Another year of progress for BLIS: 2017-2018

Field G. Van Zee

Science of High Performance Computing

The University of Texas at Austin

Science of High Performance Computing (SHPC) research group

- Led by Robert A. van de Geijn
- Contributes to the science of DLA and instantiates research results as open source software
- Long history of support from National Science Foundation
- Website: https://shpc.ices.utexas.edu/

SHPC Funding (BLIS)

- NSF
  - Award ACI-1148125/1340293: *SI2-SSI: A Linear Algebra Software Infrastructure for Sustained Innovation in Computational Chemistry and other Sciences.* (Funded June 1, 2012 - May 31, 2015.)
  - Award CCF-1320112: *SHF: Small: From Matrix Computations to Tensor Computations.* (Funded August 1, 2013 - July 31, 2016.)
  - Award ACI-1550493: *SI2-SSI: Sustaining Innovation in the Linear Algebra Software Stack for Computational Chemistry and other Sciences*. (Funded July 15, 2016 – June 30, 2018.)

SHPC Funding (BLIS)

- Industry (grants and hardware)
  - Microsoft
  - Texas Instruments
  - Intel
  - AMD
  - HP Enterprise
  - Oracle
  - Huawei

Publications

- *"BLIS: A Framework for Rapid Instantiation of BLAS Functionality"* (TOMS; in print)
- *"The BLIS Framework: Experiments in Portability" (TOMS; in print)*
- *"Anatomy of Many-Threaded Matrix Multiplication"* (IPDPS; in proceedings)
- *"Analytical Models for the BLIS Framework"* (TOMS; in print)
- *"Implementing High-Performance Complex Matrix Multiplication via the 3m and 4m Methods"* (TOMS; in print)
- *"Implementing High-Performance Complex Matrix Multiplication via the 1m Method"* (TOMS; accepted pending modifications)

Review

- BLAS: Basic Linear Algebra Subprograms
  - Level 1: vector-vector [Lawson et al. 1979]
  - Level 2: matrix-vector [Dongarra et al. 1988]
  - Level 3: matrix-matrix [Dongarra et al. 1990]
- Why are BLAS important?
  - BLAS constitute the "bottom of the food chain" for most dense linear algebra applications, as well as other HPC libraries
  - LAPACK, `libflame`, MATLAB, PETSc, numpy, gsl, etc.

Review

- What is BLIS?
  - A framework for instantiating BLAS libraries (ie: fully compatible with BLAS)
- What else is BLIS?
  - Provides alternative BLAS-like (C friendly) API that fixes deficiencies in original BLAS
  - Provides an object-based API
  - Provides a superset of BLAS functionality
  - A productivity multiplier
  - A research environment

Review: Where were we a year ago?

- License: 3-clause BSD
- Most recent version: 0.4.1 (August 30)
- Host: https://github.com/flame/blis
  - Clone repositories, open new issues, submit pull requests, interact with other github users, view markdown docs
- GNU-like build system
  - Support for `gcc`, `clang`, `icc`
- Configure-time hardware detection (`cpuid`)

Review: Where were we a year ago?

- BLAS / CBLAS compatibility layers
- Two native APIs
  - Typed (BLAS-like)
  - Object-based (libflame-like)
- Support for level-3 multithreading
  - via OpenMP or POSIX threads
  - Quadratic partitioning: herk, syrk, her2k, syr2k, trmm
- Comprehensive test suite
  - Control operations, parameters, problem sizes, datatypes, storage formats, and more

So What's New?

- Five broad categories
  - Framework
  - Kernels
  - Build system
  - Testing
  - Documentation

So What's New?

