HPC Libraries and Frameworks

John E. Stone

Theoretical and Computational Biophysics Group

Beckman Institute for Advanced Science and Technology

University of Illinois at Urbana-Champaign

http://www.ks.uiuc.edu/~johns/

Petascale Computing Institute,

National Center for Supercomputing Applications,

University of Illinois at Urbana-Champaign

Overview

- Benefits of libraries and frameworks
- Exemplary libraries
- Open vs. proprietary/commercial
- State-of-the-art library technologies
- Trade-offs, costs, and limitations of libraries and frameworks
- Library usage do's and don'ts

Major Types and Uses of Libraries and Frameworks

- Avoid “reinventing the wheel”:
  - Mathematical functions
  - Data structures, containers, serialization and I/O
  - Algorithms
- Hardware-optimized: abstract hardware-specific implementation
- Callable from C, C++, Fortran, Python, etc.

Role of HPC Libraries and Frameworks in Software Dev. Cycle

- Use libraries/frameworks to fill **software "gap"**
- **Profiling** to identify **performance bottlenecks**
- Find HPC libraries or algorithm frameworks covering gaps

Benefits from Using Libraries and Frameworks

- **Reduce application devel / maint cost**
- Use a **validated** implementation of tricky algorithms, e.g., solvers, RNGs
- **Hardware-specific optimizations, abstraction**
- Standardized or compatibility APIs allow libraries to be **dropped in, swapped, compared**

Library Examples

- Mathematical functions:
  - **Cephes**, SVML, GSL
- Linear algebra kernels and solvers:
  - BLAS, LAPACK, MAGMA, MKL, cuBLAS, cuSPARSE, SCALAPACK, ...
- Random, quasi-random number generation:
  - SPRNG, cuRAND, GSL
- Fast Fourier Transform:
  - FFTW, cuFFT, MKL

The list goes on and on…

Example: Dense Linear Algebra

- Due to the maturity and importance of linear algebra software, a thriving ecosystem of compatible and interoperable libraries and frameworks exist
- Libraries available for fundamental algorithms, higher level solvers, special hardware platforms, parallel solvers…
- Compatible and interoperable APIs

Dense Linear Algebra

- BLAS – Fundamental dense linear algebra
  - Level 1: Vector-Vector
  - Level 2: Matrix-Vector
  - Level 3: Matrix-Matrix
- LAPACK – Matrix solvers based on BLAS
  - Linear equations, eigenvalue problems, …
  - Matrix factorization: LU, QR, SVD, Cholesky, ...
- SCALAPACK – Parallel LAPACK
  - Extended distributed memory message passing APIs

