aleo No.3 snarkVM Mining Prove MSM Pippenger - CUDA

https://github.com/xiaoyu1998

Resolution:2560×1440(2K)
https://github.com/AleoHQ/snarkVM/tree/v0.16.8

Entries

MSM
snarkVM/algorithms/cuda/cuda/snarkvm.cu

```
for (size_t i = 0; i < gpu_count; i++) {
    pool.spawn([&, i]()
        select_gpu(dev);                                          每个连接的GPU平分运算
        const affine_t* pts = (affine_t*)(&((uint8_t*)points)[start * ffi_affine_size]);
        msm_t<bucket_t, point_t, affine_t, scalar_t> msm(dev);
        ret = msm.invoke(partial_sums[i], slice_t<affine_t>{pts, sz},
        ch.send(i);
size_t dev = ch.recv();
for (size_t i = 0; i < gpu_count - 1; i++) {
    dev = ch.recv();
    point_t::dadd(*out, *out, partial_sums[dev]);
```

invoke
sppark/msm/pippenger.cuh

```
uint32_t stride = (npoints + batch - 1) / batch;
stride = (stride+WARP_SZ-1) & ((size_t)0-WARP_SZ);          points数补齐warp倍数
temp_sz = stride * std::max(2*sizeof(uint2), temp_sz);      scalars内存数
d_point_sz *= sizeof(affine_h);                             points内存数
size_t digits_sz = nwins * stride * sizeof(uint32_t);       所有窗口points数4字节内存数
dev_ptr_t<uint8_t> d_temp{temp_sz + digits_sz + d_point_sz, gpu[2]};   gpu分配内存
vec2d_t<uint2> d_temps{&d_temp[0], stride};                 指向scalars内存
vec2d_t<uint32_t> d_digits{&d_temp[temp_sz], stride};       指向digits内存
if (scalars) gpu[2].HtoD(&d_scalars[d_off], &scalars[h_off], num);   复制scalars
digits(&d_scalars[0], num, d_digits, d_temps, mont);
if (points) gpu[0].HtoD(&d_points[d_off], &points[h_off],   复制points
for (uint32_t i = 0; i < batch; i++) {
    batch_addition<bucket_t><<<gpu.sm_count(), BATCH_ADD_BLOCK_SIZE,
    gpu[i&1].launch_coop(accumulate<bucket_t, affine_h>,
    integrate<bucket_t><<<nwins, MSM_NTHREADS,
collect(p, res, ones);
out.add(p);
```

integrate
sppark/msm/pippenger.cuh

```
while (i--) {
    p = row[i];
    do {
        if (sizeof(bucket_t) <= 128) {
            p.add(acc);
            if (pc == 1) {
                res = p;
            } else {
                acc = p;                                    acc每一行累加，acc = acc+row[i]
                if (pc == 0) p = res;                       res所有acc累加, res = res+acc
```

pippenger

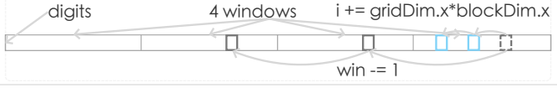
digits
sppark/msm/pippenger.cuh

```
breakdown<<<2*grid_size, 1024, sizeof(scalar_t)*1024, gpu[2]>>>(
d_digits, d_scalars, len, nwins, wbits, mont
for (win = 0; win < nwins-1; win += 2) {
    gpu[2].launch_coop(sort, {{grid_size, 2}, SORT_BLOCKDIM, shared_sz},
if (win < nwins) {
    gpu[2].launch_coop(sort, {{grid_size, 1}, SORT_BLOCKDIM, shared_sz},
```

breakdown
sppark/msm/pippenger.cuh

scalar分割

```
auto& scalar = xchange[tid/WARP_SZ](tid%WARP_SZ);
for (uint32_t i = tix; i < (uint32_t)len; i += gridDim.x*blockDim.x) {
    auto s = scalars[i];                                    当前线程的scalar
    if (mont) s.from();
    uint32_t msb = s[top_i] >> ((scalar_t::nbits - 1) % 32);   最后32位清除补齐
    s.cneg(msb);
    for (uint32_t bit0 = nwins*wbits - 1, win = nwins; --win;) {
        digits[win][i] = wval;
```



accumulate
sppark/msm/pippenger.cuh

每一个线程完成一个桶内所有点累加

```
while (x < (nwins << wbits)) {
    if ((len -= idx) && !(x == 0 && y == 0)) {
        const uint32_t* digs_ptr = &digits[y][idx];
        uint32_t digit = *digs_ptr++;
        affine_t p = points[digit & 0x7fffffff];
        bucket_t bucket = p;
        bucket.cneg(digit >> 31);
        while (--len) {
            digit = *digs_ptr++;
            p = points[digit & 0x7fffffff];
            bucket.add(p, digit >> 31);
        buckets[y][x] = bucket;
```

