GPU Acceleration of Dense Matrix And Block Operations for Lanczos Method for Systems over Large Prime Finite Field

D. Zheltkov, N. Zamarashkin

INM RAS

September 25, 2017

Scalability of Lanczos method

Consider $N \times N$ sparse system with average ρ nonzero elements per row over finite field with prime number consisting of W machine words. Such system could be solved using block Lanczos method with block size k on $p = ks$ computer nodes.

The method constist of 3 types of operations:

- Sparse matrix by block multiplication. Time — $O(\frac{\rho W N^2}{ks})$.
- Dense operations. Time — $O(\frac{W^2 N^2}{s} + \frac{W^2 k N}{s})$.
- Communication. Time — $O(\frac{WN^2}{k} + \frac{WN^2}{ks} + WNk)$.

While dense operations are fast enough method scales *almost perfectly.*

Scalability of Lanczos method

Consider $N \times N$ sparse system with average ρ nonzero elements per row over finite field with prime number consisting of W machine words. Such system could be solved using block Lanczos method with block size k on $p = ks$ computer nodes.

The method constist of 3 types of operations:

- Sparse matrix by block multiplication. Time — $O(\frac{\rho W N^2}{ks})$.
- Dense operations. Time — $O(\frac{W^2 N^2}{s} + \frac{W^2 k N}{s})$.
- Communication. Time — $O(\frac{WN^2}{k} + \frac{WN^2}{ks} + WNk)$.

While dense operations are fast enough method scales *almost perfectly*.

CPU problems

CPU are not very effective on operations over large field in comparizon with floating point operations:

- No instruction for fused multiply-add with carry.
- No appropriate vector instructions.

Rate of needed elementary integer operations is at least **32 times lower** than for single precision floating point operations.

CPU problems

CPU are not very effective on operations over large field in comparizon with floating point operations:

- No instruction for fused multiply-add with carry.
- No appropriate vector instructions.

Rate of needed elementary integer operations is at least **32 times lower** than for single precision floating point operations.

GPU advantages and problems

Advantage:

- Instruction for fused multiply-add with carry (*madc*).

Disadvantages:

- Only 32-bit version of *madc* operation (so, 4 time more elemental operations is needed for large number multiplication).
- Several (2 to 6) clocks are needed to perform *madc* operation.
- Limited register resource.

Overall, even with the same floating point performance GPU must be **several times faster** than CPU on dense operations over large field.

GPU advantages and problems

Advantage:

- Instruction for fused multiply-add with carry (*madc*).

Disadvantages:

- Only 32-bit version of *madc* operation (so, 4 time more elemental operations is needed for large number multiplication).
- Several (2 to 6) clocks are needed to perform *madc* operation.
- Limited register resource.

Overall, even with the same floating point performance GPU must be **several times faster** than CPU on dense operations over large field.

GPU architecture

GPU consist of several computational devices name *streaming multiprocessor (SM)*.
Each *SM* have access to *global* GPU memory and small amount of local memory.
Each *SM* consist of $32 - 128$ very simple *streaming processors (SP)*.
SPs are united in groups of 32 performing the same operation — *warps*.

Programming *blocks* consisting of *threads* are loaded to *SM*.
Total number of *blocks* loaded to *SM* depends of resources used by *block*.
Total number of threads on *SM* could be significally larger than number of *block*.
Blocks are dynamically scheduled to compensate instruction and data load.

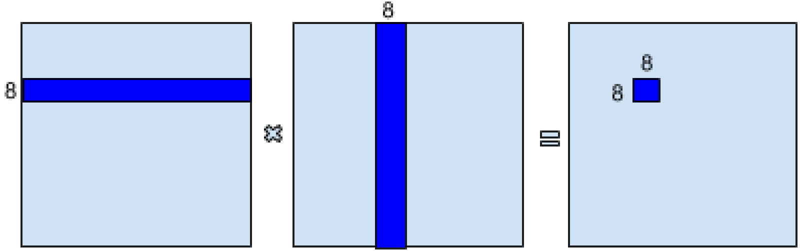
GPU resource limitation

Total number of threads loaded to *SM* is limited by several limitations:

- Total number of threads.
- Number of blocks,
- Amount of used *shared* memory.
- Number of used *registers*.

