Parallel schedulers on dense matrices

Christian Eder
joint work with Jean-Charles Faugère

POLSYS Team, UPMC, Paris, France

June 11, 2013
Basics

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

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

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

- 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

with hyperthreading: 64 cores
Preconditions II

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

NUMA

- 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

with hyperthreading: 64 cores

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)
Basics

Naive dense matrix multiplication

Naive dense Gaussian Elimination

Cache-oblivious dense Gaussian Elimination

Features of the library
We compared several variants of parallelized FOR loops:
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)
Mat Mult uint64 Matrix dimensions: 6000 x 5000, 5000 x 7000

Timings: bench-4a7a7e230bef0495ee882549092f0e33~
Mat Mult uint64 Matrix dimensions: 6000 x 5000, 5000 x 7000

GFLOPS/sec: bench-4a7a7e230bef0495ee882549092f0e33~
Basics

Naive dense matrix multiplication

Naive dense Gaussian Elimination

Cache-oblivious dense Gaussian Elimination

Features of the library
Naive dense Gaussian Elimination

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.
- Open MP behaves a bit worse when it comes to hyperthreading.
- pthread implementation slows down due to lack of real scheduler.
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.
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: test-naive-gep-hpac-talk

Naive GEP uint64 Matrix dimensions: 8192 x 8192

Number of threads
0
100
200
300
400
500
600
700
Real time in seconds

- Raw sequential
- pThread 1D
- OpenMP collapse(1) outer loop
- KAAPIC 1D
- Intel TBB 1D auto partitioner
- Intel TBB 1D affinity partitioner
- Intel TBB 1D simple partitioner
GFLOPS/sec
Naive GEP uint64 Matrix dimensions: 8192 x 8192

GFLOPS/sec: test-naive-gep-hpac-talk

Number of threads
0
2
4
6
8
10
12
14
16
18

GFLOPS per second

- 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

Number of threads
1
2
4
8
16
32
64
Naive GEP uint64 Matrix dimensions: 8192 x 8192

Speedup: test-naive-gep-hpac-talk

Number of threads
0
1
2
3
4
5

Speedup

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
Basics

Naive dense matrix multiplication

Naive dense Gaussian Elimination

Cache-oblivious dense Gaussian Elimination

Features of the library
Cache-oblivious dense Gaussian Elimination

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

Basic ideas are:

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

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

Stop recursion once parts fit in cache.
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.

![Diagram](image)
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.)
Differences to the naive approach:

- The base cases are **not parallelized**.
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:
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: test-co-gep-hpac-talk

Cache-oblivious GEP uint64 Matrix dimensions: 8192 x 8192

- Raw sequential
- pthread 1D
- OpenMP parallel sections
- KAAPIC Spawn
- Intel TBB Invoke

Timings include:
- Number of threads
- Real time in seconds
GFLOPS/sec: test-co-gep-hpac-talk

Cache-oblivious GEP uint64 Matrix dimensions: 8192 x 8192
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
Basics

Naive dense matrix multiplication

Naive dense Gaussian Elimination

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

![Graph showing GFLOPS per second vs. Number of threads for Tiled GEP double Matrix dimensions: 32768 x 32768]

GFLOPS/sec: bench-35adccead66ea99653a407c5a66039e3

Number of threads:
- 0
- 20
- 40
- 60
- 80
- 100
- 120
- 140
- 160

GFLOPS per second:
- 0
- 20
- 40
- 60
- 80
- 100
- 120
- 140
- 160

Tiled GEP double Matrix dimensions: 32768 x 32768

OpenBLAS / GotoBLAS
Features of the library

GFLOPS/sec: bench-5f898c444ab6510f97b907dfe30ec69b

Tiled GEP double Matrix dimensions: 1024 x 1024 with dimensions doubled in each step using 32 threads

OpenBLAS / GotoBLAS

GFLOPS per second

Number of increasing steps
Features of the library

GFLOPS/sec: bench-5ce3357af4f8f3b6cf377a6eabd0f2db

Tiled GEP double Matrix dimensions: 32768 x 32768
Features of the library

GFLOPS/sec: bench-f0ee92bdc4b86593fa79cffe9c29099c

Tiled GEP double Matrix dimensions: 1024 x 1024 with dimensions doubled in each step using 32 threads
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-BENCHE


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


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