Joint Laboratory for Petascale Computation

INRIA NCSA

Update on Libraries for Blue Waters

William Gropp
www.cs.illinois.edu/~wgropp

Outline

- Summary of Current Work on Libraries for Blue Waters
  - CSR updates (from paper, now accepted)
    - In particular, parallel performance
    - PETSc updates results
  - IO Tests
  - Nonblocking Allreduce in CG
- Throughout Collaboration Opportunities

Joint Laboratory for Petascale Computation

Some Libraries Requested by Applications Planning on Using Blue Waters

Required Scientific Libraries
FFTW
Hypre, P3DFFT
FFTW, HDF5, GSL, P3DFFT
BLAS, LAPACK, ScaLAPACK
P3DFFT, FFTW, ESSL
pnetCDF
FFTW, ESSL, LAPACK, boost, SPRNG
FFTW, ESSL, LAPACK
PETSc, GSL, FFTW
HDF5
BLAS, LAPACK, EINSPLINE, boost, HDF5

- Organized by PRAC application
- Not all have responded
- Usual suspects: FFT, Matrix-matrix multiply
- I/O libraries included

New Sparse Matrix Formats

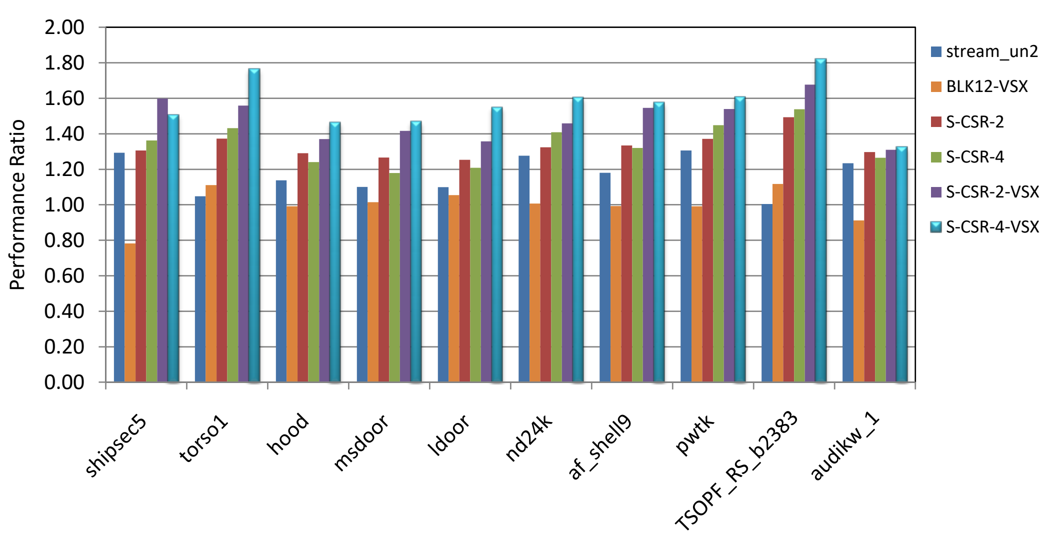
- Sparse matrix operations are memory-bandwidth bound
- Conventional formats do not exploit prefetch hardware (particularly on Power architecture)
- We have developed new “streamed” formats as variations of compressed sparse row (CSR)
- This updates results presented at previous INRIA-Illinois Joint Lab Workshops

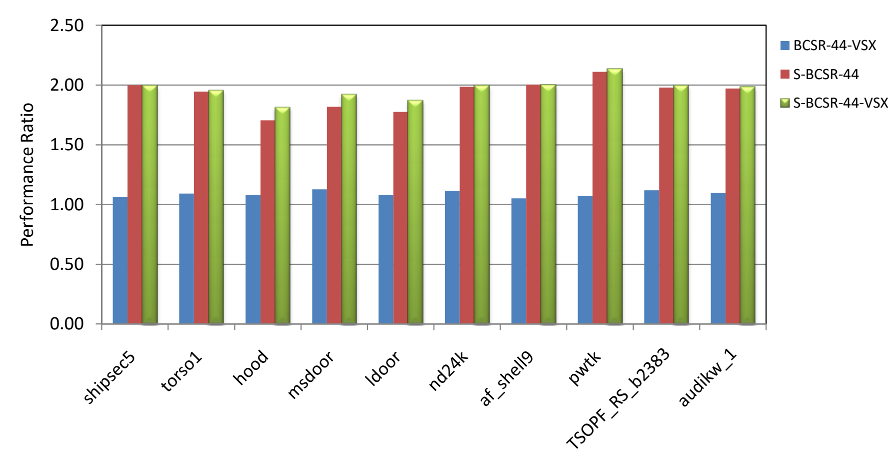
Progress in 2010

- New implementations
  - Exploit the Power7 VSX (double precision vector) instructions
  - Comparisons when all threads executing SpMV
  - Add SSOR sweep (for preconditioning) and Sparse LU triangular solve (ditto)
  - Integration into PETSc