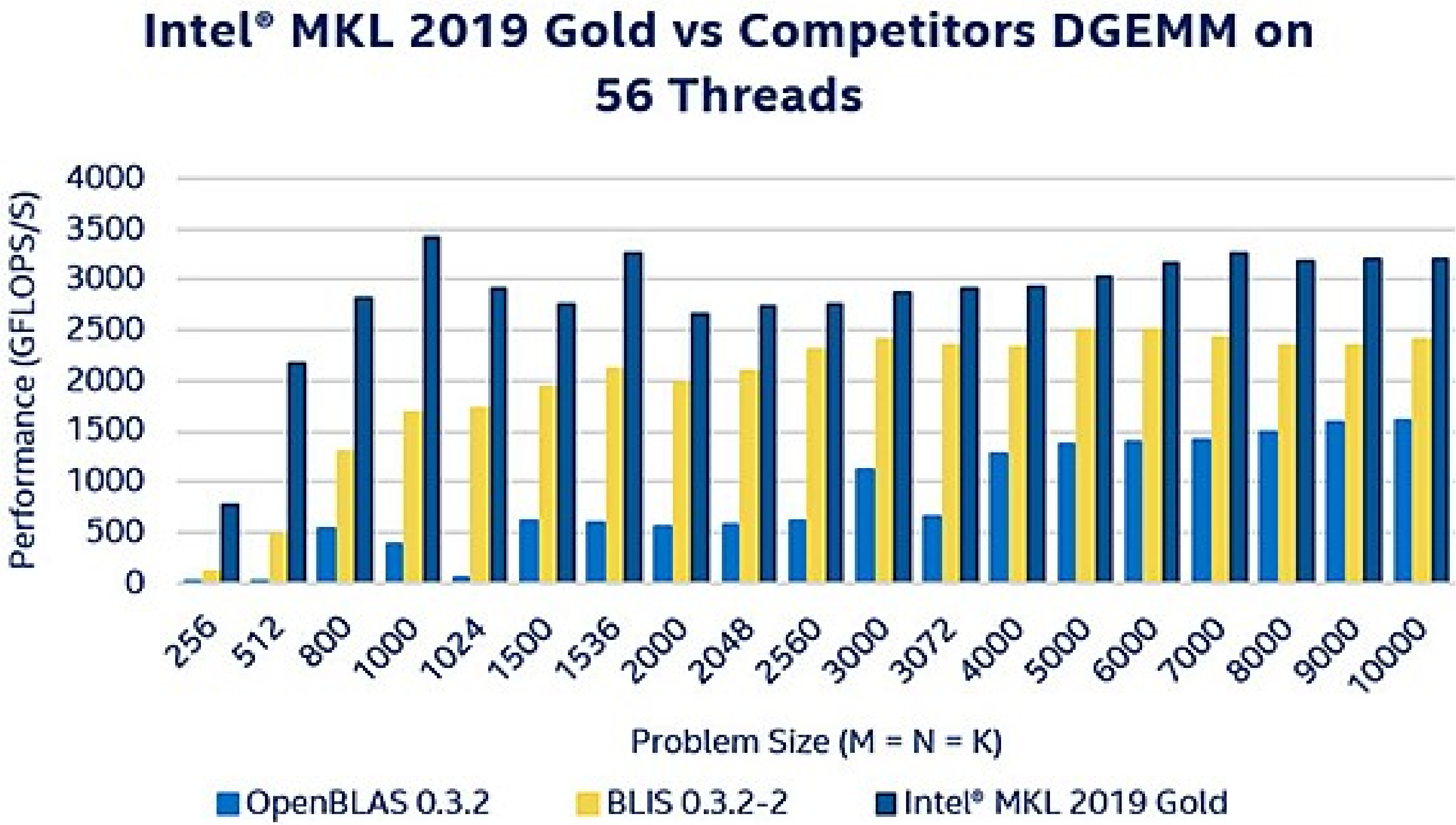
Evaluating Libraries

- Accuracy, correctness, robustness against failure(s)
- Performance
- Standard or compatible APIs
- Language bindings
- Portability, Composability, and Hardware Support
  - Thread-safe? Interferes with MPI_COMM_WORLD?
  - Compatible with OpenMP, OpenACC, CUDA, etc?
- Built-in parallelization?:
  - Intra-node (multi-core CPUs, GPUs)
  - Distributed memory

Open vs. Proprietary, Free vs. Commercial

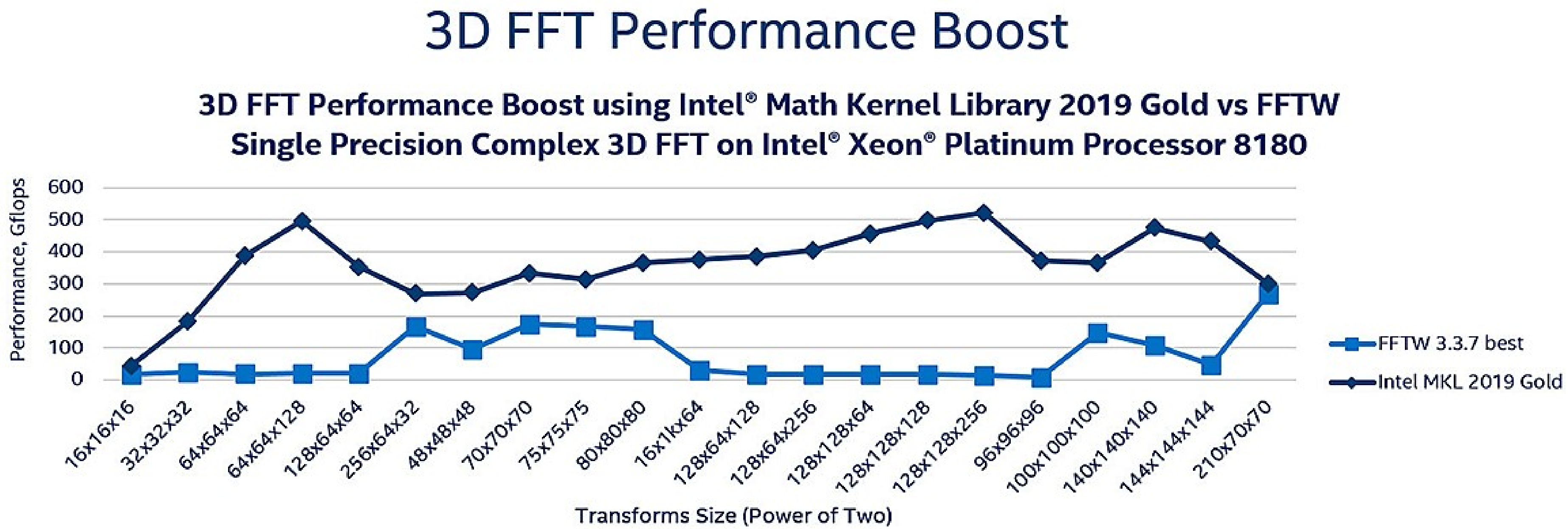
- Open libs ideal for gaining deep understanding of performance limitations imposed by APIs, application usage
- **Hardware vendor libs** try to provide optimal performance, approaching **"speed of light"** for their own platform
- Commercially licensed libs may present application distribution challenges in terms of price, ultimate scalability, etc.

