On the High Performance Implementation of Quaternionic Matrix Operations

`wavefunction91.github.io`

SIAM-CSE 2019

David Williams-Young
Scalable Solvers Group
Computational Research Division
Lawrence Berkeley National Lab

February 28, 2019

Problem Motivation

- Moving towards the end of Moore's law: Simply applying existing algorithms and data structures not sufficient.
- Quaternion symmetry is very common in many scientific and engineering disciplines, especially those whose target involves physical space.
- Much research has been afforded to real / complex linear algebra algorithms to exploit this symmetry (Dongerra, et al, 1984; Shiozaki, 2018)

The quaternion algebra is [...] somewhat complicated, and its computation cannot be easily mapped to highly optimized linear algebra libraries such as BLAS and LAPACK.

Problem Statement

How can we leverage techniques such as auto-tuning and microarchitechture optimization to provide optimized implementations of quaternion linear algebra software?

This talk will attempt to answer (discuss) three questions:

- What are quaternions and why do we care?

This talk will attempt to answer (discuss) three questions:

- What are quaternions and why do we care?
- What possible use could I have for matrices of quaternions?

This talk will attempt to answer (discuss) three questions:

- What are quaternions and why do we care?
- What possible use could I have for matrices of quaternions?
- What does all of this have to do with auto-tuning?

Quaternions: Formally

Quaternions are defined as the set $\mathbb{H}$ of all q such that

$$q = q^0 e_0 + q^1 e_1 + q^2 e_2 + q^3 e_3, \qquad q^0, q^1, q^2, q^3 \in \mathbb{R}$$

with

$$e_0 e_j = e_j e_0 = e_j, \qquad j \in \{0, 1, 2, 3\},$$

$$e_i e_j = -\delta_{ij} e_0 + \sum_{k=1}^{3} \varepsilon_{ij}^{k} e_k, \qquad i, j \in \{1, 2, 3\},$$

Quaternions: Formally

$$a, b, c \in \mathbb{R}, \quad c = ab, \qquad [a, b] = 0$$

$$w, v, z \in \mathbb{C}, \quad z = wv = w^0 v^0 - w^1 v^1 + (w^0 v^1 + w^1 v^0) i \quad [w, v] = 0$$

Quaternions: Formally

$$a, b, c \in \mathbb{R}, \quad c = ab, \quad [a, b] = 0$$

$$w, v, z \in \mathbb{C}, \quad z = wv = w^0 v^0 - w^1 v^1 + (w^0 v^1 + w^1 v^0) i \quad [w, v] = 0$$

$$p, q, r \in \mathbb{H}$$

$$r = pq = \left(p^0 q^0 - \sum_{i=1}^{3} p^i q^i \right) e_0 + \sum_{k=1}^{3} \left(p^0 q^k + p^k q^0 + \sum_{i,j=1}^{3} \varepsilon_{ij}^k p^i q^j \right) e_k,$$

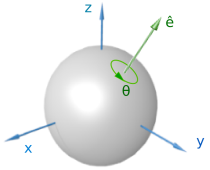
$$[p, q] = \sum_{i,j,k=1}^{3} \varepsilon_{ij}^k \left(p^i q^j - p^j q^i \right) e_k \neq 0.$$

Quaternion Applications: Spacial Rotations

Topologically, the set of unit quaternions (versors)

$$\mathbb{V} = \{v \in \mathbb{H} \text{ s.t. } ||v|| = 1\}$$

is S^3, and thus isomorphic to SU(2) which provides a double cover of SO(3) (rotations in $\mathbb{R}^3$).



