rocBLAS Device Memory Management

Lee Killough
lee.killough@amd.com
June 2019

Statement of Problem

- Some rocBLAS kernels need temporary device memory, to be optimal
- Allocating and deallocating device memory is expensive and synchronizing
- Temporary device memory should be recycled across multiple rocBLAS kernel calls within the same stream (handle)
- The API needs to be simple and easy to use
- Changes in implementation ideally should not change the rocBLAS API
- There is a critical open bug (SWDEV-176722) asking to make TRSM (not TRSM_EX) an asynchronous call, meaning no allocation operations

Current Status

- Currently rocBLAS uses 3 different methods for allocating device memory
  1. The user passes a pointer and size of already-allocated device memory (TRSM_EX; *added to GEMM_EX but not currently used*)
  2. A fixed amount of device memory is allocated at handle creation time (TRSM, TRSM_EX, TRSV)
  3. A rocBLAS kernel allocates and deallocates device memory on every call (AMAX, AMIN, ASUM, DOT, GET_VECTOR, NRM2, SET_VECTOR, TRSM, TRTRI_BATCHED)

Considered but Rejected: Passing Pointers and Sizes as Extra Arguments to Kernels

- It requires that the user manage and pass extra arguments which are not essential to the mathematical problem at hand, for bookkeeping which should ideally be kept internal to the library and managed *out-of-band* from the regular kernel calls
- It requires changing the API and breaking old code any time variable-size device workspace memory needs to be added to a kernel
- It makes it harder to recycle temporary device memory across different kernels, since the user must figure out the maximum which all of the kernels need, and pass the same pointer to all of them, which might not be intuitive in block-structured code
- It cannot be applied to non-EX kernels, which have up to now avoided adding expert parameters because of their burden on users, while this proposal can do it out of band
- It will be impractical to fix BLAS-1 functions which currently allocate and deallocate device memory on every call, if we pass that responsibility onto the user and unnecessarily complicate the BLAS-1 API; BLAS-1 functions can be fixed without breaking their API, if the device memory is stored in the handle and handled out-of-band

Considered but Rejected: Set a rocBLAS handle-wide Device Memory Allocation “Policy”

- Policy #1: rocBLAS never allocates device memory
  - —Still has the problem of querying the optimal size a kernel needs
- Policy #2: rocBLAS allocates extra device memory when needed
  - —Can cause surprise synchronizations to the user when it silently allocates memory during a kernel call
  - —Can still be useful as the default behavior when the user does not explicitly allocate memory
  - —To amortize the cost of allocations and reduce the chances of an unexpected synchronization, a default size can be allocated at handle creation time
- Policy #3: rocBLAS allocates and deallocates device memory at every call
  - —Too slow and synchronizing to be worth the savings in device memory

Considered but Rejected: Create Functions with `_size` Suffixes to Query optimal Sizes

```
rocblas_status rocblas_gemm_ex(rocblas_handle,…);
size_t rocblas_gemm_ex_size(rocblas_handle,…);
```

- Doubles the number of functions in the library
- Usually buffer size calculations are done in the main kernel anyway; this duplicates them in a separate function, requiring changes in two places if it ever changes
- Requires maintaining two functions with identical parameter lists
- Requires creating dummy `_size` functions which return `0` if the kernel does not use device memory, or requires breaking the invariant that ***every*** function can be queried as to its optimal device memory size
- Harder to maintain

Solution: Per-handle device memory allocation, with **out-of-band management**

`ROCBLAS_DEVICE_MEMORY_SIZE` environment variable

- If `> 0`, sets the default handle device memory size to the specified size (in bytes)
- If `==0` or unset, lets rocBLAS manage device memory, by using a default size (like 1 MB), and expanding it when necessary

```
rocblas_status rocblas_set_device_memory_size(rocblas_handle, size_t size);
```

- Changes the size of allocated device memory at runtime
- Any previously allocated device memory is freed
- A `size > 0` sets the device memory size to the specified size (in bytes)
- A `size ==0` frees the memory allocated so far, and lets rocBLAS manage device memory in the future, expanding it when necessary

```
rocblas_status rocblas_get_device_memory_size(rocblas_handle, size_t *size);
```

