Manchester Computing
Supercomputing, Visualization & eScience

X1 ScaLAPACK Optimization

Adrian Tate

THE UNIVERSITY of MANCHESTER

Contents

- ScaLAPACK
- The need for ScaLAPACK Optimization
- X1 problems
- CAF on X1
- Changes to BLACS
- Discussion of this approach
- CAF at a higher level
- Further optimizations

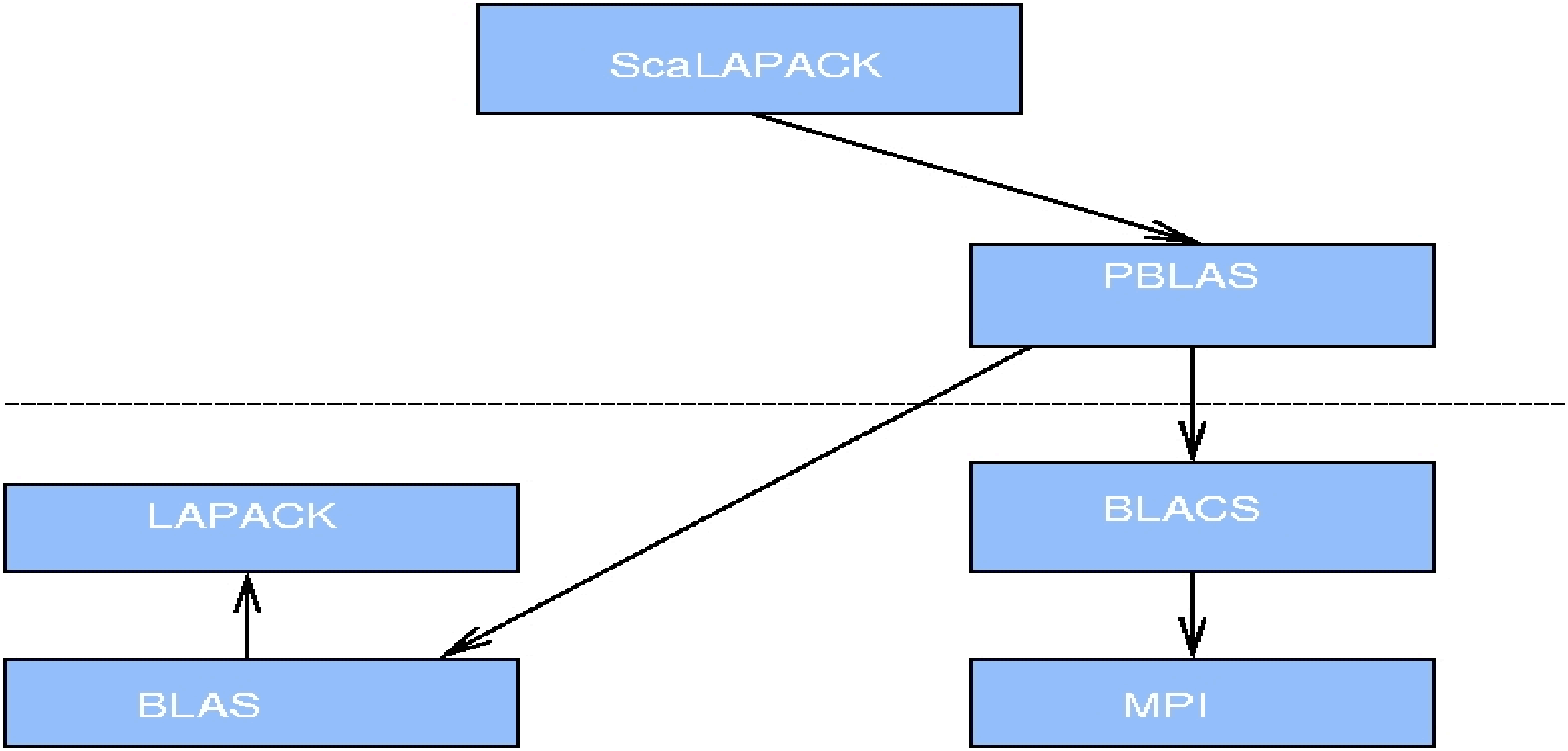
ScaLAPACK

- Parallel Dense Linear Algebra Numerical Library
- No longer funded directly, but several vendors include as a component of scientific library (Cray, SGI, Intel, IBM).
- Widely used in electro-magnetics, solid-state physics, astrophysics, climate modelling and QCD.
- Other people involved in ScaLAPACK porting, optimization and support within LibSci:
  - Mary Beth Hribar
  - John Lewis
  - Jim Hoekstra (ISU)
  - Chao Yang
- Approach - make whatever necessary alterations to ScaLAPACK to achieve good performance on X1/X1E and BW

Justification

- Distributed memory and distributed memory style programming models remain popular and are expected to remain popular
- Major architectures are DSM
- Even on SMP like systems like p690, ScaLAPACK needed.
- X1 – ratio of computation to communication is too low.
  - X1E processors will double, same network
  - Future systems, ratio will return to X1 level
- Other systems – SGI Altix more biased towards processor speed, IBM have no interconnect roadmap beyond Federation.

