How to Make Lanczos-Montgomery Fast on Modern Supercomputers?

Dmitry Zheltkov, Nikolai Zamarashkin, Sergey Matveev

INM RAS

September 26, 2023

RSA-232 factorization

Goals

- To test INM RAS parallel implementation of block Lanczos-Montgomery method for the record size problem.
- To understand the current state and the problems of the modern GNFS implementations.

The experiment

- RSA-232 has 232 decimal digits (768 bits).
- Experiment started on December 2018.
- CADO-NFS used for all the stages but the solution of the linear system over GF(2).
- Factorization obtained by February 17, 2020.

RSA-232 factorization

GNFS stages

Polynomial selection

- Performed on Zhores supercomputer on December 2018.
- Total cost – 1.5 coreyears.

Lattice sieving

- From December 2018 till March 2019.
- 80% on Lomonosov, 20% on Zhores.
- Total cost – equivalent to 550 coreyears on Zhores.
- Total number of collected relations is about 5.7 billions.

RSA-232 factorization

GNFS stages

Filtering

- Performed on June 2019 on Arkuda cluster.
- Used more than 1 TB of RAM.
- Total cost – 0.1 coreyears.
- Final matrix has size about 317 millions and average number of nonzero entries per row equal to 170.

Linear system solution

- From December 2019 till January 2020 on Zhores supercomputer.
- 20 to 40 node used, solution obtained in 38 days.
- Total cost – 62 coreyears.

RSA-232 factorization

GNFS stages

Square root computation

- Performed on February 2020 on n *m1-megamem-96* instance of Google Cloud service.
- More than 200 GB of RAM per root.
- Total cost – 0.1 coreyears.

RSA-232 factorization

Details of block Lanczos-Montgomery performance

Impact of different operations

- 90% of total time (56 coreyears) spent on sparse matrix by block products.
  - 79% of that time (44.5 coreyears) spent on computations.
  - 21% of that time (11.5 coreyears) spent on communications.
- 10% of total time (6.2 coreyears) spent on dense operations.
  - Amount of time spent on communications is negligibly small.

Question

Is block Lanczos-Montgomery performance good enough?

Block Lanczos-Montgomery performance

vs block Wiedemann-Coppersmith

Comparison to block Wiedemann-Coppersmith algorithm

- During factorization of RSA-240 similar matrix was obtained.
- Size about 282 millions, 200 nonzeros per row on average.
- Complexity estimate about 20% less.
- Computational cost – 83 coreyears on similar hardware.
- About 30% of time (25 coreyears) spent on communications.

Block Lanczos-Montgomery performance

Analysis

Communications

- Theoretical estimate of data transfers is $\frac{4N}{s_r s_c p^2}((p-1)N + p(s_r-1)N + p(s_c-1)N)$ bits per node, where N – matrix size and $ps_r s_c$ – total number of nodes.
- Zhores has communication network with 200 Gbps bidirectional bandwidth.
- Total cost estimate – 10.7 coreyears.

Block Lanczos-Montgomery performance

Analysis

Dense operations

- Theoretical estimate of complexity is $N\left(12\frac{N}{s_r s_c}+25p^2 b\right) b$ bit operations per node if naive algorithm is used, where b is block size in bits.
- Each core of Zhores cluster has theoretical peak performance $6 \cdot 10^{12}$ bit operations per second.
- Estimate of total cost is 3.2 coreyears for naive algorithm.
- Coppersmith and "4 russians" methods should provide at least 1.5–3 times speedup.

Block Lanczos-Montgomery performance

Sparse matrix by block product

- Theoretical upper estimate of complexity for CSR format is $\frac{(18+17\rho)N^2}{64ps_rs_c}$ bytes of memory transfers per node.
- Each node of Zhores cluster has theoretical peak memory bandwidth $238 \cdot 10^9$ bytes per second.
- Estimate of total cost is 13.5 coreyears.

Possible sources of problems

- ~~Non optimal block size.~~
- TLB misses.

How to make Block Lanczos-Montgomery fast?

General ideas

- General implementation improvements.
  - Complexity reduction for symmetric operations.
  - Asynchronous data transfers, dense computations and sparse matrix by block products.
- Improvement of operation implementations.
  - Faster dense operations.
  - Faster and less memory consuming sparse matrix by block products.
- Use of GPUs for sparse and dense operations.

How to make Block Lanczos-Montgomery fast?

Faster dense operations

“4 russians” method

Assume computation of $C = AB$, $A, B, C \in \mathbb{F}_2^{n\times n}$.

- Split product: $C = A_1B_1 + \dots + A_mB_m$, $A_i \in \mathbb{F}_2^{n\times k}$, $B_i \in \mathbb{F}_2^{k\times n}$
- For B_i compute matrix $T_i \in \mathbb{F}^{2^k\times n}$ of all possible linear combinations of its rows.
- With correct ordering of T_i rows result of product of j-th row of A_i by B_i is row of T_i with number, whose binary representation is j-th row of A_i.

How to make Block Lanczos-Montgomery fast?

Faster dense operations

How to implement “4 russians” method

- Theoretically optimal $k = \log_2 n$, but it doesn’t take in account cache hierarchy. In practice T_i should fit L1 cache.
- From the performance point of view n should be multiple of 512 for the modern processors.
- Should be several tables T_i at the same time in L1 cache.
- Memory accesses should be handled with care.
- Use modern vector extensions.

New implementation

- Uses $k = 4$, $n = 512$ and 16 tables in L1 cache at a time.
- Uses AVX-512.
- 5–6 times faster.

How to make Block Lanczos-Montgomery fast?

Faster dense operations

How to implement “4 russians” method on GPUs

- Ideas are similar to CPU.
- It is convenient to use n which is multiple of 1024.
- Table should store at shared memory, one table for a computational block is enough.
- Blocks are performed by single warp to avoid unnecessary synchronizations.
- Memory accesses should be handled with care.

New implementation

- $k = 4$, $n = 1024$ and single table in shared memory.
- Performance ratio to naive algorithm is worse than for CPU, but overall performance is much higher.

How to make Block Lanczos-Montgomery fast?

Faster sparse operations

CSR problems

- Up to 2 times slower for multiplication of transposed matrix.
- Memory usage scalability is not so good.
- A lot of data and TLB cache misses.

Suggestion

- Split matrix to blocks of size such that input and output blocks fit L2 cache (2^{13}–2^{15}) for the modern processors.
- Each block stored in COO format but using only 2 bytes for each index.
- Multiply one block row after another.

How to make Block Lanczos-Montgomery fast?

Faster sparse operations

New implementation

- Uses the same or less than CSR for the record size matrices.
- Memory usage scales very well.
- Better cache usage and small amount of TLB misses.
- Performance for multiplication of transposed matrix is the same.
- Achieved performance a bit better than theoretical estimate for CSR.

How to make Block Lanczos-Montgomery fast?

Faster sparse operations on GPU

Suggestion

- Use CSR and block size of 1024 bit.
- Store transposed matrix for fast operation.

Advantages

- Achieved performance a bit better than estimate for CSR.
- Memory bandwidth for GPUs is much higher, so performance on single GPU is up to 10 times faster than on single node of Zhores cluster.

Drawbacks

- High memory usage requires to use a lot of GPUs for record size problems and decreases overall efficiency of the method.

How fast is new block Lanczos-Montgomery implementation?

- Expected total cost on Zhores cluster is less than 20 coreyears – less than 2 weeks on 20 nodes.
- GPU version is faster in order of magnitude using Tesla A100 accelerators.

Thank you!