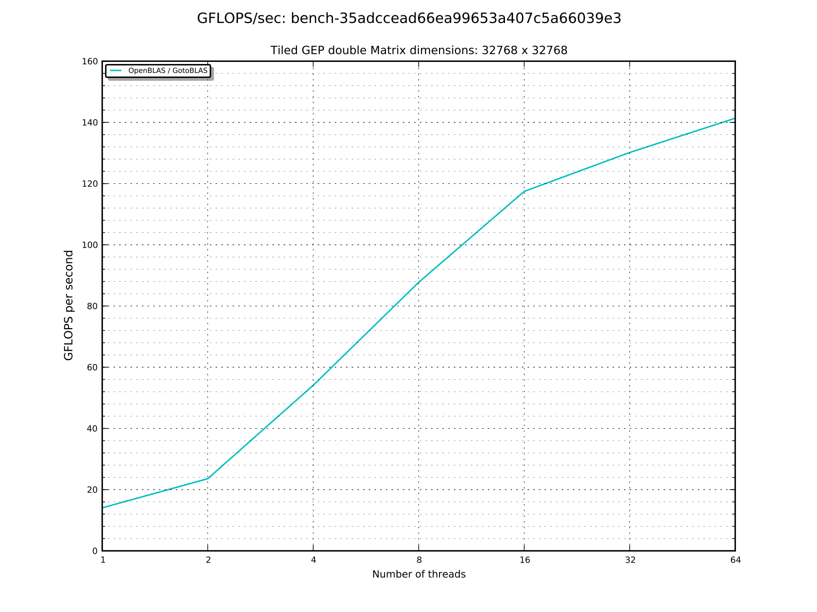
Features of the library

- Detection of available parallel schedulers

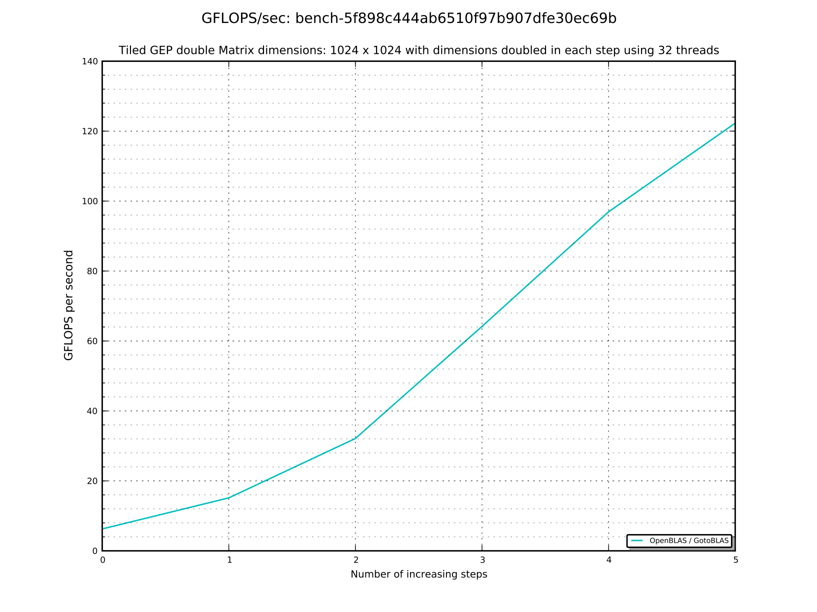
Features of the library

- Detection of available parallel schedulers
- Userfriendly interface to add new algorithms easily: For example, one can easily drop in **ATLAS**, **OpenBLAS**, **PLASMA**, etc.