We may describe spatial rotations in $\mathbb{R}^3$ via

$$\boldsymbol{r} \in \mathbb{R}^3 \mapsto r^H = r^1 e_1 + r^2 e_2 + r^3 e_3$$

$$\boldsymbol{R}(\hat{\boldsymbol{e}}, \theta) \in \mathrm{SO}(3) \mapsto \pm v = \pm \exp\left(\frac{\theta}{2}(\hat{e}^1 e_1 + \hat{e}^2 e_2 + \hat{e}^3 e_3)\right)$$

such that

$$\boldsymbol{r}' = \boldsymbol{R}(\hat{\boldsymbol{e}}, \theta)\, \boldsymbol{r} \quad \mapsto \quad r'^H = v r^H v^{-1}$$

Quaternion Applications: Spacial Rotations

The SO(3) cover has found extensive exploitation in computer graphics / vision

- (v^0, v^1, v^2, v^3) (4 real numbers) vs. $\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$ (9 real numbers)
- $v_1, v_2 \in \mathbb{H}$, $v_1 v_2$ (16 FLOPs) vs $\boldsymbol{R}_1, \boldsymbol{R}_2 \in \mathrm{SO}(3)$, $\boldsymbol{R}_1 \boldsymbol{R}_2$ (27 FLOPs)
- SLERP (**S**pherical **L**inear **Interp**olation)

Matrices of Quaternions

The algebra generated by $\{e_0, e_1, e_2, e_3\}$ is identical to the algebra generated by the Pauli matrices, thus $\mathbb{H} \cong \langle \mathrm{SU}(2) \rangle \subset \mathbb{M}_2(\mathbb{C})$,

$$e_0 \leftrightarrow \sigma_0, \quad e_1 \leftrightarrow i\sigma_3, \quad e_2 \leftrightarrow i\sigma_2, \quad e_3 \leftrightarrow i\sigma_1,$$

with

$$\sigma_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \sigma_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \sigma_2 = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad \sigma_3 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

Such that

$$q \leftrightarrow q_{\mathbb{C}} = \begin{bmatrix} q^0 + q^1 i & q^2 + q^3 i \\ -q^2 + q^3 i & q^0 - q^1 i \end{bmatrix} = \begin{bmatrix} \underline{q}^0 & \underline{q}^1 \\ -\overline{\underline{q}}^1 & \overline{\underline{q}}^0 \end{bmatrix} \in \mathbb{M}_2(\mathbb{C})$$

Matrices of Quaternions

The set of quaternion matrices, $\mathbb{M}_N(\mathbb{H})$, is defined by

$$Q = Q^0 e_0 + Q^1 e_1 + Q^2 e_2 + Q^3 e_3, \quad Q^0, Q^1, Q^2, Q^3 \in \mathbb{M}_N(\mathbb{R}).$$

Examining the $\mathbb{M}_2(\mathbb{C})$ representation of a particular element

$$(Q_{\mu\nu})_{\mathbb{C}} = Q^0_{\mu\nu}\sigma_0 + iQ^1_{\mu\nu}\sigma_3 + iQ^2_{\mu\nu}\sigma_2 + iQ^3_{\mu\nu}\sigma_1.$$

which yields the Kronecker structure

$$\begin{aligned} Q_{\mathbb{C}} &= Q^0 \otimes \sigma_0 + Q^1 \otimes i\sigma_3 + Q^2 \otimes i\sigma_2 + Q^3 \otimes i\sigma_1 \\ &= \begin{bmatrix} \underline{Q}^0 & \underline{Q}^1 \\ -\overline{\underline{Q}}^1 & \overline{\underline{Q}}^0 \end{bmatrix} \in \mathbb{M}_{2N}(\mathbb{C}), \end{aligned}$$

Matrices of Quaternions

$$Q \in \mathbb{M}_N(\mathbb{H}) \leftrightarrow \begin{bmatrix} Q^0 & Q^1 \\ -\overline{Q}^1 & \overline{Q}^0 \end{bmatrix} \in \mathbb{M}_{2N}(\mathbb{C}),$$

- Ubiquitous in quantum chemistry / nuclear physics (time-reversal symmetry).
- Applications in image processing and machine learning (quaternion PCA, etc).

