Ruby Association Grant 2017 Final Term Report

- Author: Prasun Anand
- Email: prasunanand.bitsp@gmail.com
- Github: prasunanand

Contents

Overview

Few people realise it, but even the modest computers today, including mobile phones, have powerful GPUs. And these GPUs can be used serially and in parallel to CPUs, potentially delivering awesome performance.

The RbCUDA gem that I have been developing provide mores flexibility and power to a programmers/researchers/scientists to harness and optimize their solutions for GPU computing and run across all CUDA powered hardwares.

The main objective of RbCUDA are:

1. Map all of CUDA into Ruby
2. Ready-made on-GPU linear algebra, reduction, scan using cuBLAS, cuMath, cuSolver libraries.
3. Random Numer generator using cuRand
4. Near-zero wrapping overhead.
5. CUDA profiler for Ruby.

Code

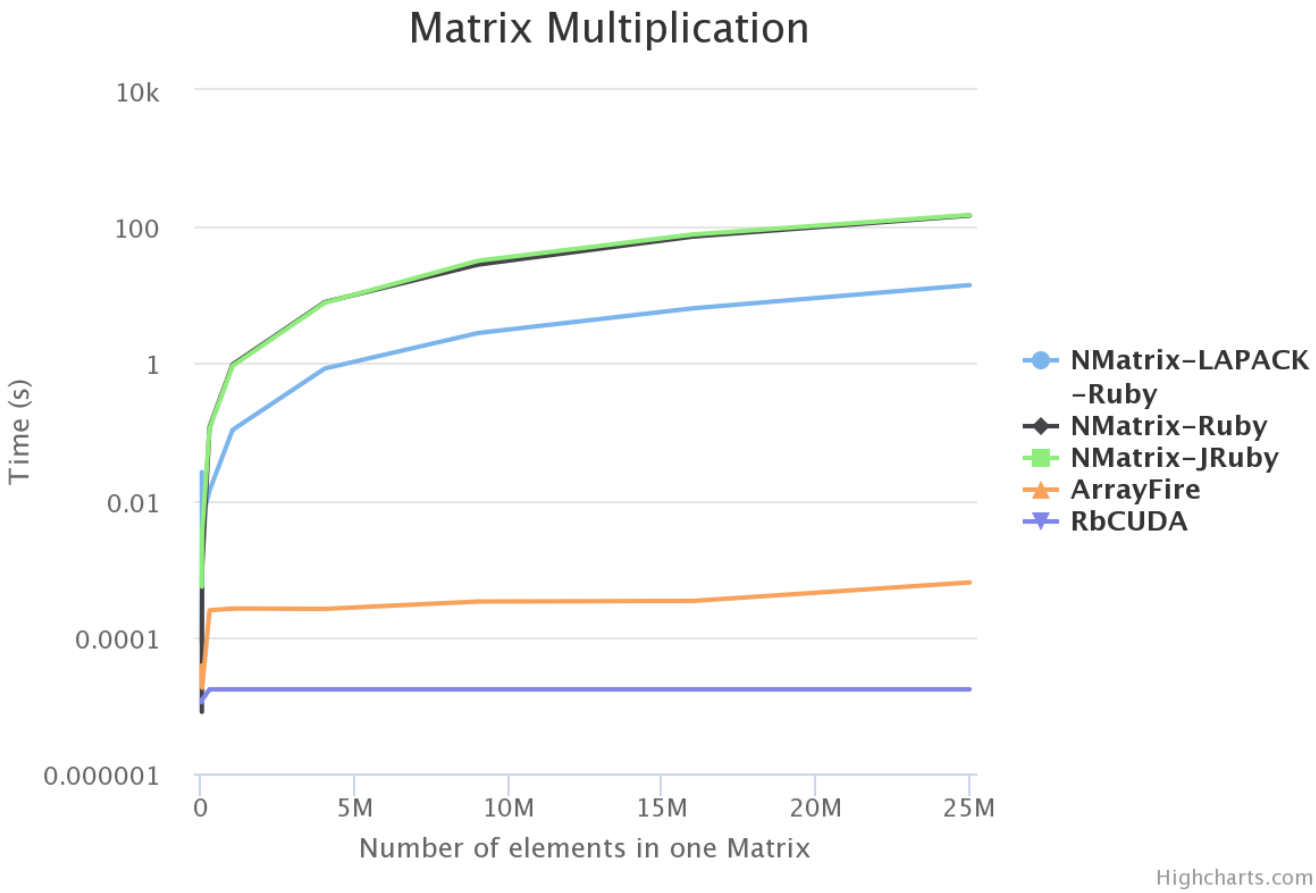
https://github.com/prasunanand/rbcuda/

Installation

1. Install nvidia cuda driversy

```
$ git clone https://github.com/prasunanand/rbcuda
$ cd rbcuda
$ rake compile
```

Benchmarks



(Note: The above benchmarks have been done on an AMD FX 8350 octacore processor and Nvidia GTX 750Ti GPU. CUDA backend of ArrayFire was used with double floating points.)