Software structure



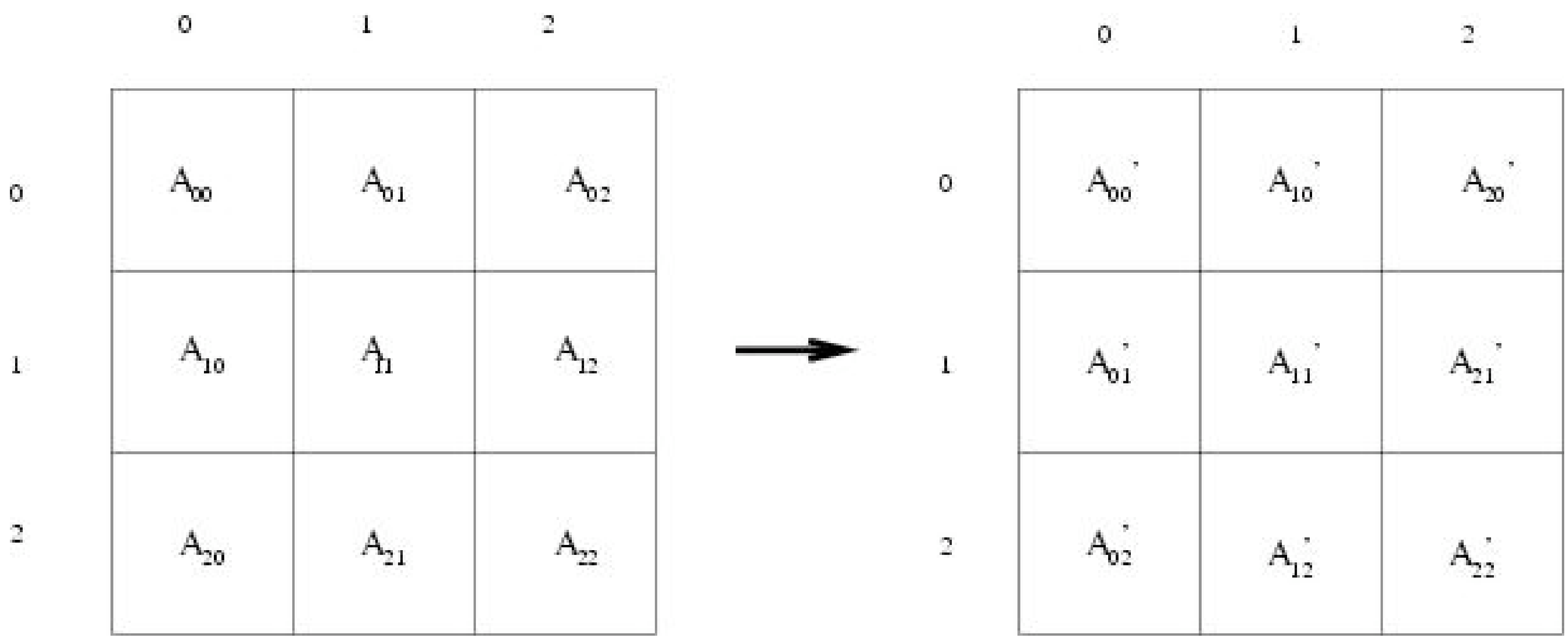
Problems

- Can get lower latency, higher bandwidth than the current MPI based comms layer gives.
- To integrate Fortran and C with MPI, many intermediate routines are called, too many function calls.
- C/C ratio low
- Leads to bottlenecks on X1.

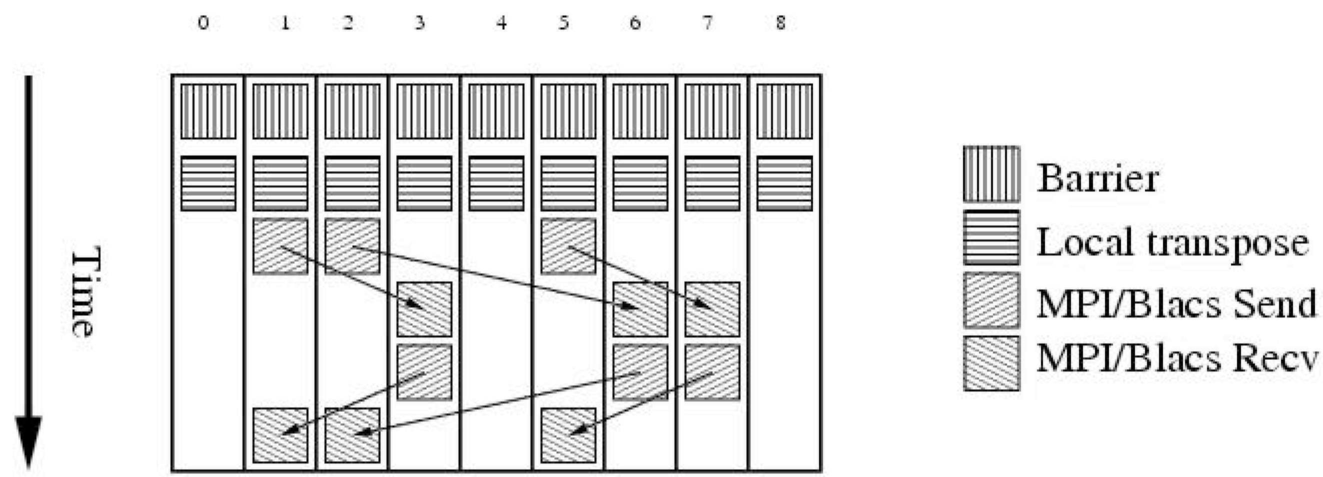
Co-array Fortran

- First step in the optimisation is to make alterations to the communications layer.
- Plan - to replace MPI with Co-array Fortran
  - One sided transfer
  - Lower latency
  - Higher bandwidth
  - No buffering
  - No function call
- First point of this list is important in itself

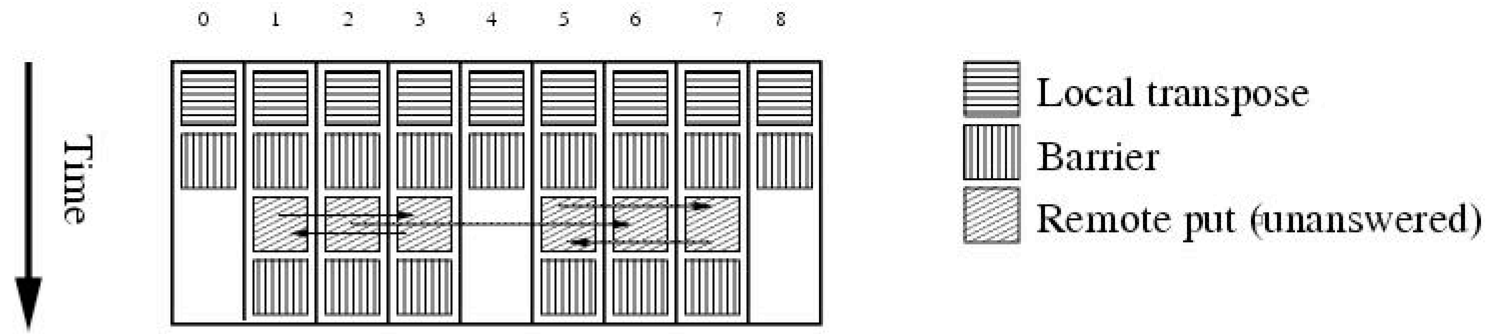
1 sided versus 2-sided. blocked parallel transpose



One sided vs 2-sided



(a)



(b)

Very simple CAF code

```
temp(:,:) = transpose(a(:,:))
call sync_all
a(:,:)[partner] = temp(:,:)
call sync_all
```

How to achieve a CAF ScaLAPACK

- We can directly replace MPI in BLACS layer
- Pass regular arrays into comms routine, use co-arrays inside.
- Can achieve this using a co-array of derived type.
  - Most powerful feature of CAF programming on X1

Using pointers to access non-symmetric memory

```
subroutine cafp ( A, C, len , dest )
type caf
real, pointer, dimension ( : , : ) :: co
end type

real :: A(*),C(len)
type (caf) :: B[*]
integer :: len,dest


B%co => A(1 : len)



call sync_all()

B[dest]%co( 1 : len ) = C(1 : len)


end subroutine
```

```
subroutine nonsymtrans(A,m,n,iam,dest)
Real :: A(len), C(len)  ,D(*)
Pointer(aptr,D)
Integer   :: iam, dest
integer*8 :: flag
call shmem_pu64(flag, loc(A), 1 , dest)
call shmem_barrier_all()
aptr = flag
flag = 0
call shmem_put(D, C, len ,dest)
end subroutine
```

(LESS POWERFUL)

