ON THE IMPACT OF HPF DATA LAYOUT ON THE DESIGN OF EFFICIENT AND MAINTAINABLE PARALLEL LINEAR ALGEBRA LIBRARIES

Christian H. Bischof[1], Steven Huss-Lederman[2], Elaine M. Jacobson[2], Xiaobai Sun[1], and Anna Tsao[2]

Abstract

In this document, we are concerned with the effects of data layouts for nonsquare processor meshes on the implementation of common dense linear algebra kernels such as matrix-matrix multiplication, LU factorizations, or eigenvalue solvers. In particular, we address ease of programming and tunability of the resulting software. We introduce a generalization of the torus wrap data layout, that results in a decoupling of "local" and "global" data layout view. As a result, it allows for intuitive programming of linear algebra algorithms, *and* tuning of the algorithm for a particular mesh aspect ratio or machine characteristics. This layout is as simple as the proposed HPF layout, but, in our opinion, enhances ease of programming as well as ease of performance tuning. We emphasize that we do not advocate that all users need be concerned with these issues. We do, however, believe, that for the foreseeable future "assembler coding" (as message-passing code is likely to be viewed from a HPF programmers' perspective) will be needed to deliver high performance for computationally intensive kernels. As a result, we believe that the adoption of this approach would not only accelerate the generation of efficient linear algebra software libraries, but would also accelerate the adoption of HPF as a result. We point out, however, that the adoption of this new layout would necessitate that an HPF compiler insure that data objects are operated on in a consistent fashion across subroutine and function calls.

[1]Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439-4801
[2]Supercomputing Research Center, 17100 Science Drive, Bowie, MD 20715

Introduction

In this document, we present a generalization of the torus wrap data layout for processor arrays of arbitrary aspect ratio. This data layout is currently being investigated as part of the PRISM (Parallel Research on Invariant Subspace Methods) project and has been effectively utilized in the development of a highly efficient general code for matrix multiplication on the Intel Touchstone Delta [7]. We propose that this data layout, which we call *virtual 2-D torus wrap*, be considered for inclusion into the High Performance Fortran standard, because we believe it to have great potential in the design and development of dense linear algebra algorithms. This data layout is a natural extension of the virtual contiguous block data layout described in papers such as [8] and subsumes the block scattered decomposition (e.g., [4]) currently being suggested for the HPF standard.

Virtual 2-D torus wrap was motivated by our desire to exploit the advantageous features of block torus wrap (see, for example, [2, 6]), while at the same time maintaining the simplicity and cost effectiveness of the communication patterns found in many algorithms for square meshes. In particular, virtual 2-D torus wrap has the following desirable properties:

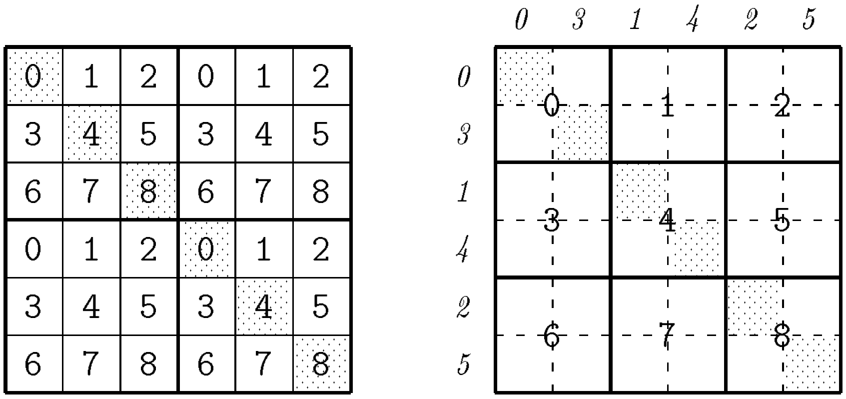
- Virtual 2-D torus wrap decouples the processor view from the physical mesh configuration, allowing the programmer to view a processor array of arbitrary aspect ratio as a square mesh of processors. This simplifies algorithm design, reduces coding and maintenance effort and facilitates flexibility in multiprocessor utilization.
- Virtual 2-D torus wrap allows a variety of useful *virtual to physical* mappings of data, including contiguous blocking and the block scattered decomposition currently supported by the draft of the HPF standard. Code tuning can be achieved by simply varying this mapping, *without affecting the "node" code.*
- The mapping used in our work, like the block scattered decomposition, is advantageous for row- or column-oriented algorithms because it allows physically spreading of row or column blocks across processor rows or columns. However, symmetric operations such as transposition and tridiagonalization are greatly simplified.
- Virtual 2-D torus wrap provides data layouts which maximize the granularity of local computations by ensuring proper maximal alignment in algorithms such the distributed matrix multiplication algorithms found in [8, 7]. Hence, this layout can make full use of, for example, assembler-coded node BLAS, or LAPACK routines.