Example of Proprietary Lib Perf.



https://software.intel.com/en-us/mkl/features/benchmarks

Example of Proprietary Lib Perf.



https://software.intel.com/en-us/mkl/features/benchmarks

Libraries vs. Frameworks

- Libraries typically "canned", not much caller-specialization possible
  - Example: Matrix Multiply
  - **Caller runs the code**
- Frameworks combine some existing code with caller-provided code to achieve application-specific functionality
  - Example: PETsc, AI stacks, OptiX Ray Tracing
  - **Framework typically runs the code**

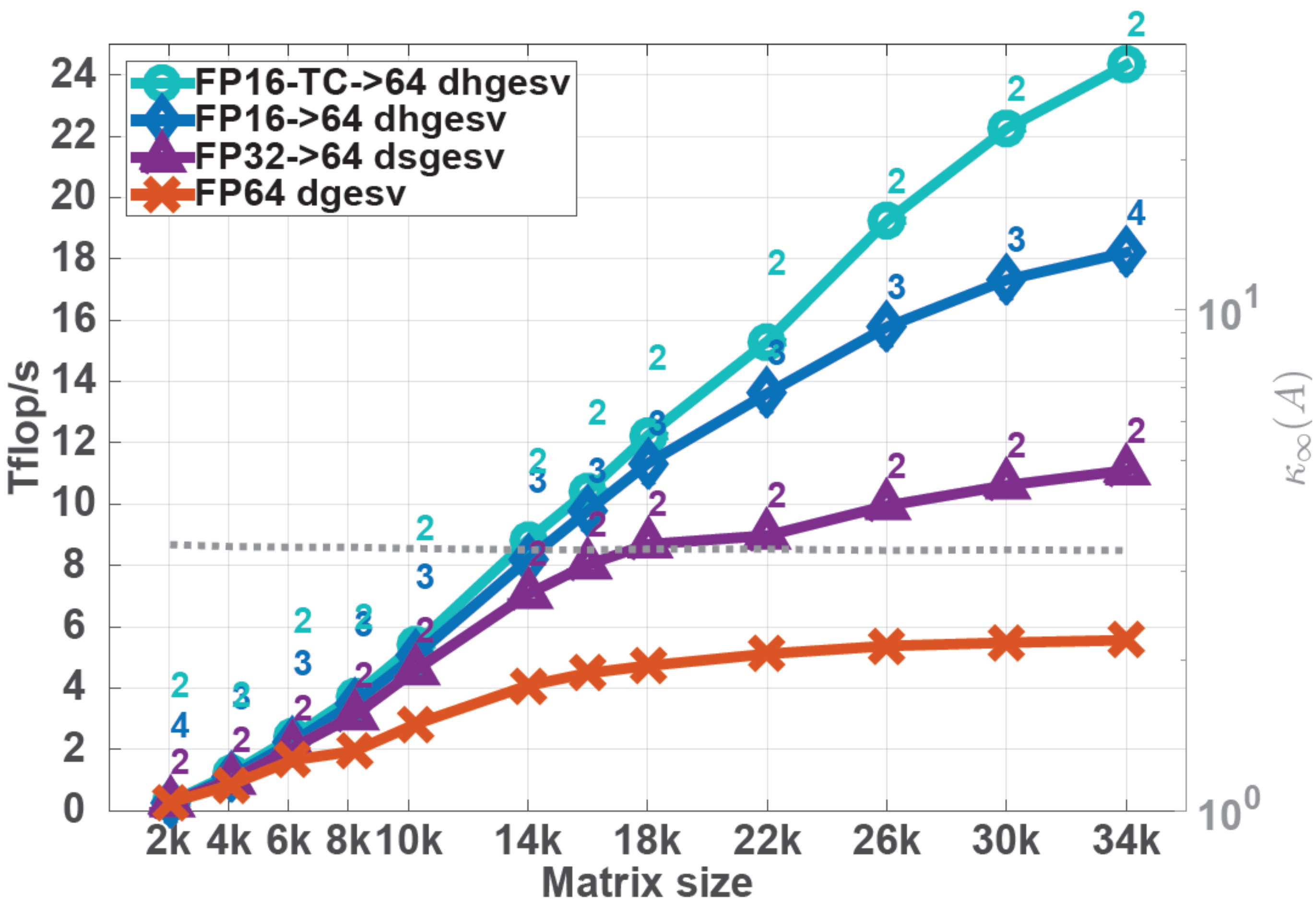
C++ Template Libraries

- Potential for generality across many types/classes
- Performance opportunities:
  - Template specialization, template metaprogramming
  - Compile-time optimization of per-thread ops by constant folding, loop unrolling, etc.
- Eigen linear algebra template library
- NVIDIA GPU accelerated template libraries:
  - **Thrust:** STL-like vector ops on GPUs (incl sort/scan)
  - **CUB:** per-block, device-wide sort/scan/reductions/etc
  - **CUTLASS:** matrix linear algebra ops