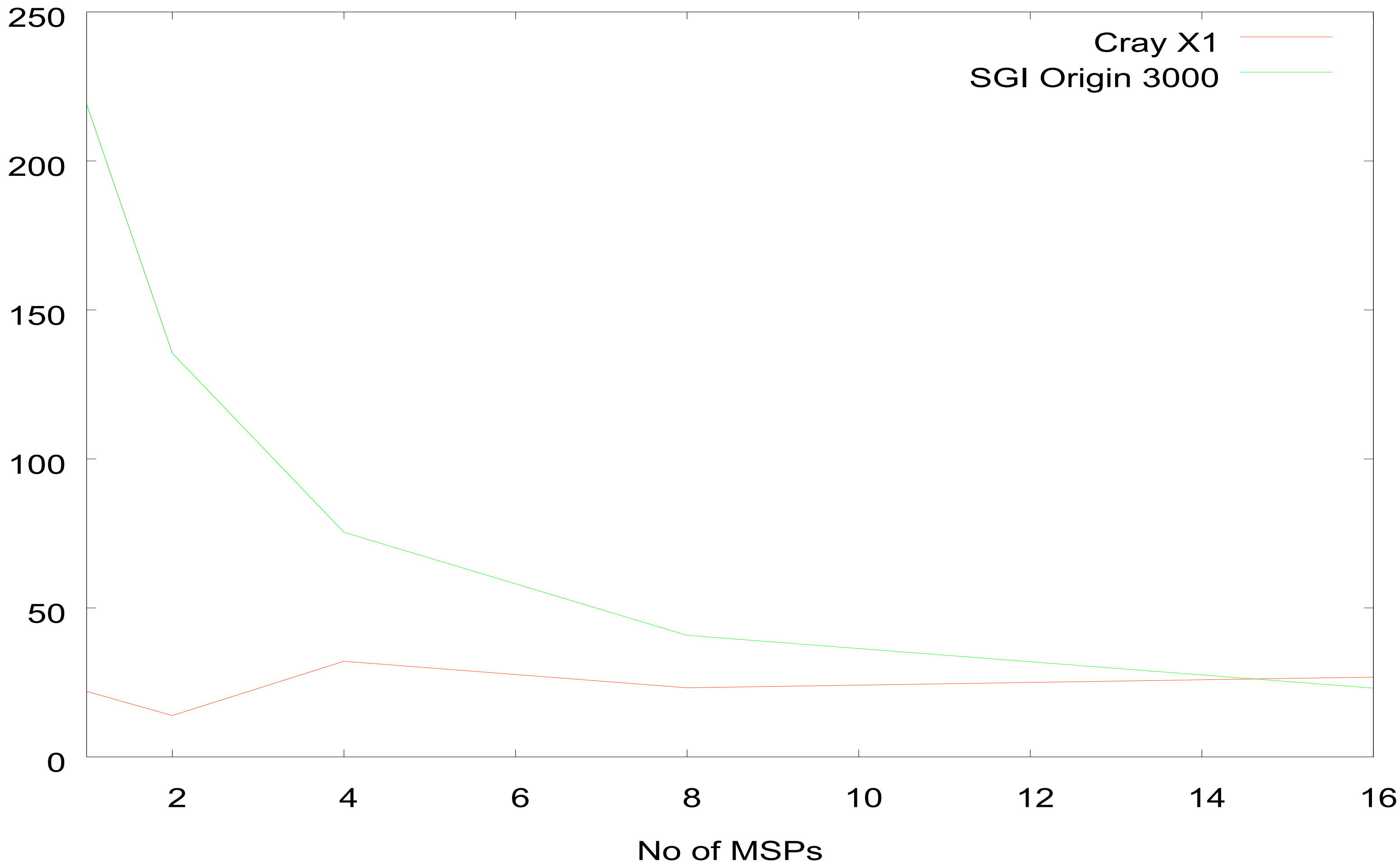
Modifying BLACS

- Improvements can be made by extending the functionality of BLACS
- pXswap routine, formally used a blacs point to point sends and receives, now replaced with a routine that performs a swap within single routine – less synchronization
- Used heavily in LU factorization
- Used CAF, with pointer method to make a CAF vector swap BLACS routine.

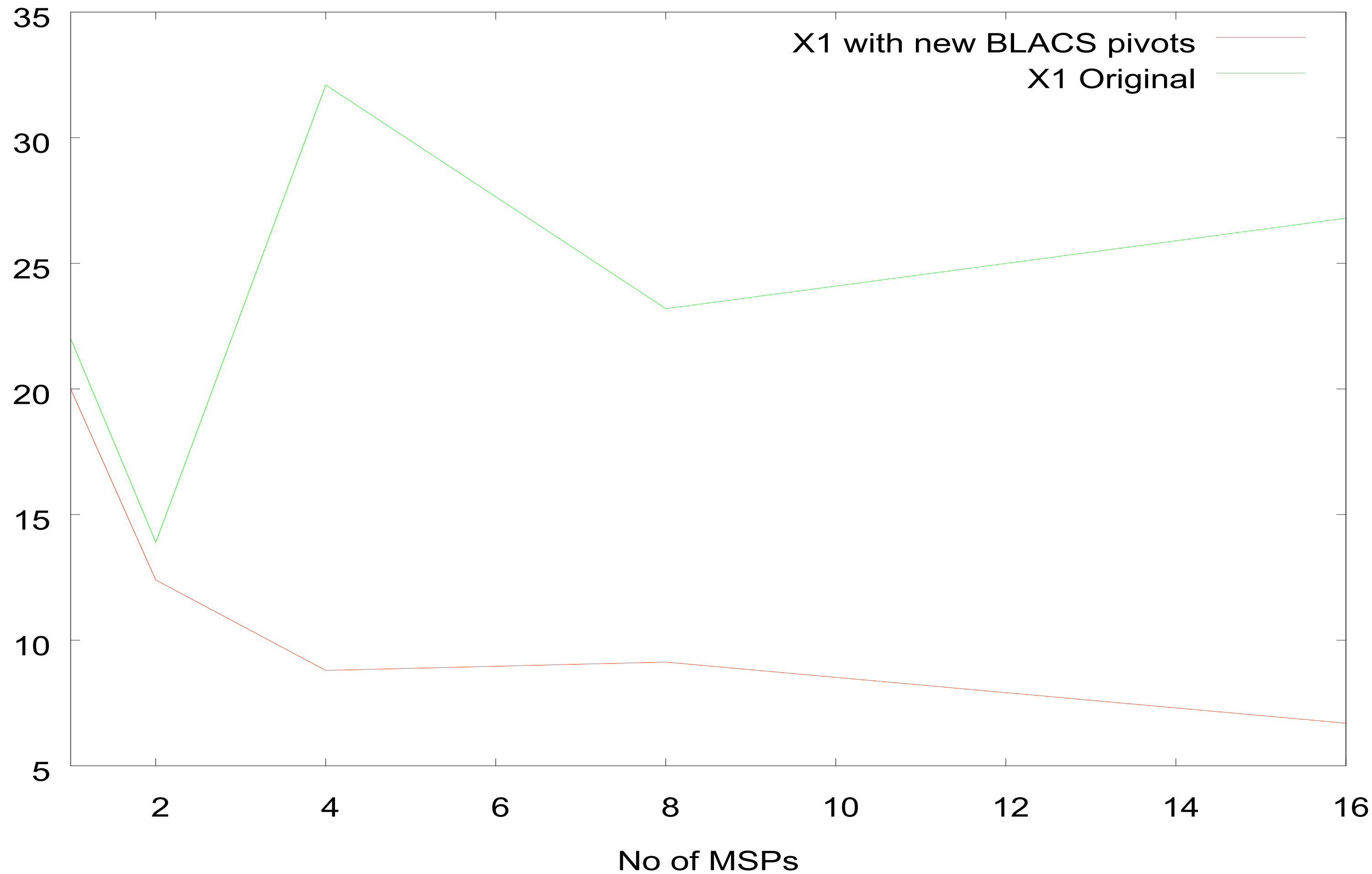
LU factorization

- Used heavily by ORNL, plus (probably) other sites.
- Shows poor performance in row pivoting area
- In addition to problems already mentioned, MPI packs and unpacks non-contiguous data into contiguous buffers, this is directly avoided in new routine.
- New BLACS CAF pivoting routine added to libsci