batch_addtion

sort
sppark/msm/sort.cuh

```
sort_row(inouts[win], len, temps[blockIdx.y], histograms[win],
```

sort_row
sppark/msm/sort.cuh

```
if (wbits > DIGIT_BITS || (lg_gridDim && wbits > lg_gridDim+1)) {   wbits很大切成两部分
    upper_sort(temp, inout, len, lsbits, top_bits, low_bits, histogram);   上半部分排序
    for (uint32_t i = blockIdx.x; i < 1<<top_bits; i += gridDim.x) {
        lower_sort(inout, temp, slice.x, slice.y, histogram, low_bits);    下半部分
} else if (blockIdx.x == 0) {
    lower_sort(inout, temp, 0, counters[0], histogram, wbits);             wbits小直接排序
```

为啥累加每wrap所有点?

batch_addition
sppark/msm/batch_addition.cuh

```
for (size_t i = tid; i < ndigits; i += gridDim.x*blockDim.x/degree) {
    uint32_t digit = digits[i];
    affine_t p = points[digit & 0x7fffffff];
    acc.add(p, digit >> 31);
for (uint32_t off = 1; off < warp_sz;) {
    bucket_t down = acc.shfl_down(off*degree);
    off <<= 1;
    if ((xid & (off-1)) == 0) acc.uadd(down);
if (xid == 0) ret[tid/warp_sz] = acc;
```

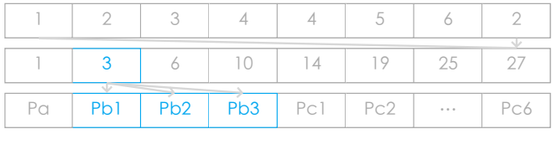
sort

msm_t
sppark/msm/pippenger.cuh

```
npoints = (np+WARP_SZ-1) & ((size_t)0-WARP_SZ);             points数补齐warp倍数
wbits = 17;                                                 每window位数
nwins = (scalar_t::bit_length() - 1) / wbits + 1;           windows数
uint32_t row_sz = 1U << (wbits-1);                          每一行window分割后
                                                            相同系数合并后，bucket数每一行bucket数
size_t d_buckets_sz = (nwins * row_sz)                      所有bucket数
    + (gpu.sm_count() * BATCH_ADD_BLOCK_SIZE / WARP_SZ);
size_t d_blob_sz = (d_buckets_sz * sizeof(d_buckets[0]))    buckets内存数
    + (nwins * row_sz * sizeof(uint32_t))                   buckets4字节内存数
    + (points ? npoints * sizeof(d_points[0]) : 0);         points内存数
d_buckets =                                                 GPU分配所有内存
reinterpret_cast<decltype(d_buckets)>(gpu.Dmalloc(d_blob_sz));
d_hist = vec2d_t<uint32_t>(&d_buckets[d_buckets_sz], row_sz);   第一行第一个桶
d_points = reinterpret_cast<decltype(d_points)>(d_hist[nwins]);   第一个点
gpu.HtoD(d_points, points, np, ffi_affine_sz);              复制所有点到GPU，如果有点
```

lower_sort
sppark/msm/sort.cuh

```
count_digits(src += base, len, mask);                       同一个桶里的点数
for (uint32_t i = 0; i < N_SUMS; i++) {
    uint32_t sum = counters[off];
    sum = sum_up(sum);
    if (i > 0) sum += __shfl_sync(0xffffffff, prefix_sums[i-1], WARP_SZ-1);
    prefix_sums[i] = sum;                                   每wrap里前面线程点数的累加
if (laneid == WARP_SZ-1) counters[warpid] = prefix_sums[N_SUMS-1];
uint32_t carry_sum = laneid ? counters[laneid-1] : 0;       每wrap所有线程点数的总和
carry_sum = sum_up(carry_sum, SORT_BLOCKDIM/WARP_SZ);
carry_sum = __shfl_sync(0xffffffff, carry_sum, warpid);
carry_sum += base;                                          每wrap所有线程点数的总和向前累加
for (uint32_t i = 0; i < N_SUMS; i++)                       为啥counters头32个数字可以被累加占用
    counters[lane_off + i*WARP_SZ] = prefix_sums[i] += carry_sum;
for (uint32_t i = 0; i < N_SUMS; i++, lane_off += WARP_SZ) {
    if (lane_off < 1<<bits) histogram[lane_off] = prefix_sums[i];   同一个桶的点数
scatter(dst, src, len, mask);                               同一个桶的点连续放入digits
```



Warp线程簇32个线程，一个流多一次操作一个Warp