Formal theory for quaternion linear algebra has been developed

- QR Algorithm
- Diagonalization, SVD
- LU, Cholesky, LDLH Factorizations

Performance Considerations

Table: Real floating point operations (FLOPs) comparison for elementary arithmetic operations using $\mathbb{H}$ and $\mathbb{M}_2(\mathbb{C})$ data structures.

Operation	FLOPs in $\mathbb{H}$	FLOPs in $\mathbb{M}_2(\mathbb{C})$
Addition	4	8
Multiplication	16	32

$$
\begin{aligned}
p + q \quad &\longleftrightarrow \quad p_{\mathbb{C}} + q_{\mathbb{C}}, \\
pq \quad &\longleftrightarrow \quad p_{\mathbb{C}} q_{\mathbb{C}},
\end{aligned}
$$

Performance Considerations

Table: Real floating point operations (FLOPs) comparison for common linear algebra operations using $\mathbb{M}_N(\mathbb{H})$ and $\mathbb{M}_{2N}(\mathbb{C})$ data structures.

Operation	FLOPs in $\mathbb{M}_N(\mathbb{H})$	FLOPs in $\mathbb{M}_{2N}(\mathbb{C})$
Addition	$4N^2$	$8N^2$
Multiplication	$16N^3$	$32N^3$

$$
\begin{aligned}
P + Q \quad &\longleftrightarrow \quad P_{\mathbb{C}} + Q_{\mathbb{C}}, \\
PQ \quad &\longleftrightarrow \quad P_{\mathbb{C}} Q_{\mathbb{C}},
\end{aligned}
$$

Performance Considerations

Quaternion arithmetic offers:

- 0.5x required FLOPs
- 0.5x memory footprint (4x / 8x floats)
- 2x arithmetic intensity (FLOPs / byte)

We should be using quaternion arithmetic!

$\mathbb{H}$AXX

- `gh/wavefunction91/HAXX`
- Optimized C++14 library for quaternion arithmetic
- $\mathbb{H}$BLAS: Optimized quaternionic BLAS functionality
- $\mathbb{H}$LAPACK: Optimized quaternionic LAPACK functionality (in progress)
- Intrinsics + Assembly kernels

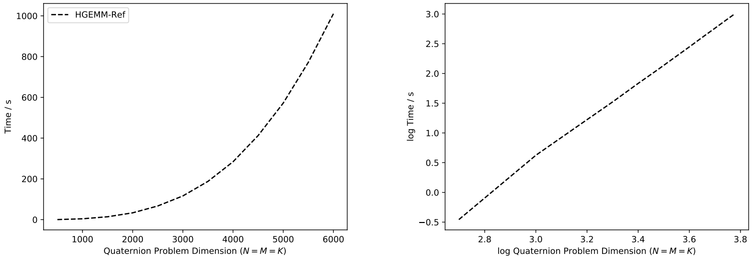
Quaternion Matrix Multiplication

The most fundamental linear algebra operation is the general matrix multiply (GEMM).

Quaternion Matrix Multiplication

The most fundamental linear algebra operation is the general matrix multiply (GEMM).

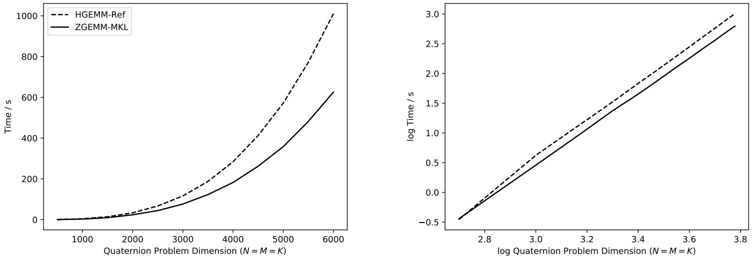
Great! Quaternion GEMM $\Longrightarrow$ HP Quaternion Linear Algebra, Right?



