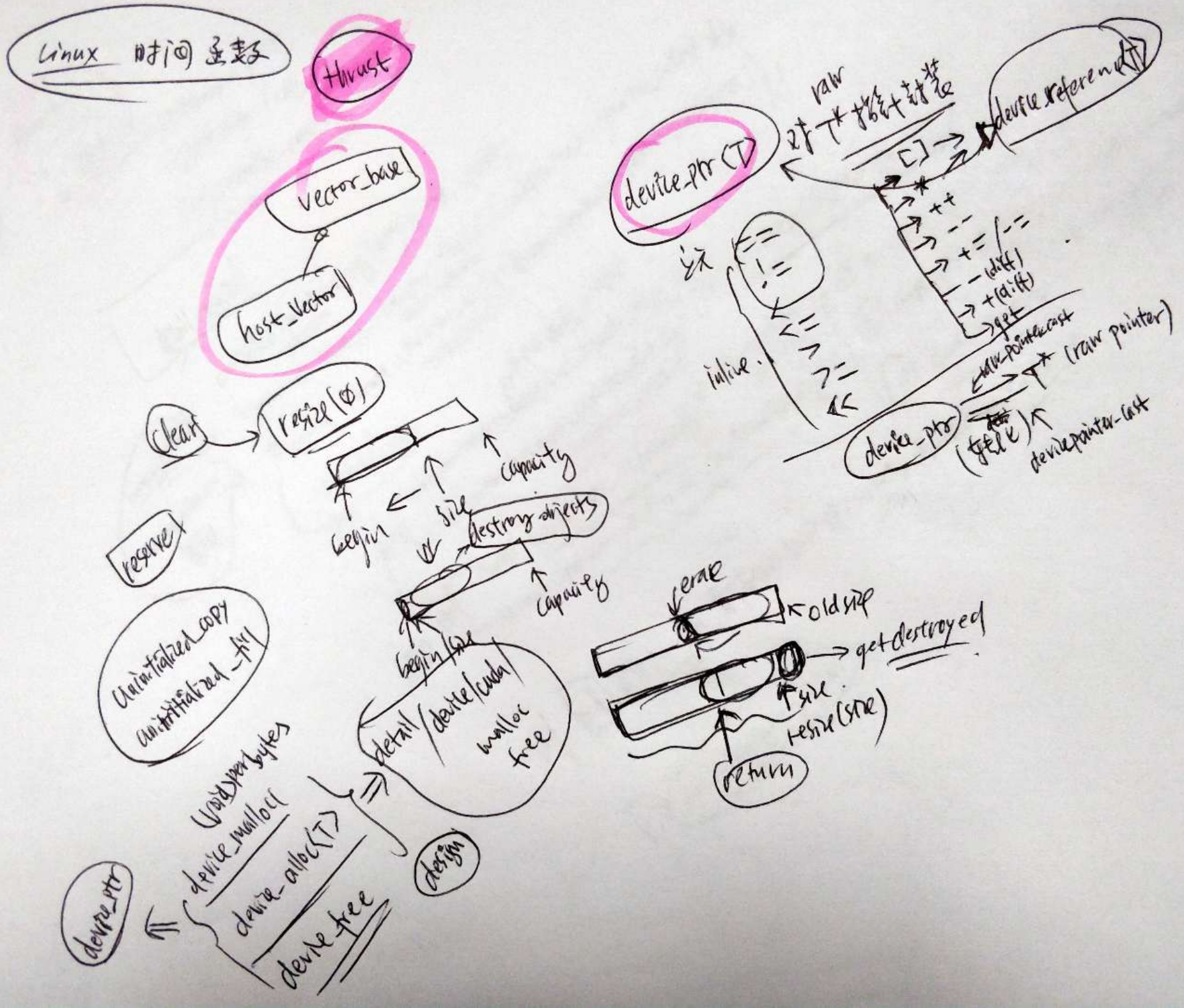
Linux 时间函数
Thrust
vector_base
host_vector
device_ptr<T>
raw 指针封装
device_reference<T>
[]
++
--
+=
-=
-(diff)
+(diff)
get
比较
!=
<=
>=
>
<
inline
device_ptr
(get)
raw_pointer_cast
(raw pointer)
devicepointer_cast
clear
resize(0)
capacity
size
begin
reserve
destroy_objects
Uninitialized_copy
uninitialized_fill
erase
oldsize
get_destroyed
resize(size)
return
detail
device/cuda
malloc
free
WorldSpecBytes
device_malloc
device_alloc<T>
device_free
device_ptr
design

thrust::**copy**(InputIterator first,
InputIterator last,
OutputIterator result);

detail/copy.inl

```
detail::dispatch::copy(begin, end, result,
    typename thrust::iterator_traits<InputIterator>::iterator_category(),
    typename thrust::iterator_traits<OutputIterator>::iterator_category());
```

InputIterator ⇒
OutputIterator ⇒
类型

dispatch/copy.h

- thrust::random_access_host_iterator_tag ⊇ input_host_iterator
- thrust::random_access_host_iterator_tag ⊇ input_host_iterator 直接 ⇒ std::copy(begin, end, result)

H2D
- thrust::input_host_iterator_tag; ⊇ input_host_iterator ← 不是random access
- thrust::random_access_device_iterator_tag;
  - ① malloc host memory
  - ② duplicate input to malloced array → may not random access
  - ③ cudamemcpy

  再 copy in device

类型不match，要先把 host 转 convert，再 copy in device.

