Document Number: P1798R0
Date: 2019-06-18
Authors: Michael Wong
Project: Programming Language C++, SG14 Games Dev/Low Latency/Financial Trading/Banking/Simulation/Embedded
Reply to: Michael Wong <michael@codeplay.com>

SG14: Linear Algebra SIG Meeting Minutes 2018/10/10-2019/06/06

Contents

Minutes for 2018/10/10 SG14 Conference Call 2
Minutes for 2018/11/09 SG14 San Diego F2F 12
Minutes for 2018/12/05 SG14 Conference Call 26
Minutes for 2019/01/02 SG14 Conference Call 35
Minutes for 2019/02/06 SG14 Conference Call 45
Minutes for 2019/02/20 SG14 Kona LEWGI review 49
Minutes for 2019/02/22 Kona SG14 SG19 Review 57
Minutes for 2019/03/06 SG14 Conference Call 61
Minutes for 2019/04/03 SG14 Conference Call 70
Minutes for 2019/05/01 SG14 Conference Call 78
Minutes for 2019/06/06 SG14 Conference Call 91

Minutes for 2018/10/10 SG14 Conference Call

Minutes by Mark Hoemmen

Attendees

=========

Lawrence Crowl

Guy Davidson

Carter Edwards

Mehdi Goli

Mark Hoemmen

David Hollman

Nasos Iliopoulos

Bryce Lelbach

Duane Merrill

Bob Steagall

Michael Wong

Discussion

==========

MW: SG14 had discussion on this at CppCon. No paper ready at that time. Bob and Guy, have things changed? See San Diego Wiki under SG14.

BL: I wanted to get together authors for mdspan proposal plus others to talk about this. SG14 wanted to go this way. What are requirements and goals? Guy, since you and Bob have thought about this in graphics context, could you share what you have in mind?

GD: I could show slides I showed at SG14. [looks them up]

BL: [summarizes mdspan]

MW: Does mdspan give me a mathematical matrix?

CE: Yes, that was the original motivation, to match or exceed Fortran capabilities.

MW: This is what I need to build a machine learning library, eigen library, and linear algebra library.

BL: It supports different layouts.

MW: Does that mean we no longer need an array TS, the one that was killed?

BL: I think that is an accurate statement.

MW: Does it implement it efficiently?

CE: Yes.

BL: It comes from Kokkos.

MW: I know the answers to these questions but I want to advertise them to the rest of this group.

??: Current prototype, transparent to the compiler, can see through 5- or 6-D references, beyond that is sketchy.

MW: Tricycle has implemented that.

??: I corresponded with Ronan on that.

GD: [shows slides] "Towards Standardization of Linear Algebra," Guy Davidson and Bob Steagall.

This effort arose out of the graphics proposal. Bob and I realized that we needed not just a matrix, but a row vector and column vector type.

"Customisable": User chooses storage and implementation.

"Promotable": std::complex<double> and std::complex<float>, should promote like integer types.

MH: Mixed precision?

BS: yes. abstract and concrete. concrete:

- element layout. (tri, diag, banded, sparse)

- storage management, orthogonal to element layout

- resize-ability?

2 promotion problems:

1. Element type promotion: scalar of double, times matrix of complex<float>, would like complex<double> matrix. For MRI, receive data as complex 32-bit integers, Fourier space, and you need to transform to say complex<double>.

2. Engine (representational type / storage) promotion: fixed-size vs. dynamically-sized engine.

Idea is to expose customization points via type traits.

We confine ourselves to matrices and vectors, not tensors.

LC?: Think about whether what you're doing extends to higher dimensions. Would rather not repeat what we did with valarray, and force people to switch to a different datatype to get to higher-order algorithms.

MH: Many linear algebra algorithms do not extend naturally to higher than rank-2 tensors.

DH: mdspan will handle the datatype stuff, for non-sparse matrices at least. That extends naturally to higher dimensions. I wanted to agree with MH.

??: We do need 2, 3, 4-rank, but have to keep in mind, when we solve a specific problem, we know the dimension ahead of time. Don't need dynamic dims.

MW: Datatype, is that same as storage type? Storage has been pointers, but want to template, so we can use things like buffer data types -- storage class. We can discuss this later on.

DH: mdspan currently provides that.

BL: our 1-D algorithms, std library, those algorithms we have an abstraction, iterators, a view into the data, agnostic of storage or allocation. Just provide an iterator. In same way, mdspan provides sort of the same thing for linear algebra or multi-d algorithms.

MH: Probably not the right abstraction for high-rank sparse tensors, but OK.

NI?: Have you defined operator+ anywhere? I see operator+=.

GD: Yes, but not as a member function.

GD: [continues to explain operations]

GD: "REP": the actual implementation happens there. Default naive implementations; can substitute superior versions from the specific domain. For example, I come from game development, with 3x3 and 4x4 matrices, so I would declare e.g., float_22.

DM: In deep learning: we want to accumulate in a different, higher-precision type (say 32-bit) but write out result in 16 bits.

MW: Important when transferring to different inference engines.

DM: Operator itself has some type-specific stuff. I have to go, but I want to make this an API request.

BS: We see this as customizable and thus that we can fulfill DM's requirements.

BL: Need to be cautious with operator overloading language. We went down this path with valarray. Valarray is almost never used because, after we standardized it, people came up with things like eigen, based on expression templates. We have to compete with that. We want implementations to be able to use expression template optimizations.

?? (NI?): We've been working with cuBLAS (?) since 2009/10, and have experience with expression templates. Linear algebra library needs expression templates. You gain a lot of performance for cache locality optimizations.

BS: We totally agree and want our API not to preclude optimizations.

DH: I've had several discussions like this w/ Eric Niebler on lazy ranges and their extension to linear algebra. Seems to align with this pretty well.

BL: If we want to allow these sorts of optimizations then we want something like lazy ranges, a lazy API where you can build in an expression tree.

DH: That's concepts, not class templates.

BL: This is new territory for the library.

?? (NI?): We have some ideas based on ranges. I have an early draft that I would be happy to share. Have similar heterogeneous computing interests and needs. Our library is concepts driven.

CE: First, make sure linear algebra operators decoupled from storage representation. mdspan focused entirely on storage and layout, representation.

??: We agree CE, definitely.

CE: I didn't see enough detail to be sure of that. Second, on what parallel resources will you perform your linear algebra operations? If large, want all threads, or large set of small system?

MH: Batched linear algebra.

GD: Could extend the interface; have to start somewhere.

BS: Decomposition of abstract and concrete, esp. element layout and storage vs. arith operations themselves, 3 orthogonal axes. Customization points.

?? (NI?): I agree with Bob; this is our requirement. Need to be able to communicate with third-party libraries etc. that have storage format requirements.

LC: When this reaches rest of C++ committee: Why should this be in the standard? Need to point out use cases for average programmers. How does this help a broader community?

BS: My experience in functional MRI. 4-D signal. Solving a multiple regression problem, highly overdetermined. I ended up writing a linear algebra library to solve this problem, with a lot of the features of this library. 90% of the code I wrote was interface to wrap these concepts.

Real work done by BLAS MKL; it did all the hard work. This library would have made medical imaging developers' lives easier.

LC: It takes almost nothing to convince _me_ that we need this, but I'm not the committee as a whole. Having a good story ready to go will help grease the wheels.

BL: NI talk now in 10 minutes.

DH: What GD said 5 minutes ago: I agree we need to get something ready and have to start somewhere, but potential to have situation with ranges and executors, where hard to build lazy rep on top of eager rep, but other way around is much more straightforward. Thus, thinking about these things early is a good idea.

NI: [showing slides] "Examples of an index based C++ linear algebra system" -- abstract your reductions and for loops based on indices. "Index" is a placeholder for evaluation iteration. It doesn't know how to iterate yet. e.g., "B(j,i) = A(i,j)" does the transpose, where i and j are each instances of this special index type [templated on a name thing]. A(i,j) generates an expression; operator= on an expression generates loops. Can use Einstein [tensor index] notation: e.g., "double s = v(i);".

DH: Have you seen this analogous effort in quantum chemistry? About a dozen libraries that do this.

NI: Yes, I have seen this. Tacho(?) and F(?)tensor(?).

DH: Tiled array, Cyclops. 2014: summit of these groups that met to establish a standard for Einstein summation libraries. Just curious if you've coordinated with their effort.

NI: Not yet.

NI: [numerical differentiation example; subarrays at compile time; matrix compositions; closest points between two sets]

NI: Sparse container [filter].

BL: Wrap-up first, then let people chat after. What do we do going forward? At Kona, GD and BS are working on a proposal. Do we have an interest in building a wider gropu to work on a unified proposal, or do we want people to work on different ideas? Evening session at Kona or in SG6 where we talk about this problem space?

LC: Best to explore the space early on, and reach agreement for how much of the space we want to tackle. Standard is large and growing; biggest criticism of C++. Want mechanism with a lot of power to cover more space.

BL: Is this sufficiently large to have its own study group, or can we use an existing study group, e.g., numerics?

LC: May not need study group route at this point. Right now, people that are interested should be getting together informally. There are a lot of ideas in this space that would be very helpful to a lot of people, but not helpful if delivered inuncoordinated fashion.

MW: Still too fragmented now. Let's see where BS and GD are going. Specific study group not important. SG14 meets monthly to talk about this. SG14 meets more regularly than SG6 or 7. We should probably keep it in SG14. Doesn't have to be in SG14 either.

BL: My preference would be informal discussion group.

MW: Yes, that's fine, I have no problem with that. SG14 has enough to work on already. This is big enough group that it could gather its own momentum.

BL: Yes,Want to make sure this doesn't get time boxed.

MW: We will track this in SG14 for now. Also, proposal from Herb to split up Evolution Working Group into e.g., long-term vs. short-term proposals.

BL: Getting kicked out of conf room now. Let's meet again in 2 weeks.

MW: Please copy SG14.

BL: Yes.

Minutes for 2018/11/09 SG14 San Diego F2F

Minutes by Mark Hoemmen

Present:

Bob Steagall
Carter Edwards
Hal Finkel
Robert Douglas
Li-Ta Lo, ollie@lanl.gov
Brian Von Stralen
Rene Rivera
Damien Lebrun-Grandie
Graham
Mauro Bianco
Mark Hoemmen (scribe)
Daniel Garcia
Michael Wong
BS: Michael Wolf described this meeting as a friendly discussion, not for making decisions. He charged me to say there are two items on the agenda:

Defining scope
Observe that this is a big enough topic that we may want separate linear algebra phone calls, separate from SG14
RR: Does this come out of SG14? / 13?

BS: Given uncertain state of 2-D graphics post Jacksonville, Guy Davidson decided to try standardizing linear algebra as a smaller building block of graphics, but it's not connected to 2-D graphics in any process-based way. I volunteered to help him.

Graham: Hopefully he was aware of nongraphics can of worms that is linear algebra.

BS: Yes. Guy is a math person and is aware of non-graphics interest.

BS: We could decide on a frequency and schedule for phone calls, and publish on reflector.

Graham: Is there an SG14-specific reflector? I should sign up for it.

BS: Yes.

RR: I thought linear algebra was also discussed in numerics.

BS: I'm not aware of that. There should perhaps be discussions of it there.

HF: Numerics covers this case.

Graham: Where is reflector?

HF: Once there are proposals, numerics will want to look at them. I don't think numerics specifically looked at the matrix classes that were in the Graphics TS. I think it was also clear from discussions that, should such a thing move towards the standard, then the matrix stuff would have been separated off anyway, else one would have to include a graphics header to get a standard matrix class.

MB: mdspan proposal can cover any kind of a 2-D array. Not sure about sparse cases, dense for sure.

HF: nods

RR: ... core proposals about matrices and comma operator [Isabella Muerte's proposal].

MB: In some other meetings when were the special math functions being proposed, first and second reaction was implementors didn't want to get into trouble of implementing them, because they didn't want to have a scientist come to them and scold their rounding error. Linear algebra is not an easy topic; research still being done in algorithms and implementation. Not clear to me the focus / target / 80% of beneficiaries.

RR: Definitely game developers.

BS: When I was working on time series analysis of medical images, multiple regression. Supporting linear algebra needs of game and graphics, multiple regression, linear least squares. Very common problem, would serve a lot of domains.

MH: Less controversy with matrix multiply than SVD.

MB: With matrix multiply, people want to use n^3 vs. Strassen.

RR: Game developers write their own, don't care about [accuracy]

MB: Bitwise reproducibility ...

MW: This is an informal meeting; no decisions can be made. Use this chance to discuss scope, design issues, frequency of meetings. Big topic, could be either in SG14 or not. SG14 meets on second Wed of every month, two hours. Could also do out of band and report in SG14 meetings. I'm only here to facilitate; BS will run the meeting.

HF: Scope in terms of graphics and game development etc., and we were talking about relevance of BLAS interfaces ...

BS: Not only in first telecon but in other discussions here, unless there's a compelling performance reason to pick a particular interface design over another, driven by performance, my feeling is we should concentrate on interface design. First define scope of problems users want to solve, then design interface, rather than quickly getting into implementation decisions.

Graham: I'll agree with that. Interfaces we can standardize anyway. In this case, might be unique -- e.g., the worry of std library implementers not necessarily wanting to get into this domain, due to community concerns about reproducibility and performance. A bit unique, we have many vendors willing to give us high-performance BLAS libraries, absolutely not interested in providing us standard C++ library implementations, way more than they want to do. If we standardize in a way either to require ... we don't want to require people who want linear algebra in C++, to have to provide a full library implementation; conversely if we don't allow a way to hook in these specialized implementations, we'll exclude a swath of users.

MH: [Explains analogy with Fortran matrix-matrix multiply intrinsics; low-level vs. high-level]

RR: Game devs may want to replace those low-level...

MH: Even the compiler might want to replace them.

RR: Right.

BS: I implemented a linear algebra library a while back for medical imaging. I wrote a high-level interface, and wrapped MKL, using a set of traits that one could specialize to provide different implementations of fundamental operations. That's like the approach Guy and I are working on for this paper. We want to give flexibility for sophisticated users to drop in services, while implementers would provide basic fundamental performance. Analogy with writing a custom allocator.

Graham: Let's circle back to scope discussion.

HF: You've wrapped various things, implemented traits ... lots of prior art in this space, e.g., Eigen, Boost microBLAS.

MH: Even POOMA did [give ability to plug in different low-level implementation details], historically.

BS: Strawman scope: basic matrix arithmetic, linear algebra 101 textbook operations, important transforms for multiple regressions or linear least squares, LU, Cholesky, SVD. A good design would let that serve needs of game developers and machine learning people.

RR: Do we want to define scope in sense of linear algebra operations? That set alone would exclude game developers at this point, because you would need to add quaternions. Not sure if it helps us to be that fine-grained. We should start by talking about which fields and people we want to serve.

HF: To make progress, -- we all represent different communities, so we should translate their interests into actual requirements. We should all figure out what those sets of things are, then discuss them and find overlap.

Graham: To rephrase: in past few meetings, increasing pressure in community to talk about features in terms of basis -- minimal set of things that all / most communities need. Yes, collect requirements, then distill out of that a basis set.

1: It's been a while when I looked at quaternions; are they just float vectors?

RR: Yes, but if you don't call it quaternions, game developers get angry.

1: I see.

[discussion of quaternions]

MH: [tensors want different interface]

Graham: Or we could give them facilities to build up what they need....

MH: Yes, e.g., they already use the BLAS....

BS: Is it helpful to think of this in terms of groups of related mathematical operations, rather than communities. EVerybody knows what "matrix arithmetic" means. Then we say, we want to make tools for solving linear least-squares problems. Is that helpful for determining scope?

HF,Graham: Yes.

HF: I don't recommend generalizing based on classes of people rather than problems, as we'll always get that wrong. "Scientists need X" is not helpful.

BS: Or helpful to think in terms of providing an interface that provides a clean way of doing all the operations that one would find, say, in LAPACK.

Graham: My community would be thrilled with that approach, but not necessarily best for all here.

MH: [That would be an LAPACK binding]

BS: Just using LAPACK as a strawman set of capabilities, not necessarily actually the thing.

CE: What you want to solve -- but also how you want to solve. Very large matrix? on a single thread? multithreaded? parallel? Or matrix operations on a lot of matrices, single threaded, multithreaded, in parallel? All the matrices with the same shape? In scope: What you're solving. Another axis: How you're solving. Can't neglect that.

MB: Another axis: sparse or dense. For dense, we know what to do.

Robert Douglas: Owning or nonowning? Expected to be fast, or go off to some server for a matrix that can't be represented on your machine? Heterogeneous types within your data?

MW: I think this group can lead the pack and make sure this thing can live heterogeneously or homogeneously, possibly in parallel. ... have you discussed mdspan and how it fits in ?

BS: Not yet.

MW: I'll wait for that.

BS: In correspondence with Guy, two obvious realms of irreducible: abstract and concrete. In abstract: Matrix, element, row and column vector. Well understood mathematical objects. Matrix arithmetic, transforms, decompositions. Abstract only in the sense that they transcend implementation. Concrete: element layout, e.g., dense, symmetric, banded. Storage stack-based or dynamically allocated. Sizes: compile-time or run-time? Resizable? Owning or not? Concurrency / parallelism. Five concrete, somewhat orthogonal axes.

RD: Inspired by STL and Ranges, are there a set of matrix types optimized for different matrix properties, and a set of solvers that can work on common interface, rather than dilute....

MH: This is totally appropriate for Krylov subspace methods; see e.g., "Templates for the solution of linear systems"; but for things like banded dense solves, the structure of the matrix is more tightly coupled to good algorithms for performance.