In the remainder of this document, we first review torus wrap and the block scattered decomposition and then describe the virtual 2-D torus wrap data layout. We will use simple examples to describe each of the data layouts presented.

0	1	2
3	4	5
6	7	8

FIGURE 1. 3×3 mesh node numbering

2-D torus wrap on square meshes.
Consider a 3×3 physical processor mesh, numbered (0, 1, ..., 8) as in Figure 1. Panel the matrix A in both dimensions with panels of width r and s, respectively; this results in a decomposition of A into $r \times s$ blocks. We illustrate the data layout from two different perspectives, for a matrix having six panels (*0*, *1*, ..., *5*) in each dimension, i.e., A is a $6r \times 6s$ matrix. Figure 2(a) (matrix point of view), shows the processor assignments superimposed on the blocks of A; the template in Figure 1 is replicated until all blocks have been assigned. The numbers at the left (top) edge of the processor mesh depicted in Figure 2(b) (processor point of view) refer to the row (column) panels of the matrix and show the processor row (column) in which each row (column) panel of A resides. Figure 2(b) also shows the actual local arrangement of blocks within each processor. In both figures, the shaded blocks represent the diagonal blocks of the matrix when $r = s$. This special case is used in algorithms such as the solution of triangular systems [5] and the reduction of a symmetric matrix to tridiagonal form in [1, 9, 3]. Notice that in this case the diagonal blocks lie in the processors along the diagonal of the processor array.



(a) Matrix point of view (b) Processor point of view

FIGURE 2. Data distribution for 2-D torus wrap

We now describe two generalizations of this data layout for non-square processor meshes:

the block scattered decomposition, which is currently suggested for inclusion in the Fortran HPF standard, and the virtual 2-D torus wrap, which we suggest as an alternative.

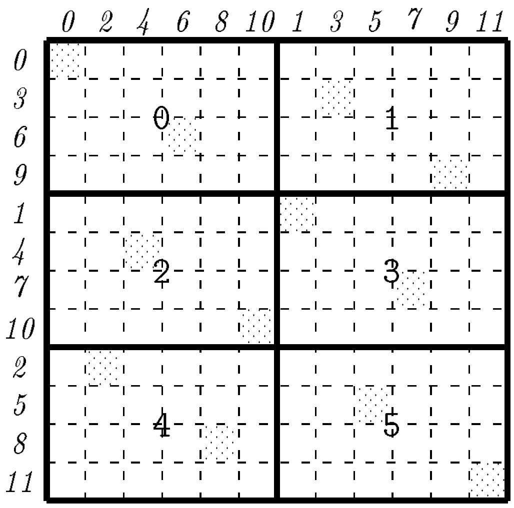
Block scattered decomposition.
Consider a 3×2 physical processor mesh numbered 0, 1, $\ldots$, 5 as in Figure 3.

