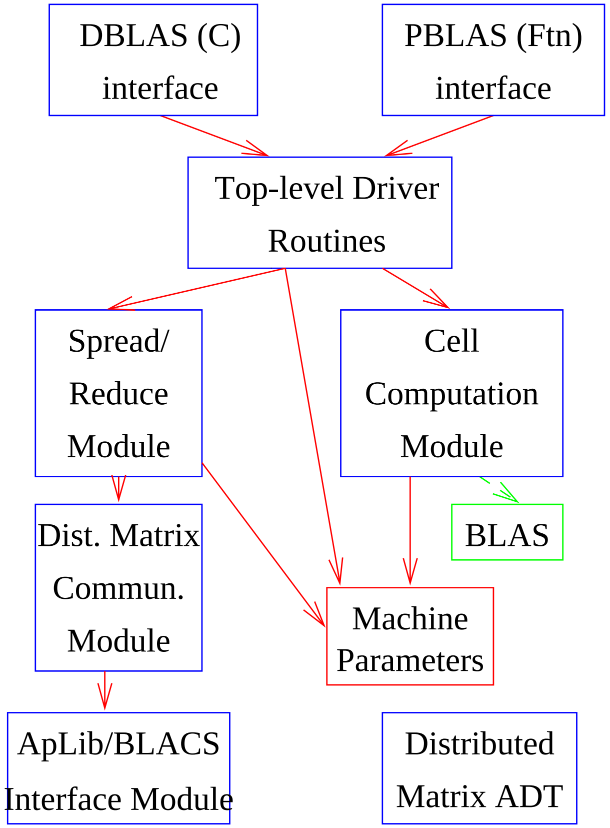
1. DBLAS: a better, portable Distributed BLAS

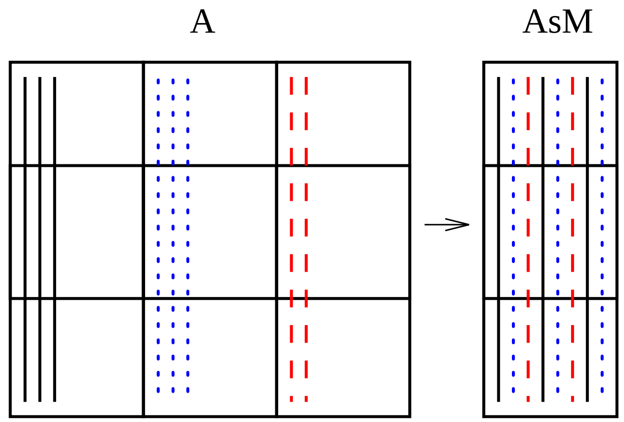
- supports the general $r \times s$ block-cyclic matrix distribution for levels 1, 2, and 3 parallel BLAS (single and double precision)
  - block alignment required on only 2 of the 3 matrix dimensions
    $\Rightarrow$ allows non-square blocks for parallel LAPACK-style algorithms
- efficient algorithms for proper parallel 'Level 3' computations
  $\Rightarrow$ important for optimal performance on memory-hierarchy multiprocessors, eg. the Fujitsu AP series
- C interface arguably simpler than that for serial BLAS, eg.:

```
P[j] = j + DDAMAX(M-j, SubMat(A, j, j), &Ajmx);
DDSWAP(N, TransSubMat(A, j, 0), TransSubMat(A, P[j], 0));
DDSCAL(M-(j+1), 1.0/Ajmx, SubMat(A, j+1, j));
DDGER(M-(j+1), N-(j+1), -1.0, SubMat(A, j, j+1),
      SubMat(A, j+1, j), SubMat(A, j+1, j+1));
```

  details of parallelism and matrix distribution are almost completely hidden!
- uses UTK/ORNL's PBLAS (Parallel BLAS) interface for Fortran programs

2. Architecture of the DBLAS





Row 'spread' of the distributed matrix A (having 8 block columns), across the rows of a 3×3 processor grid. The result is stored in `AsM`, which is replicated across all processor columns.

- powerful *data abstraction*: local and global matrix transposition implicit in `DistMat` ADT; supports C and Fortran conventions equally.
- powerful *procedural abstraction*: all communication operations expressed in terms of two generic complementary operations, called *spread* and *reduce*.