Result

RbCUDA is the fastest of all the Ruby libraries. The time taken for matrix multiplication is 0.000017s on my machine. The plain C code takes me 0.000013s for this calculation.

RbCUDA is 24x faster than ArrayFire for matrix multiplication, Most of the speed gain going straight to CUDA is probably from removing an interaction layer (and buffers) as well as how the data is organized and fed to the underlying architecture.

Hence, an overhead of 0.000004s over plain C code makes, it highly efficient Maths library in Ruby.

RbCUDA Modules

The following section walks through the code organisation of RbCUDA.

CUDA Driver

```
 CUDA = rb_define_module_under(RbCUDA, "CUDA");
  rb_define_singleton_method(CUDA, "cuGetErrorString", (METHOD)rb_cuGetErrorString,
1);
  rb_define_singleton_method(CUDA, "cuGetErrorName", (METHOD)rb_cuGetErrorName, 1);
  rb_define_singleton_method(CUDA, "cuInit", (METHOD)rb_cuInit, 1);
  rb_define_singleton_method(CUDA, "cuDriverGetVersion",
(METHOD)rb_cuDriverGetVersion, 1);
  rb_define_singleton_method(CUDA, "cuDeviceGet", (METHOD)rb_cuDeviceGet, 1);
  rb_define_singleton_method(CUDA, "cuDeviceGetCount", (METHOD)rb_cuDeviceGetCount,
0);
  rb_define_singleton_method(CUDA, "cuDeviceGetName", (METHOD)rb_cuDeviceGetName,
2);
  rb_define_singleton_method(CUDA, "cuDeviceTotalMem_v2",
(METHOD)rb_cuDeviceTotalMem_v2, 1);
  rb_define_singleton_method(CUDA, "cuDeviceGetAttribute",
(METHOD)rb_cuDeviceGetAttribute, 3);
  rb_define_singleton_method(CUDA, "cuDeviceGetProperties",
(METHOD)rb_cuDeviceGetProperties, 1);
  rb_define_singleton_method(CUDA, "cuDeviceComputeCapability",
(METHOD)rb_cuDeviceComputeCapability, 1);
  rb_define_singleton_method(CUDA, "cuDevicePrimaryCtxRetain",
(METHOD)rb_cuDevicePrimaryCtxRetain, 1);
  rb_define_singleton_method(CUDA, "cuDevicePrimaryCtxRelease",
(METHOD)rb_cuDevicePrimaryCtxRelease, 1);
  rb_define_singleton_method(CUDA, "cuDevicePrimaryCtxSetFlags",
(METHOD)rb_cuDevicePrimaryCtxSetFlags, 2);
  rb_define_singleton_method(CUDA, "cuDevicePrimaryCtxGetState",
(METHOD)rb_cuDevicePrimaryCtxGetState, 1);
  rb_define_singleton_method(CUDA, "cuDevicePrimaryCtxReset",
(METHOD)rb_cuDevicePrimaryCtxReset, 1);
```

Implentation of `cuDeviceGet` is as follows:

```
static VALUE rb_cuDeviceGet(VALUE self, VALUE ordinal){
  device_ptr* pdevice = ALLOC(device_ptr);
  CUresult result = cuDeviceGet(&pdevice->device, NUM2INT(ordinal));
  return Data_Wrap_Struct(RbCuDevice, NULL, rbcu_free, pdevice);
}
```

