Standard libraries and acceleration

Igor Sfiligoi – San Diego Supercomputing Center (UCSD)

Optimization is hard

- Properly optimizing parallel code is hard!
  - Compute vs memory tradeoffs
  - Generic vs specialized algorithms
  - Use of HW-specific instructions
- Can take FTE years to get max performance out of specific HW
  - And may need a completely different approach for different HW!

Optimization is hard

- Properly optimizing parallel code is hard!
  - Compute vs memory tradeoffs
  - Generic vs specialized algorithms
  - Use of HW-specific instructions
- Can take FTE years to get max performance out of specific HW
  - And may need a completely different approach for different HW!
- **Many “standard problems” thus have vendor-provided libraries**
  - Standardized API
  - Different HW vendors may have completely different backend implementations (often with model-specific optimizations)

Most often used standard libraries

- BLAS/LAPACK – Linear algebra
- FFTW – Linear algebra
- ML/AI tensor libraries

Most often used standard libraries

- BLAS/LAPACK – Linear algebra
- FFTW – Linear algebra
- ML/AI tensor libraries

Originally developed for CPUs

Most have Open Source implementations.
But vendors often provide optimized versions, too (e.g. INTEL MKL)

Most often used standard libraries

- GPU massive parallelization, but also function calling limits, required slight change in API
  - E.g., batching
  - NVIDIA leading with a large set of libraries
  - Others following

	NVIDIA	AMD
Math Libraries	cuBLAS	rocBLAS
	cuBLASLt	hipBLASLt
	cuFFT	rocFFT
	cuSOLVER	rocSOLVER
	cuSPARSE	rocSPARSE
	cuRAND	rocRAND
Communication Library	NCCL	RCCL
C++ Core library	Thrust	rocThrust
	CUB	hipCUB
Wave Matrix Multiply Accumulate Library	WMMA	rocWMMA
Deep Learning / Machine Learning primitives	cuDNN	MIOpen
C++ templates abstraction for GEMMs	CUTLASS	Composable Kernel

cuTENSOR

Tensor Linear Algebra on NVIDIA GPUs

Many fields have their own libraries

- Most fields have specialized “standard libraries”
  - Specific for their science/engineering domain
- I will not try to guess what is most relevant to you

- Ask your peers and use search engines
  - Most likely than not, someone had a similar problem and implemented it
  - But make sure it is a supported version (lots of abandonware/prototypes out there)
  - And was developed with large-scale compute in mind
    (Reminder: What works great on a toy problem may be extra slow when scaled)

When **not** to use standard libraries

- No library for your specific problem
- Repeatedly solving a trivial problem (but make sure it stays trivial as you scale)
    - Data transforms and invocation may be more expensive than execution
    - Especially on GPUs
- Requires many steps using standard libraries
    - The intermediate buffers may make it memory-bound
    - If you have the know-how, try to fuse all operations in an inner loop
- Simple functionality outside the critical path
    - Carrying dependencies can be a problem, long term

GPU limits make it especially tricky

- Reminder:
  Hard to invoke external functions from GPU code
    - That includes standard libraries, unless they can be inlined
- Most “heavy” standard libraries must be invoked from CPU code
    - Requiring coordination between CPU and GPU code paths
    - Potentially splitting GPU code in many smaller sections
        - Adding to the GPU invocation costs
- Not a problem when std.lib. solves a big problem
    - But can be problematic if having many small problems

GPU limits make it especially tricky

- Reminder:
  Hard to invoke external functions from GPU code
  - That includes standard libraries, unless they can be inlined
- Most “heavy” standard libraries must be invoked from CPU code
  - Requiring coordination between CPU and GPU code paths
  - Potentially splitting GPU code in many smaller sec
    - Adding to the GPU invocation cost
- Not a problem when std.lib. so
  - But can be problematic if having

Even though the CPU invokes the std.lib., the buffers must be in GPU memory.
(although some libraries can do it for you, at high cost)

Many GPU standard libraries allow for batching

- cuFFT is a textbook example
- Original FFTW logic was that you used one CPU thread per “small problem”
- cuFFT added concept of ”batch of small problems”
    - Single invocation
    - Backend spreads the problem
    - FFTW since added the capability for CPUs, too

```
plan = fftw_plan_dft_1d(M, …)
#pragma omp parallel for
  for (int i=0; i<N; i++) {
    fftw_execute_dft_c2r(plan, A(:,i), B(:,i))
}
```

```
cufftPlanMany(plan,M,N, …)
cufftExecZ2D(plan,A(:,:),B(:,:))
```

Many GPU standard libraries allow for batching

- cuFFT is a textbook example
- Original FFTW logic was that you used one CPU thread per “small problem”
- cuFFT added concept of ”batch of small problems”
  - Single invocation
  - Backend spreads the problem
  - FFTW since added the capability for CPUs, too
- But may add some memory overhead for repeat invocations

```
plan = fftw_plan_dft_1d(M, …)
#pragma omp parallel for
  for (int i=0; i<N; i++) {
    fftw_execute_dft_c2r(plan, A(:,i), B(:,i));
    …
    fftw_execute_dft_r2c(plan2, B(:,i), C(:,i))
}
```

```
cufftPlanMany(plan,M,N, …)
cufftExecZ2D(plan,A(:,:),B(:,:))
…
cufftExecD2Z(plan,B(:,:),C(:,:))
```

Ready for some more hands-on?