VSIPL Linear Algebra

Presented at VSIPL Tutorial & Users Conference
27 February 2002
Sharon M. Sacco, Ph.D., SKY Computers
Copyright © 2002 SKY Computers, Inc.
All rights reserved.
VSIPL Linear Algebra

- VSIPL Linear Algebra includes all matrix operations except
  - Elementwise operations
    - These are operations that can be viewed as a series of vector operations such as vector multiply over the rows of the matrix or a single operation such as summing over all the elements of a matrix.
    - They do not naturally rearrange the data organization.
      - Matrix transpose can be done with copy functions, but the input and output views have radically different strides.
  - 2D convolution and correlation
VSIPL Linear Algebra

• VSIPL Linear Algebra functions are not designed to replace packages such as LAPACK
  – Functional emphasis is on what is really used by the vector and signal processing communities
  – Does not explicitly support sparse matrices
VSIPL Linear Algebra

• VSIPL Linear Algebra functions can be divided into two categories
  – Matrix and vector operations
  – Linear systems solvers
    • VSIPL does not include matrix inverse or eigenvector solvers.
• Notation used in function names:
  – $d$ is the dimension, either $c$ for complex or a missing $r$ for real.
  – $s$ is the object shape, either $v$ for vector or $m$ for matrix.
  – $p$ is the precision of the calculation. Common values in many implementations are $f$ for single precision floating point or $d$ for double precision floating point.
  – When not a part of name (e.g. vsip_cmprod$h$p),
    • $j$ indicates complex conjugate
    • $t$ indicates transpose
    • $h$ indicates hermitian transpose
• Dot Products: vsip_cvjdot_p, vsip_dvdot_p
  – Dot products, although they are vector operations, are considered to be closely related to matrix-vector operations under VSIPL, so the manual pages are in the linear algebra section.
  – Dot products are included in the Core Lite Profile
    • In a Core Lite implementation, dot products can be used to generate other linear algebra functionality. This may improve performance over straight compiled code. However, it does require the user to have a good knowledge of the algorithm to write the program with dot products.
• Transposes: vsip_dmtrans_p, vsip_cmherm_p
  – Transposes are included in the Core profile
  – In Core Lite implementations, copy functions can replace the transpose function and vector conjugate functions can replace the hermitian transpose. This may lose performance from an optimized transpose on many systems, particularly those with data caches.
• **Matrix Products**
  - One version allows transpose of one of the matrices. The idea here is to improve performance by avoiding strided accessing of memory.
  - There are versions to allow conjugation and hermitian transpose of complex matrices.
  - Includes specialized versions for 3x3 and 4x4 matrices. Again, these are specified on hope of optimization.
• **Matrix/Vector and Vector/Matrix Products**
  – These functions could be performed by matrix operations, however there can be optimizations made for knowledge that one matrix is really a vector.
  – Includes specialized versions of matrix/vector product for 3x3 and 4x4 matrices. On some systems there may be methods to optimize for these special cases.
## VSIPL Matrix and Vector Operations

- **Products and Profiles**

<table>
<thead>
<tr>
<th>Matrix and Vector</th>
<th>Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>In Core:</strong></td>
<td></td>
</tr>
<tr>
<td>vsip_mvprod_p</td>
<td>vsip_dmprod_p</td>
</tr>
<tr>
<td>vsip_vmprod_p</td>
<td>vsip_cmprodh_p</td>
</tr>
<tr>
<td></td>
<td>vsip_cmprodj_p</td>
</tr>
<tr>
<td></td>
<td>vsip_dmprodt_p</td>
</tr>
<tr>
<td><strong>Outside of profiles:</strong></td>
<td></td>
</tr>
<tr>
<td>vsip_dmvprod3_p</td>
<td>vsip_dmprod3_p</td>
</tr>
<tr>
<td>vsip_dmvprod4_p</td>
<td>vsip_dmprod4_p</td>
</tr>
</tbody>
</table>
• VSIPL General Matrix Product and Sum
  – vsip_dgemp_p
    • \( C = \alpha \cdot \text{op}(A) \cdot \text{op}(B) + \beta \cdot C \)
  – vsip_dgems_p
    • \( C = \alpha \cdot \text{op}(A) + \beta \cdot C \)

where \( \text{op}(A) \) can mean \( A, A^T, A^H, \) or \( A^* \), the latter two applying to complex only.
VSIPL Matrix and Vector Operations