Runtime

```
Runtime = rb_define_module_under(RbCUDA, "Runtime");
  rb_define_singleton_method(Runtime, "cudaDeviceReset", (METHOD)rb_cudaDeviceReset,
0);
  rb_define_singleton_method(Runtime, "cudaDeviceSynchronize",
(METHOD)rb_cudaDeviceSynchronize, 0);
  rb_define_singleton_method(Runtime, "cudaDeviceSetLimit",
```

```
(METHOD)rb_cudaDeviceSetLimit, 2);
  rb_define_singleton_method(Runtime, "cudaDeviceGetLimit",
(METHOD)rb_cudaDeviceGetLimit, 2);
  rb_define_singleton_method(Runtime, "cudaDeviceSetCacheConfig",
(METHOD)rb_cudaDeviceSetCacheConfig, 1);
  rb_define_singleton_method(Runtime, "cudaDeviceGetSharedMemConfig",
(METHOD)rb_cudaDeviceGetSharedMemConfig, 0);
  rb_define_singleton_method(Runtime, "cudaDeviceSetSharedMemConfig",
(METHOD)rb_cudaDeviceSetSharedMemConfig, 1);
  rb_define_singleton_method(Runtime, "cudaDeviceGetByPCIBusId",
(METHOD)rb_cudaDeviceGetByPCIBusId, 1);
  rb_define_singleton_method(Runtime, "cudaDeviceGetPCIBusId",
(METHOD)rb_cudaDeviceGetPCIBusId, 2);
  rb_define_singleton_method(Runtime, "cudaIpcGetEventHandle",
(METHOD)rb_cudaIpcGetEventHandle, 1);
  rb_define_singleton_method(Runtime, "cudaIpcOpenEventHandle",
(METHOD)rb_cudaIpcOpenEventHandle, 1);
  rb_define_singleton_method(Runtime, "cudaIpcGetMemHandle",
(METHOD)rb_cudaIpcGetMemHandle, 2);
  rb_define_singleton_method(Runtime, "cudaIpcOpenMemHandle",
(METHOD)rb_cudaIpcOpenMemHandle, 2);
  rb_define_singleton_method(Runtime, "cudaIpcCloseMemHandle",
(METHOD)rb_cudaIpcCloseMemHandle, 1);
  rb_define_singleton_method(Runtime, "cudaThreadExit", (METHOD)rb_cudaThreadExit,
0);
  rb_define_singleton_method(Runtime, "cudaThreadSynchronize",
(METHOD)rb_cudaThreadSynchronize, 0);
  rb_define_singleton_method(Runtime, "cudaThreadSetLimit",
(METHOD)rb_cudaThreadSetLimit, 2);
  rb_define_singleton_method(Runtime, "cudaThreadGetLimit",
(METHOD)rb_cudaThreadGetLimit, 1);
```

Implementation of `cudaStreamCreateWithPriority` is as follows:

```
static VALUE rb_cudaStreamCreateWithPriority(VALUE self, VALUE flags, VALUE
priority){
  custream_ptr* p_stream = ALLOC(custream_ptr);
  cudaError error = cudaStreamCreateWithPriority(&p_stream->stream,
NUM2UINT(flags), NUM2INT(priority));
  return Data_Wrap_Struct(RbCuStream, NULL, rbcu_free, p_stream);
}
```

CuBLAS and CuBLAS_XT

CuBLAS APIs have been added for double arrays. CuBLAS_XT apis have also been added however they have not been test because I didn't have access to multiple GPU cards on a single machine.

```
  CuBLAS_v2 = rb_define_module_under(RbCUDA, "CuBLAS_v2");
  rb_define_singleton_method(CuBLAS_v2, "cublasCreate_v2",
(METHOD)rb_cublasCreate_v2, 0);
  rb_define_singleton_method(CuBLAS_v2, "cublasDestroy_v2",
(METHOD)rb_cublasDestroy_v2, 1);
```

```
  rb_define_singleton_method(CuBLAS_v2, "cublasGetVersion_v2",
(METHOD)rb_cublasGetVersion_v2, 1);
  rb_define_singleton_method(CuBLAS_v2, "cublasDnrm2_v2", (METHOD)rb_cublasDnrm2_v2,
4);
  rb_define_singleton_method(CuBLAS_v2, "cublasDdot_v2", (METHOD)rb_cublasDdot_v2,
6) ;
  rb_define_singleton_method(CuBLAS_v2, "cublasDscal_v2", (METHOD)rb_cublasDscal_v2,
5);
  rb_define_singleton_method(CuBLAS_v2, "cublasDaxpy_v2", (METHOD)rb_cublasDaxpy_v2,
7);
  rb_define_singleton_method(CuBLAS_v2, "cublasDcopy_v2", (METHOD)rb_cublasDcopy_v2,
6);
  rb_define_singleton_method(CuBLAS_v2, "cublasDswap_v2", (METHOD)rb_cublasDswap_v2,
6);
  rb_define_singleton_method(CuBLAS_v2, "cublasIdamax_v2",
(METHOD)rb_cublasIdamax_v2, 4);
```