RD: Similar problem for iterator categories in STL. Not every iterator category is appropriate for every container. Can provide more optimized algorithm by specializing on the iterator category, but don't have to.

Graham: At this early stage, I'll ask MW a question: previous words from the Direction group, being careful from getting too specialized, solving domain-specific problems. How close are we skating to this; how much math can we talk about in the std?

MW: I'm less worried about that, because we're crossing domains anyway: finance, HPC, educators, games, ... From Bjarne, make sure any C++ solution you're crafting solves at least two problems [and] crosses at least two domains. With linear algebra, that's a foregone conclusion. Failed / negative example: Transactional memory. Our solution was very specific to TM. Because block-based sync technique, did not apply well to thread-based or executor based, it didn't fly and probably never will, unless we slim it down.

[Brian Von Stralen enters]

Graham: We need to talk about it in a way common to all those domains, with our terminology.

MW: Right.

BS: Back to scope question: We talked about ways to decompose the problem ... fundamental linear algebra "stuff," problem space of linear least squares. Are there other problem spaces besides those two that we should consider for our scope?

CE: Or can we layer the scope, peeling the onion back? ... e.g., BLAS has "levels" (level 1, level 2, level 3) My hypothesis is that lowest level is most general, higher level could be more specialized. Lowest level could be where to start with consensus.

BS: I was thinking more along the lines of not necessarily doing all levels at once, but at what level up do we say that we are done? The higher level for which we shoot, the more complex the design.

Graham: I think what CE is saying, is that we don't have to decide in advance at what level up to stop.

CE: Stop once no consensus.

HF: We need to have some idea of what set of problems we're solving, and then, decompose into pieces.

BVS: Basic / minimum set of functionality that would be valuable enough to justify doing the work and get enthusiasm up past threshold.

MB: The BLAS layers you (CE) were talking about were not software layers. Level 3 BLAS uses different algorithms; it does not use level 1 BLAS.

BVS: You don't implement BLAS 3 in terms of BLAS 2.

MH: [We should go back to the BLAS oral history, collected circa 2004. BLAS levels "cut through" the stack, but one could still implement algorithms in terms of lower levels, by analogy with iterator categories and standard algorithms.]

Graham: How much are we talking about implementation at this point? Not, how are we going to implement these different levels, but the question is what they need to support.

HF: Fine line; the interfaces "must support efficient implementation," so you must understand enough about implemention to know you aren't precluding efficient implementation. I have some users who would be happy with basic matrix arithmetic and entry access; I have some who want more than that. Definitely there are relatively easy to define independently useful subsets of functionality.

CE: Look at investment going into dense matrix-matrix multiply...

HF: Figuring out relationship to mdspan. If we have a basic matrix arithmetic; is mdspan ....

[MH: we've looked at doing that with mdspan]

[MH: describes ... templated BLAS and pain of wrapping BLAS with fortran mangling etc.]

POLL: 8 people in room out of 13 have written templated BLAS wrappers.

BVS: That doesn't mean we're all coauthors of a paper though ...

BS: Trickiest problem is, once we've decided on fundamental set of linear algebra objects (matrix, row vector, column vector), whatever they are, defining the implementation behind them that can support these multiple orthogonal axes of concern that we have -- storage layout, source of memory, resizability, owning, concurrency/parallelism. That will be the hardest part of the design of the library. Getting that right will be driven by performance concerns....

Graham: Question for room: Related to all things BS just listed, are there any we think we should not solve with allocators plus executors?

BVS: I think, lacking the execution context, and/or the equivalent of HWLOC in the standard library that tells you memory affinity, doing dense linear algebra without portable way of discussing affinity is a nonstarter.

CE: Only if you care about performance.

BVS: I wouldn't even behind (?) the effort.

Graham: With or without allocators and executors, we don't have that anyway.

BVS: They will expect a BLAS 3 function to get near peak of machine. If they get 1% of peak -- if you don't get affinity right -- tile sizes wrong ... this might be the good example that justifies and drives execution context affinity design.

MW: That's a good idea.

MB: One sentence you said -- I don't completely agree -- languages that support matrix-matrix multiply don't necessarily wrap the BLAS and are not necessarily efficient, but people still use them.

BVS: In terms of rapid prototyping.

MB: I believe you need an efficient implementation, but am not sure you can provide an executor which is an abstract machine sophisticated enough to allow for that.

RD: I would be reluctant to predicate this on having that facility, under the notion that those that are serious are already doing that, we just have to make sure the design we come up with here

would not preclude doing that. Predicating this on having that would potentially sink this from getting in there.

BVS: I would drop that then.

RD: Calling "cpuset" is annoying ...

BVS: Give somewhere as a place where good things can come and live; on-ramp good codes....

RD: But we want to avoid a solution that spins up threads in the background....

BVS: Executors....

BS: Let's not forget needs of other end, small fixed size for graphics. Executors fixed size approach not applicable.

MB: BLAS not even good for that.

BS: Yes.

CE: Does this mean batched? or just one problem at a time?

BS: I'm thinking of rotation matrix to transform a large set of 3-D vectors.

CE: You are doing a matrix operations a lot.

BS: Yes, but I also care about the small case. I want to serve games and low latency, but also professors and grad students, want correct and accurate results.

MH: We've found both batched and single-small-op useful.

BVS: Good time; reproducible BLAS and IEEE floating-point getting ratified now, going into voting and being hammered out. There is possibility that grad student debugging their code could use specialization for reproducibility. They don't have to be stuck in Matlab; they can program compilable code and verify their work. The time seems to be right for "std::blas" ....

RD: Topic change: For matrix applications to graphics, should we talk about interpolation? or do we punt on that?

BS: Add to list of topics to ponder.

Graham: "See if a paper materializes"

[discussion of whether we should "seed the thunderclouds" to draw interest and get more papers and ideas]

BS: Quaterions are representable as 4-element vector, but quaternion math is not vector math. Is quaternion math in scope of this? It's useful for games, and slurps....

RD: We used quaternions in our medical imaging program, when I was rendering hearts.

[MH: Would it make sense to treat quaternions as the entry type of a matrix or vector.]

BVS: That is, is it like complex?

CE: It is, in that respect?

MH: Do you normally think of it that way?

BS,CE: Yes.

RR: We do sometimes operate on entries of a single quaternion, treating it as a vector.

[Lawrence Crowl enters]

BS: Next steps?

CE: First step: fundamental representations, mdspan is supposed to be foundation by which you can do these things. Has layouts and submatrices / subspans. Next plugin layout for mdspan which covers the set of BLAS matrices. Use a "using" statement to alias something to "linear algebra - style matrix."

[CE,MH: discussion of mdspan layout left vs Kokkos::View layout left -- mdspan needs to support strided layout left]

BVS: Some type aliases to mdspan ... give you linear algebra.

BS: You're not suggesting that you constrain storage representation to mdspan, are you?

CE: How so? mdspan is nonowning. ...

MW: Do we need a brief presentation of mdspan?

RD: mdspan as a single underlying container, vs. possibly a variety. What limitations do you put on algorithms to understand ... container ... not saying mdspan is not a good structure to use, but ... [want to see whether it should be the right container].

HF: Questions raised here are about flexibility of layout adapters.

CE: span works with any continuous thing.

RD: The standard algorithms work on more than just span; they work on many different containers. Can layer views with ranges etc. Would hesitate to define algorithms to work only on mdspan.

HF: i.e., should define in terms of concepts,

CE: Yes, then mdspan could be concrete version

RD: e.g., different concepts, like delayed load, or expression template combination, ... I'm throwing out ad-hoc terms ... would welcome people to consider if there is anything missing. Don't want to exclude whole domains by constraining to mdspan.

BVS: I think mdspan is existence enough that we don't have to be stalled. We can be so hung up on conceptualizing linear algebra, that we won't get concrete... it's OK to put a BLAS forward paper-wise, and afterwards abstract it, as opposed to starting abstract and then trying to see if it could be made concrete. I think we could build a spec out of mdspan.

HF: We have a spec for mdspan; we know what the interface is. There's a trivial generalization, to say the algorithm just conforms to an interface [to which mdspan conforms]. LEWG would insist on us following the general design pattern of algorithms in the standard library, that we design to a concept, not a concrete implementation. If we decide not to do that, we would need a rationale.

[MH: mdspan looks pretty abstract to us, but we don't want to exclude farther-out things that RD discussed]

CE: Start with "matrix concept."

RD: That would get people, at least to complain and stir up discussion.

BVS: Shame them into participating.

RD: By providing obviously wrong strawman.

MW: We do that a lot.

LC: Not necessarily intentionally.

[haha]

RD: I think it's a perfectly fine starting plan, to base something off mdspan, and flush out those concepts early on.

BS: I've been conceptualizing those around engines; engines define element layout and storage format. Given my very incomplete understanding of mdspan, seems like mdspan could be a

template parameter that provides / lets one create an "mdspan engine" for element layout and storage.

CE: It's always nonowning

MH: Depends on your pointer type [whether it is nonowning]!

BVS: Its goal is to describe element layout

MW: I want my matrix to be a wrapped data structure, not a set of pointers. I need mdspan for that. Wrapped data structure works better in SYCL.

[MH: mdspan and engine overlap functionality; may want to refactor if you want to use mdspan]

RD: Dimensions of mdspan: always compile-time or run-time?

CE: Rank is always compile time; dimensions can be mixed (compile / run).

[discussion of mdspan capabilities]

MW: Please sign up to the SG14 Google Group. It's an open group; has over 800 members. Future announcements about this meeting and future discussions will be on SG14.

LC: How much mail?

[very little / a lot]???

MW: Very technical discussion. Prefix all your discussions with "linear algebra".

HERE IS THE LINK: https://groups.google.com/a/isocpp.org/forum/?fromgroups#!forum/sg14

MH: I just joined the above group.

MW: [Describes history of group.] Group is mostly about low latency. Monthly meetings 2nd Wed, 2-4 Eastern time, about 8 meetings a year. We run it like a regular SG, and also at CppCon face to face. Usually 50-60 people show up. Now I'm getting more requests to have face-to-face meetings. Starting in Kona, will meet [at WG21 meetings] face to face. I probably won't be able to chair it. Looks like I will start up a machine learning SG, SG20. I will have to chair that. We will probably start having regular chairing meetings, next 3-4 meetings. All my cochairs are mostly in Europe. Expect SG14 meetings moving forward in this capacity. ... In the past, SG14 has pushed forward proposals in C++17 like uninitialized memories (?). We're pushing number proposals like likely / unlikely [branch probability]. SG14 was original idea for incubator. For a long time, SG14 was the incubator group, not just for low latency. After a while, we broke out true incubators from SG14.

MW: We should talk about frequency of meetings.

BS: Does it make sense for us to have a call, with some to-be-determined frequency, jsut for linear algebra, outside of regular SG14 call.

BVS: That's my vote. I don't want to be pulled into SG14 vortex.

BS: Frequency? Once a month?

BVS: Yes. I would probably hot-swap my spot with other people in my math group. I'm the C++ person but not the hard linear algebra user in my group; they know the requirements better than me. Early days will be requirements gathering and scope of work. When we get down to API designs, I could [contribute] more there. Once a month is the right frequency to keep momentum.

MW: Could accelerate the meeting schedule as you get closer [to the paper deadline].

BVS: Or small subgroups with tigher schedule.

Graham: Once a month would mean at most two calls before the next mailing.

MW: Would that be enough time to write any papers?

BS: Complicated by holidays etc.

Graham: Mailing is four weeks before the meeting. Three months between meeting and next mailing.

MH: We don't have to do everything synchronously.

RR: Depends on how much do you expect to get done initially.

[discuss calendar]

BS: Could we schedule three phone calls between now and next meeting?

RD: Expect that we schedule the telecons now, nobody writes papers until after holidays, end up canceling one of the meetings

BS: Some latency from the low-latency team [haha]

RD: Nothing to stop us from canceling or handling [discussions] over e-mail.

HF: I would recommend setting up doodle poll to set up time. We can always cancel meetings, but if they don't get on my calendar, then I can't join, because my calendar will fill up.

Graham: I'm comfortable w/ a semiweekly [bimonthly, fortnightly] cadence, but don't want to push.

BS: ACTION ITEM: I will volunteer to put up a Doodle poll for days of the week.

MW: Book fortnightly, cancel every other meeting.

MB: When you set a meeting time, please consider times friendly to European participants.

MH: Could also start common documents to dump ideas.

BS: I can resend link to Guy's repo with current state of his project and paper in progress. I have a repository where I keep my experimental stuff with strawman interface. I'll send a link to that to the group.

BS: Any further topics?

CE: On github site for mdspan, requests for expression templates with mdspan. We've told them that this would be add-on, not part of mdspan. If you're considering expression templates ...?

BS: That's an implementers' decision. It's how you implement matrix arithmetic.

CE: Did you think of that as in scope or out of scope?

BS: I thought of it as out of scope.

HF: The only thing in there in scope, is if you want to allow expression templates on types, you must be careful about how to specify return types.

BVS: What Hal said. It does show up in interface. "Expression templates" means arithmetic operations generate new types. If you want to support that -- the new type can be the same type. That would be a valid implementation. Matrix + matrix is a new type.

MH: Be careful of compile times.

MW: How to manage conference calls? BlueGene? WebEx?

HF: We can do BlueGene or Zoom?

Graham: Let HF or myself know; we can set it up.

HF: BlueGene and Zoom work on Windows, Linux, or Mac.

DG: Zoom worked very well in premeeting.

HF: Yes, good point.

[discussion of telecons and chairs]

MH: Where shall I put the minutes for this meeting?

RR: There is an SG14 page on the wiki.

MH: OK.

CE: We should notify non-C++ linear algebra community.

BVS: I'm sure Jim Demmel will have a suggestion of people who know C++ and could contribute. Jack Dongarra as well.

BS: How many plan to be at Kona.

Minutes for 2018/12/05 SG14 Conference Call

Minutes by Mark Hoemmen

1. Introductions

Wong explains that the conversation is recorded to simplify note-taking; these notes are for internal use only

Douglas makes his required disclaimer

Patrice will be taking notes

Wong will be taking care of listing participants (Thanks!)

2. Administrivia

Wong : we target first week of the Month...

Steagall : first Wednesday of the Month at 20:00 UTC seemed most popular option

Wong : let's be careful when conflicts occur with WG21 meetings

Bryce : we should announce the meetings on the EXT mailing list

Bryce : we should also add SG6

Wong : SG19 too, but they don't have a reflector yet

(we decide, at Wong's suggestion, to swap points 3 and 4)

4. Discussion of tentative scope and goals for the LA-SIG (Linear Algebra SIG)

Wong : do we need different dimensions?

Searles : I'm concerned about multi-platform support

Wong : do games need more than four dimensions?

Searles : up to four is sufficient for us

Matthieu : four dimensions is a tensor

Nasos : we need arbitrary number of dimensions

Steagall : I think we should distinguish rank and dimension, and have terms to distinguish them

Bryce : I think we should use P0009 terminology

Matthieu : it's not the consensus in the scientific community at all

Nasos : the mdspan proposal is closer

Wong : dynamic extent is dynamic rank

Nasos : extent == size in Fortran

Searles : are we going to support such things as Eigenvalues or will we stick to something simpler

Steagall : scope is something we should determine. Our job is to define an interface than does not preclude important optimizations we need

Steagall : we need to define the boundary / scope of that interface

Bryce : from my perspective, I can't imagine standardizing something that's not general enough to serve the purpose of all stakeholders

Wong : if we end up supporting three kinds of linear algebra, it will get confusing

Bryce : we need to pick a good, reasonable starting point. A minimum viable product to build on

Hammond : data structures and algorithms are two different directions to explore

Hammond : we have to get data structures sorted out with good primitives before we get working on advanced mathematics

Hammond : can mdspan deal with slices and subarrays?

Bryce : yes

Nasos : spares arrays and sparse matrices are important

Bryce : mdspan is particularly well-suited for dense representations. It can do sparse, but we have less field experience with that. I'd prefer some mdspan experts to participate in the discussion before going further

Bryce : sparse is important in the HPC world, but I'd be fine not addressing them in version 1

Matthieu : sparse might be a different data structure

Nasos : the moment you slice, you have a sparse container (discussion ensues)

Matthieu : it could be seen as fancy indexing over a dense array; that makes it different from sparse algebra

Wong : thanks, this gives us an idea of what we're doing

Steagall : we're trying to elicit requirements. It's very useful

Cem : I'd like to know where we stand on compile-time, and on SIMD. Some of these concerns are not compatible

Bryce : I disagree, particularly with C++20

(A short discussion ensues as to who could help this SIG. I did not get the chance to note it all, but Peter Gottschling and Niall Douglas were mentioned)

Searles : such things as float size will need our scrutiny

Wong : we'll get back to this issue later if there's time

3. Presentation, review and discussion of papers

3.1 D1166 (attached), Guy Davidson

Davidson :

- this paper is trying to establish what we need for a linear algebra library
- first section states we should be able to use vectors and matrices of any type, not just fundamental types
- in games, vectors and matrices can benefit from compile-time sizes in some domains

- in general, dynamic size matrices are useful in many domains
- sparse matrices can be nice if they interact with constexpr; they are often implemented with intrinsics

Bryce : with C++20 constexpr, we have is_constant_evaluated() which helps with this

Davidson :

- in some domains, we have SIMD instruction available, not in others, so we want a customizable interface
- put otherwise, we want interfaces that people can customize for their domains
- we need a "done" point, a target at which point we can be confident that we are service 80% of use-cases
- the library should be offering algorithms to do such things as decomposition
- we have a wish list (section 4 of D1166)

Hammond : I worked with people who did parallel Eigen solvers. This is where some of the type stuff comes in : we have "standard" back-ends we can wrap, but these solutions are for fixed sets of types and dimensions

Hammond : some things such as Tensor SVD are still active areas of research and are not ready for standardization

Davidson : I'm not suggesting we solve Tensor SVD in the library, but that the interface allows mixing with such a solution

Hammond : is this working group closed to ISO experts or open?

