Parallel solution of linear systems and optimal multiplication of polynomials over GF(2)

I.V. Oseledets

Institute of Numerical Mathematics, Moscow

15 September 2008.

Linear system

$$Bx = f.$$

Linear system

$$Bx = f.$$

B is sparse.

Linear system

$$Bx = f.$$

B is sparse.

Known methods

- Conjugate gradients
- GMRES
- Lanczos

Number of steps: $N_{\mathrm{iter}} N$, usually we do not allow $N_{\mathrm{iter}} = N$

Linear system

$$Bx = f.$$

B is sparse.

Finite fields

$$GF(2) = \{0, 1\}, \text{ no norm:}$$
$$x = (1, 1, 1, 1)^\top, \|x\| = 0$$

Number of iterations is $\mathcal{O}(N)$. And if $N = 10^8$?

We focus on GF(2) — no stability issues

Can use any "unstable" algorithm.

Parallel solution of linear systems, Toeplitz matrices and polynomial multiplication

We focus on GF(2) — no stability issues

Can use any "unstable" algorithm.

Parallel solution of linear systems, Toeplitz matrices and polynomial multiplication

Wiedemann algorithm

Minimal polynomial: $\sum_k c_k B^k = 0$,

Instead: two vectors x, y and solve for

$\sum_j (x^\top B^{i+j} y) c_j = 0$).

Hankel system! If found, c_j — coefficient of minimal polynomial of B

Wiedemann algorithm

Minimal polynomial: $\sum_k c_k B^k = 0$,
Instead: two vectors x, y and solve for
$\sum_j (x^\top B^{i+j} y) c_j = 0)$.
Hankel system! If found, c_j — coefficient of minimal polynomial of B

For parallel computations — block version

Coppersmith algorithm

$$a_i = X^\top B^i Y, \quad i = 0, \dots,$$

Kernel of block Hankel matrix:

$$\sum_j a_{i+j} c_j = 0.$$

For parallel computations — block version

Coppersmith algorithm

$$a_i = X^\top B^i Y, \quad i = 0, \ldots,$$

Kernel of block Hankel matrix:

$$\sum_j a_{i+j} c_j = 0.$$

Simple justification:

$$(X^\top B^{i-1}) \sum_{j=0} B^{j+1} Y c_j = 0,$$

If block Krylov subspace $X^\top B^i$ has full dimension, then

$$Bw = 0, \quad w = \sum_{j=0} B^j Y c_j.$$

Simple justification:

$$(X^\top B^{i-1}) \sum_{j=0} B^{j+1} Y c_j = 0,$$

If block Krylov subspace $X^\top B^i$ has full dimension, then

$$Bw = 0, \quad w = \sum_{j=0} B^j Y c_j.$$

Many interesting questions ...

- Dimensions of binary block Krylov subspaces
- Matrix-by-vector product efficiently
- Kernel of the block Hankel matrix

We will focus only on the last

Block Hankel-kernel

$$\sum_j A_{i+j} c_j = 0,$$

A_i is $q \times q$, usually $q = 512, 1024$.

GF(2): Challenge for Toeplitz-Hankel specialists!

$O(N \log^\alpha N)$ methods are hard to find.

Block Hankel-kernel

$$\sum_j A_{i+j} c_j = 0,$$

A_i is $q \times q$, usually $q = 512, 1024$.

GF(2): Challenge for Toeplitz-Hankel specialists!
$O(N \log^\alpha N)$ methods are hard to find.

- No Fourier transform
- No nonsingularity of leading submatrices (even for HH^*)

Division-free algorithms

- D.Coppersmith, 1994 — $\mathcal{O}(N^2)$
- E. Thome, Fast computations of linear generators for matrix sequences 2001 — $\mathcal{O}(M(n)\log n)$,

$M(n)$ — time to multiply two matrix polynomials with $q \times q$ matrix coefficients.

That is what we are going to discuss

Classical polynomial multiplication problem:

Main problem

$$c(x) = a(x)b(x),$$
$$a(x) = \sum_{i=0}^{n-1} a_i x^i, \quad b(x) = \sum_{i=0}^{n-1} b_i x^i.$$

Coefficients can be complex, integer, matrices ...

