Spec for Zexe Algorithmic/Performance Optimisations

- Batch Affine Ops with Montgomery's inversion trick
- Batch Subgroup checks with random buckets
- GLV
- w-NAF
- GPU tabled GLV approach

Elliptic Curve Batch Affine Operations

The affine formulas for group addition and doubling on short weierstrass curves involves an inversion. Taking the case of addition, the result of adding $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$, to obtain $P_3 = (x_3, y_3)$ can be expressed as follows:

$$\begin{aligned} \lambda &= \frac{y_2 - y_1}{x_2 - x_1} \\ x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned}$$

To perform the inversion efficiently, we can utilise the Montgomery trick. The general strategy is as follows:

Given a set of non-zero field elements $\{a_i\}_{i=0}^{n-1}$, compute

$$\begin{aligned} \left\{ b_i = \prod_{j=0}^{i} a_j \right\}_{i=0}^{n-1} \\ i = b_{n-1}^{-1} \\ c_{n-1} = i, c_i = c_{i+1} \cdot a_{i+1} \\ a_i^{-1} = c_i \cdot b_{i-1}, i > 0, a_0^{-1} = c_0 \end{aligned}$$

Batch Subgroup Checks

Consider the following procedure: given M buckets and N curve elements that we want to check is in the subgroup, where $M \ll N$. We randomly assign each element a bucket and add it to the bucket, wherein each bucket initially starts with the point at infinity. We then check each of the M buckets using the mul by group order check, a deterministic test.

Consider an element not in the subgroup. The only way adding it to a given bucket will result in the procedure returning `ACCEPT` is if that the buckets excluding that element has the following state: the non-subgroup projection of the particular bucket it is being added is the inverse of the non-subgroup projection of the element, and every other bucket is in the subgroup. For instance, if each of the non-subgroup parts is 1 in a subgroup of order 2. Since this fact is true for every element, the probability of the adversary passing the test given that at least 1 element is not in the subgroup is at most $1/M$.

Then, repeating this procedure over multiple rounds, we can obtain any constant error probability.

In our case, we set the extremely conservative 2^{-128} error upper bound. Realistically, this could be lowered to something like 2^{-32} since the randomness is chosen by the verifier.

Implementation of Batched Bucket Additions

The objective of this optimisation is to identify an addition tree of independent elliptic curve group additions for each bucket, and to batch the independent additions using the batch affine inversion method described above.

The strategy taken is to sort a list of bucket assignments of all the elements (which we can for most intents and purposes, think of as being uniformly random) by bucket, so that indices corresponding to elements that must be added together are physically collocated in memory. Then, in the first round, we proceed to perform independent additions producing intermediate results at the greatest depth for each addition tree (each corresponding to a bucket), and write the result to a new vector. We do so to improve cache locality for future rounds, and take advantage of the CPU-intensive nature of elliptic curve operations along with prfetching to hide the latency of reading from essentially random locations in memory.

Subsequently, we perform the additions in place, and the second operands become junk data. Finally, when we only have the buckets left (no more additions left to perform), we copy the result into a destination “out” slice.

GLV

Main Reference (original GLV paper) (https://www.iacr.org/archive/crypto2001/21390189.pdf)

The following is meant to be an elaboration of missing elements of the orignal paper along with an explanation of a slight deviation in computing an integer approximation to a rational solution to a linear equation (which only affects the performance, and does not impact correctness in any way).

The approach taken for GLV is to precompute values that make the actual scalar decomposition cheaper.

First, we construct a lattice basis. In particular we want $v_1, v_2 \in \mathbb{Z} \times \mathbb{Z}$ that are short and are in the kernel of the function $f : \mathbb{Z} \times \mathbb{Z} \to \mathbb{F}_n$, $f((x, y)) = x + \lambda y \mod n$. Notice that we can find numbers that satisfy this condition by taking advantage of Bezout's identity since λ and n are coprime, by utilising the extended Euclidean algorithm (wikipedia for a basic reference) to find integers s_i, t_i, r_i satisfying

$$s_i n + t_i \lambda = r_i$$

Let us assume that $n \gg \lambda$. This means in particular that $q_1 > 1$.
Notice that we have the following important properties:

1. $r_i > r_{i+1} \geq 0$. Hence $q_i \geq 1$, and in particular, $q_1 > 1$.
2. for $i \geq 1$, the following propositions indexed by the natural numbers holds $P(i)$: $\text{sign}(t_i) = -\text{sign}(t_{i+1})$, $1 \leq |t_i| < |t_{i+1}|$.
3. $r_i |t_{i+1}| + r_{i+1} |t_i| = n$ (proved inductively)

Proof of 2.

Inductively, since we have $t_{i+1} = t_{i-1} - q_i t_i$,
and also, $t_1 = 1$, $t_0 = 0$, we have $P(1)$:

$$t_2 = 0 - q_1 < 0$$
$$|t_2| = |q_1| > 1 = |t_1|.$$

and referring again to the identity $t_{i+1} = t_{i-1} - q_i t_i$, we also have $P(i-1) \implies P(i)$:

$$|t_i| > |t_{i-1}| \ \wedge q_i \geq 1 \implies \text{sign}(t_{i+1}) = -\text{sign}(t_i)$$
$$\text{sign}(t_{i-1}) = -\text{sign}(t_i) \implies |t_{i+1}| = ||t_{i-1}| + q_i |t_i|| > |q_i||t_i| \geq |t_i|$$

□

We run this process until $|r_i| \geq \sqrt{n}$. Then, we set our basis vectors to be $v_1 = (a_1, b_1) = (r_i, -t_i), v_2 = (a_2, b_2) = (r_{i+1}, -t_{i+1})$. Notice that from the above properties we proved, $\text{sign}(b_1) = -\text{sign}(b_2)$, and $f(v_i) = 0$. We can also in general find $||v_i|| < \sqrt{2n}$. We test for this in our parameter generation script.

