Hybrid
Matrix Multiplication
&
Gaussian Elimination

Christian Eder, Jean-Charles Faugère, Fayssal Martani
and Bjarke Hammersholt Roune

HPAC Meeting – Lyon

December 10 – 11, 2013
Fast linear algebra for computing Gröbner bases

Hybrid Matrix Multiplication

Next steps on matrix multiplication

First steps on Gaussian Elimination

Next steps
First attempts

In 2011 the original LELA project for fast linear algebra on special structured hybrid-sparse-dense matrices was started at the university of Kaiserslautern in the Singular team by Bradford Hovinen.

https://github.com/Singular/LELA
First attempts

In 2011 the original LELA project for fast linear algebra on special structured hybrid-sparse-dense matrices was started at the university of Kaiserslautern in the Singular team by Bradford Hovinen.

https://github.com/Singular/LELA

In 2012, doing a master thesis at the INRIA PolSys team Fayssal Martani optimized this code and added a completely new component with a specialized algorithm for computing the Gaussian Elimination.

https://github.com/martani/LELA
First attempts

In 2011 the original LELA project for fast linear algebra on special structured hybrid-sparse-dense matrices was started at the university of Kaiserslautern in the Singular team by Bradford Hovinen.

https://github.com/Singular/LELA

In 2012, doing a master thesis at the INRIA PolSys team Fayssal Martani optimized this code and added a completely new component with a specialized algorithm for computing the Gaussian Elimination.

https://github.com/martani/LELA

Also in 2012, Bjarke Hammersholt Roune started working on Linear Algebra for computing Gröbner bases in the MATHICGB package at the university of Kaiserslautern.

https://github.com/broune/mathicgb
The idea for a specialized Gaussian Elimination was initially presented by Faugère and Lachartre ([3]).
Basic idea

The idea for a specialized Gaussian Elimination was initially presented by Faugère and Lachartre ([3]).

- Special structure of the matrices due to Gröbner basis computations.
The idea for a specialized Gaussian Elimination was initially presented by Faugère and Lachartre ([3]).

- Special structure of the matrices due to Gröbner basis computations.
- Restructure matrix, swap rows and columns, take care of underlying monomial order
The idea for a specialized Gaussian Elimination was initially presented by Faugère and Lachartre ([3]).

- Special structure of the matrices due to Gröbner basis computations.
- Restructure matrix, swap rows and columns, take care of underlying monomial order
- Split matrix in 4 parts:
How our matrices look like
Fast linear algebra for computing Gröbner bases

Hybrid Matrix Multiplication

Next steps on matrix multiplication

First steps on Gaussian Elimination

Next steps
Hybrid Matrix Multiplication
\[ B \leftarrow A^{-1}B \]
$B \leftarrow A^{-1}B$ – Block Version
\[ B \leftarrow A^{-1} B \quad \text{– Block Version} \]
$B \leftarrow A^{-1}B$ – Block Version
$B \leftarrow A^{-1} B$ – Block Version
LELA implementation

```c
#ifdef L1 //LEVEL 1 PARALLELISM
omp_set_nested(1);
#pragma omp parallel num_threads(NUM_THREADS_OMP_MASTER) {
#endif
//for all columns of blocs of B
#ifdef L1 //LEVEL 1 PARALLELISM
#pragma omp for schedule(dynamic) nowait
#endif
for (uint32 ii = 0; ii < nb_column_blocs_B; ++ii) {
  //for all rows of blocs in A (starting from the end)
  for (uint32 jj = 0; jj < nb_row_blocs_A; ++jj) {
    const uint32 first_bloc_idx =
      A.FirstBlocsColumnIndexes[jj] / A.bloc_width();
    const uint32 last_bloc_idx = MIN(A[jj].size() - 1, jj);
    Level1Ops::memsetToZero(dense_bloc);
    Level1Ops::copySparseBlocToDenseBlocArray(R, B[jj][ii], dense_bloc);
    //for all the blocs in the current row of A
    for (uint32 k = 0; k < last_bloc_idx; ++k) {
      if (A[jj][k].empty() || B[k + first_bloc_idx][ii].empty())
        continue;
    }
  }
}
#endif L2 //LEVEL 2!
#pragma omp parallel num_threads(NUM_THREADS_OMP_SLAVES_PER_MASTER) {
#pragma omp for schedule(dynamic) nowait
#endif
for (int i = 0; i < DEFAULT_BLOC_HEIGHT / 2; ++i) {
  uint8 is_sparse = 0;
```
### Some examples – Structures