• VSIPL General Matrix Product and Sum (cont.)
  – Included in Core Profile
  – Very versatile functions from the user point of view
  – Should be thought of as “Swiss Army Knives”
    • They serve a lot of different purposes
    • They can be heavy, i.e. they may introduce a lot of unused code into an executable
    • They may not be the most optimal tool for the job, i.e. less versatile functions may give better performance on some systems
• VSIPL Outer Products
  – Vector Outer Product, vsip_dvouter_p
    • \( C_{ij} = \alpha \cdot x_i \cdot y_j^* \)
  – Kronecker Product, vsip_dskron_p
    • \( C = \alpha \cdot x \otimes y \) for vectors or \( C = \alpha \cdot A \otimes B \) for matrices.
• Most Linear Systems Solvers are included in the Core Profile
  – Special Linear Systems Solvers include Covariance Systems, Linear Least Squares Problem and Toeplitz Systems. These are single function calls.
  – Other systems solvers include LUD, Cholesky, and QRD. These methods require more than one function call to solve the equation.

• Singular Value Decomposition is included in VSIPL, but not required by Core.
• Covariance System Solver, vsip_dcovsol_p
  – Solves $A^TAX = B$ or $A^HAX = B$
  – The output matrix $X$ overwrites the input matrix $B$
  – $A$ is of order $M$ by $N$ with rank $N$, $M \geq N$. $B$ is a matrix of order $N$ by $K$.
  – Function may allocate and free temporary workspace internally.
    • This can be a performance problem.
    • This can also contribute to memory fragmentation on some systems if used many times.
    • If performance and memory are a concern, QRD may be preferable.
• Linear Least Squares Problem Solver, vsip_dllsqsol_p
  – Solves $\min_x ||AX - B||_2$
  – The output matrix X overwrites the input matrix B
  – $A$ is of order $M$ by $N$ with rank $N$, $M \geq N$. $B$ is size $M$ by $K$.
  – Function may allocate and free temporary workspace internally.
    • This can be a performance problem.
    • This can also contribute to memory fragmentation on some systems if used many times.
    • If performance and memory are a concern, QRD may be preferable.
• Toeplitz System Solver, vsip_dtoepsol_p
  – Solves a real symmetric or complex Hermitian Toeplitz system
    \[ T x = b, \text{ where } t_{ij} = t_{ji} \text{ (real)} \text{ or } t_{ij} = t_{ji}^* \text{ (complex)}. \]
  – Input matrix is stored as a vector, saving memory.
  – Calling sequence includes an explicit work buffer so memory allocation occurs outside of function.
  – Common methods used are Levinson’s algorithm and Schur’s method.
The general square linear system solver is a suite of functions using LU decomposition and back substitution.

- Create an LU decomposition Object, `vsip_dlud_create_p`
  - The creation of the LU object allows reuse of the decomposition.
  - This function allocates memory to store temporary buffers, permute matrices, and other information either needed by LU decomposition or produced by LU decomposition.
  - The organization of the object is implementation dependent. It is considered to be private memory.
VSIPL General Square Linear System Solver

- Compute an LU decomposition of a square matrix, vsip_dlud_p
  - Uses either row or column partial pivoting in an effort to improve the stability of the algorithm.
    - For row pivoting, $A = PLU$, for column pivoting, $A = LUP$, where $P$ is the permutation matrix.
    - The choice of row or column pivoting belongs to the implementation.
  - The input matrix $A$ may be overwritten with the lower and upper matrices.
    - Once the decomposition has been made, $A$ should not be used until the decomposition is no longer needed.
  - Crout’s algorithm is commonly used.
VSIPL General Square Linear Systems Solver

– Solve a General Linear System, vsip_dlusol_p
  • Solves the equation $\text{op}(A)X = B$, where $\text{op}(A)$ can be $A$, $A^T$, or $A^H$.