Quaternion Matrix Multiplication

The most fundamental linear algebra operation is the general matrix multiply (GEMM).

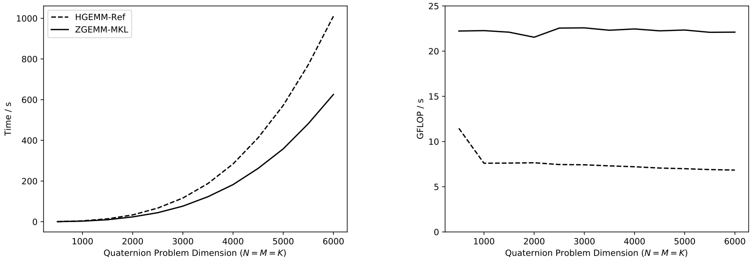
Great! Quaternion GEMM $\Longrightarrow$ HP Quaternion Linear Algebra, Right?



Quaternion Matrix Multiplication

The most fundamental linear algebra operation is the general matrix multiply (GEMM).

Great! Quaternion GEMM $\Longrightarrow$ HP Quaternion Linear Algebra, Right?



Reference GEMM

Algorithm 0: Reference GEMM

Input : $A \in \mathbb{M}_{m,k}(\mathbb{F})$, $B \in \mathbb{M}_{k,n}(\mathbb{F})$,
$C \in \mathbb{M}_{m,n}(\mathbb{F})$,
Scalars $\alpha, \beta \in \mathbb{F}$

Output: $C = \alpha AB + \beta C$

```
for j = 1 : n do
    Load C_j = C(:,j)
    C_j = βC_j
    for l = 1 : k do
        Load A_l = A(:,l)
        C_j = C_j + αA_l B_lj
    end
    Store C_j
end
```

Pros:

- ✓ Able to implement in an afternoon
- ✓ Architecture agnostic

Cons:

- ✗ No caching of B
- ✗ Reloads all of A for each C_j
- ✗ For large m, k, A load boots C_j from cache
- ✗ Relies on optimizing compiler for SIMD, FMA, etc
- ✗ (Scalable) parallelism is non-trivial
- ✗ **Not tunable**

High-Performance Matrix-Matrix Multiplication

A layered (Goto-style) algorithm significantly improves performance

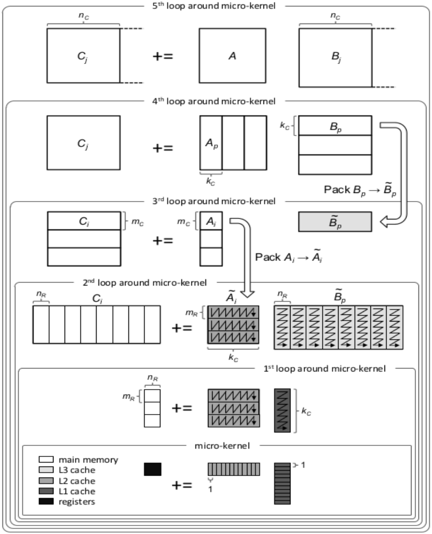
Pros:

- ✓ Caches parts of A, B for maximum resuability
- ✓ Factors architecture specific μ-ops into single micro-kernel
- ✓ Obvious avenue for SMP
- ✓ **Tunable!**

Cons:

- ✗ Significantly more complicated than naive algorithm
- ✗ Requires allocation of auxiliary memory
- ✗ Micro-kernel must be written for each architecture

Van Zee, F.G., *et al* **2017**, *ACM TOMS* 7:1-7:36.



High-Performance Quaternionic GEMM (HGEMM)

The optimized implementation of GEMM in $\mathbb{H}$BLAS utilizes the Goto algorithm. In essence, Goto's original algorithm may be extended to $\mathbb{H}$ by specialization of two sets routines:

- Micro-kernels which perform the $\mathbb{H}$ rank-1 update (assembly / intrinsics)
- Efficient matrix packing routines (intrinsics)

and optimization of 3 caching parameters, m_c, n_c and k_c for the architecture of interest.