In the following, we try to find a linear combination of our basis that sufficiently approximates the vector $(k, 0)$. Note that the accuracy of approximation only affects performance, but has no bearing whatsover on correctness. The correctness comes from the fact that $\vec{r} = (k, 0) - \beta_1 \vec{v_1} - \beta_2 \vec{v_2}$, the vector we construct whose values represent our decomposition, evaluates to k under f. The point of approximation is to make it as short as possible, or as short AND in as computationally inexpensive a way as possible.

Approximation details

To express $(k, 0)$ in terms of our basis, we have to solve the following linear equation in $\mathbb{Q} \times \mathbb{Q}$: $\beta_1 \vec{v_1} + \beta_2 \vec{v_2} = (k, 0)$

$$\beta_1 \cdot a_1 + \beta_2 \cdot a_2 = k$$
$$\beta_1 \cdot b_1 + \beta_2 \cdot b_2 = 0$$

Notice, remembering that b_1 and b_2 have opposing signs, that this is solved by $\beta_2 = \frac{k|b_1|}{n}$, $\beta_1 = \frac{k|b_2|}{n}$

This can be worked out as follows:

$$\beta_1 = \beta_2 \left|\frac{b_2}{b_1}\right|$$
$$\beta_2(a_1 \cdot \left|\frac{b_2}{b_1}\right| + a_2) = k$$
$$\beta_2 = k|b_1|/(a_1 \cdot |b_2| + a_2 \cdot |b_1|) = k|b_1|/n$$

Then from there, we can approximate with some bounded error these coefficients in $\mathbb{Z} \times \mathbb{Z}$. Call the resulting vector $\vec{v}$, and their difference $\vec{r} = (k, 0) - \vec{v}$. Notice then that $f(\vec{r}) = f((k, 0) - \vec{v}) = f((k, 0)) = k \mod n$. So we have taken advantage of our construction that the lattice basis are in the kernel of f.

Because the basis is short, we also have that, from error of approximation of rational coefficients and (with additional error that is bounded by that error due to precomputation) $|\vec{r}| = |(k, 0) - \vec{v}| < (|\vec{v_1}| + |\vec{v_2}|) < 2\sqrt{2n} = \sqrt{8n}$.
So we have succeed in finding a vector $\vec{r}$ that evaluates to k under f and also is short.

The approximation we use to offload integer division to the precomputation stage is as follows:

Choose $R > r$. Then, one precomputes $q_1 = \lceil R|b_2|/n \rfloor$, $q_2 = \lceil R|b1|/n \rfloor$. Let us continue while dropping the indices $1, 2$.

Let $k, b < n$, we can write that round function to be the solution to q in $kb + (n+1)/2 = qn + t$, $0 \leq t < n$, in other words $q = \lceil kb/n \rfloor$. So q has a distance from kb/n by at most $1/2$.

Note that if instead look for a solution to $Rb + (n+1)/2 = q'n + t'$, $0 \leq t' < n$, in other words $q' = Rb/n + 1/2 - t'/n + \epsilon$, $q' = \lceil Rb/n \rfloor$. Notice $\epsilon = 1/2n$.

Then, if we take $c = q'k/R = k/R \cdot (Rb/n + 1/2 - t'/n + \varepsilon) = kb/n + k/2R - t'k/nR + \varepsilon'$. Here $\epsilon' = k/2nR < 1/2R$. This expression has a distance from kb/n by at most $||k/R(1/2 - t'/n) + \epsilon'||$. Notice that since $|1/2 - t'/n| \leq 1/2$ $|k/R| \leq 1 - 1/R$, by triangle inequality this is $< 1/2(1 - 1/R)$ + $1/2R = 1/2$.

Since we are rounding to this approximation with error at most 1/2, our total error of approximation is at most $1/2 + 1/2 = 1$.

Comparison to Implementation

In our implementation, we write $q_i = \lceil \frac{R|b_{(i+1) \mod 2}|}{n} \rfloor$ (or actually, $q_i = \lfloor \frac{R|b_{(i+1) \mod 2}|}{n} \rfloor$, but morally, this is the same), $c_i = \lceil \frac{q_i k}{R} \rfloor$, $d_i = c_i|b_i|$ and where our solution vector has components $\vec{r} = (k_1, k_2)$, our decomposition reads as $k_2 = -(c_1 b_1 + c_2 b_2)$, $k_1 = k - \lambda k_2$. Since recall that we require $f((k_1, k_2)) = k_1 + \lambda k_2 = k$.

In our implementation, we also ensure $(|b_1| + 2)(|b_2| + 2) < 2n$ in order to ensure that a particular modular reduction in the scalar decomposition implementation is correct.

w-NAF

See standard reference

GPU Approach

To be continued