Implementation of `cublasDnrm2_v2` is as follows:

```
static VALUE rb_cublasDnrm2_v2(VALUE self, VALUE handler_val, VALUE n, VALUE x,
VALUE incx){
  rb_cublas_handle* handler;
  Data_Get_Struct(handler_val, rb_cublas_handle, handler);

  dev_ptr* ptr_x;
  Data_Get_Struct(x, dev_ptr, ptr_x);

  double result;

  cublasStatus_t status = cublasDnrm2_v2(handler->handle, NUM2INT(n),
                                         ptr_x->carray, NUM2INT(incx), &result);
  return DBL2NUM(result);
}
```

CuSolver

```
  CuSolver = rb_define_module_under(RbCUDA, "CuSolver");
  rb_define_singleton_method(CuSolver, "cusolverDnCreate",
(METHOD)rb_cusolverDnCreate, 0);
  rb_define_singleton_method(CuSolver, "cusolverDnDestroy",
(METHOD)rb_cusolverDnDestroy, 1);
  rb_define_singleton_method(CuSolver, "cusolverDnSetStream",
(METHOD)rb_cusolverDnSetStream, 2);
  rb_define_singleton_method(CuSolver, "cusolverDnSpotrf",
(METHOD)rb_cusolverDnSpotrf, 8);
  rb_define_singleton_method(CuSolver, "cusolverDnSpotrs",
(METHOD)rb_cusolverDnSpotrs, 9);
  rb_define_singleton_method(CuSolver, "cusolverDnDgetrf",
(METHOD)rb_cusolverDnDgetrf, 8);
  rb_define_singleton_method(CuSolver, "cusolverDnDlaswp",
(METHOD)rb_cusolverDnDlaswp, 8);
  rb_define_singleton_method(CuSolver, "cusolverDnDgetrs",
(METHOD)rb_cusolverDnDgetrs, 10);
```

```
  rb_define_singleton_method(CuSolver, "cusolverDnDgeqrf",
(METHOD)rb_cusolverDnDgeqrf, 9);
```

Implementation of `cusolverDnDpotrs` is as follows:

```
static VALUE rb_cusolverDnDpotrs(VALUE self, VALUE handler_val, VALUE uplo, VALUE n,
VALUE nrhs, VALUE A, VALUE lda, VALUE B, VALUE ldb, VALUE devInfo){
  rb_cusolver_handle* handler;
  Data_Get_Struct(handler_val, rb_cusolver_handle, handler);

  dev_ptr* ptr_A;
  dev_ptr* ptr_B;
  Data_Get_Struct(A, dev_ptr, ptr_A);
  Data_Get_Struct(B, dev_ptr, ptr_B);

  int dev_info = NUM2INT(devInfo);

  cusolverStatus_t status = cusolverDnDpotrs(handler->handle,
rbcu_cublasFillMode_t(uplo),
                                           NUM2INT(n), NUM2INT(nrhs),
                                             ptr_A->carray, NUM2INT(lda),
                                             ptr_B->carray, NUM2INT(ldb),
&dev_info);

  return Qnil;
}
```

CuRand

```
CuRand = rb_define_module_under(RbCUDA, "CuRand");
  rb_define_singleton_method(CuRand, "curandCreateGenerator",
(METHOD)rb_curandCreateGenerator, 1);
  rb_define_singleton_method(CuRand, "curandCreateGeneratorHost",
(METHOD)rb_curandCreateGeneratorHost, 1);
  rb_define_singleton_method(CuRand, "curandDestroyGenerator",
(METHOD)rb_curandDestroyGenerator, 1);
  rb_define_singleton_method(CuRand, "curandGetVersion",
(METHOD)rb_curandGetVersion, 0);
  rb_define_singleton_method(CuRand, "curandSetStream", (METHOD)rb_curandSetStream,
2);
  rb_define_singleton_method(CuRand, "curandSetPseudoRandomGeneratorSeed",
(METHOD)rb_curandSetPseudoRandomGeneratorSeed, 2);
  rb_define_singleton_method(CuRand, "curandSetGeneratorOffset",
(METHOD)rb_curandSetGeneratorOffset, 2);
  rb_define_singleton_method(CuRand, "curandSetGeneratorOrdering",
(METHOD)rb_curandSetGeneratorOrdering, 2);
  rb_define_singleton_method(CuRand, "curandSetQuasiRandomGeneratorDimensions",
(METHOD)rb_curandSetQuasiRandomGeneratorDimensions, 2);
  rb_define_singleton_method(CuRand, "curandGenerate", (METHOD)rb_curandGenerate,
3);
  rb_define_singleton_method(CuRand, "curandGenerateLongLong",
(METHOD)rb_curandGenerateLongLong, 3);
```

```
  rb_define_singleton_method(CuRand, "curandGenerateUniform",
(METHOD)rb_curandGenerateUniform, 3);
  rb_define_singleton_method(CuRand, "curandGenerateUniformDouble",
(METHOD)rb_curandGenerateUniformDouble, 3);
  rb_define_singleton_method(CuRand, "curandGenerateNormal",
(METHOD)rb_curandGenerateNormal, 5);
  rb_define_singleton_method(CuRand, "curandGenerateNormalDouble",
(METHOD)rb_curandGenerateNormalDouble, 5);
  rb_define_singleton_method(CuRand, "curandGenerateLogNormal",
(METHOD)rb_curandGenerateLogNormal, 5);
  rb_define_singleton_method(CuRand, "curandGenerateLogNormalDouble",
(METHOD)rb_curandGenerateLogNormalDouble, 5);
  rb_define_singleton_method(CuRand, "curandCreatePoissonDistribution",
(METHOD)rb_curandCreatePoissonDistribution, 1);
  rb_define_singleton_method(CuRand, "curandDestroyDistribution",
(METHOD)rb_curandDestroyDistribution, 1);
```