Main problem

$$c(x) = a(x)b(x),$$
$$a(x) = \sum_{i=0}^{n-1} a_i x^i, \quad b(x) = \sum_{i=0}^{n-1} b_i x^i.$$

Schoolbook: n^2 multiplications, $n(n-1)/2$ additions.

Usually
multiplication time is much large addition time.

Main problem

$$c(x) = a(x)b(x),$$
$$a(x) = \sum_{i=0}^{n-1} a_i x^i, \quad b(x) = \sum_{i=0}^{n-1} b_i x^i.$$

You can ask: What is unusual here?

Fourier: interpolation

$$\text{«Forward» step: } c(w^j) = a(w^j)b(w^j), \quad j = 0, \ldots, 2n-1$$
$$w = e^{\frac{2\pi i}{2n-1}}$$

«Backward» step: Inverse Fourier transform.

Computational complexity

$O(n \log n)$ operations.

Main problem

$$c(x) = a(x)b(x),$$
$$a(x) = \sum_{i=0}^{n-1} a_i x^i, \quad b(x) = \sum_{i=0}^{n-1} b_i x^i.$$

It work only in complex field.
And what in GF(2), only two elements available?

- There are $2n-1$ roots of unity — Fourier.
- There are $2n-1$ different elements — Toom-Cook, $a(i) = b(i)c(i), i = 0, ..., \ldots 2n-2$. Division is needed.
- Karatsuba-type algorithms.

Main problem

$$c(x) = a(x)b(x),$$
$$a(x) = \sum_{i=0}^{n-1} a_i x^i, \quad b(x) = \sum_{i=0}^{n-1} b_i x^i.$$

The most interesting case — field with two elements, $0-1$, which we will study.

Please, remember that

$$a - b = a + b.$$

Main problem

$$c(x) = a(x)b(x),$$
$$a(x) = \sum_{i=0}^{n-1} a_i x^i, \quad b(x) = \sum_{i=0}^{n-1} b_i x^i.$$

The most interesting case — field with two elements, $0-1$, which we will study.

Please, remember that

$$a - b = a + b.$$

Karatsuba algorithm

$$a(x) = a_0 + a_1 x, \quad b(x) = b_0 + b_1 x,$$
$$p_0 = a_0 b_0, \quad p_1 = (a_0 + b_0)(a_1 + b_1), p_2 = a_1 b_1,$$
$$c_0 = p_0, \quad c_1 = p_0 + p_1 + p_2, \quad c_2 = p_2.$$

Karatsuba algorithm

$$a(x) = a_0 + a_1 x, \quad b(x) = b_0 + b_1 x,$$
$$p_0 = a_0 b_0, \quad p_1 = (a_0 + b_0)(a_1 + b_1), p_2 = a_1 b_1,$$
$$c_0 = p_0, \quad c_1 = p_0 + p_1 + p_2, \quad c_2 = p_2.$$

Requires 3 multiplications and 4 additions.

Karatsuba algorithm

$$a(x) = a_0 + a_1 x, \quad b(x) = b_0 + b_1 x,$$
$$p_0 = a_0 b_0, \quad p_1 = (a_0 + b_0)(a_1 + b_1), p_2 = a_1 b_1,$$
$$c_0 = p_0, \quad c_1 = p_0 + p_1 + p_2, \quad c_2 = p_2.$$

Requires 3 multiplications and 4 additions.

What for?

Recursive application: $O(n^{\log_2 3}) \approx O(n^{1.58})$

Matrix polynomials: a_i, b_i — binary matrices

Then multiplication time is thousand times more than addition time

What for?

Recursive application: $O(n^{\log_2 3}) \approx O(n^{1.58})$

Matrix polynomials: a_i, b_i — binary matrices

Then multiplication time is thousand times more than addition time

We will talk about algorithm with minimal number of multiplication over GF(2).

There are no roots of 1, no divisions, we can not even divide by 2!($a + a = 0$).

As an application, we consider:

Matrix polynomial multiplication

$c(x) = a(x)b(x)$, Coefficients a_i — bit matrices of size, say 512×512.