Exploit New Hardware and Algorithmic Advances

- Library abstraction allows replacement of conventional solver with iterative refinement
- Mixed precision solvers, e.g. half-, single-, double-precision
- Example: Make use of special purpose hardware such as NVIDIA Tensor cores for higher performance dense linear algebra…

"Harnessing GPU Tensor Cores for Fast FP16 Arithmetic to Speed up Mixed-Precision Iterative Refinement Solvers", Haidar et al., SC2018



(a) Matrix of type 1: diagonally dominant.

State-of-the-Art Library Runtime Technologies

- Runtime dispatch of hardware-optimized code paths: MKL, CUDA Libraries
- Autotuners: FFTW "Plan"
- Built-in runtime systems for scheduling work in complex multi-phase parallel algorithms, heterogeneous platforms: MAGMA (UTK)

Library Performance Considerations

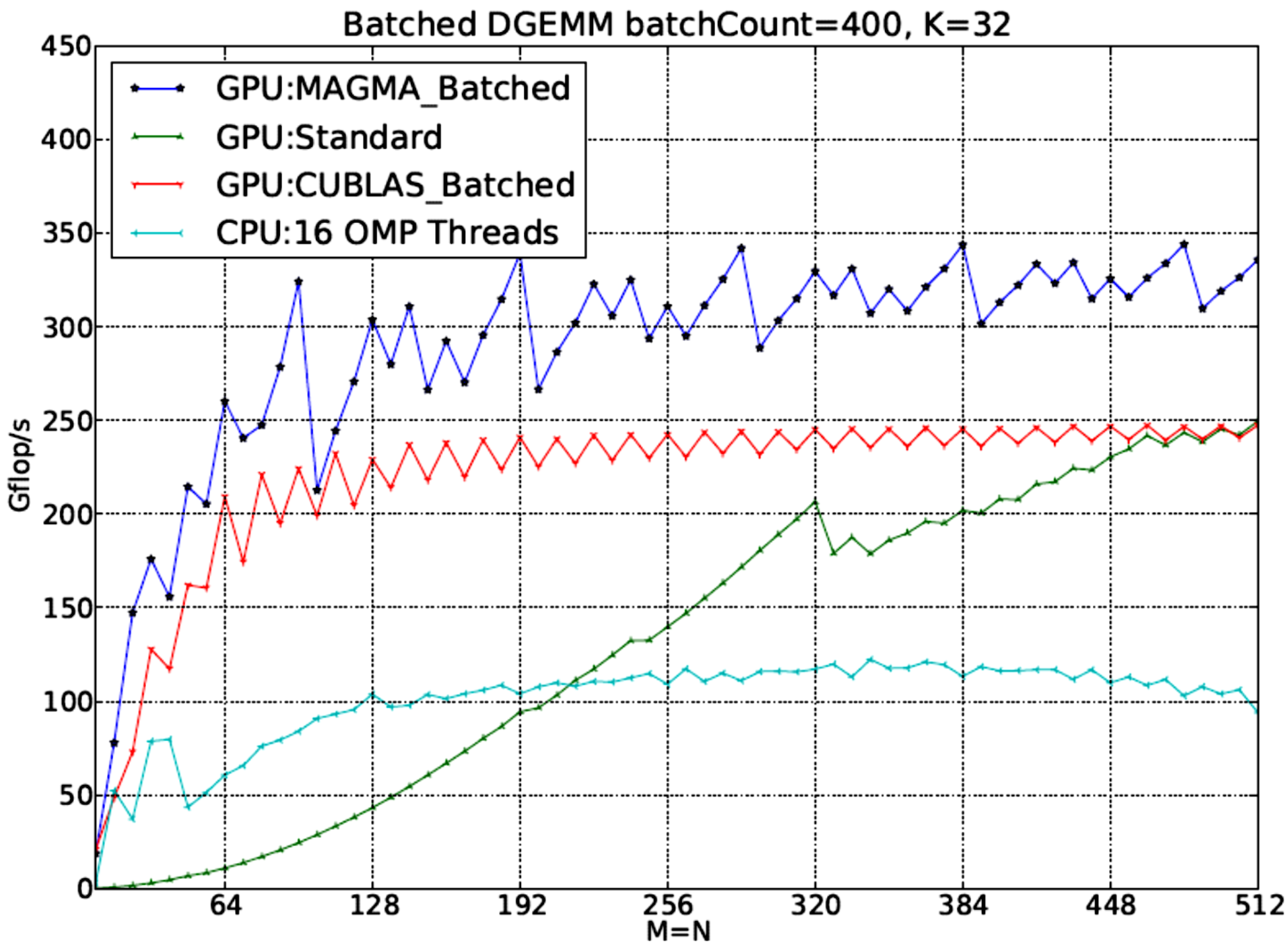
- How does library perform with varying problem size?
- Libraries may provide special APIs for **batching of large numbers of "small problems**
- May have significant startup cost:
  - Autotuners
  - JIT code generators
  - GPUs or other accelerators