IV | [illegible] 类型不match

D2H [cudamemcpy D2H

thrust::count

统计序列中 == value 的个数

transform_reduce(first, last, op, CountType(0), bin_op);

用 difference_type 作返回类型

bin_op: thrust::plus<CountType> bin_op

op: detail::count_make_predicate<InputType, EqualityComparable, CountType> op(value);

functor: `__host__ __device__ CountType operator()(const InputType1& lhs) const { return lhs == rhs ? 1 : 0; }`

transform_reduce:
- InputIterator
- InputIterator
- UnaryFunction (predicate)
- OutputType ← init 0
- BinaryFunction

cpu logic

```
while(first != last) {
    if (op(*first)) {
        cnt = bin_op(cnt, 1);
    }
    ++first;
}
```

gpu logic

用它的判断返回值 1/0，再做统计计算

① transform

```
while (first != last) {
    counts[i++] = op(*first);
}
```

⇒ output boolean array [1, 0, 1, 0, 0, 1 ...]（op 判断的一个值）

② reduce

accumulate(counts, init); 类型为 OutputType.

⇒ total count = 1.

[1|0|1|0|...|0|1|...]

init → bin_op. 也就是统计计数.

dispatch::transform_reduce

根据输入数据类型选择设备

- host:

```
OutputType result = init;
while(begin != end){
    result = binary_op(result, unary_op(*begin));
    begin++;
}
```

- device:

detail::transform_reduce_functor<...> func;（func: 定义 operator()，apply unary_func）

device::cuda::reduce(func, end_start, init, binary_op);

reduce (InputFunctor input,
size_t n,
OutputType init,
BinaryFunction binary_op

BLOCKSIZE = 256; ⇒ 一个block中线程数

n < Blocksize ⇒ __thrust_serial_reduce_kernel<<<GRID, 1>>>
{input, n, block_result.get(), binary_op};
一个线程

n > Blocksize ⇒ __thrust_unordered_reduce_kernel<<<GRID_SIZE, BLOCKSIZE>>>
{input, n, block_result.get(), binary_op};
⇒ per-block result — cuda malloc ondemand

⇒ copy per-block results to host
⇒ reduce on host finally with init

n (arraysize)
i
(i + grid)

functor

if (threadIdx.x == 0) {
OutputType accum = input[0];
for i = 1 ⇒ n {
accum = binary_op(accum, input[i]);
}
block_results[0] = accum;
}

block
i (thread index)
grid
accum = input[i];
i += grid_size;
while (i < n) {
accum = binary_op(accum, input[i]);
i += grid_size;
}

shared_memory
same bytes of thread# * sizeof(type)
sdata[threadIdx.x] = accum;
local thread idx in block
每个threads单独执行
⇒ __syncthreads();

256
128
128
64
64
32
16 32
2

128个线程工作
64个线程工作
32个线程工作

threadIdx.x == 0
⇒ 最后reduce的结果

i
i+128
+128
i, 和 i+128 进行 reduce
放入 i 位置
同步 __syncthreads()

transform

归纳下面几种类型:

① output ⇐ unary_op(input1)

② output1 ⇐ binary_op(input1, input2)

③ output1 ⇐ unary_op(input1) if predicate(stencil)

④ output ⇐ binary_op(input1, input2) if (predicate(stencil))

transform_if { ③ ④

⇓ dispatch on category

dispatch::transform

- ⇒ host2host ⇒ 直接 std::transform ⇒ ~~thrust~~::host::transform_if
- ⇒ ~~host2device~~
- ⇒ device2device
  - unary_op版本 提供 unary_transform_functor (⇐ input ⇓ unary_op ⇒ output)
  - binary_op版本 提供 binary_transform_functor (⇐ input1/input2 ⇐ binary_op ⇒ output.)
  - 对if版本 只需要提供不同的 functor wrap if (predicate(stencil)) ......
  - 最终都会调用 vectorize

调用 device::cuda::vectorize 对 functor 执行 apply functor.

vectorize

kernel 实现

```
Grid size = (n + blocksize - 1) / blocksize;
vectorize_kernel<<<Gridsize, Blocksize>>>(n, f);

for (i = blockIdx.x * blockDim.x + threadIdx.x;
     i < n;
     i += gridsize)
    f(i);
```