LU performance



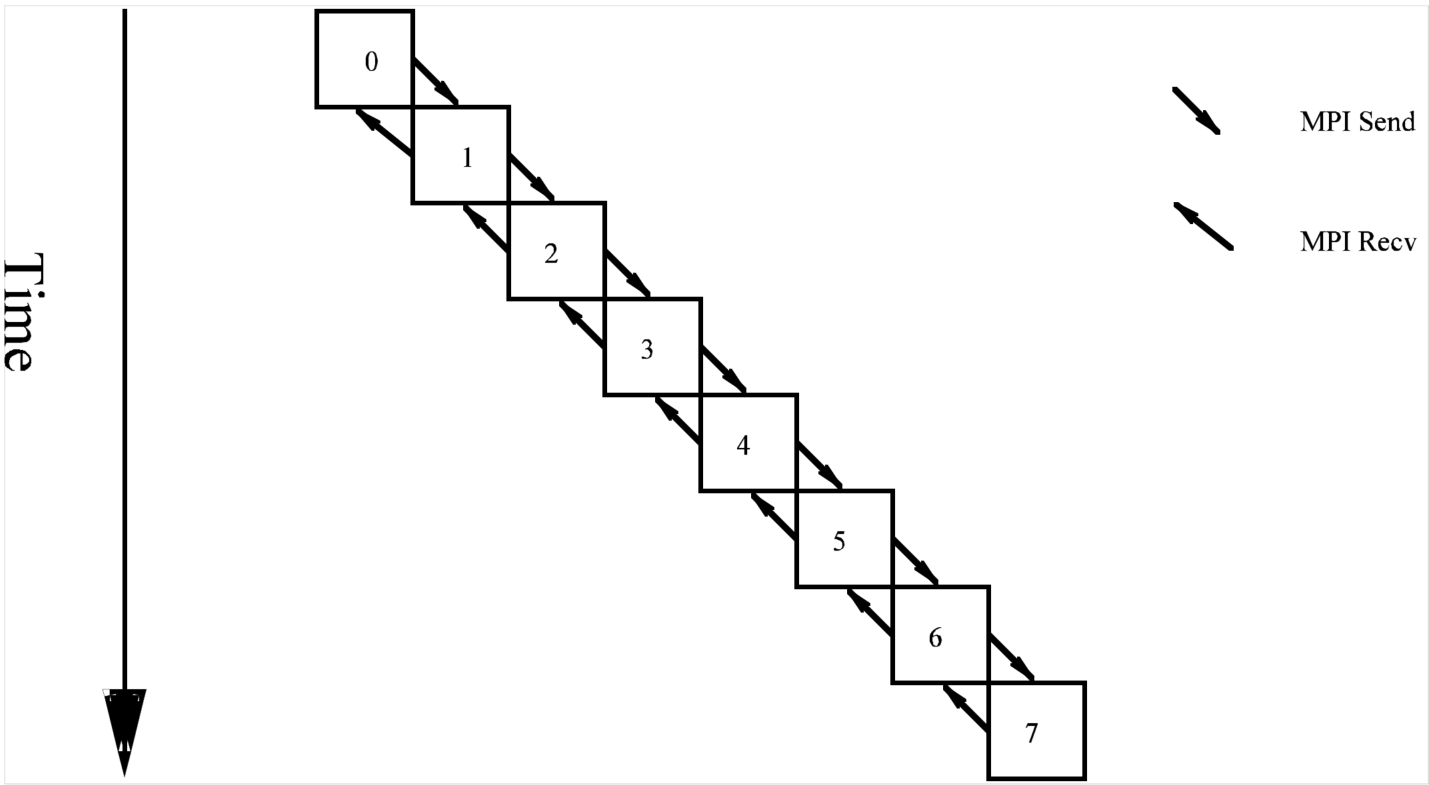
1^{st} level of Optimization



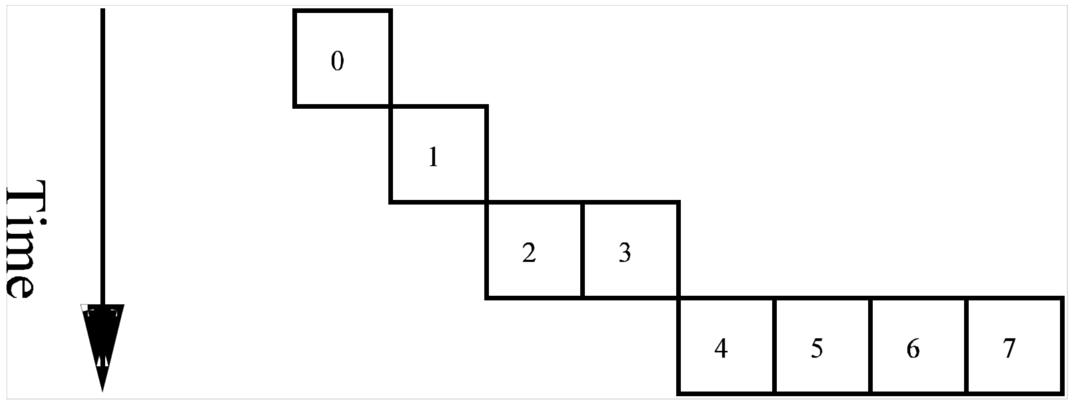
Blacs Broadcasts

- CAF Can give excellent performance for collective communications
- In a broadcast, each processor can simultaneously get the source data from the source processor.
- No memory or network contention due to intelligent memory structure of X1.
- 1st round of broadcasts came in 5.2, next set are coming soon.