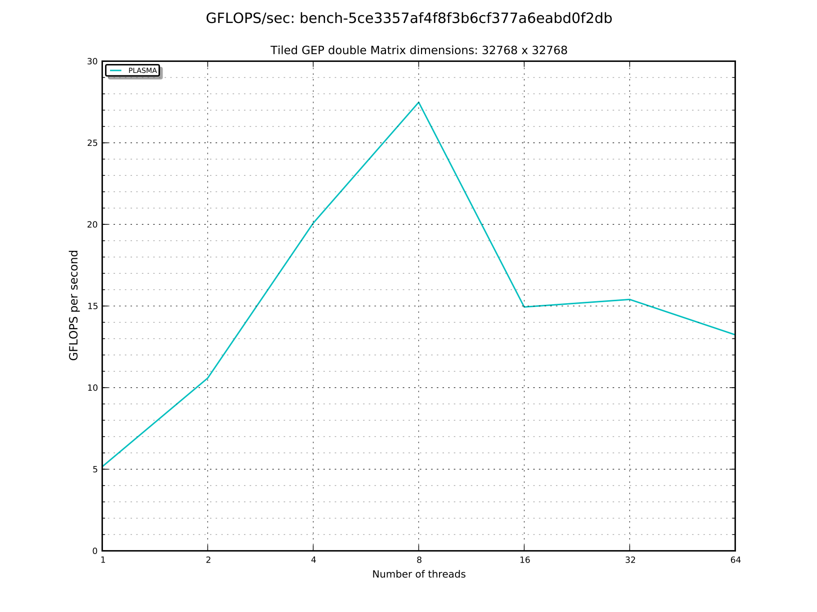
Features of the library



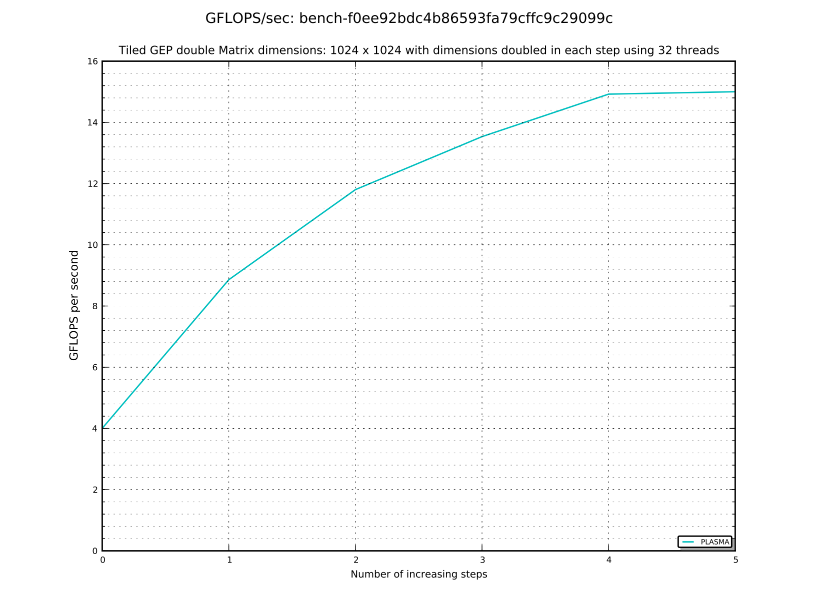
Features of the library



Features of the library



Features of the library



Features of the library

- Detection of available parallel schedulers
- Userfriendly interface to add new algorithms easily: For example, one can easily drop in **ATLAS**, **OpenBLAS**, **PLASMA**, etc.
- Easy to use and highly customizable, Python-based benchmarking tools including plotting functionality

Features of the library

- Detection of available parallel schedulers
- Userfriendly interface to add new algorithms easily: For example, one can easily drop in **ATLAS**, **OpenBLAS**, **PLASMA**, etc.
- Easy to use and highly customizable, Python-based benchmarking tools including plotting functionality
- Publicly available: `https://github.com/ederc/LA-BENCHER`

Bibliography

[PL13] E. Agullo et al. PLASMA Users' Guide: Parallel Linear Algebra Software for Multicore Architectures, Version 2.0

[OB13] Z. Xianyi, W. Quian and Z. Chothia. OpenBLAS, `http://xianyi.github.com/OpenBLAS`

[CR10] R. A. Chowdhury and V. Ramachandran. The Cache-Oblivious Gaussian Elimination Paradigm: Theoretical Framework, Parallelization and Experimental Evaluation

[WP04] R. C. Whaley and A. Petitet Minimizing development and maintenance costs in supporting persistently optimized BLAS

[WPD01] R. C. Whaley, A. Petitet and J. J. Dongarra Automated Empirical Optimization of Software and the ATLAS Project

[WD99] R. C. Whaley and J. J. Dongarra Automatically Tuned Linear Algebra Software