Wong : the point of a group such as this is to make experts collaborate regardless of affiliation

Hammond : if we want to move into maths, we need mathematicians as customers

Bryce : that's for the future

Nasos : we can work for any type / architecture

Cem : what kind of problems are not solved?

Nasos gives some (could not take notes fast enough)

Knott : we have to distinguish implementation from our work on the interface

Bryce : bullet 7 (interface customizable and overloadable) does not convince me (gives some examples)

Steagall : with respect to expression templates, we hope the decision to use them or not should rest on the implementers

Bryce expresses discomfort with operator overloading in the standard library

Steagall : if your goal is to have a set of types and expressions that mimic common mathematical notation, we'll need some operator overloading

Hammond : are we targeting C-like maths or do we go for general types and concepts? Limiting the number of types can be helpful in going faster, further

Cem : we can choose the back-end when we hit the assignment operator

Nasos : a linear algebra library for C++ needs to work with arbitrary types

Bryce : in version 1 or at some point?

Nasos : my feeling is in version 1

Wong : is there a way we could list features and separate them in version 1, 2, 3...?

Bryce : version 1 should only be the most basic things that the following versions can be built on

Bryce : we should target an operations library

Dalton : it will still need to be customizable on the back-end

Dalton : numerical stability is important

Bryce : I agree that the back-end needs to be customizable, if only to benefit from what is already there

Cem : I think Eigen has it right by letting client code decide whether computation is being run on the CPU or on a GPU

Wong : I would like someone to work on the list of items in version 1

Davidson : I will do it

Searles : I'll help

Wong : a shared Google Drive document could be a good idea here

3.2 The Linear Algebra Prior Art paper (volunteer needed)

Hammond : I don't think we could be exhaustive with prior art. It's a moving target

Hoemme :

- I wrote this to help us avoid mistakes of the past
- People in the past have also done inspiring things for us
- I discuss so-called Object-Oriented Linear Algebra
- I discuss efforts in benchmarking libraries
- I discuss POOMA (a parallel version)
- I discuss high-performance Fortran, and some reasons for its failure (hiding from users things they would have preferred to be able to customize)

Nasos : do we want contravariance in C++?

Hoemme : it depends on whether the library wants to represent distributions

Ronan : we want to think about heterogeneous memory, but not in version 1

Hoemme : in historical libraries, it's been done

Nasos suggests we summarize the discussions from the first half of the meeting for Hoemme, who missed it

Wong does so

Wong : we'll only have another meeting (in January) to get paper ready for the pre-Kona 2019 mailing

Wong : feedback on this paper?

Hoemme : it's a work in progress. I welcome contributors

Cem : don't POOMA and Blitz++ support multiple dimensions?

Nasos : some libraries like Taco have interesting interfaces to look at if we seek something to mathematical notation in code

Cem : there's a book called Matrix Computations that defines the basic operations. We could start from that

Hammond : that book covers a subset of the problem domain, mostly. I'm not sure it's the right place to start

Steagall comes back to the fact that we have different needs depending on the user, and will need a proper interface

Hammomd suggests another book which seems closer to our needs in his eyes

Discussions on the importance of customization follow

Steagall : with regards to ease-of-use interfaces / numerical sophistication. A question I expect to see raised is teachability; we should try to keep this in mind

Nasos : I agree. I see some notation in Python Numerics as inspiring

Wong : I'd like to return to Guy (Davidson)'s paper

On the wish list, item 1 :

- Wong : we want to know if it's going to be parameterized
- Steagall : do we allow arbitrary numerical types?
- Cem : could we have a concept of a number?
- Dalton : it's hard. The Ranges people are working on this
- Nasos : I'd make sure addition and multiplication are
- Hoemme : I'd like to know if they are thread-safe; if they do dynamic allocation; if they are allocator-aware; etc.
- Searles : we don't want further standards to break the original
- Dalton : I'm in favor of the paper's position
- Hammond : array primitives can be defined on any type. I other cases such as SVD, it's less clear
- Nasos : some things are algorithm-dependent more than type-dependent
- Dalton : type promotion is an issue
- Knott : Guy (Davidson), were you planning on presenting some of your CppCon material here?
- Davidson : I can
- Knott : it could be informative
- Davison : I'll share my slides to the reflector
- Marco : do we want fixed-size containers? Dynamic-size? Dynamic-rank?
- Matthieu : we can use both
- Nasos : can we change the number of dimensions dynamically?
- Matthieu : for machine learning, definitely
- Hammond : Fortran has RESHAPE which is useful
- Steagall : does anthing beyond rank 2 count as linear algebra?
- Hammond : definitely
- Steagall : should be support it?
- Hammond : definitely
- Cem : in my library, we are trying to implement this

- Reverdy : does mdspan have dynamic ranks?
- Hoemme : I think the dimensions on the underlying fixed size are not stored; only the dynamic dimensions are

On the wish list, item 2 :

- Steagall : I think we've pretty much discussed this already

On the wish list, item 3 :

- Steagall : I think we should leave the return types unspecified to allow for expression templates
- Nasos : we don't want to expose expression templates to user code
- Steagall : that's the point of leaving it unspecified in the standard
- Knott : we still need some concept for the return types
- Nasos : will we use internals of matrices or return types for expression template details?
- Steagall : it's a question I think should be left to implementors
- Knott : debugging this sort of code is difficult
- Hoemme : the representation of a matrix differs from the representation of an expression
- Cem : could be provide two layers, one functional and one operator-based?
- Steagall : do we want expressions to convert to matrices, to simplify passing return values from functions or operators as arguments to functions?

Steagall : we need to move on to logistics

Wong : next meeting would be on January 2nd

Searles : do we have enough time for Kona?

Wong reads the schedule

Steagall : we'll be able to reconvene on the 1st week of February if you have updates

Searles : do we continue this discussion next week in SG14?

Wong : no, full plate there with a discussion on the difference between "embedded" and "hosted"

Wong reminds people of the importance not to distribute the recording

Patrice will send minutes to Wong and Steagall. Participants are welcome to verify if they were quoted appropriately

Nasos : I see this as a big undertaking

Wong : we have many major undertakings. This one could take two-three years before something surfaces

Wong : if anyone wants to go to Kona, book your hotel now

Steagall ends the call.

Minutes for 2019/01/02 SG14 Conference Call

Minutes by Mark Hoemmen

20:00 UTC, 02 Jan 2019

Present:
- Mark Hoemmen (mfh)
- Bob Steagall
- brett
- Guy Davidson
- John McFarlane
- Philippe Groarke
- Cem Bassoy
- Patrice Roy
- vincent
- Michael Wong
- Jayesh Badwaik
- Nasos Iliopoulos
- Charles Bay

Bob: One goal of this meeting is to look at the Google doc we've written, and separate out the "beginning," "intermediate," and "advanced" levels of the design.

Michael Wolf (MW): The sg14 Google group has been having an issue (on Google's side) that prevents many people (including us) from posting. std proposal group has been having the same problem. If you are trying to send messages to SG14 and they are not showing up, this is what's going on.

Here is a link to the document that Bob mentioned:

https://docs.google.com/document/d/1poXfr7mUPovJC9ZQ5SDVM_1Nb6oYAXlK_d0ljdUAtSQ/edit

Brett will drive; looking at document:

Brett: Regarding Layer 0: Why treat row vectors and column vectors as different types?

Guy: Bob and I decided on this design so that we can overload operator* to compute both inner and outer products. [For example, a row vector times a column vector is a matrix, the result of an outer product; a column vector times a row vector is a scalar, the result of an inner product.] Another choice would have been to have used explicit function names like "inner_product" and "outer_product," rather than overloading operator*.

Brett: What about vectors of different sizes? e.g., what if [you attempt to do arithmetic with] a length 3 row vector and a length 4 row vector?

Guy: It would be a compile-time error if the vectors had a different number of entries.

mfh: Compare to span and mdspan: Just like span and mdspan allow either compile-time or run-time sizes, it could either be a compile-time or a run-time error (e.g., to compute the inner product of two vectors with different lengths).

Bob: Guy's reason for distinguishing between row vectors and column vectors in the type system is right on. We want to use the type system to minimize surprise. Second, we imagine these row and column vector and matrix types as adapter types. They would wrap "engines" that provide the actual implementation. Examples of engines: fixed size and fixed capacity (compile-time dimensions), or dynamic capacity and dynamic sizes, or fixed capacity but dynamic sizes. Engines handle implementation details, and traits classes provide best performance.

Brett: I was just thinking that row vs. column could be an enum.

Bob: Then you would need to make a run-time decision as to whether it's a row vector or a column vector. What does operator* return? Depends on whether the inputs are row vectors or column vectors.

mfh: Could just use explicit names [like inner_product and outer_product, instead of overloading operator*].

Guy: Could also ignore vectors and treat everything as a matrix; row vectors would have one column, and column vectors would have one row.

mfh: Explains how the linear algebra library we built with Kokkos::View (inspiration for mdspan) works. operator() could be more efficient if one knows at compile time that there is only one dimension: e.g., x(i,0) could be less efficient than x(i), because the compiler may not realize that it's stride-one access. Thus, our library has some run-time special cases for "matrices" with a single row or column, where it specializes from a rank-2 mdspan (x(i,j)) to a rank-1 mdspan (x(i)). If a library treats all vectors as matrices, then it might not be possible to have efficient specializations like this without reaching inside of the interface. That is, we would want the interface to be a zero-overhead abstraction that suffices to implement any linear algebra algorithm; if we can't express "vector" at compile time, it might not be a zero-overhead abstraction.

Brett: What about the type of a list of eigenvalues? complex? float or double?

Bob: "Matrices" generically mean matrices, row vectors, and column vectors. "Matrices" should support element types that make sense. float and double, complex float and double. It may make sense to allow them to work with signed integer types. For medical imaging, MRI reconstructions, raw data from scanner are signed 32-bit complex integers. Guy and I think it

makes sense also to support element types which are arithmetical, that are not part fo the standard, e.g., fixed-size floats, or high-precision or dynamically resizable floats -- anything that behaves like a real number.

Nasos Iliopoulos: Why not a general field? Why not allow anything?

Bob: We haven't figured out the set of constraints that should be imposed on the element type of the matrices. Concepts may help with this, e.g., a "field" concept.

mfh: Computer science concepts like "trivially memcpy-able" may be just as important for the implementation as mathematical concepts like "semiring" or "field."

Bob: Another important design point: Source of memory? Addressing model? Natively, one addresses elements through normal pointer arithmetic, but "fancy pointers" should work too.

mfh: mdspan takes care of all of that except for actual allocation.

Bob: Reasonable. Related question: Memory ownership? Could have an engine type that leverages mdspan; it could be an owning or nonowning view into mdspan.

Bob: Capacity and resizability. std::string, std::vector have capacity and size. In LAPACK, one can specify column capacities > column sizes [via arguments like LDA, "leading dimension of the matrix A"]. Useful for functional MRI. Dynamically resizable memory could have row and column sizes and capacities.

Bob: Element layout?

Guy: When we originally presented this at Cppcon, the room assumed I was talking about mdspan. I wasn't, but I'm pleased at the progress mdspan has made. mdspan could be an implementation detail of this library.

Brett: Is our focus to try to emulate mdspan?

Guy: Not at all. mdspan can be used to implement this linear algebra library.

Bob: Yes. mdspan could turn out to be a useful piece for how a library implementer would actually implement some of the default set of underlying engines.

Guy: mdspan and linear algebra are at different levels of abstraction.

Brett: What engines are we going to support?

Bob: Will come to that. Now: element access and indexing. User of this library would like to access individual elements in both const and mutable fashion. 1-D indexing for vectors, 2-D for matrices. Guy and I differ on 0-based or 1-based indexing. Guy favors 0-based, I lean towards 1-based.

?: Could we do both? Is that possible?

Bob: Possible.

mfh: It could be a function of the layout, in mdspan terms. [With Kokkos::View, we have explored layouts that allow different index bases, to help simplify structured grid codes.]

Bob: Element type promotation: We think it's important to allow expressions that have mixed element types. If you multiply a matrix of double times a matrix of complex double, it should be a matrix of complex double. Element type promotion should occur inside expressions, so that precision is never lost if possible.

John McFarlane (JMF): Could that depend on types? [yes] I would prefer leaving that to the types themselves. I can give examples.

Bob: I would like to see those examples.

JMF: Two examples: fixed point and units (for dimensional analysis). If you multiply two numbers together, decltype(left*right) should be the type, else it could cause a conversion. Any other option could only produce as much or more work and constrain the behavior as much or more as just letting them behave as they usually do. Also: if you multiply two shorts together, you get int. signed char * short, results in int. We have been trying to get GLM to work with fixed-point types; the units guys have been trying similar thing; the first thing you notice is if you add two vectors together, whatever the scalar type fo the two operands, that is the type of the result.

Bob: Could [customizable] element promotion traits help with that?

JMF: I guess so? Seems like you could just wrap the input arguments' element types in another type that changes the result type. Would it not overstep the bounds of a linear algebra library to permit customizing the type of the result of an expression?

mfh: [Explains related history of the POOMA project, discussed in the history paper in progress.]

Bob: We conceive the result type as customizable via element traits. This also defines the result type for cross-engine arithmetic expressions. Default is still decltype(A*B).

JMF: Do you promote before or after operating? That's crucial.

Bob: That's an implementation detail.

Nasos: Why would a type promotion facility be a part of an algebra proposal, and not a proposal on its own?

Bob: Traits would need to interoperate with linear algebra. But that's an interesting idea -- breaking arithmetic promotion traits out of linear algebra is analogous to breaking linear algebra out of graphics. I think we should try it inside the linear algbra proposal first before breaking it out.

Brett: But isn't that part of the type traits library? Shouldn't we do that as a separate ... run time type traits ... to be supported?

...

Bob: What do you do when you have expression that mixes engine types? e.g., fixed size, fixed capacity matrix, times dynamically resizable matrix? Analogy with element type promotion. We think promotion should be in the direction of more generality. For example, towards dynamic. But if all fixed size compile time, results should be too.

JMF: Could you perhaps do this (defining the result type of expressions) by specializing common_type for pairs of engines? That might be blunt, since different operations might want to be different things, but that would be first goto.

Nasos: Could "engine" include GPU, MPI?

Bob: Currently: Fixed size and fixed capacity; dynamically allocated memory that might be mdspan'd; transpose engine. Role of engine is to manage storage and layout.

Cem: What does "engine" really solve? If that many possibilities, e.g., with MPI governing parallelism.

Bob: Engine is the mechanism of customization. Matrix will be parametrized in terms of engine. If you want a new storage layout or if you want to support a new type of hardware, you would do this by writing a new engine type and instantiating your matrices in those terms. Analogy is with custom allocator.

Cem: What about default engines?

Bob: Set of default engines -- some immediately obvious, e.g., fixed size fixed capacity engines like 3x3, 4x1 commonly used in graphics.

[some confusion about what "engine" means]

mfh: [Read history paper in progress, in particular the POOMA section, for a definition of the "engine" idea.]

Bob: Concurrency and parallelism are similiar but different. Interface specification should not preclude implementations from exploiting these. Custom engines come in here.

Cem: It seems like an engine should just be an std::tuple of template parameters for your matrix, like how the Eigen library does it.

mfh: Engine should not _be_ the template parameters; one should _deduce_ the engine type from the matrix's template parameters. [This seems like a fine distinction, but it lets matrices have a default engine if users don't specify it, and would also let users specify only some but not all optional template parameters. Compare to how Kokkos::View deduces the layout, execution space, and memory space from its template parameters.]

Bob: Engine comes from template parameters, but the engine could itself be parameterized.

mfh: I would prefer factoring out the allocation aspects of engine from the layout aspects of engine. [Compare to how Kokkos::View factors out execution and memory spaces from the array layout.]

Bob: This depends on how you factor out the template parameters. It's an aesthetic choice, which of the 8-9 aspects belong to engine vs. adapter types. Layout is an intrinsic property of the engine itself, though one could define a parameterized engine by layout.

Nasos: I think Mark as a point. "Engine" from HPC side sounds like computer architecture, e.g., shared memory, GPGPU, MPI. We should factor out architecture from layout.

mfh: An engine could have a default preferred layout. For example, in Kokkos, the "Cuda" execution space defaults to use column-major ("layout left") matrices, and CPU execution spaces default to use row-major ("layout right") matrices.

Nasos: That makes sense to me.

Bob: ...

mfh: Regarding the boundary between Layers 1 and 2 in your document: Layer 1 conflates computer science / BLAS ideas (e.g., transpose) with mathematical / LAPACK ideas (e.g., inverse).

Guy: I culled Layer 1 from a standard linear algebra textbook. what do you mean?

mfh: Implementing inverse depends on numerical analysis knowledge; implementing transpose does not.

Cem: ... layer 1 as fundamental layer ...

Guy: ...

mfh: [Explains above:] The BLAS and LAPACK are separate, because they express a dividing line between computer science domain knowledge, and numerical analysis domain knowledge. For example, you may need a low-level understanding of computer architecture in order to

implement dense matrix-matrix multiply efficiently, but you don't need to understand floating-point arithmetic. In order to implement LU decomposition correctly, you need to know something about floating-point arithmetic and numerical analysis, but you don't necessarily need a low-level understanding of computer architecture.

mfh: To relate this back to a C++ standard library context: The people most likely to implement and maintain a standard C++ library are unlikely to be trained as numerical analysts, and vice versa.