3. Portability Issues: Communication

- goal: portability of *performance* across memory hierarchy parallel processors
- *spread* / *reduce* operations require row & column broadcasts and point-to-point receive-add communications for one-dimensional strided vectors
  - the 'size' of such a vector element is large (see the `AsM` workspace of the diagram above)
- ApLib interface: can use standard ApLib. Also 'stride' ApLib developed:
  - minimizes communication startup and buffer copying overheads
  - on AP1000, uses stride line sending:
    - more efficient than packing, provided overheads due to cache line size granularity $< \frac{1}{8}$
    - ring-buffer wraparound presents special challenges!
  - on AP+, uses stride PUT operation to the ring buffer
- BLACS (Basic Linear Algebra Communication Subroutines) interface also developed (but is inherently slightly less efficient)

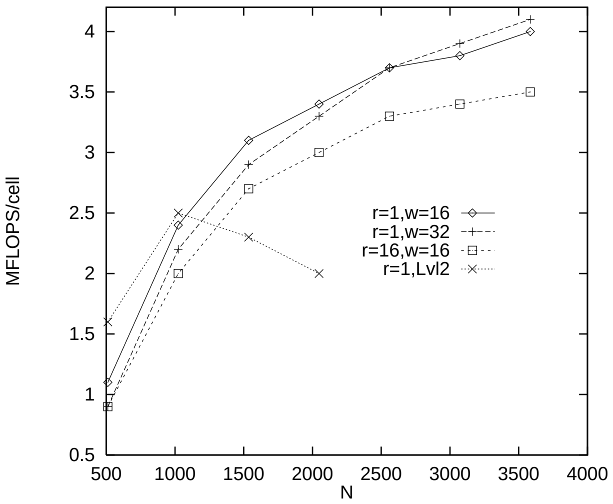
4. Portability Issues: Cell Computation Performance

- provide a standard (vendor-supplied) cell BLAS interface for the low-level routines
  - the BLAS itself cannot handle distributed triangular matrices: these must be partitioned using a set of (tunable) architecture-dependent parameters
- C versions of all low-level routines also provided:
  optimal only for cells with 16 (double precision) registers   eg. for SPARC 1–10's
- optimization for the 32-register UltraSparc required more radical techniques:
  deeper *software pipelining*, and (top-level) *cache-lookahead*.
  Results (MFLOPs) on a 170 MHz UltraSparc I (A^T, B are $K \times N$, with $K = 128$) (for the third row, partitioning methods have been applied):

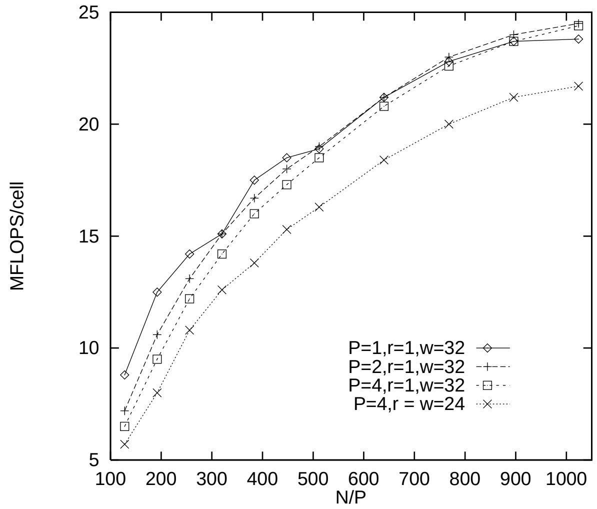
N	32	64	128	192	256	384	512
C+=A^TA	316	299	311	283	260	238	52
C+=A^TB	312	295	269	256	71	52	37
C+=A^TB			218	257	263	240	236

  shows fast but 'finicky' performance (due to memory sub-system effects)

5. Applications of the DBLAS



QR factoriz'n. on an 8×8 AP1000



LU factorization on a $P \times P$ AP+

- powerful *data abstractions* in DBLAS enables 4 of the variants of the `DDLARFB()` routine (used for QR factorization) to be expressed using a single section of code
- above results are for an $N \times N$ matrix with $r \times r$ blocks and panel width ω
- show benefits of 'distributed panels': $\omega > r$, eg. $rP|\omega$
- high performance achieved for AP/AP+; AP+ scalability demonstrated. Software overheads reducing performance for small N are still problematic