Improving Performance with Batching APIs

- Trivial example:
  - Replace separate sin() and cos() calls with **sincos()** (C99 math lib standard)
  - Input angle **domain checking logic is amortized**, approach ~2x speedup
- Mainstream examples:
  - FFTW, MKL, cuFFT batched FFTs
  - MAGMA:

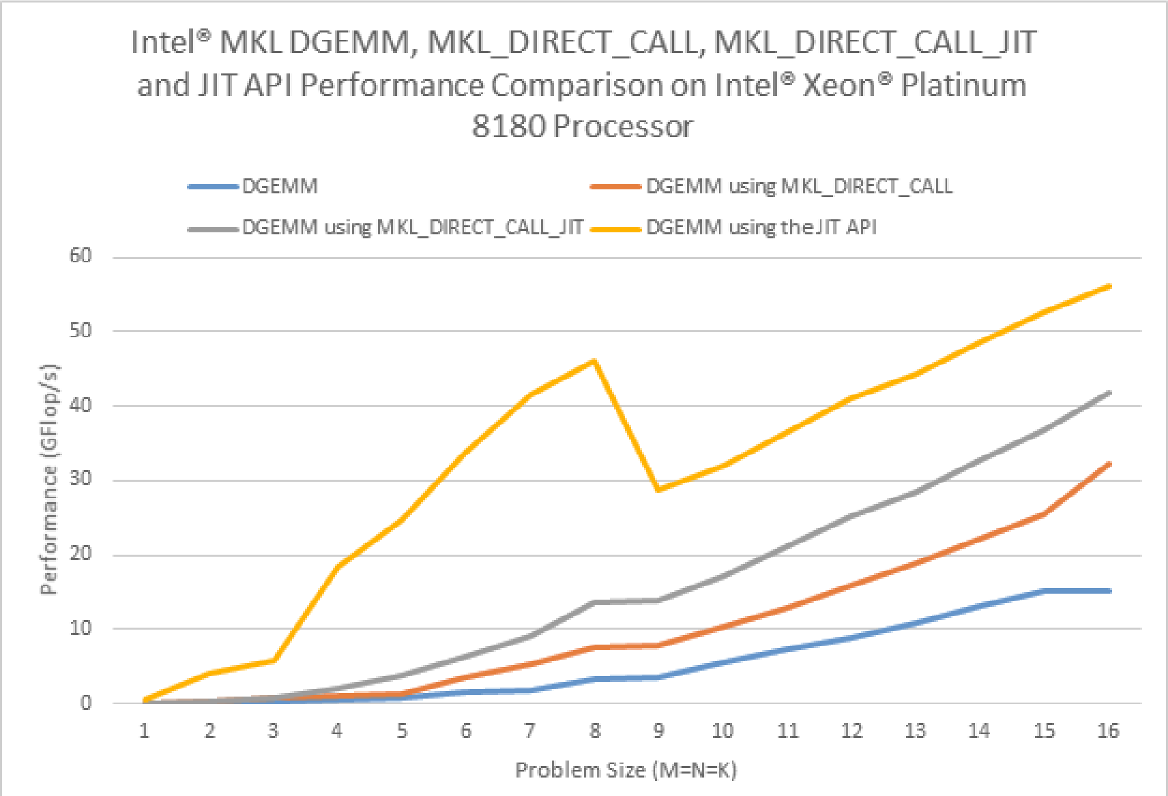
"MAGMA Batched: A Batched BLAS Approach for Small Matrix Factorizations and Applications on GPUs", Dong et al., ICL Tech Rep. 2016