0	1
2	3
4	5

FIGURE 3. 3×2 mesh node numbering

0	1	0	1	0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3	2	3	2	3
4	5	4	5	4	5	4	5	4	5	4	5
0	1	0	1	0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3	2	3	2	3
4	5	4	5	4	5	4	5	4	5	4	5
0	1	0	1	0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3	2	3	2	3
4	5	4	5	4	5	4	5	4	5	4	5
0	1	0	1	0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3	2	3	2	3
4	5	4	5	4	5	4	5	4	5	4	5

(a) Matrix point of view

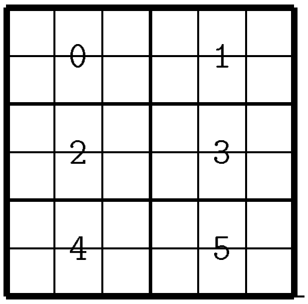


(b) Processor point of view

FIGURE 4. Data distribution for block scattered decomposition

Panel the matrix A in both dimensions to obtain $r \times s$ blocks. We illustrate using a matrix having 12 panels (*0*, *1*, $\ldots$, *11*) in each dimension, i.e., A is a $12r \times 12s$ matrix. Figures 4(a) and 4(b) give the matrix and processor points of view, respectively. Again, the shaded blocks represent the diagonal blocks of the matrix when $r = s$. Notice that the diagonal blocks of the matrix are spread out all over the mesh and in particular, are not generally in contiguous rows or columns of memory.

Virtual 2-D torus wrap.
Again consider a 3×2 physical processor mesh as shown in Figure 5(a). For virtual 2-D torus wrap, we virtualize the physical mesh to a 6×6 square mesh. Figure 5(b) shows the two-dimensional processor numbering for the resulting virtualized mesh. Note that we can actually virtualize a $p \times q$ to a $\alpha \times \alpha$ mesh where the least common multiple of p and q divides α. Panels are then assigned as if on a $\alpha \times \alpha$ mesh.



(a) 3×2 physical mesh

00	01	02	03	04	05
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

(b) Virtualized as 6×6 mesh

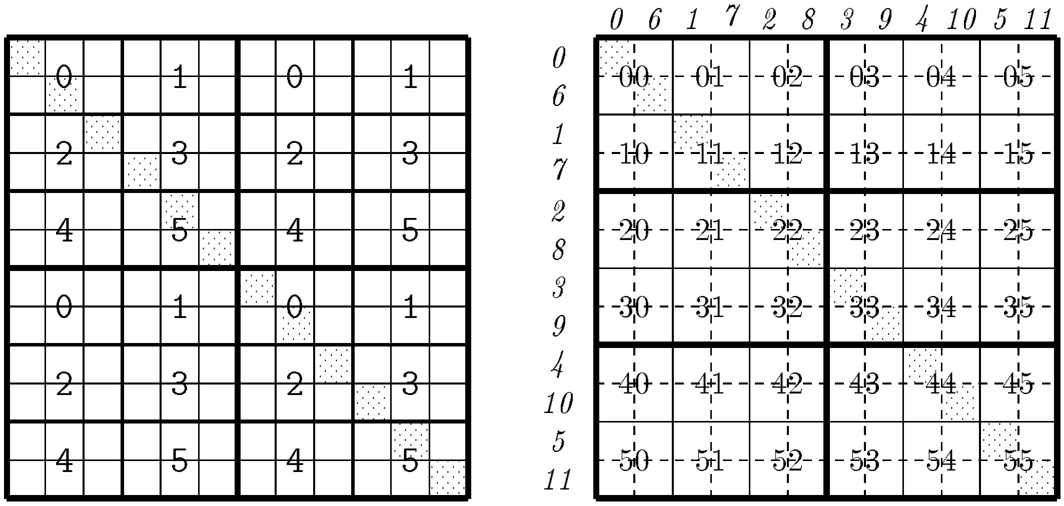
FIGURE 5. Virtualized 3×2 mesh

If we distribute the panels in torus wrap fashion, i.e., the 6×6 template in Figure 5 is replicated over the blocks of A, we have a virtual analog of torus wrap. The latter results in the matrix and processor points of view depicted in Figure 6, for the same example used in the block scattered decomposition. The matrix is arranged in memory so that the data corresponding to all virtual processors within a node is stored in a contiguous buffer. This allows the user to achieve maximal granularity in block algorithms. Notice that if $r = (\#\text{ rows of } A)/6$ and $s = (\#\text{ columns of } A)/6$, we have the contiguous block data layout [8].