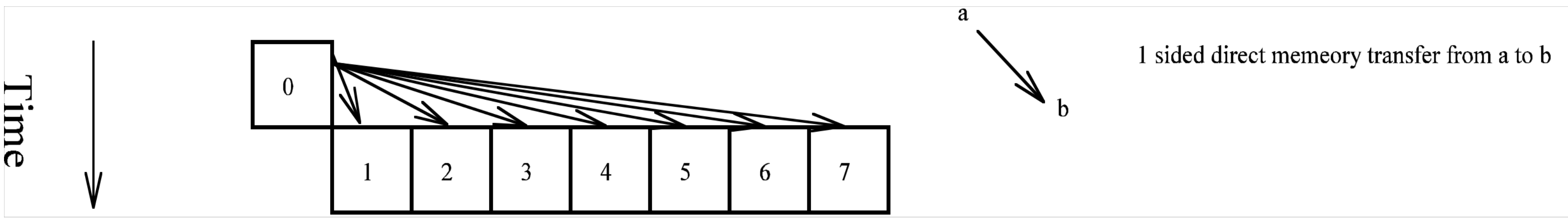
Broadcast Algorithms – ring broadcast



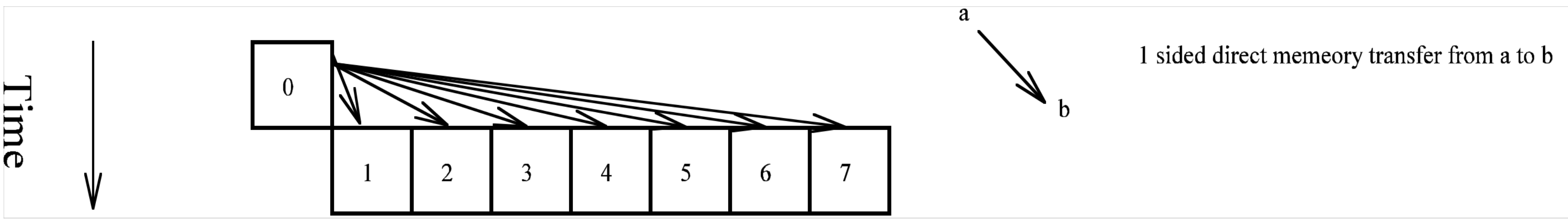
Broadcast algorithms – 1-tree



Broadcasts with one-sided



Broadcasts with one-sided



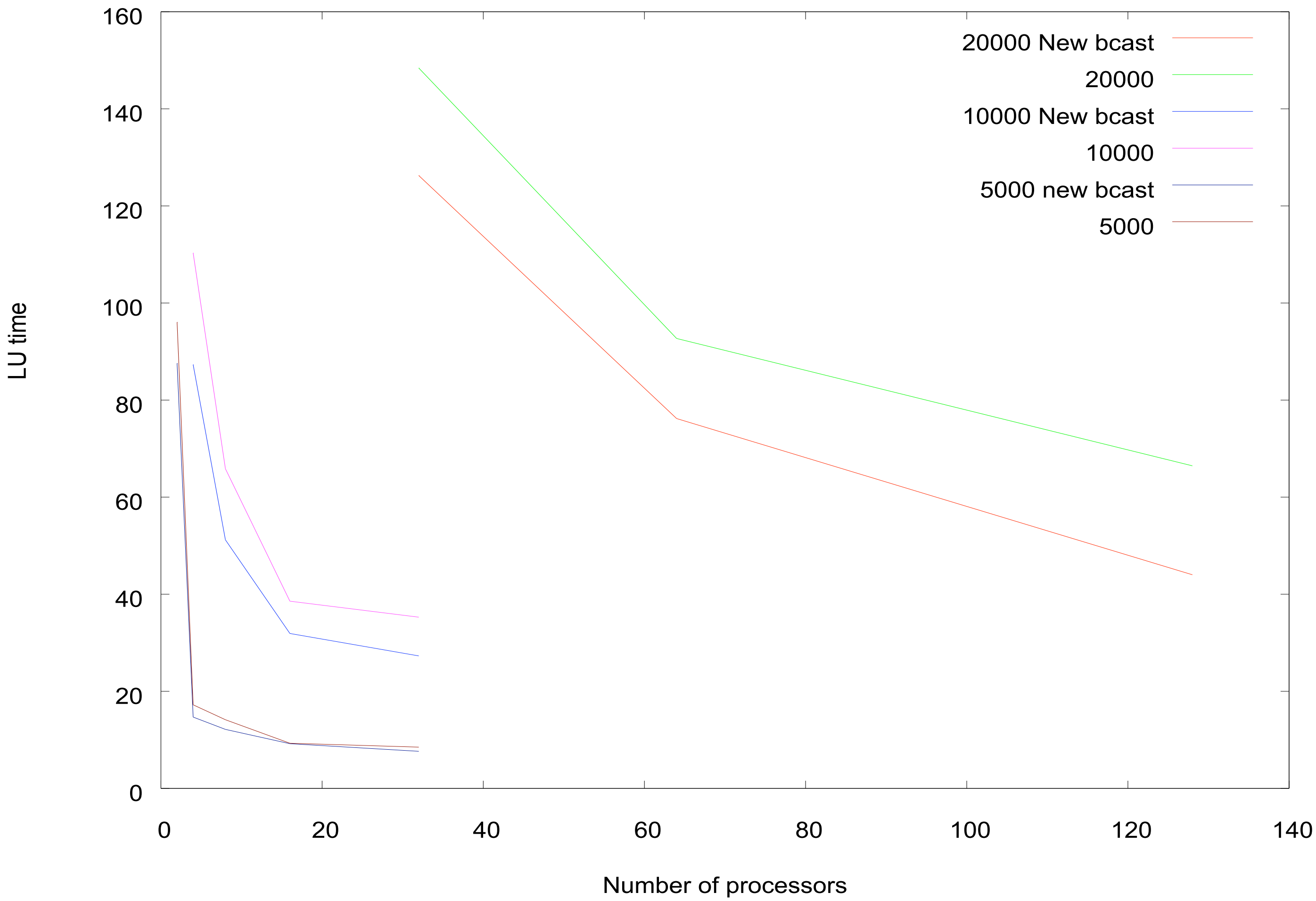
Direct Broadcasts

- Requires an intelligent memory system that can allow each processor to make simultaneous copies.
- Also requires intelligent interconnect technology, since there is potential for a bottleneck.

- Paul Burton, Bob Carruthers, Greg Fischer, Brian Johnson and Robert Numrich *Converting the Halo-Update Subroutine in the MET Office Unified model to Co-array Fortran,* ECMWF World Scientific, January 2001.