Karatsuba-like algorithms for high degrees

$$n = 3 — 6 \text{ multiplications},$$
$$n = 4 — 9 \text{ multiplications},$$
$$n = 5 — 13 \text{ multiplications},$$
$$n = 6 — 17 \text{ multiplications}.$$

These are results from year 2005!

Montogomery P.L., Five, six and seven Karatsuba-like formulae, IEEE Trans on Computers.

What to say about practical algorithms for the multiplication of polynomials of degree, for example, 100.

- A general approach was obtained for the construction of algorithms with minimal number of multiplications.
- Code generator was written (for $n = 128$ the program length is $\sim$ 25000 lines
  It is faster (10 times) than recursive Karatsuba.

Let us give main ideas.

General scheme for bilinear algorithms

Output vector: c_i, length $2n-1$

Input vector: a_i, b_i length n.

General scheme for bilinear algorithms

Output vector: c_i, length $2n-1$
Input vector: a_i, b_i length n.

All fast algorithms has the form

$$c = V\left((Ua) \circ (Ub)\right),$$

U, V — matrices of size $r \times n$ and $m \times r$ respectively,
$\circ$ — elementwise product of vectors.
r — rank of the algorithm (minimal number of multiplications).

All fast algorithms has the form

$$c = V\left((Ua) \circ (Ub)\right),$$

It should be an identity: setting $a = e_i, \quad b = e_j$:

Trilinear decomposition:

$$C_{ijk} = (x^i x^j)_k = \delta_{(i+j)k} = \sum_{\alpha=1}^{r} u_{i\alpha} u_{j\alpha} w_{k\alpha}.$$

It should be an identity: setting $a = e_i, \quad b = e_j$:

Trilinear decomposition:

$$C_{ijk} = (x^i x^j)_k = \delta_{(i+j)k} = \sum_{\alpha=1}^{r} u_{i\alpha} u_{j\alpha} w_{k\alpha}.$$

Three-dimensional tensor is conveniently represented as a set of $2n-1$ matrices:

$$C_1, C_2, \ldots, C_{2n-1}.$$

It is easy to see, that

Equivalent formulation

$$C_k = \sum_{\alpha=1}^{r} w_{k\alpha} R_{\alpha},$$

$R_\alpha = u_\alpha u_\alpha^\top$ — symmetric rank-1 matrices

Equivalent formulation

$$C_k = \sum_{\alpha=1}^{r} w_{k\alpha} R_\alpha,$$

$$R_\alpha = u_\alpha u_\alpha^\top \text{ — symmetric rank-1 matrices}$$

For given matrices $C_1, \ldots, C_{2n-1}$ find rank-1 matrices such that

$$\mathrm{Span}(C_i) \in \mathrm{Span}(R_s).$$

Equivalent formulation

$$C_k = \sum_{\alpha=1}^{r} w_{k\alpha} R_\alpha,$$

$R_\alpha = u_\alpha u_\alpha^\top$ — symmetric rank-1 matrices

For given matrices $C_1, \ldots, C_{2n-1}$ find rank-1 matrices such that

$$\mathrm{Span}(C_i) \in \mathrm{Span}(R_s).$$

For polynomials $\mathrm{Span}(C_i)$ is a space of all Hankel matrices,

$$C = [c_{i+j}]$$

For polynomials $\mathtt{Span}(C_i)$ is a space of all Hankel matrices,

$$C = [c_{i+j}]$$

For complex or real number the answer is known and easy:

$$r = 2n - 1,$$

because there are $2n - 1$ linearly independent matrices of rank 1:

$$(H_k)_{i+j} = \rho_k^{i+j},$$

ρ_k — different nodes.

In GF(2) situation is diffent: Too few rank-1 matrices

Theorem

There are 3 matrices of rank 1, 5 matrices of 2, 9 matrices of 3, $\dots 2^k + 1$ matrices of k, such that they are linearly independent alltogether.