- Five broad categories
  - Framework
  - Kernels
  - Build system
  - Testing
  - Documentation

Runtime kernel management

- Runtime management of configurations (kernels, blocksizes, etc.)
  - Rewritten/generalized configuration system
  - Allows multi-configuration builds (“fat” libraries)
    - CPUID used at runtime to choose between targets
  - Examples:
    - `./configure intel64`
    - `./configure x86_64`
    - `./configure haswell   # still works`
  - Or define your own!
    - `./configure skx_knl   # with ~5m of work`

Runtime kernel management

- For more details:
  - `docs/ConfigurationHowTo.md`

Self-initialization

- Library self-initialization
  - Previously status quo
    - User of typed/object APIs had to call `bli_init()` prior to calling any other function or part of BLIS
    - BLAS/CBLAS were already self-initializing
  - How does it work now?
    - Typical usage of typed/object API results in exactly one thread calling `bli_init()` automatically, exactly once
    - Library stays initialized; `bli_finalize()` is optional
  - Why is this important?
    - Application doesn't have to worry anymore about whether BLIS is initialized (esp. with constants `BLIS_ZERO`, `BLIS_ONE`, etc.)
  - Implementation
    - `pthread_once()`

Basic + Expert Interfaces

- Separate “basic” and “expert” interfaces
  - applies to both typed and object APIs
- What is the difference?

Basic + Expert Interfaces

```
// Typed API (basic)
void bli_dgemm
     (
       trans_t transa,
       trans_t transb,
       dim_t   m,
       dim_t   n,
       dim_t   k,
       double* alpha,
       double* a, inc_t rsa, inc_t csa,
       double* b, inc_t rsb, inc_t csb,
       double* beta,
       double* c, inc_t rsc, inc_t csc
     );
```

```
// Object API (basic)
void bli_gemm
     (
       obj_t*  alpha,
       obj_t*  a,
       obj_t*  b,
       obj_t*  beta,
       obj_t*  c
     );
```

Basic + Expert Interfaces

```
// Typed API (expert)
void bli_dgemm_ex
     (
       trans_t transa,
       trans_t transb,
       dim_t   m,
       dim_t   n,
       dim_t   k,
       double* alpha,
       double* a, inc_t rsa, inc_t csa,
       double* b, inc_t rsb, inc_t csb,
       double* beta,
       double* c, inc_t rsc, inc_t csc,
       cntx_t* cntx,
       rntm_t* rntm
     );
```

```
// Object API (expert)
void bli_gemm_ex
     (
       obj_t*  alpha,
       obj_t*  a,
       obj_t*  b,
       obj_t*  beta,
       obj_t*  c,
       cntx_t* cntx,
       rntm_t* rntm
     );
```