SpMV - Serial Run on Power7

- The tests are on a P7 machine (BlueBiou in Rice University).
- The streamed format S-CSR-2/S-CSR-4 with VSX intrinsic functions achieves 30 – 80% performance improvement compared to the traditional CSR format.
- The streamed 4x4 blocked format achieves 70 to over 100% improvement over the BCSR 4x4 format.
- Hand-written VSX code for S-BCSR-4 format doesn't help much.

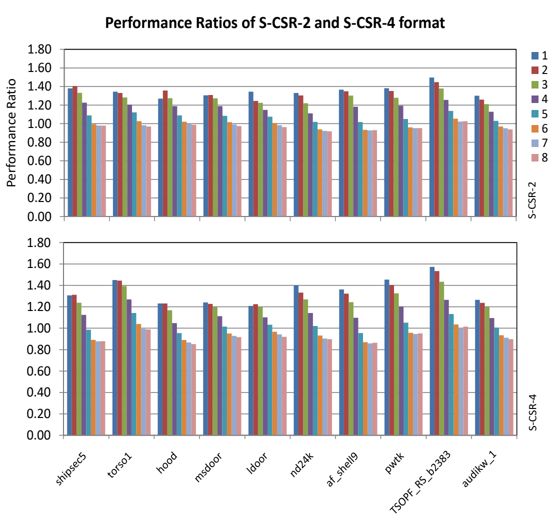


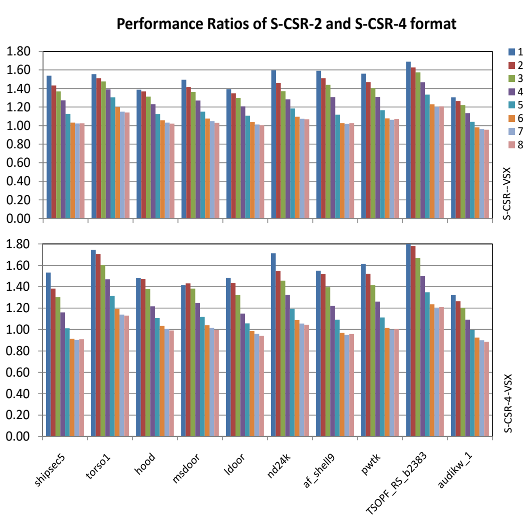


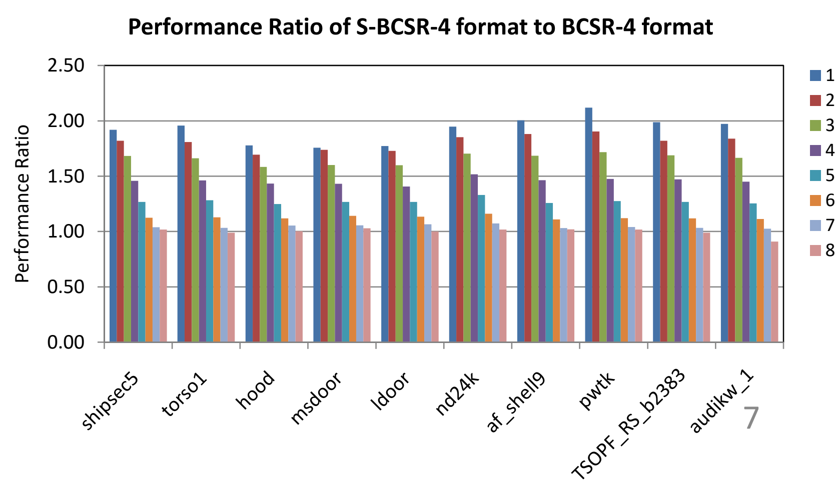
SpMV--- Multi-Thread Tests on P7

The thread number changes from one to eight on the same chip. Every thread uses an individual core and computes a complete SpMV.