– Destroy and LUD object, vsip_dlud_destroy_p
  • Frees any memory associated with an LUD object that has been created by vsip_dlud_create_p.

– Get the attribute of an LU decomposition object, vsip_dlud_getattr_p.
  • An LU attribute structure contains the length of the rows or columns of $A$. 
• Cholesky decomposition is a special case of LU decomposition, sometimes referred to as a square root decomposition.
  – For symmetric matrices, where $a_{ij} = a_{ji}$ (real) or $a_{ij} = a_{ij}^*$ (complex), the matrix $A$ can be decomposed into
    
    $A = LL^T$ (real) or $A = LH^*$ (complex),
    
    where $L$ is a lower triangular matrix or
    
    $A = R^TR$ (real) or $A = RH^*$ (complex),
    
    where $R$ is an upper triangular matrix.
  – Cholesky is known for its stability.
– Create a Cholesky decomposition object, vsip_dchold_create_p

- Creation of the Cholesky decomposition object allows reuse of the decomposition.
- This function allocates memory to store temporary buffers and other information either needed by the Cholesky decomposition or produced by it.
- The organization of the object is implementation dependent. It is considered to be private memory.
– Compute the Cholesky decomposition of a symmetric positive definite matrix, vsip_dchold_p

• The input matrix A may be overwritten by the decomposition.
  ▪ Once the decomposition has been made, A should not be used until the decomposition is no longer needed.

• Common methods used are Gram-Schmidt, Givens rotations, and Householder transformation.
– Solve a symmetric positive definite matrix from its Cholesky decomposition, vsip_dcholdsol_p
  • Solves the equation AX = B.
  • B is overwritten by X

– Destroy a Cholesky object, vsip_dchold_destroy_p
  • Frees any memory associated with the Cholesky object that has been created by vsip_dchold_create_p

– Get attribute of Cholesky object, vsip_dchold_getattr_p
  • A Cholesky attribute structure contains the length of the rows or columns of A.
The overdetermined linear systems solver is a suite of functions using QR decomposition.

- Input matrix $A$ can be rectangular, $M$ by $N$, where $M \geq N$.
- QR decomposes $A$ into an orthogonal (or unitary) matrix $Q$ and an upper triangular matrix $R$, $A = QR$. 
– Create a QR decomposition object, `vsip_dqrd_create_p`.
  - Creation of object allows reuse of the decomposition.
  - R is always saved.
  - Option allows Q to be
    - Not saved
    - Save the full Q
    - Save the “skinny Q”. The QR product can be viewed as Q being two horizontally adjacent matrices, [Q1 Q2] and R as a square upper triangular matrix with a zero matrix adjacent below.
  - The organization of the object is implementation dependent. It is considered to be private memory.
Overdetermined Linear System Solver

- Compute the QR decomposition, `vsip_dqrd_p`
  - The input matrix A may be overwritten by the decomposition.
    - Once the decomposition has been made, A should not be used until the decomposition is no longer needed.
    - Common methods used are Gram-Schmidt, Givens rotations, and Householder transformation.
Overdetermined Linear System Solver

– Solve a linear system based on R from QR decomposition, vsip_dqrdsolr_p
  • Solves a triangular linear system of the form,
    \[ \text{op}(R) X = \alpha \cdot B \]
    where \( \text{op}(R) \) is \( R, R^T, \) or \( R^H \).
  • Input matrix B is overwritten by X.