- Expect to perform much better, especially at high process counts (e.g 64 processors doing an ‘All’ broadcast’)

Broadcast performance



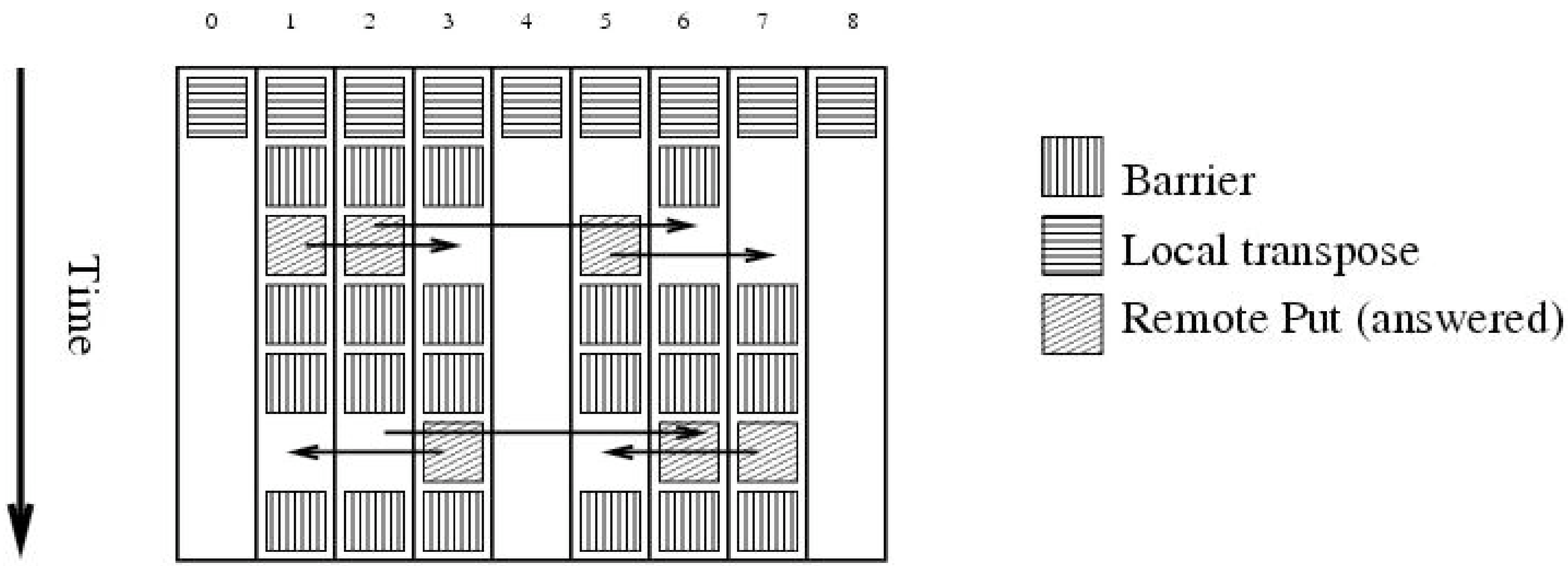
Troubles

- Important information in the BLACS is stored in external C structures that are not easily accessible from the new Fortran90 routines.
  - Needed to develop a mechanism for information sharing
  - Needed to make several changes to Blacs grid initialization routines to support this
  - Fortran 2003 allows interoperability between C structures and Fortran derived types
- Other problems held up bug fixes and prolonged development.

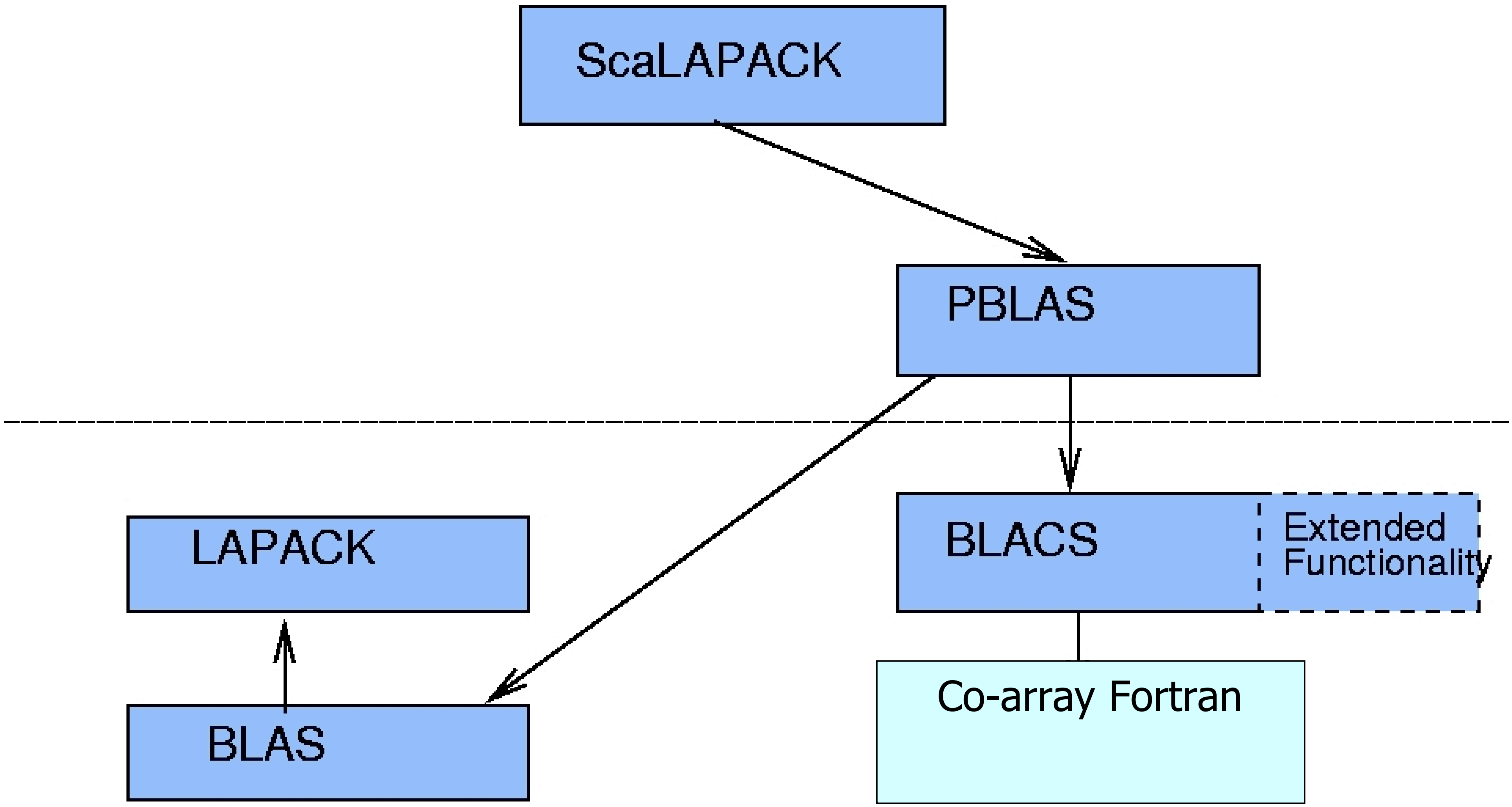
CAF ScaLAPACK

- This idea of having CAF inside communications routines is not ideal
  1) Much of BLACS code is made redundant
  2) Higher function call count
  3) Pointer method inefficiency (?)
  4) Current PBLAS algorithms are written for 1-sided communications
     - Consider the same blocked transpose, where we make direct, generic replacements to BLACS.