Guy: Some things in Layer 1 are possibly too simple for 1 and belong in Layer 0; some things in Layer 1 are too complex and should be in Layer 2. Layers could be separated either by domain [as above, in mfh's argument], or by complexity of functionality. We called Layer 1 "geometry" because it's the simplest set of things you could do with linear algebra. In terms of the textbook I've been reading, it was a set of operations that required reading the least number of pages into the textbook. It would be daft for a foundation paper not to include at least some features from Layer 1. A first paper could include just the "trivial" functions from Layer 1; determinant and inverse could come later. I included inverse because the immediate audience of graphics developers would want inverse.

Brett: I would agree with that. Do we need modulus, identity, subvector?

Guy: There are a few ways to write the paper. The beauty of linear algebra is that dependencies are well defined, so you can stop at any point to standardize, and write successive papers later to standardize more features.

mfh: Skill sets of likely implementers should define the boundary between layers. There's not much overlap between "numerical analysts" and "C++ standard library implementers."

Guy: Yes, I agree.

Vincent: "Geometry" -- does that mean "geometry of matrices," like transposing, or the "geometrical interpretation of matrices"?

Guy: I mean the latter.

V: Then, if we start talking about that, we enter a whole new domain. If we only do the Euclidean part, it looks simple, but if we do more general geometry, we need tensor algebra and differential geometry. That's the right way to do it, but would take a lot more work.

Guy: This is a linear algebra paper, not a tensor algebra paper. Once you go that far it stops being "linear."

V: Not sure how to define this correctly without tensor algebra.

Bob: "Geometry" means common kinds of geometry operations one might use when implementing a game. Not thinking about geometry of multidimensional space in the differential geometry / tensor calculus sense.

Guy: That's quite right. Layer 1 is about applications of those. We certainly do open the door of tensor analysis, but we don't have to go through it in this paper.

V: Yeah, sure.

Guy: Whole point of successor papers -- I don't want to tackle the whole field of linear algebra in a single paper, otherwise we'll get nothing until 2029.

Brett: That's why ... layer 0 right now.

Cem: Guy, What's your favorite textbook?

Guy: I've read a lot of textbooks. my favorite is by Seymour Lipschutz:

https://www.amazon.co.uk/Schaums-Outline-Linear-Algebra-Outlines/dp/1260011445/ref=sr_1_5?s=books&ie=UTF8&qid=1546462998&sr=1-5

Bob: I used Howard Anton's "Elementary Linear Algebra." See also David C. Lay, "Linear algebra and applications."

Cem: What about Gene Golub's book, "Matrix Computations"?

mfh: It's good but not laid out pedagogically.

Cem: Good to have standard functions we could use for linear algebra. "Geometry" stuff might be specicial to geometry, but linear algebra needs more complicated stuff like LR decompositions.

Guy: Those could be in successor papers. Plenty of scope for growth. I took my favorite early concepts for broad use. "Geometry" tag for Layer 1 simply tries to find a domain for them.

Bob: Our goal is to define an interface that allows for good implementations. Interface should be implementable so people can get the performance they like. To come back to what Mark said a few minutes ago: Where do we draw the line? You made a point about inverse -- an example of one kind of decomposition, like least squares or SVD. Are these things that should be part of first paper, or follow-on papers? Should they have a default implementation provided by the standard library for at least float and double? These are all questions to be answered. We mentioned SVD last call -- several objections, it's really hard, how could we make it right for everyone all the time? I don't think that's achievable, but by analogy with allocators, one could perhaps implement specialized versions of those algorithms for special element types. Decompositions and eigenvalue problems belong at least to Level 1, and more likely to Level 2.

mfh: One way to define Layer 1: Does it suffice to implement all the algorithms in Gene Golub's textbook?

Cem: I got confused because "inverse" only needs a matrix decomposition for large matrices. For small matrices, like 3x3, one could implement inverse directly, as an explicit formula. That's why I got confused, because it's in Layer 1. I was thinking that the BLAS 1, 2, and 3 could define Layer 1. Layer 2 could be more complex operations, like what LAPACK provides.

Guy: I'll have to think more about this.

mfh: If we had parallel Ranges, would that undermine Layers 0 and 1 somewhat? For example, would a fully parallel Ranges version of transform_reduce make "inner product" unnecessary? Imagine a C++ developer who is not a linear algebra person; they might object to a linear algebra library adding redundant and "more special" ways of spelling existing things like transform_reduce.

Guy: I have thought about Ranges and Layer 0. We could implement Layer 0 in terms of Ranges. But here we are defining vocabulary.

mfh: So, for example, transform_reduce could be an implementation detail of inner product?

Guy: Yes.

Bob: We're trying to provide set of vocabulary types that mimic mathematical notation on paper. Hence e.g., row vs. column vectors.

Guy: We are without an agenda, an therefore without a schedule. It's late in UK.

Brett: I'll have to leave pretty soon myself. I think we got a lot of info we need in order to work with, an we just need to start getting some papers out.

MW: Sounds like we've done a lot of good work, and maybe it's time to come to a reasonable agreement on paper(s) for the 21 Jan (Mon 10 am ET) Kona paper deadline. Do we have 1-2 papers? This proposal + history document?

Guy: I already have a paper number, 1166, into which I would fold this document and discussion. Bob and I are also working on a different paper, 1385 (?), that will describe our first iteration of a response to 1166 (as an API or design sketch).

mfh: Did you mean to fold the history paper into 1166?

Guy: No.

mfh: Would it be helpful to request a separate paper number for the history paper, so that Guy and Bob can cite it?

Guy,MW: Yes, that would be good.

MW: This is last linear algebra conference call before the Kona paper deadline. Guy, would you like others to collaborate with you on this paper? Also, I hope you have booked your hotel rooms for Kona. If not, please do so as soon as possible; it may be too late. If there are enough people at Kona, I plan to hold an SG14 face-to-face meeting to present these papers.

Guy: We are by no means excluding collaboration. Regarding presenting at Kona, we will present in SG14 Friday morning.

MW: [confirms SG14 Friday morning]

MW: SG19 (machine learning) is Friday afternoon.

Guy: I have a rolling presentation. I can make it again in Kona. I will make it in two weeks, I'll be speaking at Prague, avast meetup, live streaming. 16 Jan?

MW: Please post [streaming] link to SG14.

Guy: I will.

MW: I think this call went really well.

Bob: Slack has "sg14" organization -- could be alternative means of communication, in case the Google group doesn't work.

brett: It's on cpplang, probably #sg14.

Bob: Next call will be first Wed of Feb: 06 Feb. 3pm ET, 20:00 UTC, per google poll.

Minutes for 2019/02/06 SG14 Conference Call

Note Taker: Jayesh Badwaik

Bob lists out the agenda and asks for volunteer to discuss them.
Mentions P1385R1 to be complete by Kona
Asks if someone wants to discuss P1166R0? No takers.
Asks if someone wants to discuss P1416R0? No takers.
Discussion about P1416R0 deferred to Kona face to face meetings.

Asks for volunteers for P1417?
Jayesh talks about Joel's observations about expression templates.

Cem Bassoy mentions how there are two types of expression templates and their impact on the decision.

Michael Wong mentions that he has his own ET types and will follow up on it.

Cem Bassoy talks about fixing misleading description of matrix operations about Matlab in the paper.

Nothing more on the paper. The dicussion proceeds to P1385 where Bob starts describing the paper.

Bob:
By high performance, the paper means: reasonably high performance. Not extreme high performance.
There are people working in labs are looking for highest possible performance.
For them, ability to tune and customize is key.

And take care to ensure that the tunability and customization
of interface is maintained to ensure that high performance algorithms can be implemented.

The objective is to provide an interface specification and not an implementation.

The objective is for the interface to be intuitive, expressive and provide Out of the box performance (Rivaling calls to Blaze/LAPACK) within 80-90% of the possible performance range.

It should provide basic build blocks to manage things like layout, access to memory.

Building blocks could also be used to represent other mathematical things of interest like Quaternions, Tensors etc, but that is for later.

It should provide straightforward set of facilities for customization.

Bob then describes the notation in the paper to solve ambiguities between fields of computer science and mathematics and points out definitions which are different in the two fields.

Cem Bassoy talks about using field as element types. Bob responds that talking to Casey Carter indicates that using Concepts to do that is extremely difficult.

Jayesh says that one of the best ways to do it is manually specifying which type is field, since the actualy float point numbers (as specified by the IEEE Standards) are not a field due to NaNs and negative and positive zero.

Bob explains 3.2 with special mention to MathObj. He explains how rank for tensor and matrix are ambigous.

Cem Bassoy: says that rank is also referred to an order and may be that can be used.

Bob then proceeds to overloaded terms and disambiguate the two things.

Bob then proceeds to Layered approach about linear algebra and explains that the paper is the fundamental services only proposal as listed in 4.1 of the paper.

Bob then moves onto tensor and explains why to exclude tensors from the design and not to combine the design. Similar logic for quaternions and octonions.

Cem Bassoy asks a question about quaternions and octonions.

Bob clarifies that the only intent is to separate quaternions completely, and clarifies that using the example of compledx numbers. Even though complex number can be represented by the vector, it is not a vector and hence should be modeled by a different type.

Bob then goes onto the section of design aspects (Section 5.3)

Jeff Hammond notes that Fortran allows any indexing range and says that C++ should go that route for allowing arbitrary indexing. He says there is tremendous value in having both 0 and 1, and once you are supporting two, you might as well support arbitrary ones. He gives an example of angular momentum.

Ronan Keryell says that mdspace proposal is very similar. Michael supports the

idea or 0 or 1.

Bob explains his motivation of 1-based indexing, that it's easier for people to transfer algorithms from book to code directly.

Bob then moves onto Element Type.

Bob asks question about mixed element types (section 5.11) and how to deal with it.

Bob brings up constexpr of linear algebra and requests for ideas on how to make it work with linear algebra in general.

Cem Bassoy wants some elaboration on parallelism and SIMD.

Bob gives an example of how a multiplication trait can be specialized for a certain type to give a SIMD implementation of the type multiplication.

Cem Bassoy asks if there has been any thought about execution polcies.

The discussion now moves onto section 6. Even though its an R1 draft, the interface is R0. No big surprises there. Fundamental design principles are unchanged. They take a bottom up approach instead of top down. An engine type is something that is responsible for something managing storage.

The real difference for the dynamic engine is in the methods having to do with changing the size and capacity.

Jeff Hammond: Says that for complex numbers, its better to have much better to have more, because Lapack sometimes wants transpose without conjugate, sometimes without. Jeff asks for a template parameter based engine. Bob says that it will be possible to do so based on templated engine. And since, engine knows the element type, it can be then be optimized for it.

Jeff says that part of the thinking in being careful about this is that, the design should not actively interfering with SIMD as the similar fault exists with `std::reduce`.

Bob would allow one to create additional engines or at least not preclude one from getting performance.

Now, we move onto the traits.

Partial specialization of traits in 6.3 traits is suggested to be allowed.

Bob mentions the repo of the paper and the interface code and mentions that in the repo.

Unfortunately, the standard library does not allow mixed element types in complex which is the subject of a different paper by Bob.

Promotion traits for negation are trivial.

Bob explains, why have a separate promotion type for engines. The reason to keep them separate, that way the implementer might to use expression templates as the result. by not specifying that, the freedom is to the implementer and this promotion type traits allow that flexibility.

Bob then descirbes the Mathematical Types in Section 6.2

Guy Davidson tells us that he presented the paper on Tuesday on CpponSea and the reception of encouraging. There is real enthusiasm for the paper.

Bob then closes out the meeting.

Minutes for 2019/02/20 SG14 Kona LEWGI review

D1385R1: LEWGI

DH: Davis Herring
LC: Lawrence Crowl
RK: Ronan Keyrell
WS: Bill Seymour
OL: Ollie Lo
DS: Dan Sutherland
AT: Alan Talbot
CC: Chris Connelly
BvS: Brian van Straden
JB: Jorg Brown
GL: Graham Lopez
S: Sebastien
AL: Andrew Lumstine
VR: Vincent Reverdy
GD: Guy Davidson
DL: Damien L-G
EF: Eric Fiselier
MH: Mark Hoemmen
BS: Bob Steagall
BL: Bryce Lelbach

D1385R1:
(13h58)
BS (presenting): The scope is perhaps surprisingly narrow.
Initial requirements (or goals) for "typical users" and "super users". Typical users want ease-of-use with mathematical syntax and reasonably high performance. Some have said operator overloading is a "must-have".
GD: In the gaming community, they are already using such operators and would ignore a library without them. All the homegrown libraries are practically identical.
BL: Is there prior art listed?
GD: They're proprietary.
BL: Please include whatever prior art you can in the next revision.
BS: Super users value performance from customization.
BS: Goals are mathematical vocabulary types/operations; a public intuitive interface; easy LAPACK-like performance; infrastructure for memory storage/layout (might also be extended to quaternions/octonions/tensors); customization facilities; ...at reasonable granularity.
BL: Is there any discussion of the goals?
MH: We'll have to see what the valid element types are.
BS: We have to distinguish mathematical terms of art from overlapping computer science terms, especially matrix, dimension, rank, and vector.
DH: Is rank tensor rank or matrix rank?

BS: Matrix rank, but we try to avoid it.
BS: We distinguish (indexed) arrays and (...) matrices.
MH: We use extent in mdspan.
BS: "MathObj" means a matrix, row_vector, or column_vector. An engine manages the resources for a MathObj (including exeution context), selected by template parameter but otherwise private.
BS: Row size is number thereof, bounded by row capacity. Engines may be fixed-size or just fixed-capacity or neither.
BS: We want a layered, iterative, and incremental approach, leaving higher-level functionality for further proposals on top of a foundational layer of matrix/vector arithmetic that supports new element types, engines, and arithmetic operations.
BS: We do not address tensors because they have additional invariants and different interfaces.
GL: Does the same apply to vectors? A matrix with one row or one column is not the same as a vector. What about outer products?
BS: The outer product is the matrix outer product.
GL: How I do get a vector, not a Kronecker, outer product?
BS: Sorry -- we do compute the actual outer product.
GL: But the Kronecker product has a different size. A row_vector is not a matrix, then.
MH: The differences between mathematical objects are less important than the difference betwen mathematical objects and computer science objects.
GL: But if we're going to be precise about matrices vs. tensors, we should be precise about operations too.
BS: It depends on which operators you want.
VR: From a differential geometry background, it's important to distinguish matrices and tensors.
AL: A vector exists in the abstract without being a row or column vector.
BS: I want a scalar product from row_vector*column_vector, not a 1x1 matrix.
AL: But should * perform the Hermitian inner product for complex vectors?
GD: Thanks for pointing that out.
BS: Memory can come from various places and utilize fancy pointers. Ownership and capacity (as distinct from size) are also important. In MKL LAPACK, there are 18 parameters to the SVD function. I can add a row when doing multiple regression.
MH: In some applications, you incrementally update the factorization of a matrix.
BS: We have the usual C/Fortran element order issue and 0/1-based indexing.
AL: There's no debate about 0/1.
BL: We can take a poll.
BS: Some people want arbitrary bases.
RK: In Fortran you can pick any start.
MH: Some libraries use negative indices to wrap around.
BS: We have to support heterogeneous operations between elements via "element promotion". The underlying engines might be different, so we do "engine promotion" (to dynamically-resizable if necessary) too.
MH: So Nx3 * 3x3 is Nx3 (with only N dynamic).
DH: What about engines with different execution contexts?
BS: We've focused on the storage issue (adding execution context only Monday evening).
BS: We want to customize for performance based on element layout or parallelism/SIMD, so we have "operator traits promotion" as well.

BS: We haven't done constexpr yet, but we can do it.
BS: That's the introduction.
BL: Are there examples to look at before the synopsis?
MH: We have test cases.
BS: We have trivial examples of usage.
BL: Let's look at those next.
BS: We have mixed element types and then mixed engine types.
BL: What happens with temporaries? Is there any DSEL?
BS: The design allows for it.
DH: Should we avoid auto for subtleties of memory management?
BS: It's allowed to return something convertible to the matrix type.
BL: This is the valarray issue; no one uses it due to such ineffiencies.
MH: You'd have to make sure to assign, not return by auto.
DH: Can operations on rvalue re-use storage?
BS: It's not disallowed.
MH: The customization and free-function interface could be used to control element types and storage.
BL: Eigen avoids temporary copies but also produces an optimizable expression tree.
MH: Eigen sometimes created temporaries on purpose.
BS: Various traits compute the types and values for an operator.
BS: is_complex detects complex<T>.
MH: numeric_limits can do that.
GD: OK, less wording for us.
BS: Only arithmetic or complex types are allowed as elements.
BS: matrix_element_promotion selects result types.
BL: Why not common_type?
MH: That picks complex<float> between it and double.
BL: Is this a customization point? So all engines use this trait?
BS: Yes. You can use it to multiply a matrix of doubles by a row_vector of bignums.
BL: common_type should support that.
BS: If we don't end up needing the trait, that's fine.
BvS: Do we need the operator()(i,j)? Distributed memory can't do that.
BS: You can always not instantiate them, or we could SFINAE them away.
BL: What's size_tuple?
BS: A convenience for things like capacity().
S: Why disallow 0-sized matrices (for forwarding)?
BS: I'll have to think about it. fs_matrix_engine is supposed to be simple; maybe another one supports that.
BL: Does it hold data?
BS: It contains its own data.
BL: You can't do alignment or padding.
BS: Correct.
DH: The mp_bias is undefined behavior.
BL: We'll poll 1-based indexing.
RK: What about operator[]?
BS: Not now; maybe once we can have two arguments.