– Solve either a linear covariance or linear least squares problem based on a QR decomposition, vsip_dqrdsol_p.
  • Problem solved is determined by an option.
  • Input matrix B is overwritten by X.
Overdetermined Linear System Solver

- Product with Q from a QR decomposition, `vsip_dqrqrdprodq_p`
  - Option to save Q needs to be used during the QRD object creation.
  - An option allows multiplication of Q, $Q^T$, or $Q^H$.
  - An option determines whether Q is multiplied on the right or left hand side of the input matrix C.
  - C is overwritten.
Overdetermined Linear System Solver

- Destroy a QR decomposition object, `vsip_dqrd_destroy_p`
  - Frees memory allocated during `vsip_dqrd_create_p`.

- Get attributes of a QR decomposition object, `vsip_dqrd_getattr_p`
  - Allows access to the size of the original matrix A and the option chosen for retention of Q.
Singular Value Decomposition

• SVD is a suite of functions similar to the other matrix decomposition routines.
  – A major difference is that SVD does not provide a linear equation solver.
    • Systems of equations can be solved using the SVD multiplication routines.
  – SVD calculates $A = USV^T$ (real) or $A = USV^H$ (complex), where $U$ is a $M \times M$ orthogonal (unitary) matrix, $S$ is a diagonal matrix, and $V$ is an $N \times N$ orthogonal (unitary) matrix.
    • SVD is known to be a very stable method.
  – SVD is not included in Core.
Singular Value Decomposition

– Create a Singular Value Decomposition Object, \texttt{vsip\_dsvd\_create\_p}

\begin{itemize}
\item Creation of the object allows reuse of the decomposition.
\item An option determines whether \(U\) is not computed, partially computed or fully computed.
\item A similar option determines whether \(V^T\) (\(V^H\)) is not computed, partially computed or fully computed.
\end{itemize}
– Compute the Singular Value Decomposition, `vsip_dsvd_p`

- Computes $A = USV^T$ or $A = USV^H$
- The input matrix $A$ may be overwritten during the computation. $A$ must not be used until the decomposition is no longer needed.
- The diagonal matrix $S$ is stored as a vector. The singular values are positive and real that are returned in descending order.
Singular Value Decomposition

- Product with U from SV Decomposition, `vsip_dsVdprodu_p`
  
  - U must be either partially or fully saved in the SVD object.
  - An option allows U, \( U^T \), or \( U^H \) to be used.
  - An option allows either \( \text{op}(U) \) to be applied to the left or right side of \( C \).
  - C is overwritten.
    
    - The output matrix may be larger than the original matrix \( C \). In that case, it is stored in the natural order determined by the offset, row stride and column stride of the input matrix view. In these cases, the block containing \( C \) must be large enough to embed the new matrix. The view of \( C \) also needs to be sufficiently spread in the block so that the new matrix has unique memory locations for all elements.
Singular Value Decomposition

- Product with V from SV Decomposition, \texttt{vsip_dsYvdprodv_p}
  - V must be either partially or fully saved in the SVD object.
  - An option allows V, V^T, or V^H to be used.
  - An option allows either \texttt{op(V)} to be applied to the left or right side of C.
  - C is overwritten
    - The output matrix may be larger than the original matrix C. In that case it is stored in the natural order determined by the offset, row stride, and column stride of the input matrix view. In these cases, the block containing C must be large enough to embed the new matrix. The view C also needs to be sufficiently spread in the block so that the new matrix has unique memory locations for all elements.
Singular Value Decomposition

• Destroy a Singular Value Decomposition object, `vsip_dsvd_destroy_p`
  – Frees memory allocated during `vsip_dsvd_create_p`.

• Get attributes of a Singular Value Decomposition object, `vsip_dsvd_getattr_p`
  – Allows access to the size of the original matrix A and the options for saving U and V.
• A Few Selected References
  
  – Two easily available references are
  
  – Available through SIAM