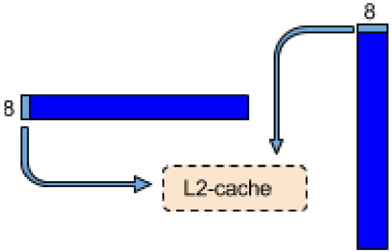
Computational scheme of matrix multiplication

Calculation in one block



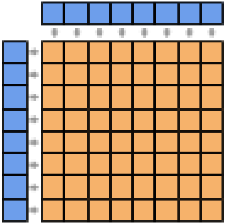
Computational scheme of matrix multiplication

Data load in naive algorithm



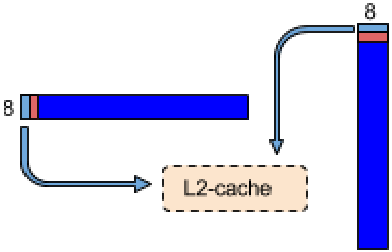
Computational scheme of matrix multiplication

Data handle



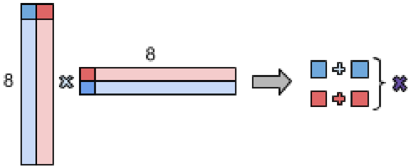
Computational scheme of matrix multiplication

Data load in Winograd algorithm



Computational scheme of matrix multiplication

Winograd computation



Winograd method for LU decomposition

Two steps of elimination:

$$A \to A - \begin{bmatrix} L_{11} \\ A_{21} U_{11}^{-1} \end{bmatrix} \begin{bmatrix} U_{11} & L_{11}^{-1} A_{12} \end{bmatrix} = A - \begin{bmatrix} L_{11} \\ \hat{A}_{21} \end{bmatrix} \begin{bmatrix} U_{11} & \hat{A}_{12} \end{bmatrix},$$

Consider matrix product:

$$C = \begin{bmatrix} A_1 & A_2 \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \end{bmatrix},$$

So

$$C_j^i = (a_{i1} + b_{j2})(a_{i2} + b_{j1}) - a_{i1} a_{i2} - b_{j1} b_{j2},$$

Numerical results

512-bit prime

Table: Naive algorithm time (sec.)

Device	C2070	K40	GTX1050
$2^{21} \times 8$	0.35	0.28	0.41
$2^{21} \times 16$	1.31	0.89	1.56
1024×1024	2.38	1.57	2.53

Table: Winograd algorithm time (sec.)

Device	C2070	K40	GTX1050	i5-4440
$2^{21} \times 8$	0.26	0.19	0.28	3.98
$2^{21} \times 16$	0.89	0.6	0.95	13.41
1024×1024	1.48	0.91	1.42	20.92

Numerical results

768-bit prime

Table: Naive algorithm time (sec.)

Device	C2070	K40	GTX1050
$2^{21} \times 8$	0.85	0.58	1.15
$2^{21} \times 16$	3.1	2	3.88
1024×1024	5.75	3.67	6.49

Table: Winograd algorithm time (sec.)

Device	C2070	K40	GTX1050	i5-4440
$2^{21} \times 8$	0.8	0.63	1	7.24
$2^{21} \times 16$	2.86	2.07	3.5	24.28
1024×1024	5.38	3.31	5.52	39.23

Numerical results

1024-bit prime

Table: Naive algorithm time (sec.)

Device	C2070	K40	GTX1050
$2^{21} \times 8$	2.04	1.06	2.83
$2^{21} \times 16$	7.59	3.91	10.82
1024×1024	13.55	7.16	18.11

Table: Winograd algorithm time (sec.)

Device	C2070	K40	GTX1050	i5-4440
$2^{21} \times 8$	1.53	1.07	2.12	12.6
$2^{21} \times 16$	5.47	3.57	6.8	54.97
1024×1024	9.37	5.77	9.74	69