An important point to realize is that successive panels in each dimension can be assigned to virtual processors arbitrarily and a very useful case occurs when panels are assigned with a "virtual panel spacing" in each dimension, s_r and s_c. Panels are assigned consecutively so that panel i in the row dimension is assigned to row virtual processor

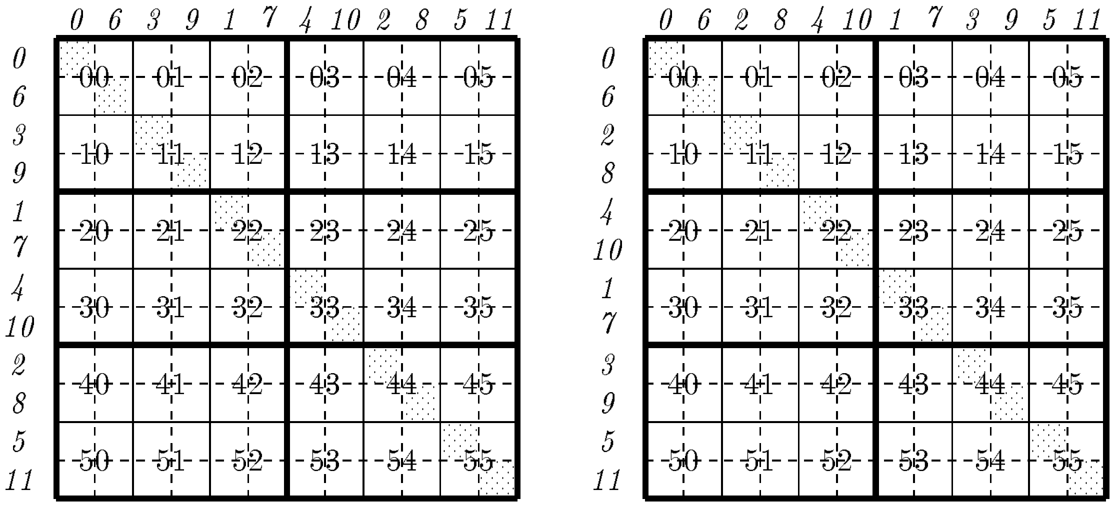
$$s_r\left(i \mod \frac{\alpha}{s_r}\right) + \left(\left\lfloor \frac{i}{(\frac{\alpha}{s_r})} \right\rfloor \mod s_r\right).$$

In the above expression, $\lfloor * \rfloor$ denotes the greatest integer function. Panels are assigned in the column dimension analogously, substituting s_c for s_r in the formula above. Qualitatively, virtual panel spacing is the number of virtual processors to the next panel.



Matrix point of view Processor point of view

FIGURE 6. Virtual torus wrap



Panel spacing = 2 Panel spacing = 3

FIGURE 7. Physical row or column panel wrap from processor point-of-view

Some algorithms are row-oriented (column-oriented) and achieve better performance and load balancing when successive row (column) panels lie in different *physical* processors. This physical spreading of rows (columns) across physical rows (columns) of the mesh can be accommodated by choosing the virtual panel spacing in the row (column) dimension to be the number of virtual processors per node in that direction. Figure 7(a) illustrates physical spreading of row panels by choosing $s_r = s_c = 2$ (α/p) and Figure 7(b) illustrates physical spreading of column panels by choosing $s_r = s_c = 3$ (α/q). We note that for $s_r =$

s_c, this data layout preserves the symmetric placement of indices onto virtual processors. *In fact, the "local processor view" in Figures 6 and 7 is identical, even though the overall load balancing properties of the algorithms are likely to be different.* We see three main advantages resulting from this property:

- Global data reorderings, or changes in block size, as motivated by considerations relating the behavior of an algorithm to the particular mesh aspect ratio, do not affect the "node code".
- Symmetric operations such as transposition and tridiagonalization are straightforward, since each virtual processor behaves as if it were a physical processor on a square mesh.
- Row and column indices are naturally aligned for matrix operands in algorithms such as the distributed matrix multiplication algorithms in [8, 7].