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>size $A$</th>
<th>density $A$</th>
<th>size $B$</th>
<th>density $B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kat12-Mat7</td>
<td>$18,890 \times 18,890$</td>
<td>0.76%</td>
<td>$18,890 \times 4,237$</td>
<td>14.02%</td>
</tr>
<tr>
<td>Kat13-Mat10</td>
<td>$40,023 \times 40,023$</td>
<td>0.50%</td>
<td>$40,023 \times 8,204$</td>
<td>13.53%</td>
</tr>
<tr>
<td>Kat14-Mat14</td>
<td>$88,941 \times 88,941$</td>
<td>0.37%</td>
<td>$88,941 \times 16,397$</td>
<td>12.03%</td>
</tr>
<tr>
<td>Kat16-Mat6</td>
<td>$82,086 \times 82,086$</td>
<td>0.22%</td>
<td>$82,086 \times 32,166$</td>
<td>3.54%</td>
</tr>
<tr>
<td>Minrank-9-10-7-Mat3</td>
<td>$18,460 \times 18,460$</td>
<td>4.01%</td>
<td>$18,460 \times 56,095$</td>
<td>14.01%</td>
</tr>
<tr>
<td>Minrank-9-10-7-Mat4</td>
<td>$34,053 \times 34,053$</td>
<td>3.52%</td>
<td>$34,053 \times 74,125$</td>
<td>38.93%</td>
</tr>
</tbody>
</table>
## Some examples – Timings LELA & MATHICGB

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Timings in seconds with given number of threads</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>K12-M7-L</td>
<td>31.09</td>
</tr>
<tr>
<td>K14-M7-M</td>
<td>32.11</td>
</tr>
<tr>
<td>K13-M10-L</td>
<td>175.41</td>
</tr>
<tr>
<td>K13-M10-M</td>
<td>176.52</td>
</tr>
<tr>
<td>K14-M14-L</td>
<td>1,285.10</td>
</tr>
<tr>
<td>K14-M14-M</td>
<td>1,2799.23</td>
</tr>
<tr>
<td>K16-M6-L</td>
<td>593.32</td>
</tr>
<tr>
<td>K16-M6-M</td>
<td>597.41</td>
</tr>
<tr>
<td>M-9-10-7-M3-L</td>
<td>1,734.00</td>
</tr>
<tr>
<td>M-9-10-7-M3-M</td>
<td>1,745.81</td>
</tr>
<tr>
<td>M-9-10-7-M4-L</td>
<td>6,410.00</td>
</tr>
<tr>
<td>M-9-10-7-M4-M</td>
<td>6,421.23</td>
</tr>
</tbody>
</table>
### Some examples – Speedups LELA & MATHICGB

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Speedup w.r.t. computation on 1 thread</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>K12-M7-L</td>
<td>1.00</td>
</tr>
<tr>
<td>K12-M7-M</td>
<td>0.97</td>
</tr>
<tr>
<td>K13-M10-L</td>
<td>1.00</td>
</tr>
<tr>
<td>K13-M10-M</td>
<td>0.99</td>
</tr>
<tr>
<td>K14-M14-L</td>
<td>0.99</td>
</tr>
<tr>
<td>K14-M14-M</td>
<td>1.00</td>
</tr>
<tr>
<td>K16-M6-L</td>
<td>1.00</td>
</tr>
<tr>
<td>K16-M6-M</td>
<td>0.99</td>
</tr>
<tr>
<td>M-9-10-7-M3-L</td>
<td>1.00</td>
</tr>
<tr>
<td>M-9-10-7-M3-M</td>
<td>0.99</td>
</tr>
<tr>
<td>M-9-10-7-M4-L</td>
<td>1.00</td>
</tr>
<tr>
<td>M-9-10-7-M4-M</td>
<td>0.99</td>
</tr>
</tbody>
</table>
Fast linear algebra for computing Gröbner bases

Hybrid Matrix Multiplication

Next steps on matrix multiplication

First steps on Gaussian Elimination

Next steps
Next steps

► Test and compare different sparse, hybrid, dense data structures in LELA and MATHICGB
► Implement and compare task-based hybrid tiled matrix multiplication with StarPU, OpenMP/KAAPI and Intel TBB
Fast linear algebra for computing Gröbner bases

Hybrid Matrix Multiplication

Next steps on matrix multiplication

First steps on Gaussian Elimination

Next steps
First step – Dense Gaussian Elimination
Some examples – Kat16-Mat9

Timings: test-tiled-gep-2395-74125-256

Tiled GEP uint64 Matrix dimensions: 2395 x 74125

- Intel TBB - bs 64
- LELA - bs 256
- StarPU - bs 96

Real time in seconds vs Number of threads
Some examples – Minrank-10-9-7-Mat4

Timings: test-tiled-gep-3614-60539-256

Tiled GEP uint64 Matrix dimensions: 3614 x 60539

Real time in seconds

Number of threads

Intel TBB - bs 96
LELA - bs 256
StarPU - bs 96
Some examples – Minrank-10-9-7-Mat6

Timings: test-tiled-gep-4040-101500-256

Tiled GEP uint64 Matrix dimensions: 4040 x 101500

Number of threads

Real time in seconds

Intel TBB - bs 128
LELA - bs 256
StarPU - bs 192
LELA implementation

- Using pthreads

- Idea of structured Gaussian Elimination:

  \[
  \text{row}_i \leftarrow \text{row}_i + \sum_{j=1}^{i-1} a_j \text{row}_j
  \]

  At step \( i \) rows 1 to \( i - 1 \) are reduced.