RK: Generic code will already be using it.
OL: What about setting capacity?
BS: It's fixed size; the capacity is equal to it.
GL: Why size_t instead of int?
BL: It's precedence and consistency; int is the modern advice, but maybe we can get away with it.
GL: So there's no reason particular to this situation.
BL: Why are the operator() overloads not noexcept?
BS: It's a good question; they have a precondition.
GD: Contracts?
BL: Not in the standard library yet.
MH: The traits and hooks are more important than the details of an engine, right?
BL: Yes, but we should look at the details too.
DH: Why are things like is_rectangular types instead of constexpr values?
BS: Consistency with allocator_traits (like POCMA) and pointer_traits.
BL: Can you copy a fixed-size engine to a dynamic engine?
BS: Yes, and vice versa if the sizes are right.
BL: Then you need constructors.
BS: The matrix knows how to copy the engines.
BS: dyn_... looks like fs_matrix_engine except for having resize/etc.
VR: Why not use extent from mdspan?
BS: This is a strawman implementation. We want to find out whether the traits system, etc. makes sense.
VR: You'll need a concept of an engine, which will depend on things like an extent.
BL: Are these engines actually part of the proposal or merely examples?
BS: The proposal says there should be basic fixed-size and dynamic engines; these are examples of what they might look like.
(break at 15h01)

(resume at 15h21)
GL: Now that I've had time to think about it: at the very top level, how attached are the authors to the idea that there need to be separate row_vector and column_vector types?
BS: It's not a matter of personal attachment but of practicality of interface: row_vector*column_vector produces effectively a scalar. If I treated them as matrices, I would get a 1x1 matrix from which one would have to extract the single element.
AL: A function called dot could give the scalar and wouldn't be too ugly.
GL: Why wouldn't operator* work?
BS: If the size were always known statically, you could overload * so as to get a scalar. With runtime sizes, that doesn't work.
GL: But vector*vector is a scalar.
AL: Unless it's the outer product.
GL: But * could mean inner product everywhere; it's the most common.
MH: This is the risk of using an operator.
GL: I can't read * without Hungarian notation if it might be an outer product.
BL: Would you be happier if we had no *?

GL: That's one way; if you do that, there's no need for two vector types. Vectors are vectors, and having to distinguish them is useless.
BL: I presume row_vector and column_vector have the same interface.
GL: But I have to keep up with which is which.
AL: But * can't be used for inner product anyway.
BL: If I have a function on a vector, is it templated on a concept or accept a base class? We have two types that are the same thing just to pick the right overload?
BS: Yes.
BL: Even with one type with a row/column tag, you have to distinguish.
GL: If you can keep track of the types, * isn't ambiguous.
OL: Do you have elementwise multiplication?
BS: No; we don't define how multiplication should work.
DH: There is a consistent definition of * here for
GL: Outer product means something different for vectors and for 1xn, nx1 matrices.
DH: As embedded into the space of matrices, you get the matrix outer product.
GL: I disagree.
DH: Using static/dynamic extents like mdspan would allow specializing to select a scalar result as appropriate.
EF: Does the grabbiness of the operator templates cause problems in a real implementation?
BS: We checked with a strawman operation.
EF: When you put this into std, they compete with many other operators even for irrelevant operands.
GD: We have Kronecker products and Hadamard products; we want * for matrix multiplication. Ideally we would have fancy Unicode operations for the exotic products, or just free functions.
GL: This is a weird middle space. You could just say that you have matrices, or 0/1/2-dimensional things, and inner product works on everything, or you can have matrices and vectors, and operator * means something else there. The blending isn't obvious.
GD: The specific types offer optimization opportunities and may be convenient for users.
GL: So you're advocating for not having any distinction and neither row_vector nor column_vector.
GD: No I'm not.
GL: But this way you're getting the ambiguity and the complication.
BvS: The vector types could just be aliases.
GD: My initial implementation did that, but it caused problems.
BL: We could disambiguate with extents.
BS: If the extents work, we could go down to two primary types.
S: If vector is just an alias, we seem to need tensors for consistency. We can't add them later for ABI reasons.
BL: Why wouldn't we just use mdspan for tensors?
S: It could be the class that unifies the concepts.
AL: Again, just use a function for dot product (and dot_conjugate). Then vector can be vector.
EF: I see static_asserts -- should we be using concepts instead?
BL: For this proposal's timeframe, yes.
MH: I argued last night that mdspan is really a low-level multidimensional array, not necessarily a matrix. We use our equivalent as if it were such a tensor in a BLAS-like fashion. There's room above it, even if it's not my priority.

S: So you can't use mdspan to unify these concepts.
MH: You can use it to unify the storage only.
GL: It's not necessarily a typedef unification.
MH: mdspan lets you make views of rows/columns/etc.
BS: The row_vector/column_vector/matrix classes wrap the engines, with SFINAE for resizing.
EF: It looks properly dependent.
BL: Let's look at the traits.
BS: Next is engine promotion traits per operation; for example, it picks an element type, an allocator for that, and makes a dynamic engine for that.
EF: Please give long names of template parameters. I want to see if there might be conflicts specializing on types I don't own.
BS: There are higher-level traits later.
DH, CC: Template parameter names can reduce visual noise.
VR: More generally, I think mdspan is a view on storage. Why can't a matrix combine it with a standard container?
MH: mdspan allows owning pointer types.
BL: We'll have to poll on that.
S: If I had a CPU and a GPU matrix, would they have different engines?
BS: Probably.
S: Then they couldn't be compatibly promoted. I probably don't want an operation on them.
BS: If you want to prevent that operation, you don't create the appropriate traits type. By default it doesn't work.
OL: Is "matrix_engine_multiply_promotion" the same as "matrix_multiplication_engine_promotion"?
BS: Yes; typo.
BS: Each of the four arithmetic operation does engine promotion. The top-level types take as template arguments the engine and an operator traits type. The latter comprises traits for different arithmetic operations to allow customization of each operator's algorithm; they aren't used directly by the class template.
DH: Is ",OT" missing from transpose_type?
BS: Yes.
MH: Couldn't they be different?
GL: They'd have to be different for row_vector/column_vector.
BS: We define hermitian_type to be transpose_type for real matrices.
DH: Shouldn't you transpose the complex case too?
BS: You have to copy it anyway.
DH: Then it's a view sometimes and a value sometimes.
BS: Jeff Hammond, I think, made the point that the hermitian transpose and regular transpose are both useful for complex matrices.
GL, AL: Indeed.
BL: We're going to finish presenting and wrap up soon to go to concurrent queues.
BS: The next layer of traits are arithmetic traits that perform the operations.
BL: Do we have to specialize these?
BS: If you want to customize.
BL: Eric and I were concerned earlier that having these traits in std invites conflicting customizations.

BS: There's a way coming to customize them.
BL: For all the traits so far?
BS: Yes.
EF: Are OP1 and OP2 in matrix_subtraction_traits the operands?
BS: Yes.
S: I'm a bit concerned about having a set of traits for each operator. Machine learning defines hundreds of operators to specialize for.
BS: The rest of the system to be presented might help.
BS: matrix_multiplication_traits handles, among other things, producing an unboxed scalar.
GL: I think you can do that with a single vector type.
BS: Yes, I can get the inner product that way. How do I represent the outer product?
GL: With a function.
AL: You'll have to do inner with a named function.
EF: Is void, as seleceted here, a generally valid engine type?
BS: There is no engine type in this case since we return a scalar. The caller knows not to use that typedef-name; it's there for documentation.
VR: I wonder if we are missing a lower-level abstraction than matrices. We have mdspan and containers; we want linear algebra. Do we just need a mathematical_array out of which to build vectors and matrices?
AL: Does it matter what's under the top interface layer?
VR: Having the mathematical objects in the names is a complication.
AL: I think mathematical_array should be a concept.
EF: Have we discussed existing practice like BLAS?
GD: There's boost::mpblas, but it hasn't been touched since 2009.
BS: Finally, we have an default_matrix_operator_traits container (in std or std::la) which is the default operator traits type for matrix.
BL: So the lower-level traits are not customization points.
BS: This is why we have operator traits promotion. If one set of operator traits evaluates eagerly and the other produces expression templates, the promotion lets the user say how to combine them.
EF: We should then avoid suggesting specializing the others.
BL: Just prefix them with "default_".
BL: We need to wrap up for concurrent queues.
BS: We have an example of how all the traits work together to pick a result type for a very heterogeneous multiplication.
EF: Is this "template<>" thing an example implicit specialization?
BS: Yes.
BS: You specialize std::is_arithmetic to allow a new element type.
DH: Can you specialize is_arithmetic?
BL: People do.
EF: Maybe you can't specialize it.
BS: You can go for is_element instead.
BS: There are examples for engine promotion traits interested only in limited types, and for explicit specializations to provide custom operator implementations for certain types.
EF: There aren't any user-defined types here, so you can't specialize it.
BS: But you can put it in a custom operator traits container.

(polls at 16h19)

We want 0-based indexing as opposed to 1-based indexing. (unanimous: 20)

EF: Boost has matrix_column and matrix_row.
AL: Those are actual pieces of matrices.

We like having separate row_vector and column_vector types in addition to matrix.
SF F N A SA (21 present)
3 0 5 4 4

BS: I like the idea of recognizing static row or column extents of 1.

We want explicitly named operations (e.g., dot and outer) in addition to operators.
SF F N A SA (21 present)
8 5 2 1 0

JB: I like the existing row_vector/column_vector approach.

BL: If we use extents, we get rid of fs_ and dyn_ types.
GD: mdspan is non-owning.
BL: But to be consistent, you don't want dynamic extents in a fs_ type.
MH: With all static extents, you wouldn't have an allocator.
BL: We're just asking the authors to explore it.
VR: Can you be fixed-size in one dimension and not in the other?
GD: We're proposing such an engine.
BS: You could make a fixed_vector-like engine.

Define engine/matrix classes in terms of mdspan + storage and mdspan concepts (e.g., extents), and expose an mdspan-esque interface. This implies that fs_ and dyn_ are combined into one template parametrized on extents (which are either static or dynamic).
SF F N A SA (22 present)
6 2 7 0 0

EF: We've been asking for not only implementation experience but usage experience. Ideally, not just tests but something on GitHub to let people try for themselves.
BL: We need prior art for gaming and graphics uses.
MH: I think it helps to review Eigen and uBLAS and explain differences.
EF: It might just be language versions.
CC: It's good to know why things are different.
(done at 16h33)

Minutes for 2019/02/22 Kona SG14 SG19 Review

Minutes by Mark Hoemmen

D1385R1: SG14/19

Bob presenting:

Bob: in terms of regular users, we're trying to support ease of use, meaning expressiveness, and reasonable performance out of the box
Bob: having overloaded operators that looks and feel like what they see in textbooks is a must-have for games programmers
Bob: others users will want the best possible performance and will want to tweak everything to get maximum out of the hardware
Bob: games programmers use lots of small matrices. They're the ideal users
Bob: HPC user matrices are very large
Vincent: a part of the HPC commnity has tens of billions of small matrices, e.g. for general relativity
Bob: we try to address these two "markets" based on our perception of their needs: an interface that's intuitive and usable, a set of reusable building blocks for the vocabulary types, as well as highly customizable facilities to allow optimization for specific problems on specific hardware
Bob: we want to minimize boilerplate code required for customization
Michael: have you thought of using concepts?
Bob: the point of the paper is to make sure the design makes sense from a high-level perspective. We're deferring constexpr and concepts, but it's on the radad
Mark: there are nice models for pre-concepts concepts in existing library
Michael: constexpr, concepts, parallelization and heterogeneous are things to consider
Bob: we try to define workable and correct definitions for the key ideas and operations expected from such a library
Bob: we also offer a list of terms from computer science (but in the C++ sense) to avoid ambiguous interpretation
Bob: en engine manages resources associated with a math object
Bob: among other things, we use the term vector in the mathematical sense
Mark: we hope to make vocabulary and concepts converge with what is being done with mdspan
Bob: our target is C++23, and we progress iteratively
Bob: we decided to delay tensor support to C++26, due to the scope of such an effort
Christian: I'd like to have a discussion on tensors later, as I'm not sure we should address this
Vincent: I have a presentation on this, later
Bob: GuyD and I are not sure we could complete such an undertaking for C++23
GuyS: for game developers, this library will be less useful without quaternions
GuyD: I agree, but we'll add it incrementally
Andrew: quaternions are otrhogonal to linear algebra

GuyD: I agree, and maths are underrepresented in the standard
GuyS: thank you, I was scared we were throwing them away
Bob: we see quaternions and octonions as complementary types
Michael: should they be developed in parallel on in sequence?
Bob: in parallel. I welcome a quaternion library; being rings, not fields, they will be a good test for the library
Graham: I hoped for sequential development, to reuse the rest of the library
GuyD: this would slow down our progress
Mark: when people develop tensor libraries, they reach for the BLAS
Bob: we allow for parametric memory source management
Bob: we also allow for parametric addressing model, using fancy pointers
Christian: do you have composable ownership semantics?
Bob: yes
(discussion ensues with respect to 1-based indexing vs 0-based indexing)
Bob: if only for performance reasons, let's stick to 0-based
Break starts at 9:48

Work resumes at 10:11

Bob: we can envision expressions with mixed element types
Bob explains how customization is expected to be done, e.g. to benefit from intrinsics or SIMD instructions
Bob presents considerations with respect to constexpr in the library
Bob suggests we skip section 6 first, seeing an example in section 7, then returning to section 6
Bob presents a design based on layered traits for elements, promotion, arithmetic, operators...
Christian: a related property of this is that for parallel computation with executors, we might hit a case where objects don't agree on execution. One can specialize operator traits to decide what to do
Bob illustrates the design using a multiplication of a fixed sized matrix of double with a dynamically resizable matrix of floats
Paul: this shows how we could special-case matrices of specific dimensions
Bob: yes, through partial specialization of names found in the std:: namespace
Mark: it's an example of customization for special use-cases
Paul: engine promotion is just a metafunction, so engine promotion would be a somewhat trivial task?
Bob explains the nuances between element_type and engine_type
Bob: there's a strawman implementation in the Github repo
Mark: since std::allocator returns raw pointers, there's no ownership attached. Matrices are owning
Bob: engine types are owning. This is intended to be similar in design to standard containers, which will be clearer when we go back and look at the implementation
Christian: we need things more like containers, which is probably the right thing for small matrices, but not for huge matrices where view semantics are better
Mark: I thing the customization points provided can deliver
Bob: engine types can be owning or non-owning

Patrice recommends Bob reads the matrix_multiplication_traits expanded example aloud at conferences, like Michael Caisse did with his Standard Slam in the past
Christian: I agree with the conceptual components proposed. I'm not sure that the general design with engines and partial specializations (instead of concept customization points) are the right approach. It's not a fundamental issue, and all components shown for customizability are the right ones, the thing that remains to be seen is if we're doing it the right way
Bob: the question is whether we cover the needs of 80% of our users out of the box
Paul: can you put in your own engine?
Bob: yes, and in your own namespace. There's an example used for debugging purposes in the document
Christian: I don't think the current engines will suffice for 80% of the users. I need a fixed sized engine with an allocator but which doesn't own its memory. There are many orthogonal dimensions
Bob: I will buy your argument, that's a very good idea
GuyD: we could add that with additional succession papers
Andreas: I agree that we should not standardize an engine that's not orthogonal enough. I think we should start with a small subset, assess what makes sense, and build from orthogonal parts
Mark: no similarly designed matrix library exists
GuyD: Matt Godbolt follows this project and will integrate it in Compiler Explorer. It will become simple to test new engine types and such, and look at the generated assembly
GuyS: I'm in the 80%. How many in this room are too? (about half of the room)
Patrice: cool, we just found co-authors to add stuff to the library!
Graham: should scalar types be first-class citizens?
Mark: they are
Graham: if I do scalar times matrix, can scalar compete? Are they equal citizens?
Bob: interesting question. I need to think about it
Paul: I think it can be implemented
Bob provides examples
GuyS: if I'm multiplying a matrix of T with a matrix of U, how does this get resolved? (Bob explains the traits manoeuver)
Mark: we could use defaults, but it's tricky if we target heterogeneous computing. Maybe no defaults is better
Bob: we have defaults in the paper, but I'm not sure they're the correct ones
Questions / answers:

Mark: to me, since you're using the same names and words than other libraries, this feels like standardizing existing practice
Bob: compiler and languages advances have really helped
Bob: we've had guidance from LEWGI which will help us progress
Bob: following feedback from Mark, Christian and Graham, we're revisiting some parts of the design. In the R2 version of the paper, we'll have a single vector type with an orientation based on context
Mark: inner product is more useful that outer product in terms of operations
Bob: this will reduce the number of traits and overloads, as well as the number of ADL attack vectors
Bob: the reference implementation on Github is for debugging purposes only

Michael: objection to encouraging further work? (none)
Christian: we should specify the scope of what we wanrt to see in C++23 in order to focus our efforts
Michael: we'll discuss in the next telecon on March 6
Applause.

Minutes for 2019/03/06 SG14 Conference Call

Minutes by Mark Hoemmen

Present:

- Bob Steagall (BS)
- Mark Hoemmen (MH)
- Jayesh Badwaik (JB)
- Cem Bassoy (CB)
- Klaus Iglberger (KI)
- Graham Lopez (GL)
- Andrew Lumsdaine (AL)
- Michael Wong (MW)
- Charles Bay (CB2)

MH: I met Jeff Hammond at SIAM CSE last week. He expressed the wish that a linear algebra proposal include or at least be open to tensor algebra. I suggested that he write a tensor proposal, since we didn't have any at the Kona meeting and he is an expert. He wishes he had time to write a tensor proposal.