MAGMA: GPU Batched DGEMM



"MAGMA Batched: A Batched BLAS Approach for Small Matrix Factorizations and Applications on GPUs", Dong et al., ICL Tech Rep. 2016

JIT Code Generation for Large Repetitions (1000x) of Small Problem Sizes

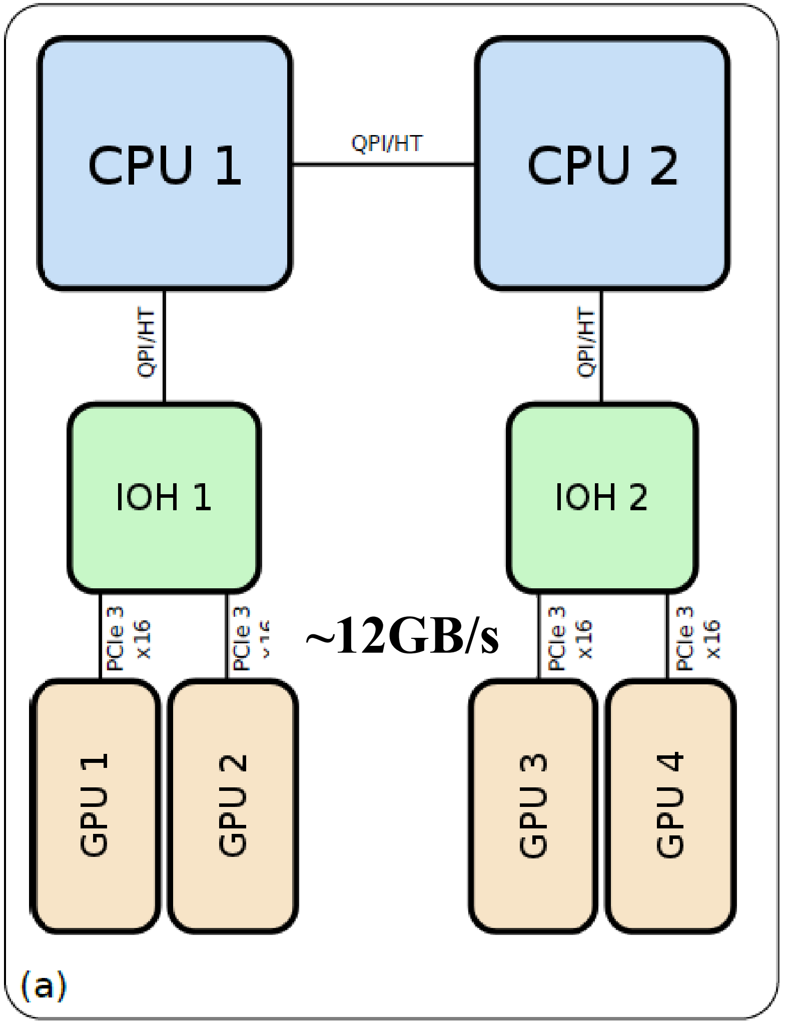


https://software.intel.com/en-us/articles/intel-math-kernel-library-improved-small-matrix-performance-using-just-in-time-jit-code