  - Reduce in parallel rows \( i \) to \( i + p \) by pivots up to \( i - 1 \)
  
  - Row \( i + \) offset waits until rows from \( i \) to \( i + \) offset \(- 1 \) are done
    \( \Rightarrow \) waiting queue
  
  - Threads share global variable \texttt{last_pivot}

- Blocksize 256 works out best for the shown examples
StarPU implementation

Tiled Gaussian Elimination

```c
struct starpu_data_filter f = {
    .filter_func = starpu_matrix_filter_vertical_block,
    .nchildren = lblocks
};

struct starpu_data_filter f2 = {
    .filter_func = starpu_matrix_filter_block,
    .nchildren = mblocks
};

starpu_data_map_filters(dataA, 2, &f, &f2);
```
StarPU Implementation

Tasks & Dependencies

```c
static int create_task_12(starpu_data_handle_t dataA, unsigned k, unsigned j)
{
    int ret;

    struct starpu_task *task = create_task(TAG12(k, j));
    task->cl = &task_12_cl;

    /* what sub-data is manipulated ? */
    task->handles[0] = starpu_data_get_sub_data(dataA, 2, k, k);
    task->handles[1] = starpu_data_get_sub_data(dataA, 2, k, j);

    if (!no_prio && (j == k+1)) {
        task->priority = STARPU_MAX_PRIO;
    }

    /* enforce dependencies ... */
    if (k > 0) {
        starpu_tag_declare_deps(TAG12(k, j), 2, TAG11(k), TAG22(k-1, k, j));
    } else {
        starpu_tag_declare_deps(TAG12(k, j), 1, TAG11(k))
    }
...
}
```
class TASK_12 : public task {
    public:
        TYPE *a;
        TYPE *b;
        TYPE offset;
        // successors, all TASK_22
        TASK_22 **task_22_succ;

        TASK_12(TYPE *a_, TYPE *b_, TYPE offset);

        task* execute();
};

for (j=k+1; j<mblocks; ++j) {
    task_12_queue[task_12_ctr] = new(task::allocate_root())
        TASK_12(&mat[tile_size*(k+k*m)], &mat[tile_size*(j+k*m)],k);
    task_11_queue[k]->task_12_succ[j-k-1] = task_12_queue[task_12_ctr];
    if (k!=0) {
        task_22_queue[task_22_start_idx+j-k]->task_12_succ =
            task_12_queue[task_12_ctr];
        task_12_queue[task_12_ctr]->set_ref_count(2);
    } else {
        task_12_queue[task_12_ctr]->set_ref_count(1);
    }
    task_12_ctr++;
}
- Fast linear algebra for computing Gröbner bases
- Hybrid Matrix Multiplication
- Next steps on matrix multiplication
- First steps on Gaussian Elimination
- Next steps
Next steps

- Finish Intel TBB implementation of dense Gaussian Elimination and compare to StarPU
- Improve scheduling for \#columns \gg \#rows
- Make these implementations available in LELA resp. MATHICGB
Directly reduce C?
Other tests – CALU 1

Timings: CALU fully-dynamic scheduling w/ MKL
Matrix dimensions: 36000 x 36000, Layout: 8 (B ~ Blocksize)
Computations done on 4 x 8 Intel Xeon E5-4620 @ 2.20GHz NUMA with 4 x 96 GB RAM

![Graph showing real time in seconds vs. number of threads for different MKL versions.](image-url)
Other tests – CALU 2

Timings: CALU fully-dynamic scheduling w/ MKL
Number of threads: 32, Layout: 4 (B ~ Blocksize)
Computations done on 4 x 8 Intel Xeon E5-4620 @ 2.20GHz NUMA with 4 x 96 GB RAM

The graph shows the real time in seconds against the size of rows/cols for different benchmarks and configurations. The y-axis represents the real time in seconds, while the x-axis represents the size of rows/cols.
Other tests – CALU 2

Timings: CALU fully-dynamic scheduling w/ MKL

Number of threads: 32, Layout: 8 (B ~ Blocksize)

Computations done on 4 x 8 Intel Xeon E5-4620 @ 2.20GHz NUMA with 4 x 96 GB RAM
Other tests – CALU 2

Timings: CALU fully-dynamic scheduling w/ MKL

Number of threads: 32, Layout: 16 (B ~ Blocksize)
Computations done on 4 x 8 Intel Xeon E5-4620 @ 2.20GHz NUMA with 4 x 96 GB RAM
Other tests – CALU 2

Timings: CALU fully-dynamic scheduling w/ MKL
Number of threads: 32, Layout: 32 (B ~ Blocksize)
Computations done on 4 x 8 Intel Xeon E5-4620 @ 2.20GHz NUMA with 4 x 96 GB RAM