- Sets `*size` to the current device memory size for the handle

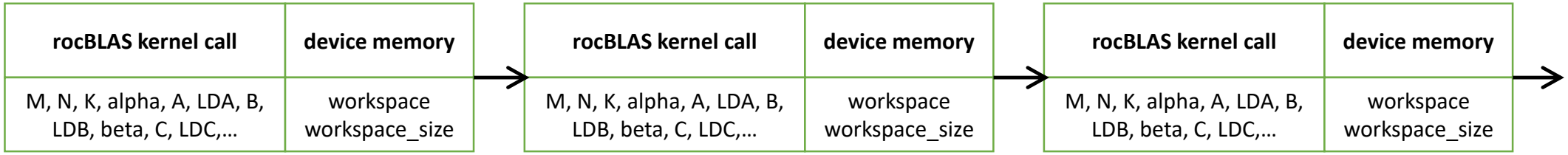
```
bool rocblas_is_managing_device_memory(rocblas_handle handle);
```

- Returns `true` when device memory in `handle` is managed by rocBLAS

In-Band vs Out-of-Band

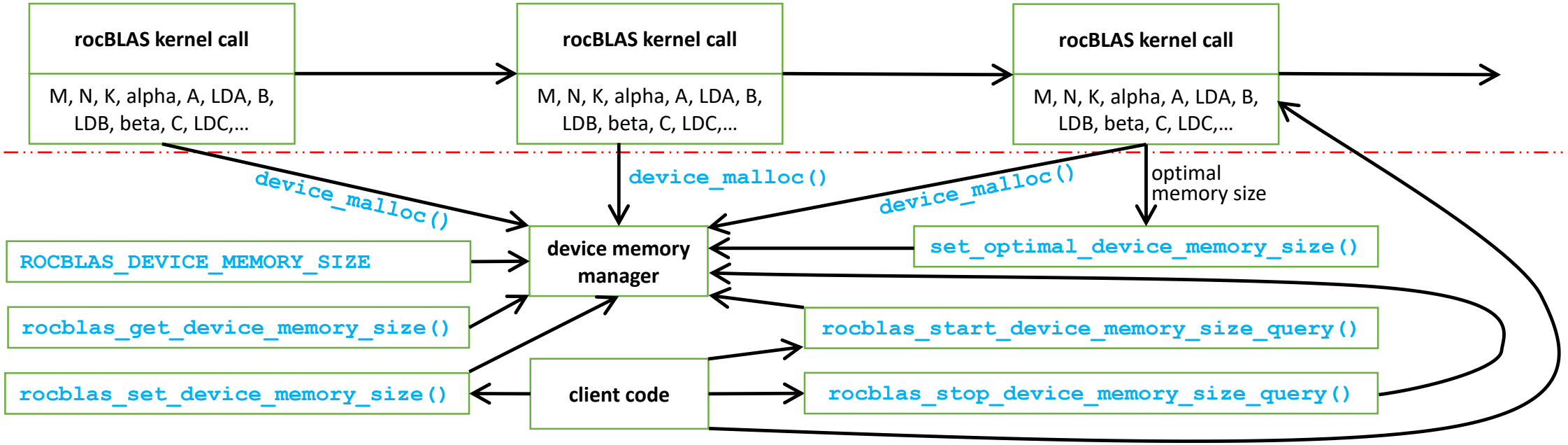
In-Band device memory allocation:

In-band



Out-of-Band device memory allocation:

Out-of-band



Device Memory Size Queries

```
rocblas_status rocblas_start_device_memory_size_query(rocblas_handle);
```

- Indicates that subsequent rocBLAS kernel calls should collect the optimal device memory size in bytes for their given kernel arguments, and keep track of the maximum
- Each kernel call can reuse temporary device memory on the same stream, so the maximum is collected
- Returns `rocblas_status_size_query_mismatch` if another size query is already in progress; returns `rocblas_status_success` otherwise
- Can be applied to existing kernels without changing their APIs, because it is encapsulated in the `rocblas_handle` class, which is universal to all rocBLAS kernels
- **Does not require `*_ex` functions with extra `void* size_t*` pointers at the end of the parameter list**
- **Can be applied to non-`_ex` GEMM, TRSM, and TRSV!!!**

```
rocblas_status rocblas_stop_device_memory_size_query(rocblas_handle, size_t* size);
```

- Stops collecting optimal device memory size information, and stores the maximum of the optimal sizes collected into `*size`
- Returns `rocblas_status_size_query_mismatch` if a collection is not underway; `rocblas_status_invalid_pointer` if `size` is `nullptr`; `rocblas_status_success` otherwise

