Parallelizing Explicit Formulae for Arithmetic in the Jacobian of Hyperelliptic Curves

P. K. Mishra and P. Sarkar

Cryptology Research Group
Indian Statistical Institute
e-mail: {*pradeep_t, palash*}*@isical.ac.in*

What is an Explicit Formula?

Koblitz proposed Cantor's Algorithm to add two divisors in the Jacobian of a hyperelliptic curve.

Spallek (1994) was first to propose an alternative: Fix the genus of the curve and find out the parameters of the resultant divisor using the parameters of the input divisors.

Since then many explicit formulae have been resulted for hyperelliptic curves of various genus due to extensive research in many parts of the globe.

Efficiency of Explicit Formulae

Table 1: Complexity of Explicit Formulae in Genus 2

Algorithm	Operation	Cost
Cantor (1989)	Add	$3[i] + 70[m/s]$
	Dbl	$3[i] + 76[m/s]$
Lange(2002)	Add	$1[i] + 22[m] + 3[s]$
	Dbl	$1[i] + 22[m] + 5[s]$
Lange (2002)	Add	$40[m] + 6[s]$
	Dbl	$47[m] + 4[s]$

Our Contributions

A general methodology for parallelizing any explicit formula.

Algorithms so obtained are provably optimal in #multiplication rounds.

Application to Lange's explicit formula for genus 2 hyperelliptic curves.

Application to encapsulated add-and-double algorithm.

Design of parallel algorithms using 4, 8 and 12 multipliers.

A concequence: parallel version of inversion free arithmetic is more efficient than the parallel version of arithmetic using affine coordinates.

General Methodology

Parallellizing the multiplications only suffices.

Preprocessing:

1. *Convert all operations to binary* :
   For example,
   $p_5 = p_1p_2p_3$ is rewritten as $p_4 = p_1p_2$ and
   $p_5 = p_3p_4$.
2. *Reduce multiplication depth* : For example:
   $p_3 = p_1^2p_2;\ p_4 = p_3p_2$. can be done as
   $t_1 = p_1^2;\ p_3 = t_1p_2;\ p_4 = p_3p_2$. or
   $\{t_1 = p_1^2;\ t_2 = p_2^2;\}\{p_3 = t_1p_2;\ p_4 = t_1t_2\}$.
3. *Eliminate reuse of variable names* :
   For example:
   $q_1 = p_1 + p_2; q_2 = p_3; \ldots; q_1 = p_4 + p_5; \ldots$
4. *Labeling each operation with unique label* :
   like DM1: $t_1 = p_1q_2$.

General Methodology (contd..)

Given a formula $\mathcal{F}$, we define a DAG $G(\mathcal{F})$ as follows:

- The nodes of $G(\mathcal{F})$: the arithmetic operations and variables of $\mathcal{F}$. Also nodes for the parameters of the input divisor(s) and the curve.
- Suppose $\mathbf{id}$:$\mathbf{r} = \mathbf{qp}$ is a multiplication operation. Then the following arcs are present in $G(\mathcal{F})$: $(q, \mathbf{id}), (\mathbf{p}, \mathbf{id})$ and $(\mathbf{id}, \mathbf{r})$.

Facts about $G(\mathcal{F})$

- The indegree of variable nodes corresponding to the parameters of the input divisors and the parameters of the curve is zero.

- The indegree of any node corresponding to an intermediate variable is one.

- The outdegree of any node corresponding to an addition or multiplication operation is one.

Note that the outdegree of nodes corresponding to variables can be greater than one.

The Graph $G^*(\mathcal{F})$

Define a DAG $G^*(\mathcal{F})$ from $G(\mathcal{F})$ as

- The nodes of $G^*(\mathcal{F})$ are the nodes of $G(\mathcal{F})$ which correspond to multiplication operation.
- An arc from node $\mathbf{id_1}$ to node $\mathbf{id_2}$ in $G^*(\mathcal{F})$ only if there is a path from $\mathbf{id_1}$ to $\mathbf{id_2}$ in $G(\mathcal{F})$ which does not pass through another multiplication node.

The graph $G^*(\mathcal{F})$ captures the ordering relation between the multiplication operations of $\mathcal{F}$. An arc $(\mathbf{id_1} \to \mathbf{id_2})$ in $G^*(\mathcal{F})$, then $\mathbf{id_1}$ must be done before $\mathbf{id_2}$.

Parallel Rounds of $\mathcal{F}$

We partition the vertex set of $G^*(\mathcal{F})$ as follows:

- $G_1(\mathcal{F}) = G^*(\mathcal{F})$ and M_1 is the set of nodes of G_1 whose indegree is zero.
- For $i \geq 2$, G_i is the graph obtained from G_{i-1} by deleting the set M_{i-1} from G_{i-1} and M_i is the set of nodes of G_i whose indegree is zero.

Let r be the least positive integer such that G_{r+1} is the empty graph.

All the multiplications in any M_i can be performed in parallel.

Intermediate Variables

For $1 \leq i \leq r$, we define:

1. I_{2i-1} contains intermediate variables which are the inputs to the multiplication nodes in M_i.
2. I_{2i} contains intermediate variables which are the inputs to the addition nodes in A_i.
3. O_{2i-1} contains intermediate variables which are the outputs of the multiplication nodes in M_i.
4. O_{2i} contains intermediate variables which are the outputs of the addition nodes in A_i.

For $1 \leq j \leq 2r$, define

$$V_j = (\cup_{i=1}^{j} O_i) \cap (\cup_{i=j+1}^{2r} I_i). \tag{1}$$

V_j contains the intermediate variables to be stored after the jth round.

Parallel version does not need more storage for intermediate variables than the sequential algorithm.

Some Terminology

We define:

Multiplication Width (MW): $\mathrm{Max}_{1 \leq i \leq r} M_i$

Buffer Width (BW): $\mathrm{Max}_{1 \leq j \leq 2r} V_j$

Critical Path Length (CPL): The length of a path from some node in M_1 to some node in M_r.

Total Number of Multiplications (TM): Total number of multiplications in the explicit formula.

An explicit formula with $\mathsf{CPL} = r$ can not be executed in less than r rounds.

Application to HECC

Inversion-free Arithmetic

Table 2: Parameters for parallel versions of explicit formula.

	MW	BW	CPL	TM
Add	8	20	8	59
Double	11	15	8	65

Both of addition and doubling can be performed in 8 parallel rounds of computations. For addition 8 multipliers are required and for doubling 11.

Arithmetic in affine Co-ordinates

Arithmetic in affine co-ordinates involve one inversion for each of addition and multiplication.

Table 3: Parameters for parallel versions of explicit formula.

	MW	BW	CPL	TM
Add	6	12	7*	29*
Double	5	13	8*	34*

* Including one inversion.

Encapsulated-Add-and-Double

Computing the scalar multiplication using Encapsulated-Add-and-Double formula is secure against Side-Channel attacks. We found parallel versions for different number of multipliers.

Table 4: Summary of algorithms with varying number of multipliers for inversion free arithmetic of genus 2 curves.

No of Multipliers	2	4	8	12
Number of rounds	54	27	14	10

Above formula has $\mathsf{TM} = 108$.
A parallel algorithm is "best" if using r multipliers if takes $\lceil \mathsf{TM}/r \rceil$ parallel rounds.
Almost all the above algorithms are best.

Future/Ongoing Work:

- Implementation.
- Optimising the buffer width.
- Obtaining bounds on buffer requirement.

Thank you !