CB: I have a couple suggestions for recent tensor papers to read.

1. "High-Performance tensor contraction without transposition," Davin Matthews. I think Jeff [Hammond] might have even worked with him. https://epubs.siam.org/doi/abs/10.1137/16M108968X

2. "Design of a High-Performance GEMM-like Tensor-Tensor Multiplication," Paul Springer: https://dl.acm.org/citation.cfm?id=3157733

BS: In my e-mail, I circulated a write-up by JB, called "Some Observations on Implementation of P1385." Shall we begin by discussing this paper?

JB: I think some of my points in the "some observations" paper were addressed in the minutes from the Kona meeting; let's talk about that first.

BS: I just got minutes from LEWG-I yesterday, so I haven't yet

incorporated their feedback into 1385. I will do so then submit a new version of P1385 for the post-Kona-meeting mailing. What we learned:

1. Indexing: Everyone strongly in favor of zero-based indexing.
2. Against separate row vector vs. column vector types; want one vector type, with context-dependent orientation.
3. Star would default to inner; explicit outer product.
4. Explore Engine types, to implement in terms of mdspan, to get reuse. Also need storage.
5. Explciitly like named functions, in addition to operators.

Andrew Lumsdaine, original author of the Matrix Template Library (MTL), presented both in LEWG-I and at the SG14/19 combined session. At end of day, after his MTL presentation, I debriefed with him. He thinks our approach is good.

Vincent Reverdy raised a point about the naming of Vector -- see review attached to meeting notice. "Hypermatrix" etc. He objects to the name "vector." I have a different objection to the name "vector" -- it would collide with std::vector -- unless we put it in a different namespace. A proposal for named constants would live in the std::math namespace, so perhaps we could use std::math for our vector type.

MH: I brought up this issue with Jeff Hammond last week. JH pointed out that there are three main communities of tensor users: general relativity, quantum chemistry, and machine learning. Vincent does general relativity. JH says that quantum chemistry and machine learning have needs that are most alike. I'm not saying we should marginalize Vincent's needs, but it could also be true that the largest community of users is less concerned about careful distinctions between hypermatrices, matrices, vectors, and tensors.

GL: Regardless of the representation share of the tensor community, it seems like, at very start, what we're doing here with the linear algebra proposal, something like tensors could be built on top of it. We don't need to design a tensor library at the same time. If we do a good job with these lower-level designs, tensors could reuse all of this. It's a tangential issue anyway.

JB: Linear algebra objects have very common names, like "vector." This suggests that there should be a separate namespace for linear algebra. I'm not sure how enthusiastic the Committee would be about that. Did you get feedback?

BS: Titus Winters (chairman of LEWG) will push back against having new

namespaces. He doesn't want an explosion of subnamespaces. But if you can make a convincing case, I don't think he would stand against it. For example, a convincing case was made for the chrono namespace and literals.

[AL & MW enter]

GL: If we need to push for a new namespace, we would have more weight pushing for "math" than for [something like] "linear_algebra."

BS: That was feedback from LEWG-I and the SG14/19 combined meeting. More feedback: Not reflected in poll but was in LEWG-I minutes: They want P1385 to have more discussion of prior art. SG20 feedback: They want a "Gor table" (see P1362R0, Section 4.4) -- showing feature levels and anticipated user sophistication for each. SG20 would like authors of new papers to include tables like this. We generally got favorable feedback from both rooms. Working in strawman implementation repository. If anyone wants to try implementing arithmetic traits, even in naive way that returns temporaries, please feel welcome. Some people -- JB and KI already -- have already tried implementing P1385.

MH: We plan to write a paper on a low-level interface, and want that to be complementary to P1385. Christian Trott (CT) and I discussed this with you at Kona.

BS: Great idea, would like to help, map primitive ops to corresponding arithmetic traits, see how they fit.

MH: We don't have a proposal yet, and are not sure if we will make the post-Kona-meeting mailing deadline, but do plan to submit at least for the pre-Cologne-meeting deadline. We will keep in touch with you about it.

BS: Let's talk about JB's write-up "some observations."

JB: I'll share my own screen. I started implementing this before the Kona meeting, so this is based on P1385P0. First, in the Standard Template Library, we have sort algorithm, can provide it with any kind of container which has begin and end. Do we want similar thing with the linear algebra library, so that users can give it different linear algebra data structures?

AL: Yes, I think we need to. What made the STL successful was its

openness. Need not be based on iterators, could be generic indexing syntax.

MH: First, we have the experience that users treat matrix data structures as data containers to share between software modules. For example, users use Trilinos' matrix data structures well beyond the original intended use case. Thus, it would be good for a linear algebra library to be able to work with users' data structures, rather than mandating a particular data structure. Second, mdspan offers a generic indexing syntax.

AL: There's a difference between users' use of a matrix, and implementing the functions over it. mdspan could be a matrix at the implementation level.

MH: I agree. Also, mdspan is not necessarily a matrix at the user level, for reasons discussed in my Kona talk.

JB: Regarding engine promotion traits for arithmetic operations: if Engine1 and Engine2 are standard types, then we can decide what to do. If one is a user's type, the user can decide. What if both are [different] user types? Then best is that no one would specify what would happen in that case.

BS: Two situations there. JB's example, two diff engine types, one from library1 and one from library2. Implementers of the two libraries don't know anything about each other. User wants to use the lbiraries together in a mixed expression. No predefined way to determined how to promote.

JB: Yes, and best practice would be not to define it.

BS: Yes. In the strawman implementation, and how Guy and I envisioned it, that would be a case where the library provides no guidance. No default promotion. User would have to create their own traits and decide how to promote. It might be a third engine type altogether.

JB: Even then, if library1 and library2 do different things, this might introduce bugs in the code.

CB: I agree with Bob: it's the responsibility of the library user; we can't ensure, except with concepts, that there is a certain type given out.

JB: I agree with that. But even the user cannot do it very well -- they would have to redefine all the machinery on their own.

BS: Definitely that's true. However, today, if you had two linear algebra libraries and you wanted to mix their types, could you even do it?

JB: No, you couldn't.

BS: So, our hope is that at least by providing this traits mechanism, permitting user specialization, it would help a sufficiently motivated user solve this problem.

JB: I think I have a solution to this issue, which I'll present. Keep that in mind. See Point (3) of my paper.

BS: About your Point (4): It's not clear that we want to require that elements are fields. Some people wanted quaternions, which do not obey all the field axioms.

MH: Also, some people want to express graph algorithms using linear algebra operations [see e.g., the GraphBLAS effort]. In that case, the ring axioms or even less suffice for matrix entries.

JB: That takes care of Point (5). Point (6) handled in meeting minutes. Point (7): minutes said that Kronecker vs. outer product are different. I'm still not decided on the position.

BS: Vector / matrix of dimension 1 / 1x1, mathematically, is synonymous with an instance of the element type. Problem is getting C++ type system to conform.

[Charles Bay (CB2) joins]

BS: If fixed size engine, size 1, then can have conversion operator, no performance cost.

MH: That may be true mathematically, but may not be true in the computer science sense. A scalar value is immediately available. Accessing an entry in a matrix may require dereferencing a pointer, or something even more complicated [when using mdspan with a custom access policy]. The referenced datum may be expensive to access -- for example, it may live on a GPU.

BS: For static matrix storage, should be minimum run-time cost. For dynamic storage, would require run-time check. This is one reason why we pushed for row vs. column vectors to have different types -- so that the multiply operation could return a scalar in that case,

instead of a 1x1 matrix.

MH: GPU example, may not want 1x1 to behave like scalar value always.

JB: I'll show how I dealt with the problem of different traits / types. addition_trait, subtraction_trait, negation_trait. On right-hand side, I have a custom class (jvector), in my own namespace (testlac). I gave jvector an execution policy (EP) template parameter. All types that want to interact with the standard library should have this parameter. Then we can specialize the traits; they only define result type, not how to compute. Then I defined separate traits classes, called "engines," templated on EP, that define how to compute e.g., negation, addition, and subtraction. I specialize the engines for my own execution policy. I show different execution policies; some can be provided by default. Now, since execution policy is just a tag, we can demand a templated move constructor, that moves a jvector with one policy into a jvector with another policy. That's a cheap conversion from one to another. Now we can demand that for any operations, the two execution policies must be the same. Now all the arithmetic operations can work. Addition engine can just call already implemented addition engines, if they want to reuse something that's already been done. I've been able to have, for example, a "verbose" policy [representing a custom user policy] and a system-provided policy, with a static vector templated on the execution policy and other stuff, and a different vector class, and I can combine them in arithmetic operations. Another advantage: suppose you have a tridiagonal matrix, but no arithmetic -- then you can define your own execution policy, with existing containers, and you can specialize that matrix multiplication operation and use it in your code. Main idea is that we should require that every type which wants to deal with our linear algebra library, should have this template parameter. Need more experiments to see whether this works. See code in my GitHub repository.

MH: JB, you require that the matrix or vector type take this "execution policy" template parameter. What if a given execution policy is not compatible with my matrix type? For example, I may have a matrix that stores data on the GPU, so that it's not even correct to execute on the CPU.

JB: First, could use static_assert to restrict type, but then your class would need to know what to exclude.

CB: Eigen's tensor library has a solution to this. It uses expression templates for that. You don't clip [attach] the execution policy to the matrix or vector -- instead, you say to the expression that you

want to have it executed on the GPU, for instance. C.on(gpu) = A+B. When operator= evaluates the expression, the ".on(gpu)" tells it to run on the GPU.

BS: MH and Christian Trott in Kona suggested a similar syntax for this. I agree that it would make sense for our proposal, and it seems easy to do. JB, I like exec policy, ties into that conversation, but I haven't had time to study your code since the latest developments.

MH: It sounds like the "C.on(gpu)" syntax, and JB's move constructor (that is meant to allow changing the execution policy cheaply) are trying to address the same issue.

CB (?): The problem is that there are two competing execution policies with the Engine. I don't know how to solve; gets complicated... I would rather like to have an expression to be executed on the GPU.

JB: I thought about tthis. Issue is that the types of the matrix or vector can also affect how you want to execute -- e.g., sparse matrix needs different iteration. Policy is based on type as well, not just expression.

MH: JB, your "execution policy" conflates _where_ to run, vs. how to iterate or access the matrix.

JB: They are conflated. [Sparse matrix example.]

MH: I could see that, e.g., for hardware that has a special execution path for sparse matrix-vector multiply.

JB: This is my solution for the traits not matching.

BS: Any more questions or comments?

MH: We just need to look a bit more.

BS (? or JB): Should I continue?

MH: Yes, it's good for more of us to try implementing this, so we can discover any issues.

BS: I'd like some time to study what you [JB] have, and to combine that with what I've learend from Mark and Christian and Graham, and see how that impacts our paper. JB, if you have suggestions for execution policies in our paper, we'd love to hear them. Keep exploring.

BS: Next steps for P1385: Incorporate Kona feedback into post-meeting paper, now extra avenue of research raised by JB to think about before Cologne.

MH: [discusses plan for low-level linear algebra interface paper, to be complementary to P1385]

BS: Succession papers? MH mentioned one, JB may decide to write one too, up to you. Guy and I would like to encourage people to write papers. We want to build linear algebra momentum. For example, a succession paper could propose an interface for doing matrix decompositions [like LU or QR]; here's what the input and output types would look like, based on 1385. Comments or discussion?

CB: Have you thought about views? Sections or ranges of matrices and vectors?

BS: Yes, at bottom of D1385R1, Section 9, third bullet, would address that. Could have views of a vector, matrix, etc. Does that answer your question?

CB: Yes. I think we need that.

AL: This is where mdspan becomes very powerful. You can define a separately indexable thing over the same data. LAPACK and standard textbook linear algebra algorithms require / are written in terms of subviews. Also needed for blocking and parallelization.

BS: I agree with AL. This would be a perfect way to test mdspan in terms of two nonowning view types -- one const, one mutable -- need the latter for decompositions. Definitnely on TODO list.

CB: I could help with that; I've implemented such things.

BS: OK, connect with me offline.

MW: You technically don't have to go through LEWG-I [for review]. I spoke with (? ...) about this, since there has been some confusion. It's normal for papers that pass SG review to go straight to LEWG or EWG. Thus, it's up to you, Bob, to decide whether you get periodic [LEWG-I] reviews. Normally, a paper that comes out of SG14 or SG19 would go directly to LEWG or EWG. But thank you, I'm really glad of the implementation that is coming out of JB's work.

AL: Course I've been teaching at UW, simple linear algebra library, compatible with design being proposed here.

MW: Good validation.

BS: Open floor to misc topics.

MH: Bob, you mentioned getting feedback on P1385 asking for prior art. I do plan on expanding the history paper (P1417) to include new material, based on both what I've found and what AL sent me. However, I imagine that they would want P1385 to do more than just review existing work -- they would want to see how that work connects with your design. This would mean that an expansion of P1417 wouldn't suffice; you would also need to add material to P1385 relating to your design.

[Bob acknowledges this.]

Minutes for 2019/04/03 SG14 Conference Call

Minutes by Mark Hoemmen

Present:

- Bob Steagall
- Guy Davidson
- Michael Wong
- Steven Varga (Toronto)
- Mark Hoemmen
- Cem Bassoy
- Graham Lopez (ORNL)
- Christian Trott
- Jayesh Badwaik

MW: SG14 will not meet next Wednesday, due to the ACCU meeting in Bristol. Will move SG14 meeting to the followingx week.

BS: Progress report on P1385.

In our public GitHub repo, we have a new branch "la_r2" containing work in progress.

I've been implementing KONA 2019 suggestions, documented at end of P1385R1 (at wg21.link). I've removed column/row_vector and gone to a single vector type.

I took a detour and started working on something called "number traits" -- an initial attempt to differentiate between semirings, rings, and fields. Matrix elements would need to be fields. Not sure if that's the right thing yet, but I wanted to put the machinery in place.

I've also taken dynamically resizable etc. engine classes, and split them into separate classes based on whether they support matrices or vectors. Not sure if that's the right design, but I did it as an experiment. I may instead partially specialize on the number of dimensions.

BS: After this, I will look at integrating mdspan. Finally, I start thinking about executors and execution contexts. It will likely be several weeks before I get to these two tasks.

MH: Regarding semirings, rings, and fields: Different algorithms have different requirements on the matrix element type. For example, algorithms that look like tensor products (e.g., matrix-matrix multiply, dot products) only need plus and times. Factorizations may need division, but in some cases could be reformulated to work without it -- e.g., for matrices whose elements are polynomials. Would it make sense to let each algorithm constrain the matrix element type, or do you want to constrain the matrix element type for all algorithms?

BS: I was thinking about that with the traits refactor. Naively, it suffices for the matrix element type to be a ring. But what about the determinant? That needs division. For now, the test is that the matrix element type must be a field.

MH: Would you prefer to express this requirement as a Mandates (static_assert) or as a Constrains (SFINAE, that is, does not participate in overload resolution if requirement not met)?

BS: I prefer mandates (static_assert).

MH: I would have a preference for a per-algorithm requirement, and to express the requirement as Constrains (SFINAE). However, our use cases for ring or semiring matrix element types are mainly for graph algorithms, and would thus use sparse matrices.

MW: Is this work on semirings, rings, and fields for the first layer?

BS: If by "first layer" you mean the foundation layer, then yes. It's something new I thought of a few days ago, and added just to start the conversation. Not sure if these traits should be part of linear algebra, but if useful there and applicable elsewhere, then it may make sense to pull them out into their own paper, for Numerics.

MW: OK. We should check thoroughly how much we need that for first layer.

MH: ... field theory ... [I think I said something to the effect of "this is interesting and potentially useful, but we need use cases for more exotic matrix element types, and we also risk accusations of overcomplicating the design to support them." In addition, interfaces of libraries like the BLAS and LAPACK were not designed for this much generality, so we may need to look elsewhere to make sure that the interface itself does not overconstrain us -- especially as we start to include matrix factorizations.]

MW: I see what you mean.

GD: When I started working on this, I wondered how far to draw the line [with respect to how general the matrix element type should be]. I do use matrices of polynomials.

BS: At least from an interface and type design perspective, having a set of elements that's well behaved and fulfills a fixed set of requirements ... [is a good idea]. We could have an element type that represents a polynomial, that has well-formed operations ...

MH: This issue of the matrix element type matters more for matrix factorizations, than it does for algorithms that look like tensor products [that only use plus and times].

BS: Any more comments on P1385?

MW: Will you submit a revision for the pre-Cologne mailing?

BS: Yes, we would submit an R2 paper for the pre-Cologne mailing.

MW: Someone from PNNL, working w/ Andrew Lumsdaine, wants to give a talk at the SG14 linear algebra meeting in Cologne. I would book a face-to-face meeting at Cologne.

GD: I will be there.

BS: I will try to be there.

===

BS: Are people planning succession papers [to P1385]?

GD: I've started trying to work out what a geometry succession paper would look like. I started with Boost Geometry and with our games, to see what minimum set would be, so that others could write further succession papers.

MH: Geometry is numerically subtle.

GD: It is.

MH: For example, correct evaluation of Boolean predicates (like "am I inside or outside of this circle?" or "on which side of a line am I?") may call for extended-precision arithmetic.

GD: Games usually express this with an "epsilon" tolerance, that lets them trade off between speed and accuracy.

MH: Finite-element or other discretizations of partial differential equations in space favor accuracy, in order to avoid situations like inverted elements that may break the discretization.

BS: This reminds me that I'm currently using a geometry library for earth surface stuff, open source, originally Java, then ported to C++, not well. But it's mandated that we must use it.