Notice that our first example of virtual 2-D torus wrap in Figure 6 has a virtual panel spacing equal to one; furthermore, if $s_r = 2$ and $s_c = 3$, or in general, $s_r = \alpha/q$ and $s_c = \alpha/p$, we obtain the block scattered decomposition (up to local storage differences). The four special cases of virtual 2-D torus wrap discussed in this document are the only ones that currently appear to be useful, but further study of this is required. Note that the adoption of "spaced" 2-D torus wrap would necessitate that an HPF compiler insure that data objects laid out with different panel spacings are operated on in a consistent fashion across subroutine and function calls.

As we have seen, the extension of the HPF draft discussed in this document has several desirable attributes. Since "spaced" virtual 2-D torus wrap is a superset of the proposed HPF block scattered decomposition, adoption of this proposal would provide the functionality of all the special cases discussed above under a single general framework.

References

1. Anderson, E., A. Benzoni, J. Dongarra, S. Moulton, S. Ostrouchov, B. Tourancheau, & R. van de Geijn, *LAPACK for distributed memory architectures: Progress report*, Fifth SIAM Conference on Parallel Processing for Scientific Computing, Houston, SIAM, 1991.
2. Ashcraft, C. C., *The distributed solution of linear systems using the torus wrap data mapping*, Engineering Computing and Analysis Technical Report ECA-TR-147, Boeing Computer Services (1990).
3. Bischof, C., M. Marques, and X. Sun, *Parallel bandreduction and tridiagonalization*, Proceedings, Sixth SIAM Conference on Parallel Processing for Scientific Computing (Norfolk, Virginia, March 22-24, 1993) (R. F. Sincovec, eds.), SIAM, Philadelphia, 1993.
4. Choi, J., J. J. Dongarra, R. Pozo, & D. W. Walker, *ScaLAPACK: A scalable linear algebra library for distributed memory concurrent computers*, Proceedings, Fourth Symposium on the Frontiers of Massively Parallel Computation (McLean, Virginia, October 19–21, 1992), IEEE Computer Society Press, Los Alamitos, California, 1992, pp. 120–127.
5. Choi, J., J. J. Dongarra, & D. W. Walker, *Level 3 BLAS for distributed memory concurrent computers*, CNRS-NSF Workshop on Environments and Tools for Parallel Scientific Computing (Saint Hilaire du Touvet, France, September 7–8, 1992), Elsevier Science Publishers, 1992.

6. Hendrikson, B. and D. Womble, *The torus-wrap mapping for dense matrix calculations on massively parallel computers*, SAND92-0792, Sandia National Laboratories (1992).
7. Huss-Lederman, S., E. M. Jacobson, A. Tsao, & G. Zhang, *Matrix Multiplication on the Intel Touchstone Delta*, Proceedings, Sixth SIAM Conference on Parallel Processing for Scientific Computing (Norfolk, Virginia, March 22-24, 1993) (R. F. Sincovec, eds.), SIAM, Philadelphia, 1993, also to appear as Technical Report, Supercomputing Research Center, 1993.
8. Mathur, K. K. & S. L. Johnsson, *Multiplication of matrices of arbitrary shape on a data parallel computer* (1992), Thinking Machines Corporation, preprint, also released as Technical Report TR-216.
9. Van de Geijn, R. A., *Massively parallel LINPACK benchmark on the Intel Touchstone Delta and iPSC/860 systems: Progress Report*, Computer Science Technical Report TR-91-28, University of Texas (1991).