Basic + Expert Interfaces

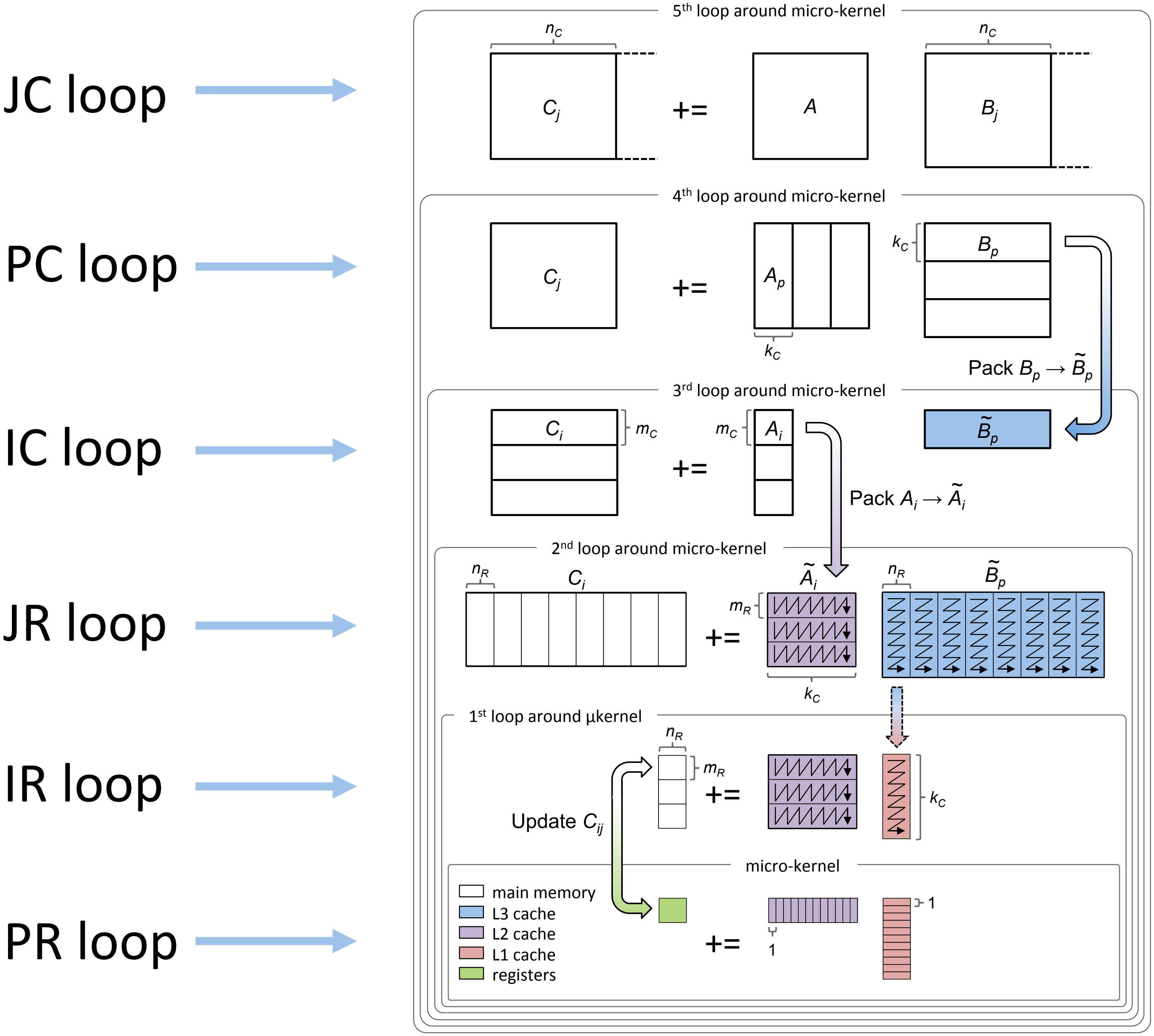
- What are cntx_t and rntm_t?
  - `cntx_t`: context encapsulates all architecture-specific information obtained from the build system about the configuration (blocksizes, kernel addresses, etc.)
  - `rntm_t`: more on this in a bit
  - Bottom line: experts can exert more control over BLIS without impeding everyday users

Basic + Expert Interfaces

- For more details:
  - `docs/BLISTypedAPI.md`
  - `docs/BLISObjectAPI.md`

Controlling Multithreading

- Reminder
  - How does multithreading work in BLIS?
  - BLIS's gemm algorithm has five loops outside the microkernel and one loop inside the microkernel
    - JC
    - PC (not yet parallelized)
    - IC
    - JR
    - IR
    - PR (microkernel)



JC loop

PC loop

IC loop

JR loop

IR loop

PR loop

Controlling Multithreading

- Previously, BLIS had one method to control threading: Global specification via environment variables
  - Affects all application threads equally
  - Automatic way
    - `BLIS_NUM_THREADS`
  - Manual way
    - `BLIS_JC_NT`, `BLIS_IC_NT`, `BLIS_JR_NT`, `BLIS_IR_NT`
    - `BLIS_PC_NT` (not yet implemented)

Controlling Multithreading

- Example: Global specification via environment variables

```
# Use either the automatic way or manual way of requesting
# parallelism.

# Automatic way.
$ export BLIS_NUM_THREADS = 6

# Expert way.
$ export BLIS_IC_NT = 2; export BLIS_JR_NT = 3

// Call a level-3 operation (basic interface is enough).
bli_gemm( &alpha, &a, &b, &beta, &c );
```