Heterogeneous Compute Node

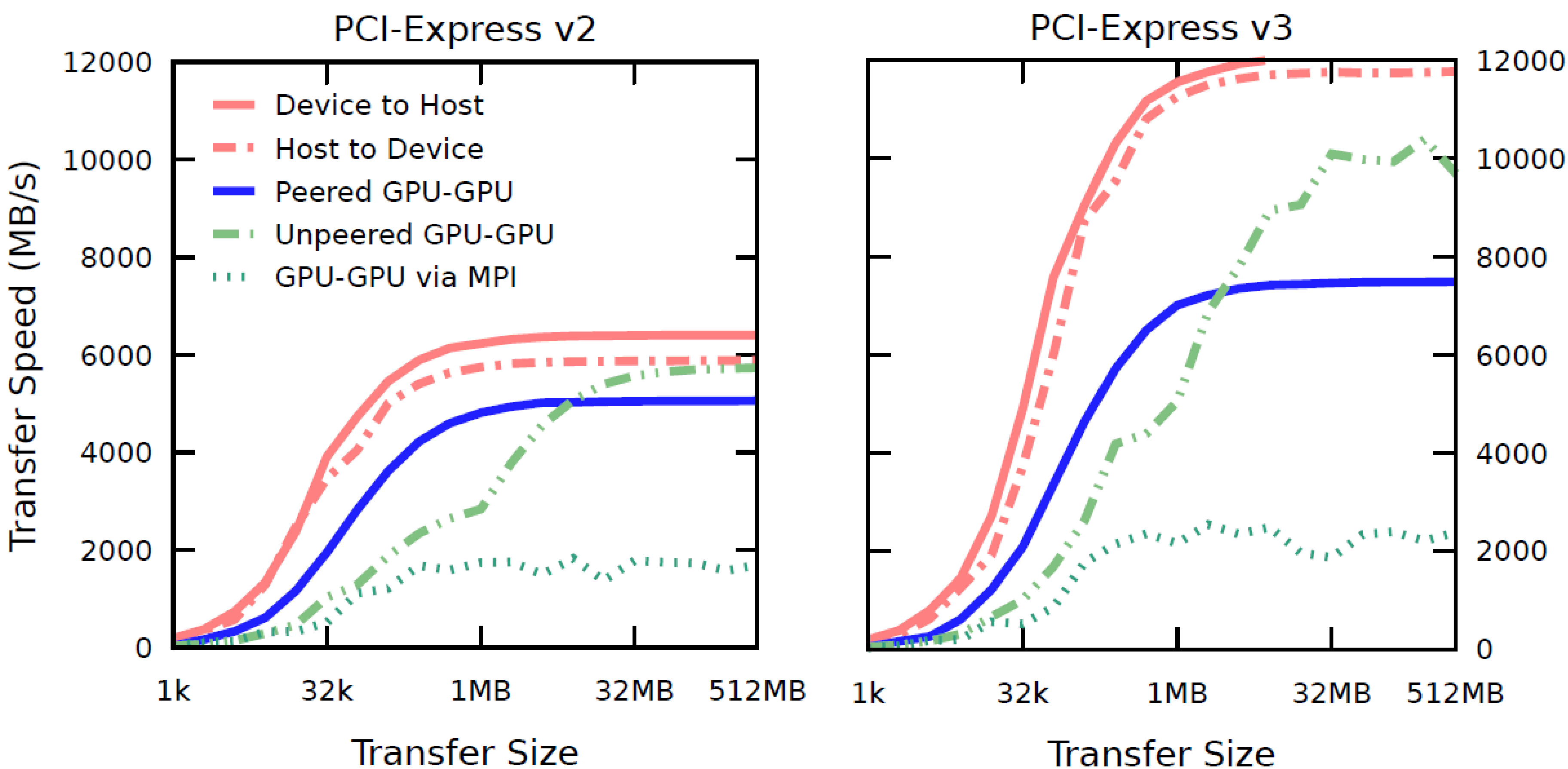
- **NUMA CPU architecture**
- Dense **PCIe-based** multi-GPU compute node
- Application would **ideally exploit all** of the CPU, GPU, and I/O resources **concurrently**…

  (I/O devs not shown)



Simulation of reaction diffusion processes over biologically relevant size and time scales using multi-GPU workstations
Michael J. Hallock, John E. Stone, Elijah Roberts, Corey Fry, and Zaida Luthey-Schulten.
Journal of Parallel Computing, 40:86-99, 2014.
http://dx.doi.org/10.1016/j.parco.2014.03.009

GPU PCI-Express DMA



Simulation of reaction diffusion processes over biologically relevant size and time scales using multi-GPU workstations
Michael J. Hallock, John E. Stone, Elijah Roberts, Corey Fry, and Zaida Luthey-Schulten.
Journal of Parallel Computing, 2014. **(In press)**
http://dx.doi.org/10.1016/j.parco.2014.03.009

Exemplary Heterogeneous Computing Challenges

- **Tuning, adapting, or developing software for multiple processor types**
- Decomposition of problem(s) and load balancing work across heterogeneous resources for **best overall performance and work-efficiency**
- **Managing data placement** in **disjoint memory** systems with **varying performance attributes**
- **Transferring data** between processors, memory systems, interconnect, and I/O devices
- …

Using Libraries for Programming Heterogeneous Computing Architectures

- Use **drop-in libraries** in place of CPU libs
  - **Little or no code development**
  - Examples: MAGMA, cuBLAS, cuSPARSE, cuSOLVER, cuFFT libraries, many more…
  - **Speedups limited by Amdahl's Law** and overheads associated with data movement between CPUs and GPUs

https://developer.nvidia.com/gpu-accelerated-libraries

Costs, Limitations, Arising from Using Libraries and Frameworks

- Lib API may require inconvenient data layout
- Lib API boundaries inhibit inlining of small functions, and prevent “kernel fusion”
- Lib implementation may sacrifice some performance to ensure generality
- Too many library dependencies create challenge for source compilation, e.g., for MPI codes