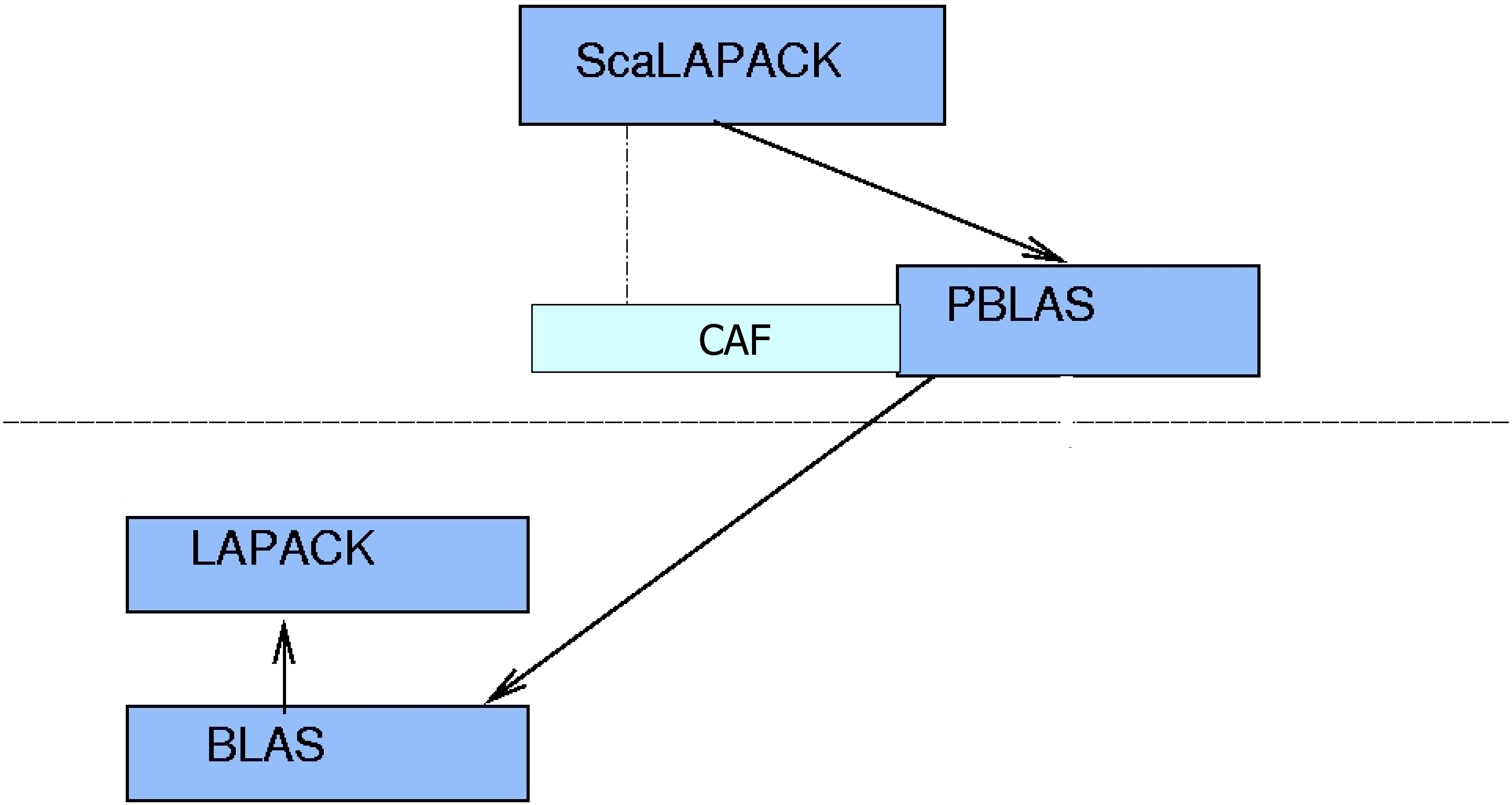
Transpose Example



Optimised software structure



Proposed software structure



Important Questions

- Is the pointer method actually less efficient than passing co-arrays?
- Are there other reasons why we might want to change to new structure?

Important Questions

- Is the pointer method actually less efficient than passing co-arrays?

Sometimes…

- Are there other reasons why we might want to change to new structure?

Important Questions

- Is the pointer method actually less efficient than passing co-arrays?

  Sometimes

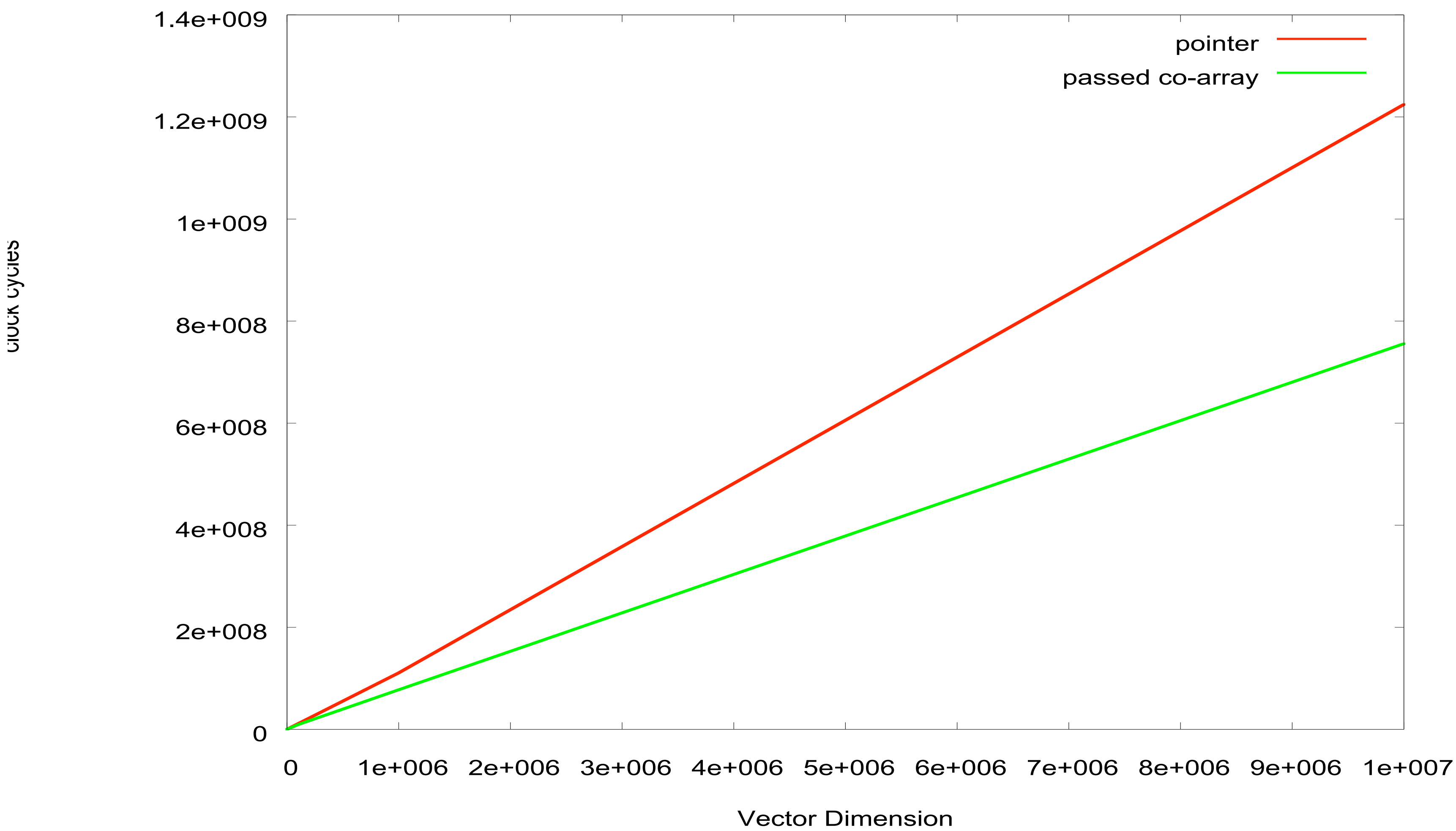
- Are there other reasons why we might want to change to new structure?