High-Performance Quaternionic GEMM (HGEMM)

The optimized implementation of GEMM in $\mathbb{H}$BLAS utilizes the Goto algorithm. In essence, Goto's original algorithm may be extended to $\mathbb{H}$ by specialization of two sets routines:

- Micro-kernels which perform the $\mathbb{H}$ rank-1 update (assembly / intrinsics)
- Efficient matrix packing routines (intrinsics)

and optimization of 3 caching parameters, m_c, n_c and k_c for the architecture of interest.

High-Performance Quaternionic GEMM (HGEMM)

The optimized implementation of GEMM in $\mathbb{H}$BLAS utilizes the Goto algorithm. In essence, Goto's original algorithm may be extended to $\mathbb{H}$ by specialization of two sets routines:

- Micro-kernels which perform the $\mathbb{H}$ rank-1 update (assembly / intrinsics)
- Efficient matrix packing routines (intrinsics)

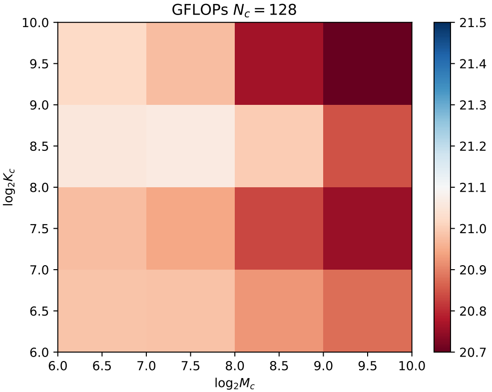
and optimization of 3 caching parameters, m_c, n_c and k_c for the architecture of interest.

Optimization of Cache Parameters

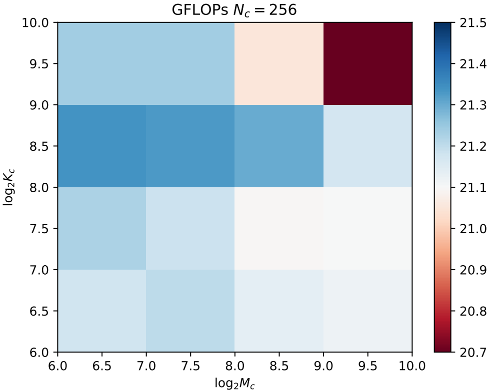
- OpenTuner: An open source Python framework for auto-tuning
- Register blocks fixed (on AVX / AVX2), $n_r = m_r = 2$.
- Integer discretize $m_c, k_c \in \{2^n\}_{n=3}^{12}$, $n_c \in \{2^n\}_{n=5}^{16}$
- Find $\{m_c, n_c, k_c\}$ which minimizes run time (maximizes GFLOP/s)
  - Average over 5 cold (cache invalidated) runs on select matrix sizes (500,1k,2k,4k)

Possible to brute force otimimize, but not convienient!