Controlling Multithreading

- We now have a second method: Global specification via runtime API
  - Affects all application threads equally
  - Automatic way
    - `bli_thread_set_num_threads( dim_t nt );`
  - Manual way
    - `bli_thread_set_ways( dim_t jc, dim_t pc, dim_t ic, dim_t jr, dim_t ir );`

Controlling Multithreading

- Example: Global specification via runtime API

```
// Use either the automatic way or manual way of requesting
// parallelism.

// Automatic way.
bli_thread_set_num_threads( 6, &rntm );

// Manual way.
bli_thread_set_ways( 1, 1, 2, 3, 1, &rntm );

// Call a level-3 operation (basic interface is still enough).
bli_gemm( &alpha, &a, &b, &beta, &c );
```

Controlling Multithreading

- And also a third method: Thread-local specification via runtime API
  - Affects only the calling thread!
  - Requires use of expert interface (typed or object)
    - User initializes and passes in a "runtime" object: `rntm_t`
  - Automatic way
    - `bli_rntm_set_num_threads( dim_t nt, rntm_t* rntm );`
  - Manual way
    - `bli_rntm_set_ways( dim_t jc, dim_t pc, dim_t ic, dim_t jr, dim_t ir, rntm_t* rntm );`

Controlling Multithreading

- Example: Thread-local specification via runtime API

```
// Declare and initialize a rntm_t object.
rntm_t rntm = BLIS_RNTM_INITIALIZER;

// Call ONE (not both) of the following to encode your
// parallelization into the rntm_t.
bli_rntm_set_num_threads( 6, &rntm );       // automatic way
bli_rntm_set_ways( 1, 1, 2, 3, 1, &rntm );  // manual way

// Call a level-3 operation via an expert interface and pass
// in your rntm_t. (NULL below requests default context.)
bli_gemm_ex( &alpha, &a, &b, &beta, &c, NULL, &rntm );
```

Controlling Multithreading

- For more details:
  - `docs/Multithreading.md`

Thread Safety

- Unconditional thread safety
- What does this mean?
  - BLIS always uses mechanisms provided by pthreads API to ensure synchronous access to globally-shared data structures
  - Independent of multithreading option

    ```
    --enable-threading={pthreads|openmp}
    ```

    - Works with OpenMP
    - Works when multithreading is disabled entirely

Sandboxes

- Motivation: what if you could provide your own implementation of gemm?
  - You could use as little or as much of the existing implementation code as you like
  - But you want to preserve everything else: build system, testsuite, utility functions, etc.
- Enter BLIS sandbox
  - Integrated into build system (no additional makefiles)
  - Requires only one header file (which can be empty)
  - Requires only one function: `bli_gemmnat()`
  - Use C (or even C++)

Sandboxes

- Enabling a sandbox in BLIS

```
# Enable sandbox named ‘ref99’ (with automatic configuration
# selection).
$ ./configure --enable-sandbox=ref99 auto

# Shorthand:
$ ./configure -s ref99 auto
```

Sandboxes

- Possible uses
  - Trying a different algorithmic path (not Goto)
  - Trying a different implementation of packm (not just packm kernels)
  - Try various optimizations: avoiding `obj_t` at a higher level, or inlining functions.
  - Create experimental implementations of new operations

Sandboxes

- NOT for doing any of the following:
  - Defining a new datatype (half-precision, quad-precision, short integer, etc.)
  - Changing existing APIs
  - Removing support for one or more datatypes (to reduce library size)
  - Change implementation of other level-3 operations such as herk or trmm
    - This may be allowed in the future

Sandboxes

- For more details:
  - `docs/Sandboxes.md`

So What's New?