Corollary: $M(n) = \mathcal{O}(n \log n)$, but with a very big constant: If $\alpha = \log_n M(n)$ then $\alpha = 1.01$ when

$$n \sim 10^{334},$$

i.e. estimate is purely theoretic.

$$C_k = \sum_{\alpha=1}^{r} w_{k\alpha} R_{\alpha},$$

Finite number of variants — exhaustive search!
For example: $n = 3$ there are only $2^3 - 1 = 7$ different rank-1 matrices,
You have to select 6, only 7 diffent sets R_α

$$C_k = \sum_{\alpha=1}^{r} w_{k\alpha} R_\alpha,$$

Finite number of variants — exhaustive search!
For example: $n = 3$ there are only $2^3 - 1 = 7$ different rank-1 matrices,
You have to select 6, only 7 diffent sets R_α

- For $n = 4$ — 5005 variants.
- For $n = 5$ — 206 253 075 variants.

For small n — exhaustive search
For large — general theory of low-rank Hankel matrices.

For small n — exhaustive search

For large — general theory of low-rank Hankel matrices.

Hankel matrix $H = [h_{i+j}]$ has rank r, when its generating vector h satisfies a short recurrence relationship of order k:

$$h_{i+r} = \sum_{s=0}^{r-1} \alpha_s h_{i+s}.$$

Polynomial $\sum_{s=0}^{r-1} \alpha_s x^s$ is called generating — it plays a key role.

In the end, everything is reduced to the right selection of polynomials.

Another intepretation

We select some set of polynomials $p_1, \dots, p_s$, and compute

$$a(x)b(x) \mod p_i,$$

then reconstruct everything back using Chinese remainder theorem.

You can loose one or two multiplications:
For $n = 5$ optimal CRT method gives 14 multiplications (not 13)
For $n = 6$ — 18 multiplications (not 17),

These numbers can be obtained by our exhaustive search method.

Right selection of polynomials set — difficult problem.
It is solved by integer programming (long, but one time).

Right selection of polynomials set — difficult problem.
It is solved by integer programming (long, but one time).

n	M_{old}	M_{new}
2	3	3
3	6	6
4	9	9
5	13	13
6	17	17
7	22	22
8	26	26
9	31	30
16	64	62
34	243	159
128	2187	749
1000	~ 50000	~ 3000

After the construction of U, V matrices nothing is finished.

What about additions?

For $n = 128$ matrix V has size 128×749 and contains approximately 50000 nonzeros, i.e. 50000 additions are needed.

After the construction of U, V matrices nothing is finished.

What about additions?

For $n = 128$ matrix V has size 128×749 and contains approximately 50000 nonzeros, i.e. 50000 additions are needed.

Fast multiplication by a given bit-matrix:
As an input — matrix, on the output — program to multiply on it.

Result

Instead of 50000 additions around 5000 additions, but tens of thousand auxilaury variables.

Then we reduce the number of auxilaury variables with the help of
Program graph decomposition:
Instead of tens of thousands — 500 auxilaury variables.

Fast multiplication by a given bit-matrix:
As an input — matrix, on the output — program to multiply on it.

Result

Instead of 50000 additions around 5000 additions, but tens of thousand auxilaury variables.

Then we reduce the number of auxilaury variables with the help of
Program graph decomposition:
Instead of tens of thousands — 500 auxilaury variables.

These are analogues of two known "compiler" techniques:

- Common expression elimination
- Register allocation via graph coloring

Both of them for these problem are inefficient

Everything consists from parts:

Summary

- Optimal algorithms for small polynomials
- Selecting of polynomial set for large n
- Optimizing multiplication by U and V («Fourier without Fourier»)
- Program graph decomposition

As a result — 10 times faster for $n = 128$.

Not only in theory, but in practice.

- I.V. Oseledets, Optimal Karatsuba-like formulae for certain bilinear forms in GF(2), Linear Algebra Appl. 2008
- I.V. Oseledets, Improved n-term Karatsuba-like formulae, IEEE Trans. Comp., submitted (2008)

- I.V. Oseledets, Optimal Karatsuba-like formulae for certain bilinear forms in GF(2), Linear Algebra Appl. 2008
- I.V. Oseledets, Improved n-term Karatsuba-like formulae, IEEE Trans. Comp., submitted (2008)

Papers can be obtained from the author or from
http://spring.inm.ras.ru/osel
Thank you! Questions?