Libraries May Sacrifice Performance to Ensure Generality

- Math lib functions do significant preprocessing and validation on input parameters for allowed function domain
- Caller may know that that input param may fall within a very limited subrange, but there's no way to exploit this in a conventional library
- **Bespoke math functions can outrun general math lib** function by significant margin for **limited input domain** or **reduced precision**

Example of Lost Vectorization or Inlining Opportunities

- Traditional math libraries don’t permit inlining of function calls into calling loop
- Significant function call overhead if the main content of loop is a library routine
- **So-called “header-only” C++ template libraries can overcome some of this**
- **Special intrinsics and short-vector math libraries** can be used to resolve cases where library calls would otherwise inhibit vectorization

MO Kernel for One Grid Point (Naive C)

```
…
for (at=0; at<numatoms; at++) {
  int prim_counter = atom_basis[at];
  calc_distances_to_atom(&atompos[at], &xdist, &ydist, &zdist, &dist2, &xdiv);
  for (contracted_gto=0.0f, shell=0; shell < num_shells_per_atom[at]; shell++) {
    int shell_type = shell_symmetry[shell_counter];
    for (prim=0; prim < num_prim_per_shell[shell_counter];  prim++) {
      float exponent       = basis_array[prim_counter     ];
      float contract_coeff = basis_array[prim_counter + 1];
      contracted_gto += contract_coeff * expf(-exponent*dist2);
      prim_counter += 2;
    }
    for (tmpshell=0.0f, j=0, zdp=1.0f; j<=shell_type; j++, zdp*=zdist) {
      int imax = shell_type - j;
      for (i=0, ydp=1.0f, xdp=pow(xdist, imax); i<=imax; i++, ydp*=ydist, xdp*=xdiv)
        tmpshell += wave_f[ifunc++] * xdp * ydp * zdp;
    }
    value += tmpshell * contracted_gto;
    shell_counter++;
  }
} …..
```

Loop over atoms

Loop over shells

Loop over primitives: largest component of runtime, due to expf()

Loop over angular momenta

(unrolled in real code)

Value of Bespoke Math Functions

- Eliminate overheads from checking / preprocessing of general input domain
- Inlinable into loop body
- Only implement caller-required numerical precision / accuracy

I/O Libraries

- I/O is now and for all time a significant concern for HPC apps
- I/O performance has plateaued at many sites
- Meanwhile, compute capabilities are growing toward exascale by leaps and bounds
- App developers need easy-to-use and performant I/O mechanisms to avoid bottlenecking
- **NetCDF and HDF5**

HDF5, NetCDF I/O Libraries

- Bindings for all major languages
- Lots of documentation and examples
- Easy to use for many HPC tools
  - Cross-platform portability, conversion of byte order to native endianism, etc.
  - HDF5 supports compression
  - User defined data blocks, organization
  - Makes it easy to author both the output code for a simulation tool and the matching input code for analysis and visualization usage
- Support integration with MPI-I/O for parallel I/O

HPC Graphics and Visualization

- Visualization:
  - VTK, VTK-m
- Rasterization: EGL and Vulkan
- Ray tracing:
  - Intel OSPRay CPU Ray Tracing Framework
  - NVIDIA OptiX GPU Ray Tracing Framework
  - Research: NVIDIA VisRTX Framework

Library "DOs"

- **Do use standardized, interoperable library APIs**
- **Do use libraries to exploit GPU accelerators**, new hardware features, new algorithms
- **Do use high level APIs**, abstractions, allow library freedom to use most efficient back-end solver
- **Do use batched APIs** for large numbers of small size problems
- **Do use Autotuning and JIT** when workload is repeated and overheads can be amortized

Library "DON'Ts"

- **Don't** use a library without considering whether it creates an avoidable obstacle to software compilation, redistribution, or usages
- **Don't** use a library or framework that harms long-term portability for short-term gain, always **leave yourself an "out" for future systems**
- **Don't** continue using a library indefinitely -- **periodically do a "bake-off"** to see how it **compares** with other **state-of-the-art choices**

Keep and Eye Out For

- New state-of-the-art libraries and frameworks arising from DOE Exascale Computing Project funding
- Ongoing advances by major library developers, hardware vendors
- Evolving and improving interoperability, compatibility APIs to ease porting

Questions?