MH: Regarding succession papers: We're working on identifying "primitives" for a linear algebra library -- low-level things in support of an interface like P1385, that vendors would most want to implement for good performance and/or accuracy. I am working on a paper that shows how a C++ linear algebra interface might develop organically, starting with a BLAS interface and developing to higher levels of abstraction. The paper is not ready yet, but I could summarize it now, if you like.

BS: Please discuss it at the next call, once it is complete.

MH: Regarding the set of supported matrix element types: I'm worried that we're taking as models interfaces that only support floating-point types. Would we need different interfaces to support fixed-point types? [For example, would they need to pass along scaling factors?] We should consult experts on fixed-point arithmetic.

GD: I'll try to get John Mcfarlane on the next call.

GL: Davis Herring at LANL has interest in fixed point and also in linear algebra.

BS: Davis was in the LEWGI review of P1385 at Kona.

GD: He took excellent minutes.

GL: He was representing one of the fixed-point papers as well.

===

BS: Misc. business -- other topics

CB: Google Summer of Code started. I am contributor to muBLAS and added tensor library to that. Next GSC topic: I want an interface to

that, similar to your paper. I asked Jayesh if he wanted to participate and he said yes. I would like to extend this ... to tensors.

BS: I'm interested in that ... based on the work we do here, some subset of us would like to put together a tensor paper proposal, once we've learned lessons from linear algebra.

CB: We're trying to use Boost expression template evaluator -- can do Eigen Tensor - like things -- specify at end of expression whether you want to evaluate in parallel, on device, etc. Trying to do that in this Google Summer of Code project. We have several good students.

MW: Would the tensor paper you're suggesting also be of interest to SG19 (machine learning)?

CB: Yes. In Boost, we are heading twoards tensor decomposition algorithms, like higher-order SVD. In machine learning, you may have other operations but similar. That would be a good thing to do.

MW: Christian Trott is here with me; add him to roll call.

SV: As I mentioned [at the beginning of this call, I work on a ... tensor solution for the HDF group, for linear algebra library, ... idea is general framework for C++ ... current status is full linear algebra support ... structured ... compiler-assisted reflection ... how does this fit into the material that I see on my screen? and how related to machine learning working groups? ...

BS: I don't know yet. I'm looking at your link.

SV: Is it OK for me to attend these meetings to get a better understanding of the trajectory towards which you are working?

GD: That would be great. These are public meetings. You don't need permission to join us.

BS: You're welcome any time.

MW: SV, since you live in Toronto, you can apply to be a Canadian expert -- just send me an e-mail with your resume. With that, you would have the power to vote officially. Every country has its own delegation -- SCC -- I am the chair of all the programming languages.

SV: One more question: I just received this material from you, BS, are these functions functional? If they are working, then I would

incorporate them.

BS: What we're doing with P1385 is to specify an interface that would become part of the Standard. Library implementers would provide their own implementations that express the interface that we propose.

[Jayesh Badwaik joins]

BS: [continues] Our work so far, up to this point and probably for the next few weeks yet, is on the interface, and perhaps a reference implementation that vendors could use or modify. Most of our efforts now are on designing the interface. Branch la_r2 that I sent you -- la_r1 branch actually compiles and executes, but with stub execution. Once I get r2 branch working, the same thing [stubs]. Beyond that, we would actually put real code that does arithmetic there, into the arithmetic traits where the work would actually be done. Also, this interface is deliberately limited to what you're calling rank 1 and rank 2 objects -- matrices and vectors.

BS: While working on traits for semiring, ring, and field -- several people asked me the same Q about element promotion traits -- they asked why we couldn't just use common_type. At the time, I didn't have a good answer. common_type is one of those things in namespace std you're allowed to specialize. Half-size floats: It might be that you want to write traits for operations, where both operands have type half-size float, but you want result type to be full-size float. It could be that common_type of half-size and half-size is half size.

MH: common_type of complex<float> and double is complex<float>; surprise!

GD: Wow.

BS: I'm writing a paper to fix that aspect of common_type; Walter Brown discouraged me from doing so, though. [The implication to the scribe was that WB thought it would be impossible to get this change through, not necessarily that WB thought it was a bad idea. I'm not sure exactly what BS meant, though.]

GD: WB also discouraged me.

MH: Would the expression C = A + B for half-precision A and B and full-precision C trigger half,half,full promotion, with correct arithmetic that accurately computes a full-precision result? or would it just truncate half+half to half precision and cast to full?

BS: The intent is that you could support the former, yes.

MW: You have two meetings before the mailing deadline -- May and June.

CB: Would you like to discuss about the view stuff?

BS: Yes.

CB: [shows an example of views] In a Householder QR decomposition, in Matlab you would use a colon operator. Here, I show "span(j,m)". The idea is to create a matrix view. No copying, it would just be a view. Do we want to support something like this in the matrix library?

BS: It's not part of the interface now, but in KONA feedback, there's a desire to have view types of columns, rows, and submatrices, that are part of the library. We haven't defined them yet, or decided whether they would use mdspan or something else.

Christian Trott: Another argument in discussions with vendors why we should use something like mdspan -- I talked with vendors about accessors that express things like streaming stores and loads -- using special capabilities in the memory subsystem -- mdspan has all the extension points needed in order to express all these things. Vendors are very interested in that. On newer systems, memory system hardware is more complex and can express more than just plain load and store.

CB: Of course mdspan supports a lot of things, but if you want to keep it a bit simpler ... [shows screen] -- operator() that takes "spans."

MH: We have a function called "subspan" that does this. Can be hard to have overloaded operator... would be good to use mdspan here since it has gone through more design review and more C++ subcommittees, but we don't mean to stomp on people.

GD: No problem from us -- we view mdspan as a crucial implementation detail, potentially as a view engine, so please don't worry.

JB: Sorry I haven't had so much time to work on this; I'm working on my thesis.

[group]: Don't worry!

===

BS: Next linear algebra call: Wed 01 May.

MW: Next SG14 call: 17 Apr, 2-4 PM Eastern.

Minutes for 2019/05/01 SG14 Conference Call

Minutes by Mark Hoemmen

Present:

Cem Bassoy (CEB)
Charles Bay (CB)
mbutler
Guy Davidson (GD)
Matt Harrington (MH) (4154764628)
Mark Hoemmen (MFH)
Jens Maurer (JM)
Bob Steagall (BS)
Michael Wong (MW joining from Singapore)

BS: Latest release of paper:

https://wg21.link/p1385

Latest version of the code:

https://github.com/BobSteagall/wg21/tree/la_r2

BS: Giving a linear algebra talk at C++Now next week. Update on progress on P1385 since our last call in April. Implemented most if not all points in implementation guidance at end of R1 version of P1385. Added several things over the last month:

- const and mutable iterator types for vectors
- const and mutable begin for vector
- row and column view types, for rows resp. columns of a matrix: read-only view type
- work in strawman implementation: filled in incomplete member functions just to make everything work; Guy working on impl for arithmetic traits
- applied constexpr to almost everything, except for dynamically resizable engines and arith traits (not gotten to it yet)
- noexcept'd most of the things
- added more debugging helpers and stubs

mdspan, executors: still on the to-do list. We wanted basic implementation done first. One major development in how traits work:

if interest, I can pop up a screen and show how it works. We've modified operation traits in such a way that it's possible to create new operation traits type, and can override traits that are part of default operation traits. Example in code repo: Overwriting the element promotion traits so that float + float gives a double. Also examples of overriding other things, like arithmetic operator. New traits system makes it easier for user to override just what they want. It doesn't change interface much. Implementation more complex, but this is hidden from user.

GD: I'm nearly done with the arithmetic traits; just need to do the complicated ones. Started looking at specializations for geometry, but that's beyond scope of this meeting.

BS: Kona feedback: Merge column and row vector types. This is all in std::math namespace. Matrix * vector means matrix * (column vector); vector * matrix means (row vector) * matrix. vector * vector is inner product. Outer product will be named function.

JM: Are you just working on implementation, or any progress on updating the paper? The paper does not mention std::math, for example.

BS: R2 version of paper will propose std::math. Not part of R1.

JM: I would encourage that the R2 update be more verbose in the changelog than what I'm currently seeing in the revision history of the paper. This is where std::math, made compatible with mdspan, etc. changes would go.

BS: OK.

JM: Changelog would help when rereading paper. Also mention things like zero-based indexing in its part of the paper, not in a remote place.

BS: OK, we'll include that in the main text for the R2 paper.

BS: You mentioned earlier, JM, that you had some questions.

JM: I started to read through the paper for this call, and made a few notes. I would be happy to talk about them with you.

BS: Sure.

JM: Concern with entire way of approaching this:

Non-concept-constrained template parameters. For example: fs_matrix_engine just takes a template class T. We're in a world now where we want to have concepts. Presumably T is an element type that must satisfy the element type constraints.

BS: Yes, adding concepts is on the to-do list. We haven't attempted to do it yet.

JM: It will change some of the presentation in here. Would be good to have an open issues list in the paper to see that it's on the to-do list, so people won't repeat tha5t concern.

GD: You're quite right, JM. In an ideal world, we would have started on this sooner, but concepts are fresh.

JM: Fine, you have a lot of concepts in the text of the paper, just not in the C++.

BS: A lot of this is just my unfamiliarity with concepts.

GD: Me too.

BS: For example: matrix element. Could even be noncommutative ring for some matrix arithmetic, but determinant requires more refinement. How do we write C++ concepts that embody these mathematical concepts? I don't know how to do that yet.

JM: My view: Ignore the question. We have historically punted on defining what we mean by "usual floating-point type." For example, with std::complex, we only allow float, double, long double. Ditto for random number generation. I understand, you don't want to constrain your library similarly. People have continually failed to describe "a type that acts like double" accurately. For linear algebra, maybe enough that it's a ring. Limited operations. Don't need sine, cosine, log. Maybe possible to describe set of required operations. Again, std::complex, people have tried and failed to make that reasonable.

MFH: ... it's a function of the operations you want to do, not of the matrix data structure, so constraining "matrix element type" is not appropriate.

JM: Right...

BS: ... it will take us a while.

JM: The C++ concepts we have mostly embody syntax, not semantics. I need type with the following operators, etc. You're not saying "this is a ring" or "this is a field."

BS: Right. A potential fly in the ointment is that of computing the result type. For example, John Mcfarlane has his elastic integer proposal, where the class is parameterized by the number of bits. The result type of addition is determined at compile time by the interaction of the template parameters of the inputs. One thing I don't understand very well is how one could cover that in a C++ concept.

JM: You don't necessarily want to ...

[Cem Bassoy enters]

BS,JM: ...

BS: Add two vectors of float16, get vector of float32.

JM: Two concept things, must distinguish them. First, two element types, may differ, the two argument types you have, they must have syntactically valid operator+. Second: btw, the result of this operation needs to be this type or convertible to this type. Maybe you should punt on the latter part. The concept requirement for you would be that you can put operator+ between those two types. For the result, you can only hope that the result type makes sense. "Matrix element" in 6.2 -- not sure whether ... essentially an explicit opt-in. If I write a new type, I must explicitly opt in to make it a matrix element type. That's not usually something we do. We don't usually have a special switch to enable this. Any technical reason why you need this?

BS: R2 version has changed this. Instead of explicitly specializing a traits class, there's a traits like allocator_traits or pointer_types, in which one describes the capabilities of the numeric type -- it's called number_traits. You specialize it for your new numeric type.

MFH: explains history

JM: Gneeral nuemric props class, tries ot embody math properties -- I am hesitant to find that a good idea. If you go there, SG6 has an opinion. Second, I undersatnd why you need for adnvanced operations a division, for others you don't. You can check with constexpr... Currently, I am amiss which other props of T which are not checkable by syntactical checks in a concepts... would be needed

to help you to implement a matrix library. If there are such properties, I owuld like to have them listed in the paper.

BS: OK. That's a good question. I admit that one motivation for number_traits is to constrain set of types of which matrices can be instantiation.

JM: Why the additional burden on the user?

CEB: Because you don't want to allow e.g., bitwise operations -- you might like to exclude some types, to help back-end developers.

JM: The T we're talking about here, element type, is supposed to be airhmetic in shape and form, not std::pair. It's totally OK if I have a back-end developer who decides that his engine is unsuitable for Ts that are not trivially copyable or larger than 8 bytes or not double. That's stuff you can formulate with concepts.

[Michael Wong]

BS: Without constraint, one could add two matrices of std::string.

JM: What's wrong with that? You can add two strings and it does something.

CEB: IF you have a matrix, you would expect as a mathematician that you would have a nuemrical type.

GD: Not necessarily. You can have matrices of functions.

BS: Philosophical point. We want linear algebra to represent math.

MFH: What about matrices of polynomials?

BS: That's math; strings are not.

JM: You could further constrain by requiring plus, minus, and multiply -- that would exclude string -- if you're worried about it. Anyone who defines these operations on a type without actually implementing a ring, they may do so.

GD: Yes, they have dug their own grave.

BS: OK. Your points on concepts are well taken; I am writing them down.

JM: Next: 6.3: talks about element promotion traits. Talks about a config mechanism to find out what the compound element type would be,

for input element types. It talks about partial specializations. I would drop partial specialziations -- all impl -- and show what kind of type unification you want here. For plain types T1 and T2, you pick maybe the larger one, and for std::complex<T1> and complex<T2> you pick complex<unification of T1 and T2>. That would be actual rules, not partial specs without rules that don't give me clue what's going on .

BS. OK. Diff in R2. This was a hallway point I got in Kona. Element promotion begins with decltype of addition operator -- let the types say it --

JM: If that actually gets rid of element propmotion thing ... fine with me.

BS: We can't actulaly get rid of it, since it defeats flexibility of the system -- ability to override the dfeault behavior -- e.g., element prom traits were float + flaot -> double.

JM: I hope, as written here, it seems the matrix elemeent promotion is a global struct.

BS: In R1, it is a global struct. R2 removes that and lets one create partrial specs in your own namespace and insert them into the operand types so they overrid3e the default traints in namespace std.

JM: OK, good. My concern here is if we have global setting, two diff parts of program get different outcome when I add two half-floats, then taost.

BS: ... [R2 fixes that] ...

JM: Yes, that's nice. I believe the only sensible behavior for namespace std is decltype(a+b).

BS: Right, that's the default.

JM: That is the only sensible thing we would have in namespace std, so we don't have to call it out. ... fine. Not user customizable. Good.

JM: Fixed-size engine in 6.4.1. First, there is no showing on the paper of the requirements of a basic engine type. It "provides a basic indexing ..." and then you go on to fixed-size engine.... which part of class declaration is part of basic engine requirements, and which are part of fixed-size engine. For purpose of exposition: would help to have class italics basic_engine italics, that show the required names.

JM: 6.4.1: First concern: Two template parameters rows and cols do not seem to be reflected as public static const data members. Template should expose them as public things.

BS: In the R2 version, the cols and rows -- all the member functions of fsengine are constexpr -- rows and cols member functions return the template parameter values.

JM: that doesn't address the problem -- if you have a run-time fsmatrixengine value, having a constexpr member function just means you can call that function in a constexpr context, but I don't think it will work ... if it's a static member function, that will work, but those functions are nonstatic here, for no good reason actually. That means I need a this pointer, and this pointer is nonconstexpr in general. That means I can't get cols and rows as compile-time constants.

BS: OK. The problem goes away if the functions are static?

JM: Yes. Also goes away for a different reason if you use mdspan extent.

JM: is_dense, is_rectangular as true_type / false_type -- instead of static constexpr bool. What's the benefit of the type as opposed to a simple compile-time boolean? I understand why traits -- must derive from true_type etc. -- but not really necessary here.

BS: Reason is precedence set by things in containers ...

JM: I'm curious to see precedence having true_type and false_type as member typedef in some larger structure.

BS: in allocator_traits: propagate on container copy assignment ... is always equal ... all four of those nested type aliases are assigned to either true_type or false_type. That precdence is what I was attempting to follow, though you're right, having static constexpr bool as variable would be cleaner and easier.

GD: [inaudible]

JM: I would expressly ask LEWG and LWG whether that is their design guidance in this area, instead of deducing anything from allocator_traits.

BS: I was [just] trying to follow an existing model.

JM: numeric_limits is closer to your domain and follows a different

model. Again, I am not a library expert in this area, just that this strikes me as odd, and we should ask for library guidance.

GD: Absolutely; your input is welcome in this regard.

JM: when you have single-argument constructors, e.g., drmatrixengine, that takes size utpule ...

BS: That should be epxlicit; that's an oversight.

JM: Yes.

JM: There is a curious amount ... you have this transpose engine here, which claims to be just a view on something else, where operator() exchanges i and j.

BS: right.

JM: Does that cuase ownership concerns? No, because you don't own the other engine type, you have a pointer to it, not a by-value copy.

BS: Correct.

JM: Do you get mmeory management concerns here? Can your original matrix go out of scope while you hold the transpose matrix?

BS: Yes, it's possible for that to occur, just like a string to which a stirntg_view refers can go out of scope.

JM: It feels odd to call this "engine," because I understood engine to own the memory of a matrix.

BS: We distinguish between owning and nonowning engines -- transpose engine is nonowning.

JM: I get a bit of stomachache ... until you introduced the transpose engine, everything was value based, e.g., return ... by value. Having a view has all the scope and dangling pointers trouble. HOwever, I'm being told that maybe this will change in R2 so I won't press the point.

BS: This design is meant to let one create expression template enginees. The enginees would be the expression templates themselves, all of which themslevs would be nonowning view types. Possible to contrive a situaiton where one instantiates a matrix whose engine is an epxxresion, and the matrices to which those expressions refer all go out of scope. The problem as I see it is no worse than string_view and string. The ability

to have expression templates and create your own ... is a critical part of the flexigblity we're trying to build in. Expression templates would improve performance ...