Answering Device Memory Size Queries

`bool _rocblas_handle::is_device_memory_size_query() const;`

- Indicates that the current kernel call is collecting information about the optimal device memory allocation size

`rocblas_status _rocblas_handle::set_optimal_device_memory_size(size...);`

- Sets the optimal size(s) of device memory buffer(s) in bytes for this kernel call
- The size(s) are rounded up to the next multiple of 64 (or some other chunk size), and the running maximum is updated
- Return status:
  `rocblas_status_size_unchanged` if the maximum optimal device memory size did not change
  `rocblas_status_size_increased` if the maximum optimal device memory size increased
  `rocblas_status_internal_error` if this kernel call is not supposed to be collecting size information
- The kernel should return the status returned by this function, so that it never returns `rocblas_status_success`

```
if(handle->is_device_memory_size_query())
{
    size_t size = m * n * sizeof(T);  // Compute optimal size
    return handle->set_optimal_device_memory_size(size);
}
```

`size_t rocblas_sizeof_datatype(rocblas_datatype type)`

- Computes the `sizeof` a rocBLAS ***runtime*** data type
- Used to calculate the device memory size in bytes for a given kernel with runtime type information
- Need to be careful to distinguish `Ti`, `To`, and `Tc` (the input, output and compute types) when calculating the device memory size, since the temporary arrays may be storing types of `Ti`, `To`, or `Tc`
- The actual size should be calculated and returned, rather than an upper bound, such as always multiplying by `sizeof(double)==8` (which is the current behavior of TRSM)

`RETURN_ZERO_DEVICE_MEMORY_SIZE_IF_QUERIED(handle)`

- Convenience macro which executes `return rocblas_status_size_unchanged;` if this kernel call is a device memory size query
- Used at the beginning of kernels which do not need any extra device memory

```
rocblas_status rocblas_kernel(rocblas_handle handle, ...)
{
    RETURN_ZERO_DEVICE_MEMORY_SIZE_IF_QUERIED(handle);
    // ...
}
```

rocBLAS Kernel Device Memory Allocation

```
auto mem = handle->device_malloc(size...);
```

- Returns an opaque RAII object lending allocated device memory to a particular rocBLAS kernel invocation
- To simplify and optimize the code, only one **successful** allocation object can be alive at a time
- The lifetime of the returned object is the lifetime of the borrowed device memory (RAII)
- If the handle's device memory is currently being managed by rocBLAS, it is expanded in size as necessary
- If the user allocated an explicit size of device memory, then that size is used as the limit, and no resizing or synchronization ever occurs
- The object evaluates to **`false`** if there aren't enough bytes available
- The object returned is convertible to **`void*`** or other pointer types, if only one size is specified
- The object can be assigned to **`std::tie(ptr1, ptr2, …)`**, if more than one size is specified
- **The allocation always has O(1) cost and no synchronization, if the available device memory is large enough, or if the user had explicitly set the allocation size ahead of time**
- This class hides the device memory allocation implementation from the rocBLAS kernel programmer, while providing fast and easy access to device memory of certain requested size(s)

Multi-Buffer Device Memory Allocation

- Convenience operation to allocate multiple buffers of device memory for a single kernel
- `device_malloc()` return type can be assigned to `std::tie(ptr1, ptr2, ptr3, …)`
- Automatically partitions allocated memory into separate buffers of different sizes
- Automatically rounds up each buffer to a multiple of 64 (or other chunk size) for alignment purposes
- Generates optimized inlined code, such as folding the sums of the sizes if they are **`constexpr`** values

  For example, **`device_malloc(1024, 256, size, 512)`** gets automatically simplified to **`size+1792`** at compile-time
- At most one successful allocation is done for the total size
- Multiple pointer assignment is inlined and optimized into individual pointer assignments

```
void *buf1, *buf2, *buf3;
size_t bufsize1, bufsize2, bufsize3;
auto mem = handle->device_malloc(bufsize1, bufsize2, bufsize2);
if(!mem)
   return rocblas_status_memory_error;
std::tie(buf1, buf2, buf3) = mem;  // Inlined and scalarized assignment
```