- Five broad categories
  - Framework
  - Kernels
  - Build system
  - Testing
  - Documentation

Kernels

- Intel SkylakeX and Knight's Landing (AVX-512)
  - native: s/d (all level-3 operations)
  - induced 1m: c/z (all level-3)
- Intel Penryn, Sandybridge, Ivy Bridge, Haswell, Broadwell, Skylake, Kaby Lake, Coffee Lake
  - native: s/d/c/z (all level-3; some level-1v, -1f)
- AMD Bulldozer, Piledriver, Steamroller, Excavator, Zen
  - native: s/d/c/z (all level-3)

So What's New?

- Five broad categories
  - Framework
  - Kernels
  - Build system
  - Testing
  - Documentation

Build system

- Monolithic header generation
  - All headers (~500) recursively inlined into `blis.h`
  - Faster compilation time
  - Easier to distribute build products
- Rewritten configure-time hardware detection
- Configuration blacklisting (assembler/binutils)
- ARG_MAX hack
  - `./configure --enable-arg-max-hack`
- Compile/link against installed copy of BLIS
  - `make BLIS_INSTALL_PATH=/usr/local`

So What's New?

- Five broad categories
  - Framework
  - Kernels
  - Build system
  - Testing
  - Documentation

Testing

- Integrated netlib BLAS test drivers
  - Carefully translated from Fortran-77 to C
  - Integrated into build system
    - `make checkblas`
- Simulate application-level multithreading in testsuite
  - Execute with arbitrary number of threads
- Travis CI now uses Intel SDE emulator to test all x86_64 kernels
  - Exception: FMA4-based Bulldozer

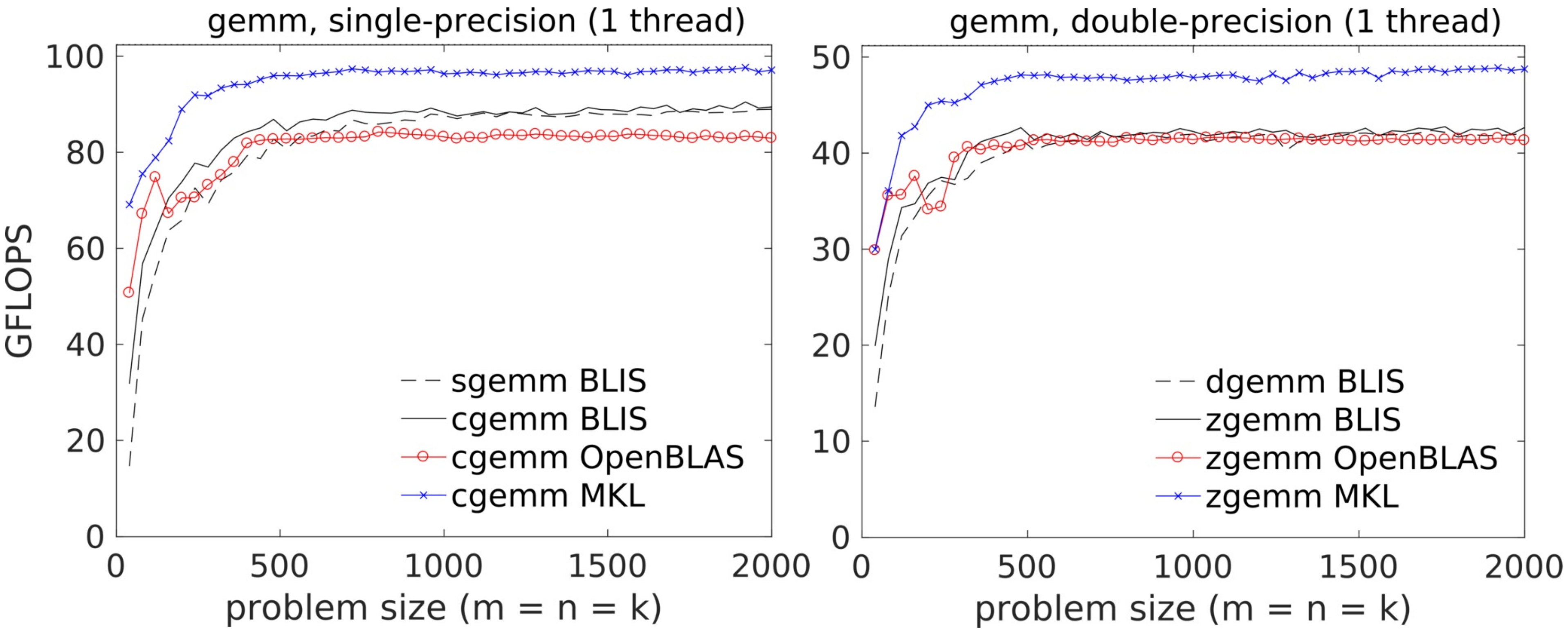
So What's New?

