

# A Matrix-type for Performance Portability

N. P. Drakenberg

Department of Microelectronics and Information Technology

Royal Institute of Technology

Electrum 204, S-164 40 Kista, SWEDEN

e-mail: npd@it.kth.se

January 30, 2004

## Abstract

We describe a datatype for (dense) matrices whose primitive operations are decomposition and composition (of submatrices), as opposed to indexed element access which is the primitive operation on conventional arrays. Using the composition and decomposition operations it is for example possible to express both recursive and traditional block matrix algorithms (*e.g.*, Cholesky factorization, QR-factorization, *etc.*) as is illustrated by the following (pretty-printed and contracted) example:

```
function cholesky: Matrix  $\mapsto$  Matrix is
  cholesky( [ a ] ) = [  $\sqrt{a}$  ]
  ⋮
  cholesky(  $\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}$  ) =
    let U = cholesky( A )
        ⋮
        Z = zeros( C )
    in  $\begin{bmatrix} \mathbf{U} & \mathbf{V} \\ \mathbf{Z} & \mathbf{X} \end{bmatrix}$ 
end cholesky
```

in which the blocking of matrices is explicit (allowing block algorithms not derivable by compilers to be expressed and used in contexts where portability is required), but at the same time block *sizes* are implicit giving compilers the freedom to choose block sizes so as to best suit the target architecture (or system). However, fully general and arbitrary composition of matrices may require data-copying and is therefore potentially very expensive. To ensure efficiency, functions taking matrix arguments and/or yielding matrix results must satisfy two additional requirements:

1. the size ( $m \times n$ ) of any matrix-valued function results must be affine functions of the sizes of the function's argument matrices. This ensures that all of the memory needed for each matrix can be allocated at a single point in time.
2. All matrix compositions must be possible to perform in-place (*i.e.*, without data copying), which implies that the ultimate storage location (through sequences of matrix compositions and function calls) of every matrix element must be known at the time when the value is computed.

These requirements are verified through program analysis (currently using abstract interpretation) and compile-time error messages are delivered for any program which cannot be determined to satisfy the requirements.

Several of the key functions/subroutines of BLAS (*e.g.*, dgemm, dsyrk, dtrsm) have been expressed in terms of the matrix-type described above, and performance evaluations of the executable code produced by our current (prototype) compiler implementation shows it to be 10–20% faster than the vendor-tuned architecture-specific BLAS implementation on SGI Octane (R12000, 400MHz) systems. This performance advantage is achieved mainly by the compiler's use of the (pipelined) L2-cache to allow larger iteration counts in inner loops and thereby more efficient software pipelining and fewer mispredicted branches.