rocblas_status_memory_error

- Used to indicate that insufficient device memory has been allocated for the successful execution of a kernel

```
size_t size1, size2;

// Compute size1, size2

auto mem = handle->device_malloc(size1, size2);
if(!mem)
    return rocblas_status_memory_error;
```

rocblas_status_perf_degraded

- Used to indicate that a slower algorithm was used because of insufficient device memory for the optimal algorithm

```
rocblas_status ret = rocblas_status_success;
auto mem1 = handle->device_malloc(size1);
if(!mem1)
{
    auto mem2 = handle->device_malloc(size2);
    if(mem2)
    {
        // Algorithm using smaller mem2 of size size2
        ret = rocblas_status_perf_degraded;
    } else {
        // Not enough device memory for faster or slower algorithm
        ret = rocblas_status_memory_error;
    }
} else {
    // Algorithm using larger mem1 of size size1
}
return ret;
```

push_pointer_mode(rocblas_pointer_mode)

- Used to save and automatically restore the current pointer mode, while switching it to a new mode
- Needed for functions which use constants on host (-1.0, 0.0, 1.0, etc.) passed to GEMM alpha, beta, etc.
- Can be compared to **rocblas_pointer_mode** values to get old value
- Avoids extra copies from device to host

```
// Switch to host pointer mode, saving current pointer mode, restored on return
auto saved_pointer_mode = handle->push_pointer_mode(rocblas_pointer_mode_host);

// Get alpha
T alpha_h;
if(saved_pointer_mode == rocblas_pointer_mode_host)
   alpha_h = *alpha;
else
   RETURN_IF_HIP_ERROR(hipMemcpy(&alpha_h, alpha, sizeof(T), hipMemcpyDeviceToHost));


// Original pointer mode is restored on return or exception
```

Example: `rocblas_trsm_ex`

```
extern "C" rocblas_status rocblas_trsm_ex(rocblas_handle handle,
                                          rocblas_side side,
                                          rocblas_fill uplo,
                                          rocblas_operation trans_a,
                                          rocblas_diagonal diag,
                                          rocblas_int m,
                                          rocblas_int n,
                                          const void* alpha,
                                          const void* a,
                                          rocblas_int lda,
                                          void* b,
                                          rocblas_int ldb,
                                          const void* invA,
                                          rocblas_int ld_invA,
                                          rocblas_datatype compute_type)
                                          rocblas_trsm_option option,
                                          size_t* x_temp_size,
                                          void* x_temp_workspace)
```

- **No need to pass extra parameters.** Query returns *optimal* size, but a smaller size can still be used with a slower algorithm, without having to explicitly specify **`option = rocblas_trsm_low_memory`**

Example: `rocblas_trsm_ex`, Cont.

```
{
    // By default return success
    rocblas_status rb_memory_status = rocblas_status_success;

    // Compute the optimal size in bytes for maximum speed
    size_t x_temp_size = rocblas_sizeof_datatype(compute_type) * m * n;

    // If this call is a device memory size query,
    // return the size in bytes recommended for maximum speed
    if(handle->is_device_memory_size_query())
        return handle->set_optimal_device_memory_size(x_temp_size);

    // Attempt to allocate the optimal size
    auto x_temp_workspace = handle->device_malloc(x_temp_size);
    if(!x_temp_workspace)
    {
        // If optimal size is not available, try the smaller size
        x_temp_size = rocblas_sizeof_datatype(compute_type) * m;
        x_temp_workspace = handle->device_malloc(x_temp_size);

        // If the smaller size cannot be allocated, return error
        if(!x_temp_workspace)
            return rocblas_status_memory_error;

        // Set return status to indicate degraded performance
        rb_memory_status = rocblas_status_perf_degraded;
    }
```

Example: `rocblas_trsm_ex`, Cont.

```
// Pass the large or small x_temp_size and x_temp_workspace
rb_status = rocblas_trsm_ex_template<TRSM_BLOCK>(
                                        handle,
                                        side,
                                        uplo,
                                        trans_a,
                                        diag,
                                        m,
                                        n,
                                        static_cast<const float*>(alpha),
                                        static_cast<const float*>(a),
                                        lda,
                                        static_cast<float*>(b),
                                        ldb,
                                        static_cast<const float*>(invA),
                                        ld_invA,
                                        &x_temp_size,
                                        static_cast<float*>(x_temp_workspace));

 return rb_status != rocblas_status_success ? rb_status : rb_memory_status;
}
```