  Maybe…

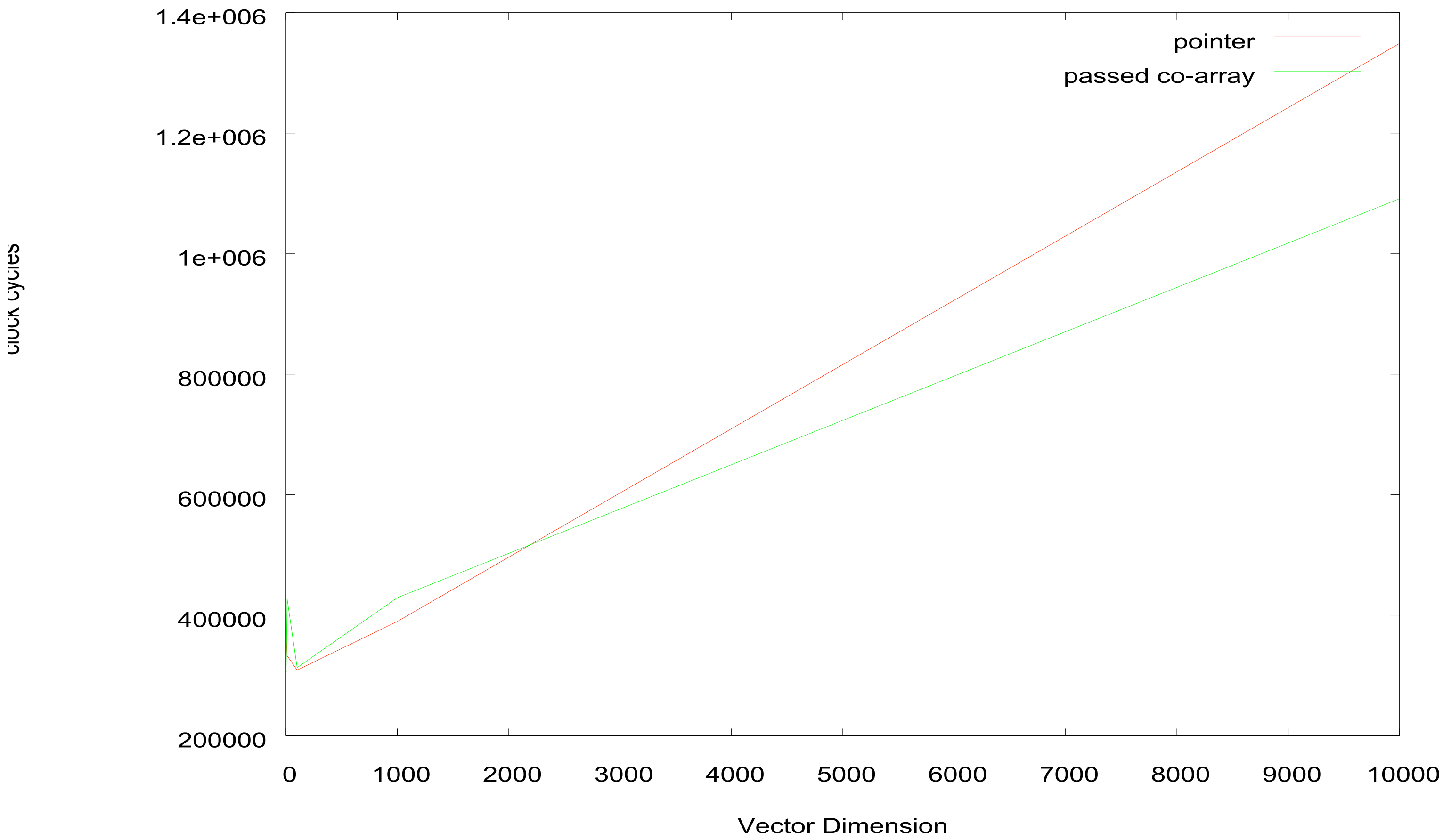
Testing pointer method

- Test code – uses CAF to perform a series of blocked transposes in three ways
- Case 1 = Co-array real argument and co-array dummy argument
- Case 2 = Co-array pointer method

Results of Pointer method test 16 MSPs



Smaller vectors



Expense

- Is it not vector dimension being passed that is the problem, but the number of array references being made.
  - Referencing a pointer is slower than referencing an array directly.
  - Repeated tests with number of references to data being constant.
    - Pointer method was slower but at a constant rate
- Can deduce two things from this
  - Each call to the pointer method involves some additional cost
    - Cost of pointer assign
  - Expense of using pointer method is related to number of array accesses
- In BLACS do we need to make many array references?
  - Even though we are only transferring data, we make array references, since we need to designate array sections (i.e. A(1: lda))
  - Sometimes need to transpose

Expense within BLACS

- Primarily though, these routines are for communication only, and shouldn't need to perform many operations.
- Unless block size is very big, it is unlikely that the overhead is going to hurt too much.
- For 64x64 block size, if address of every 4096 array elements had to be calculated individually, we don't expect a crippling loss of performance.

Further Questions

- If we make higher level changes to make ScaLAPACK arrays co-arrays, can we allow them to passed through the PBLAS 'unharmed'
  - Theoretically, yes
  - CAF interoperability will need to be improved before we can comfortably achieve this.
- Should we just re-write PBLAS in UPC? (or in Fortran and CAF?)
  - Big job.

Conclusions

- There is not sufficient overhead from pointer to warrant a re-write of PBLAS layer,
- Also, the uncertainty in mixing with C, and amount of effort in rewriting PBLAS.
- so for now, keep BLACS with imbedded CAF.

- We can still -
  - replace all MPI calls, except those that are not likely to be within loops (grid initialization etc).
  - Look for areas where 2 sided pattern is being assumed and make changes at PBLAS layer.
  - Strip away redundant code and interfaces

Additional Optimizations

- Optimal Blocking factors
- Effect of ScaLAPACK blocking factor on LAPACK blocking factor and LDA.
  - X1 gives varying performance for block sizes and leading dimensions for BLAS
  - we may want to remove the dependence of leading dimension on distribution blocking factor
  - Can we introduce a more dynamic system?
- Customer driven, routine specific optimisations.
- Address user interface.
- Parallel libraries in Cascade