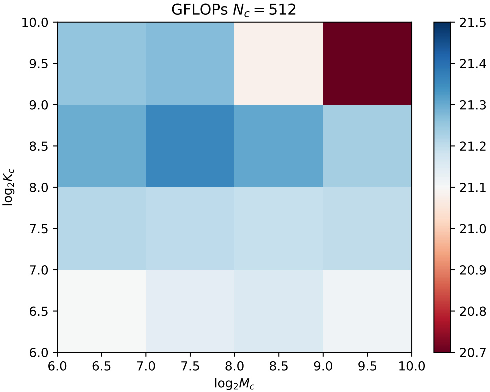
Optimization of Cache Parameters



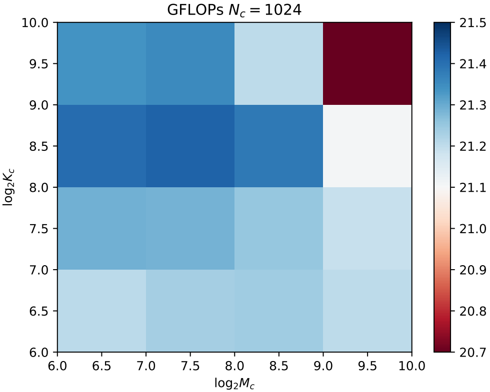
Optimization of Cache Parameters



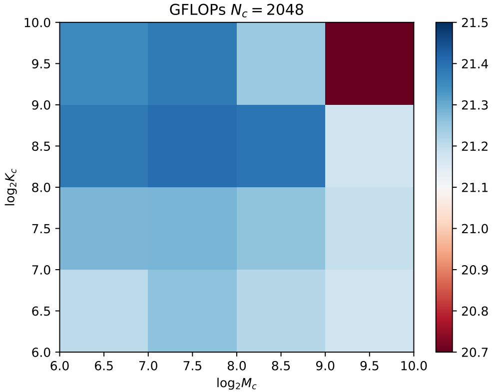
Optimization of Cache Parameters



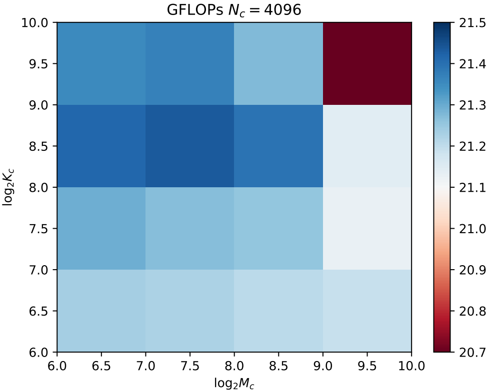
Optimization of Cache Parameters



Optimization of Cache Parameters



Optimization of Cache Parameters

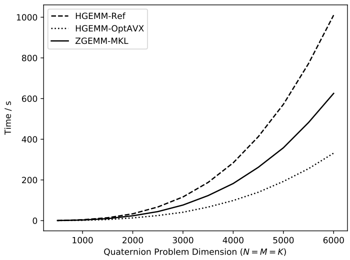


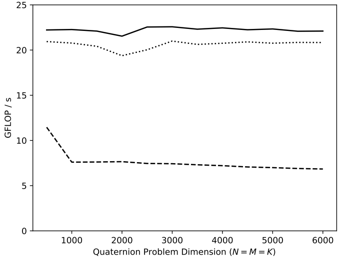
AVX Optimized HGEMM Implementation

Intel Sandy Bridge (L1d: 32k, L1i: 32k, L2: 256k, L3: 20480k)

OpenTuner results:

- 10 tests x 10 runs (˜1 hour vs 10 hours brute force)
- $m_c = k_c = 64$
- $n_c = 1024$





Conclusions

- With instruction sets newer than AVX, high-performance quaternionic linear algebra is possible and a viable alternative to complex linear algebra for appropriate problems.
- Goto's algorithm + auto-tuning drastically improves performance
  - Impractical with reference implementations.

Future Work

- Fill out $\mathbb{H}$BLAS and $\mathbb{H}$LAPACK coverage of the BLAS and LAPACK standards.
- Package autotuner and tuning methodology to automate optimization of caching parameters (+...) on new architectures.
- Address parallelism (SMP + MPI)

Acknowledgments

- Xiaosong Li (UW)
- $$$ ACI-1547580 (NSF), OAC-1663636 (NSF to XL)
- $$$ LAB 17-1775 (DOE-BES)
- Benjamin Pritchard (MolSSI)
- Edward Valeev (VT)
- Wissam Sid-Lakhdar (LBNL)
- Organizers
- Audience