Implementation of `curandCreateGenerator` is as follows:

```
static VALUE rb_curandCreateGenerator(VALUE self, VALUE rng_type){
  rb_curand_generator* generator_ptr = ALLOC(rb_curand_generator);
  curandStatus_t status = curandCreateGenerator(&generator_ptr->generator,
                                                rbcu_rand_rng_type(rng_type));

  return Data_Wrap_Struct(CuRandGenerator, NULL, rbcu_free, generator_ptr);
}
```

Interfaces

```
  cNMatrix = rb_define_class("NMatrix", rb_cObject);
  rb_define_method(cNMatrix, "to_dev_array", (METHOD)rb_nmatrix_to_gpu_ary_method,
0);

  cNArray = rb_define_class("NArray", rb_cObject);
  rb_define_method(cNArray, "to_dev_array", (METHOD)rb_narray_to_gpu_ary_method,
0);
```

Implementation of `rb_nmatrix_to_gpu_ary_method` is as follows:

```
extern VALUE rb_nmatrix_to_gpu_ary_method(VALUE nmatrix) {
  if (NM_DIM(nmatrix) > 3) {
    rb_raise(rb_eStandardError,
      "NMatrix must not have greater than 4 dimensions.");
  }

  if (NM_DTYPE(nmatrix) == nm::FLOAT64) {
    return Data_Wrap_Struct(Dev_Array, NULL, rbcu_free,
rb_nmatrix_to_dev_ary(nmatrix));
  }
  else {
    rb_raise(rb_eStandardError,
```

```
      "NMatrix should be either :complex64, :complex128, :int32 or :float64
type.");
  }
  return Qnil;
}


dev_ptr* rb_nmatrix_to_dev_ary(VALUE nm) {
  DENSE_STORAGE* nmat = NM_STORAGE_DENSE(nm);
  dev_ptr* ptr = ALLOC(dev_ptr);

  if (nmat->dtype != nm::FLOAT64) {
    rb_raise(rb_eStandardError, "requires dtype of :float64 to convert to an
Af_Array");
  }

  cudaMalloc((void **)&ptr->carray, sizeof(double) * nmat->count);
  cudaMemcpy((void*)ptr->carray, (void*)nmat->elements, sizeof(double) *
nmat->count, cudaMemcpyHostToDevice);

  return ptr;
}
```

The method `rb_narray_to_gpu_ary_method` is not ready at the the time of writing this report and will be ready soon.

Conclusion

RbCUDA is a huge project. I have successfully added the bindings to CUDA Driver and Runtime APIs. CuBLAS and CuBLASXT apis were added successfully for double dtype arrays. CuSolver and CuRand APIs were added that will help in matrix factorization and generating an array of random numbers. CUDA Profiler has been added successfully. NMatrix has been interfaced to RbCUDA. NArray interface is currently in progress.

Future Work

I have added CuBLAS, Profiler and CUDA driver usage examples. The test suite is missing and it will be done now along with code refactoring. I intend to polish this project and move it under SciRuby projects domain.

Talks and Awards

1. Invited as a Speaker at RubyKaigi 2018, Sendai, Japan to talk about High Performance GPU Computing on Ruby
2. Speaker at Ruby Conference India, 2018 to talk about GPU Accelerated Libraries for Ruby.
3. Speaker at RubyConf 2017, New Orleans, USA to talk about High Performance GPU Computing on Ruby.
4. Qualified for Fukuoka Ruby Award 2018 finals and was the winner of 'GMO Pepabo Award'.

Acknowledgements

I would like to thank my mentor Kenta Murata for guiding me developing the project.

I am very thakful to Ruby Association for funding the development of this project.