JM: You eventually need values, but I understand that there are efficiency gains, e.g., if multiplication by a constant is folded into matrix-vector multiplication. Maybe my main thing here is that the 6.4 intro to engine types does not even mention that there are owning and nonowning [engines]. it says, the underarching purpose is to perform "resource management" -- that suggests storage management. Maybe it should be an "engine view" to highlight that it's not a storage management ...

BS: OK. We allude to that in Definition 4, of what an engine is. We haven't clearly spelled out that storage management could be owning or nonowning.

JM: Frankly, with this wording, I'm not getting that this is where expression templates fit in. If engines are, then this needs to be much more explicit. Expression templates are not just about conveying access to the math elements. The expr templates , in their core meaning, just say, you're performing all computations at the operator=. There's no direct element access at that point, it's just a bulk operation that you do at one point, rather than several operations with temporaries in between. I don't read this in section ...

BS: OK, that's fine. This is good feedback.

JM: One thing I have not understood in here, is in 6.7.1, we have e.g., negation traits. When I call unary operator- on a matrix, this tells me how to actually compute that, and what the result is, ..., right?

BS: Yes.

JM: I understand that there's a class operand1, which is the matrix or mathematical vector type. Where does optraits R come from?

BS: A better way to look at the problem: Skip to Section 7.1. This explains how the various traits work together. It makes more sense from the top down.

JM: You're computing new operation traits from the two arguments ... and those traits are the ones you pass into ...

BS: Exactly [explains relation to element type promotion].

JM: OK, the thing I was missing is that the op traits were doing these promotions -- unary negation, it's a noop.

BS: It's almost trivial, but there just in case.

JM: Fine.

JM: What I was wondering is, you don't necessarily have to show this in the paper as code, but I undersatnd there are substantial algorithm gains if you special-case diagonal and band matrices?

BS: Yes. There are special cases for upper and lower triangular and also for banded matrices.

JM: Those are not yet shown here, in terms of code.

BS: that's a question of scope for P1385, whether it should include additional engines.

JM: I think P1385 will already be large enough once you're done with all the details. I'm fine with leaving those out. What I'm concerned about, is whether the infrastructure works out when those get added later.

BS: We haven't implemented those yet. That's a question of, for a given engine type that represents one of those specialized layouts, that its corresponding arithmetic operation is written correctly. There's no conceptual difference between that and say, the transpose engine. [explains] That transpose engine works with our test code, for example.

MFH: Historically, this was possible, and done in different ways.

JM: Yes, I'm not worried that it can be done. I just want to know whether the infrastructure works with it. I have the same concern... what's conspicuously absenst here is std::valarray. it was targeted to solve some of these problems, but was a dead end. I don't want a dead end like that, that does not do what it promised to do. Same for the expression templates. Do you have experimental code that does expression templates and shows that the engine mechanics work?

BS: I don't have that yet. ... binary addition. If I call m.t and return a view of the transpose, that is, in effect, an expression.

JM: That's fine. When do you trigger materializing that expression when you see the operator=? ONe of the unsolved problems with expression templates, is "auto x =" is not a value type, it's just a view. However, "specific_matrix_type x = " would hopefully make a real copy of the matrix. What's the mechanic to make that happen, and is that another operation... that needs to be in the infrastructure -- the materialization or copy operation.

BS: That's a good question: I don't have an answer yet. We will think about that. I don't want to speak for GD.

GD: We'll just have to keep writing experimental code.

CEB: In Boost, we have a rudimentary expression template system. I think we have done it with CRTP. You have a base expression structure where you derive all your types -- matrix and vector become expressions -- as JM said, you need an operator= that evaluates that expression. This is how we did it in Boost uBLAS. I have never seen that like what you have done here, with an engine ... this could be reevaluated ...

BS: A long time ago, when I was working in medical imaging, I wrote a medical imaging library for my company. some of my experiences went into this. I was looking at whether to derive matrices and vectors from base expression type or use engines. I tried the engine approach and it seems to work well. Engines avoid conceptual baggage of inheritance.

JM: I have no doubt that not using inheritance in this area is beneficial. All I'm saying is that until I've seen it [expression templates etc.] or have credible hearsay that it works, I wouldn't count on it. For example, operator= materialization. there is an operator= that takes a different engine and a different optraits and does something -- maybe that's enough to deal with the expression templates business, but I don't know.

CEB: What do you mean by "materialization"?

JM: Evaluation of the accumulated instructions -- expression templates work by accumulating mathematical operations into a complicated C++ data structure, and materialization says "go get the values."

BS: Your points are well taken, about documenting the desire to have expression templates, and proving that it works.

GD: That's good high-quality feedback, JM; thank you very much.

BS: Yes, thank you. This is the first paper I've worked on.

BS:

MFH: blas lapack wrapper paper suggestions?

BS:

mdarray? BS would like
paper
proposal

BS: strawman mostly not designed for optimal perf, designed to test
how pieces of interface work together, to get the compile-time
behavior we expect.

We had talked about mdspan as common language for element layout,
perhaps within and certainly beyond the lin alge librayry ith the
low-level stuff. I like it just haven't had time to think about it.
could use mdarray

accessor for expression templates

CB: Lurking. I'm learning allt he things about writing a paper ...

MW: marks paper is good
use of expression tempaltes in library has been borught up
do you think it could be problem with library evolution
not in the C++ standard

BS: I don't know how poeple will react to that.
the operators would be value based at this point.
Our original thinking was to leave it up to implementer,
whether operators work by expression templatse or values
Values are easier to program. We could provide impl
of expression template engines in std::experimental to get more
experience with them

MFH: valarray: optional whether it uses expression tempaltes.

MW: don't get me wrong, i am supportive of using expression templates,
but you're not making it a requirement.

BS: We are not.

BS: Thank you JM!

MW: Yes, thank you JM!

next call is beofre paper deadline. Deadline is 17 June-ish.

Minutes for 2019/06/06 SG14 Conference Call

Scribes: Mark Hoemmen, Bob Steagall

Participants

- Jayesh Badwaik (Jayesh)
- Charles Bay
- Matthew Butler
- Guy Davidson (Guy)
- Ronen Friedman
- Mark Hoemmen (mfh)
- Klaus Iglberger (Klaus)
- Graham Lopez (Graham)
- Maikel Nadolski (Maikel)
- Rene Rivera
- Brett Searles (Brett)
- Bob Steagall (Bob)
- Michael Wong (Michael)

Bob: Update on P1385. As discussed in May, Guy and I made changes to improve customizability. We presented that at C++Now in Aspen. I got good feedback. Here is a summary:

One-vector approach (that is, not having row and column vectors be separate types) turned out to be a controversial design decision.

Audience favored using the std::math namespace.

Regarding the traits-based approach: I got feedback that it seemed like a "1990's approach" to writing an interface. The commenter didn't suggest a way to implement this functionality without traits. However, I talked with Zach Laine who did outline an approach. The idea is to remove the second template parameter (operation traits) from the matrix and vector class types.

Positive feedback on customizability -- people liked arithmetic traits -- some discussion of predicate traits, e.g., number traits for ring vs. field, etc. We were asked why we didn't just use concepts. The answer is that we haven't had time to figure out how to use concepts to do that yet.

I had a conversation w/ Lisa Lippincott. She liked the analysis that preceded the actual design.

I had a conversation w/ David Hollman (Sandia). David mentioned the papers Sandia is developing.

Guy and I are currently working on a revision of P1385 for pre-Cologne submission.

Bob: Klaus has done a nice writeup of feedback for P1385.

Klaus: We [the proposal?] do not deal w/ complex expressions. It is possible w/ infix [arithmetic operator] notation to combine arbitrary number of vectors and matrices [in an expression]. How do we want to deal with this? e.g., A+B+C is easy. but (A+B)*C ? This makes us decide how to deal with the addition. There are 3 different approaches, each of which has been taken by different libraries:

1. Do nothing / forbid this expression from compiling (Eigen)

2. Treat A+B as an expression (template), so that the matrix-matrix multiply evaluates A(i,j)+B(i,j) every time it needs that entry of the expression.

3. Create a temporary matrix for the matrix sum A+B.

We must decide whether or not this expression is allowed. We can't let whether or not it works be implementation defined.

Bob: I don't have a direct answer for that in terms of P1385. Guy and I hesitate to prescribe how to evaluate expressions; I would rather call this "implementation defined."

Maikel: "You want to specify order of evaluation more strictly?"

Klaus: I don't think you should define the underlying technology, but you specify whether it's possible. Regarding Maikel's question, it's not about evaluation order, but whether to allow certain expressions. For example, (A+B)*C is a valid compound expression, but should we allow it? How it works is not what we should specify, but whether it works is.

Klaus: Also, should taking a submatrix of an expression, like "submatrix(A*B)," be allowed? or should "submatrix" only work on a concrete matrix type [not an expression]? mdspan may not be enough to support this use case if we need taking a submatrix of an expression to work.

mfh: It's possible to do some expression templates w/ mdspan through its Accessor policy; the question is whether you want to do expression templates.

Klaus: That's right.

Klaus: Last time, everybody wanted begin and end (iterators) for vectors. However, they did not argue for iterators for matrices, because this is more difficult. But you'll need them once you introduce sparse matrices. Should we deal with this now or wait until add sparse matrices? Once we have iterators on matrices, should we introduce Standard-compliant begin and end member functions that take no arguments, or have them take an index argument and thus allow traversal of a single row or column? The former (zero arguments) has subtle implementation disadvantages: more memory, larger iterator, and slower. The latter would not be STL compliant, but it might work better for matrices in general.

mfh: mdspan only allows indexing; it does not offer iterators. This is a deliberate choice.

Klaus: But sparse matrices.

mfh: The issue with sparse matrices is that users are less likely to want to construct many different kinds of arithmetic expressions with them. However, I can suggest prior work -- in C++, even -- on using relational algebra to define something like iterators over different sparse matrix data structures, for the automatic construction of efficient computational kernels like sparse matrix-vector multiply. See "The Bernoulli Generic Matrix Library" and other publications by Nikolay Nateev, Keshav Pingali, and Paul Stodghill.

Brett: Could we modify Ranges for spanning matrices? I know it "uses iterators" yet it has better capabilities.

mfh: Ranges are 1-D, just like iterators.

Klaus: Regarding the examples of static and dynamic matrix type promotion in Section 6.6 of P1385: The entire text focuses on multiplication. For addition, dynamic + static could perfectly well result in a static matrix. Guy has talked about this at Meeting C++, but it is just not described yet in the paper. Feedback: Be more specific about how result types are evaluated, and give more examples in this section.

Guy: You're correct, we did talk about this at Meeting C++, but only multiplication. Addition is different because dimensions of output are same as those of input. I haven't given thought to the other operations other than multi and their effects on the return types. I will think about that more; thank you.

Klaus: Might need to introduce operations that don't make sense in generic case, like e.g., Kronecker product. If we have an operation for matrices, we should also think about whether it makes sense for vectors.

Klaus: I have 6 pages of findings that I can send around.

Guy: Thank you, Klaus.

Klaus: Any more questions?

Bob: Klaus, you expressed an opinion in favor of row and column vector being separate types. I'd like to hear your thoughts.

Klaus: I think it's valuable to have both, in my experience. I'm happy to distinguish between the two. I deal with more complex stuff, like treating a vector like it's a matrix -- for that, it's helpful to know whether it's a row or column vector. Also, sometimes we want to treat a vector as a matrix, or a matrix as a vector. I like the fact about math that it's exact, but here we're being vague -- we're interpreting something because we feel that it's what users want.

Graham: This opens a can of worms...

Bob: Please go ahead.

GL: 1. We want to have an interface... it's impossible to cover every edge case, so we want to cover the most common but allow special cases. For example, if you want to treat a vector as a single-column matrix, you already have the machinery to do that. We can provide helpers to do that cheaply. But, it's much more often that you want to treat a vector as a vector, not in the linear algebra textbook sense (which is the only place where you differentiate between row and column vectors). 2. Treating a vector as a matrix is type punning, which is generally not accepted for interface design. I'd like to call types what they are.

mfh: Our job is to create low-level machinery, not to please users w/ terms... [goes on to explain that decisions like whether row and column vectors should have different types, or whether vectors get

automatically promoted to matrices or vice versa as needed, are high-level decisions on which users disagree.]

Bob: The point of 1385 is to create basic facilities and interfaces for users, and let us gain experience w/ that, and to add more sophisticated things later. I would like to punt on the idea of how to manage expressions. We don't have to handle that in this paper. Could leave for follow-on papers.

mfh: Precedent for that in valarray. [valarray permits but does not require expression templates. There is a bit of history there. Daveed Vandevoorde attempted to add expression templates to the valarray proposal, which had been abandoned by its original authors, but it was too late in the standardization process and the Committee instead accepted the current language. Some implementations of valarray do actually use expression templates and have other optimizations.]

Bob: We did start out with a small set of requirements we were trying to fulfill. Our paper tries to fulfilll those requirements. We can ask whether they are the right set of requirements.

Graham: I agree with trying to scope it down to provide the low-level operations. That's gives us a smaller goal and less to argue about. I would like to avoid trying to make things nicer for the user prematurely. Nevertheless, I would be open to arguments about the two-vector system, if arguments could be made for it independent from usability. Are there performance implications? Could you not express some operations in the one-vector system, that you could in the two-vector system?

Klaus: That would only come up later (more operations, not defined in the paper). I think there will be second, third, and fourth steps. By saying there is only one kind of vector, do we restrict ourselves in the future?

Bob: Going back to a conversation Graham and I had in Kona: my original thinking for row and column vectors was to distinguish bewteen inner and outer products when using operator*.

Maikel: My experience from using Blaze or Eigen is that differentiating between row and column vectors caught a lot of bugs at compile time.

Jayesh: I agree, esp. in generic physics code.

Graham: Outer and inner product are not the only possible vector products. Row and column vector thus won't get you all the way.

Bob: Any more comments? [none]

Bob: Klaus, thank you. Turn over to Jayesh.

Jayesh: [shares screen] I was wondering how to deal w/ problems like what Klaus raised today, w/ submatrix(A*B). Also, as we put stuff into the Standard Library, we want it to be generic and customizable -- this puts a lot of pressure on the library. Is there a way to let users extend the Standard library, or to use their own library w/ Standard Library types? e.g., std::sort or boost::sort w/ custom containers. Problem is not the named functions themselves, but with infix operators like "a+b". Is this an expression template or an owning type? What if a and b belong to different libraries? Here is an example.

Sorry for "engine" overloaded term w/ P1385. Please don't confuse w/ how I'm using the term. Also, this could be written in terms of concepts, but I'm not so familiar w/ concepts so I'm writing using traits.

[presents design; the idea is to distinguish between owning types, consuming owning types, and expressions (that don't own). Define operations on engine types. Objects like vector can change engine. operator+ defined on T&& and U&& but constrained so they use a particular engine. Example shows how to add two vectors, one that uses a Standard Library type and the other that uses a custom type.]

Jayesh: I have a proof of concept that the two mechanisms work, and am currently trying to combine them. ... if we place similar requirements on our vector types, e.g., that they should have an Engine template parameter, export types, and have the three change_engine functions, then we can get interoperability between libraries.

[Around this time, Guy starts having internet trouble and dropping out.]

Jayesh: ... and we do not have to restrict how the execution of the addition is being done. e.g., if you want to do a different addition for sparse matrices...

Bob: Feedback?

mfh: Sparse matrices are a different world. The kinds of operations users want to do with sparse matrices is often very different than with dense matrices. Thus, you don't have to feel like you need to solve that problem.

Jayesh: Point is to let the Standard Library move slowly....

Bob: Other comments or questions for Jayesh?

mfh: Just wanted to say that this is good stuff! It looks like this could help decouple use of arithmetic operators from linear algebra.

Michael: Jayesh, will you submit this to Cologne?

Jayesh: Not sure; depends if this paper is ready. It is very preliminary. However, I will be there in Cologne.

Michael: I will try to schedule a [face-to-face] SG14 meeting in Cologne for Friday morning.

Bob: Jayesh, this is interesting work. Even if you can't finish the paper in time for the deadline, please keep us appraised.

[At this point, mfh talks about the 3-4 papers on linear algebra that Sandia intends to submit.]

[Rene Rivera and Charles Bay enter the meeting.]

--- Bob taking notes now ...

mfh: several papers under development.

* md_array proposal, like span, but an owning container like std::vector. Will let small data structures be passed around by value.

* another papers that describe lowest-level things that could possibly work. Started with idea of building upon BLAS; an experience paper describing how to wrap BLAS in C++; another aspect is how to simplify how arguments are managed

* another paper is how to support batched linear algebra operations;

* papers still under review, not quite ready for publication

* does not compete with 1385, but provides a set of "atoms" upon which 1385 might be built.

* may be able to present all 3 at Cologne

* not attempting to provide expression support

* could do simd inside

Michael: To the group: Is it appropriate to shoot for a TS?

Bob: Not sure; whichever approach gets linear algebra into '23 fastest is what I want.

mfh: There seems to be discouragement against using TS's lately, don't think it's the right way to go.

Michael: Not so much for Library, but agreed, a TS may be too much.

Michael: Do you intend for your proposal to be a TS?

Bob: no

mfh: no

--- mfh taking notes now ---

[We interpreted general feedback from previous meetings as discouraging a TS.]

Michael: SG14 monthly call next Wednesday; it will be the last before 17 June pre-Cologne mailing deadline.

Bob: 2019-07-03 15:00-05:00 will be next call.