- Five broad categories
  - Framework
  - Kernels
  - Build system
  - Testing
  - Documentation

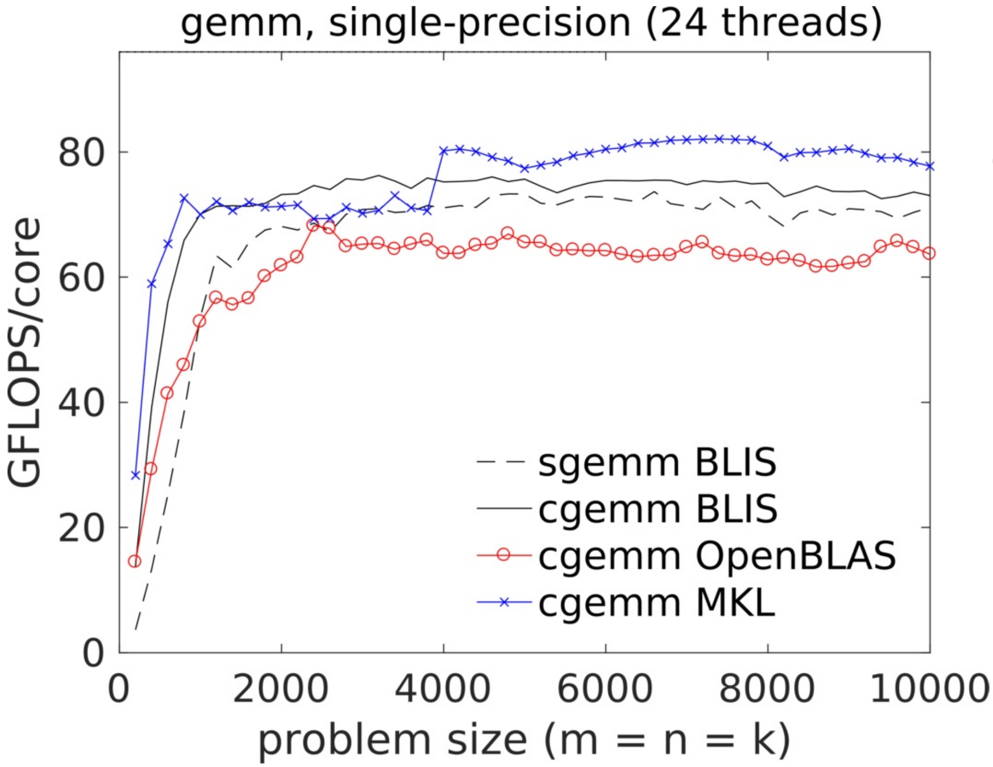
Documentation

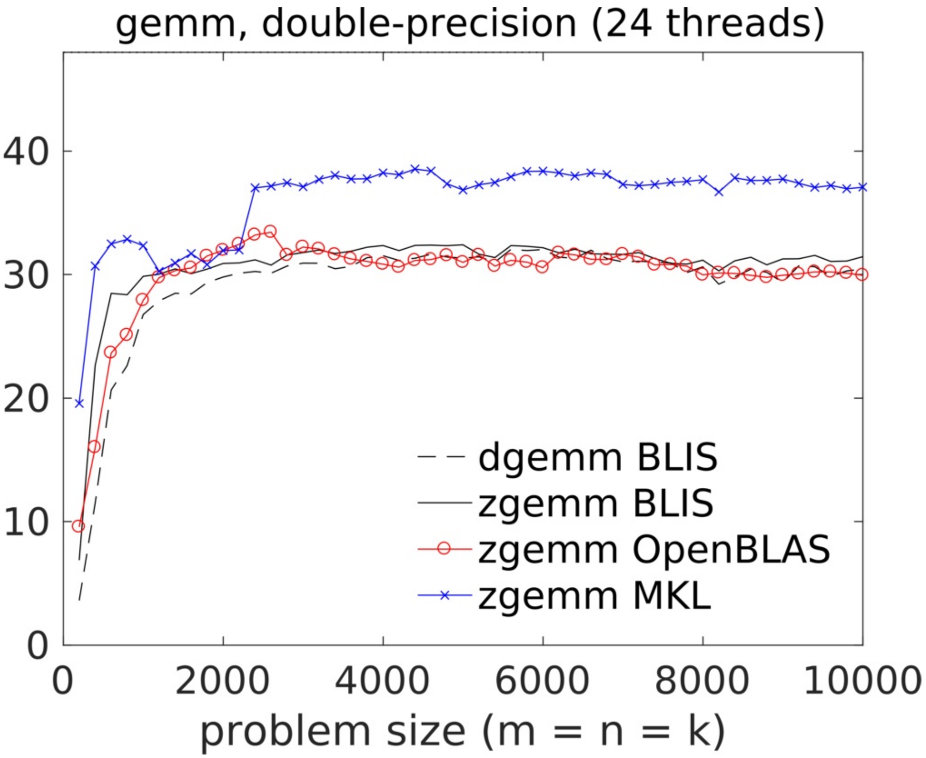
- Example code
  - Typed API: `examples/tapi`
  - Object API: `examples/oapi`
  - Makefiles included
  - Set up like a tutorial: read code alongside the executable output
- Documentation
  - typed API, object API, build system, configurations, hardware support, kernels, multithreading, sandboxes, testsuite, release notes

Performance



Performance





GitHub Stats

- Total BLIS contributors to-date: 62
  - non-UT contributors: 52
- Issues closed: 115
  - by non-UT contributors: 86
- Pull requests closed: 88
  - virtually all accepted
- Average unique clones per two-week period: ~50
  - total clones per two-week period: ~500
- Average unique visitors per two-week period: ~350
  - total visitors per two-week period: ~1500

What's new? (review)

- Five broad categories
  - Framework: runtime config management; library self-init; basic+expert APIs; per-call multithreading specification; unconditional thread safety; sandboxes
  - Kernels: zen support; Devin's assembly macro language
  - Build system: monolithic header generation (faster build time); rewritten configure-time hardware detection; config blacklisting; ARG_MAX hack; BLIS_INSTALL_PATH
  - Testing: integrated netlib BLAS test drivers (translated to C); simulate application-level threads in testsuite; Travis CI now uses Intel SDE
  - Documentation: example code (typed and object APIs); API documentation (typed and object APIs); moved wikis into source distribution

Conclusion

- BLIS…
  - is rapidly maturing
  - is feature-rich
  - is well-documented
  - has a community to support its developers/users
  - has been embraced by industry
  - provides competitive (or superior) performance relative to other leading open-source solutions (and some vendor libraries!)

Further Information

- Website:
  - http://github.com/flame/blis/
- Discussion:
  - http://groups.google.com/group/blis-devel
  - http://groups.google.com/group/blis-discuss
- Contact:
  - field@cs.utexas.edu

Thank you!