Date:	5 February 1997
To:	X3J3
From:	Van Snyder
Subject:	Impact of Uniform Syntax on Parallel Computation

One significant problem with parallel computation today is that it is difficult to express algorithms that are portable between and work well on scalar machines, ensemble machines having shared memory, and ensemble machines that use message passing.

The difficulty arises from using a procedure library, say MPI, to manage message passing.

Accessors might provide a method to allow portable expression of parallel algorithms without significant compromise of efficiency in scalar environments.

Consider the Householder transformation. Let v be an arbitrary column of a matrix A. From v we derive t and α. Then the transformation is applied to another column, w (in place), by replacing w by $(\mathrm{I} - \alpha t t^T)w$. One way this has been implemented is:

$$\begin{aligned} t_i &:= v_i,\ i > 1 \\ t_1 &:= v_1 + \mathrm{sign}(||v||, v_1) \\ \alpha &:= \frac{1}{||v||\,|t_1|} \\ \beta &:= \alpha(t^T w) \\ w &:= w - \beta t \end{aligned}$$

Fortran statements to accomplish the above calculations using BLAS-1 might be:

```
real A(M,N)
do i = 1, n
  call h1 (a(i:,i), i) ! array or function reference for A
end do ! i

contains
  subroutine H1 (V, i) ! argument association for V and
  ! host association for A prevents fetching i'th column twice
    real V(:) ! and other declarations as necessary
    vn = s2nrm(v)
    v(1) = v(1) + sign(vn, v(1))
```

```
      alpha = 1 / (vn * abs(v(1)))
      do j = i, n
        call APPLY (a(i:,j)) ! array, or function + updater, for A
      end do ! j
      return
      contains ! notice the utility of two-level nesting
      subroutine APPLY (w)
        real, intent(inout) :: W(:)
        w = w - ( alpha * sdot(v, w)) * v
        return
      end subroutine APPLY
    end subroutine H1
```

Using a naive decomposition in which each processor "owns" one column, one would insert some calls to a communication library, say MPI, to bring the j^{th} column of v to the processor that "owns" the i^{th} (or vice versa). (Maybe this is a stupid strategy – I just want to illustrate the principle.) Inserting this kind of call is what makes it difficult to transport a program developed for a parallel system that uses message passing to a system that has shared memory, or to a scalar machine.

Accessors allow one to "hide" the details of message passing (or the lack thereof) *in the declaration for A*. On a parallel system that uses message passing, one would change *only the declaration for A* from `REAL A(M,N)` to

```
  real function A(I, J) result(C)
    sequence I ! first:last:stride of abstract "object" A
    integer J
    real C(I)
    ! MPI calls to fetch the I rows of column J of the abstract
    ! array A from its owner, and put it into C
  updater
    ! MPI calls to send the I rows of column J of the abstract
    ! array A from C to the owner of the column
  end function V
```

and change none of the usages!. A *single algorithm* with a change *only in the declarations* is transportable between scalar and parallel machines.