- The streamed format still achieves over 20% improvement for most of the sparse matrices when four or five cores on the same chip are employed.
- The performance ratios of the streamed formats degrade when the thread number increases, because
  - the L3 cache is totally 32 MB for the eight cores, and it can be easily used up when more and more data is prefetched into the cache.
  - There is only one memory controller on the chip. Too many prefetch requests exhaust the load channels.
- The multi-thread tests on the P7 MR machine show better results, since there are two memory



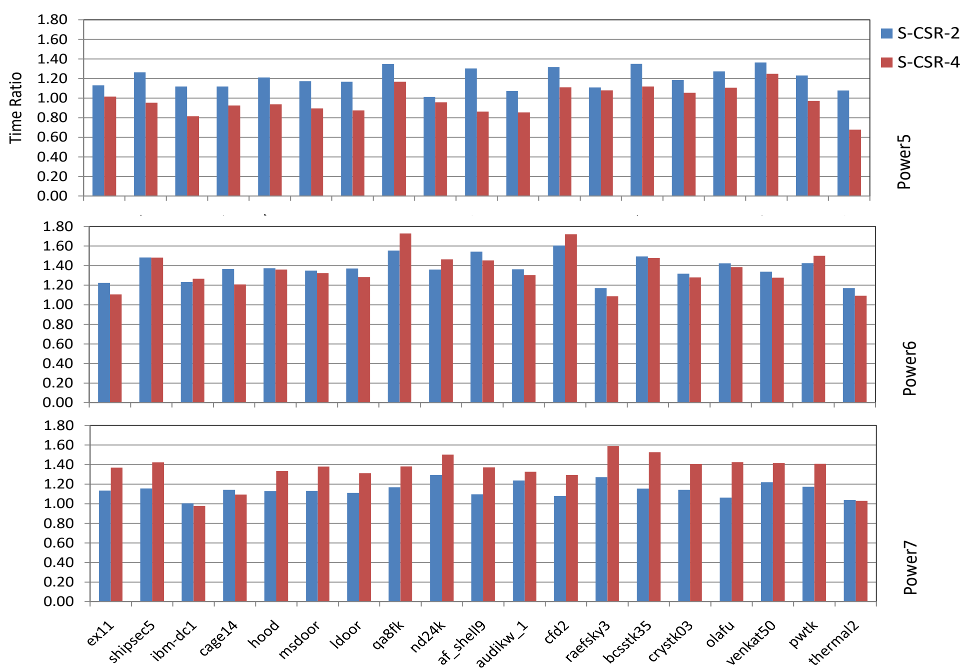


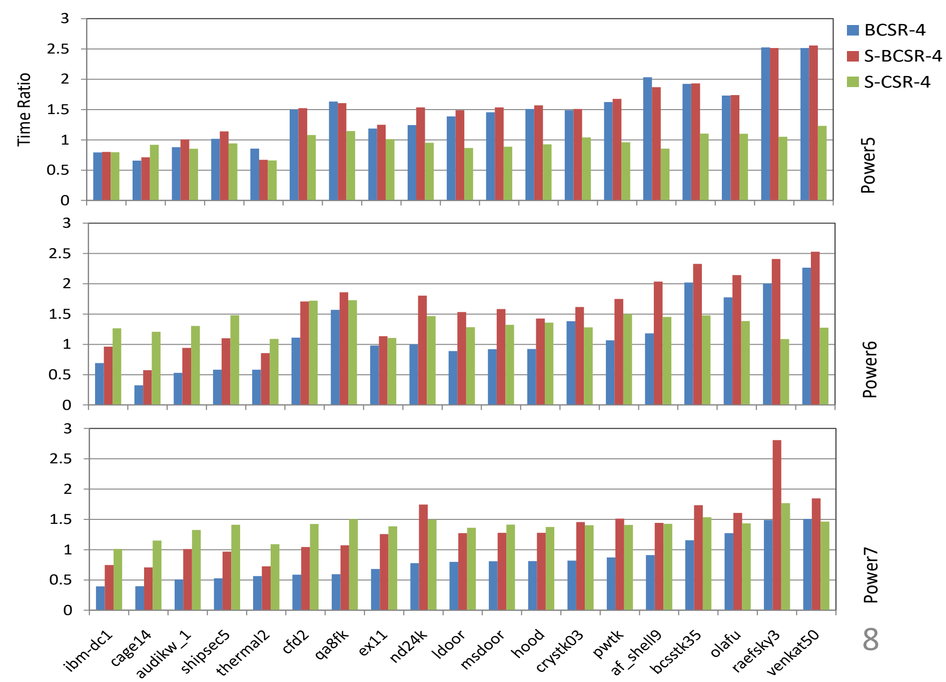


SSOR iteration test results

Either the S-CSR-2 or the S-CSR-4 format is better than the CSR format, the time ratio between them can achieves as much as 1.30 on Power5, 1.70 on Power6, and 1.55 on Power7.

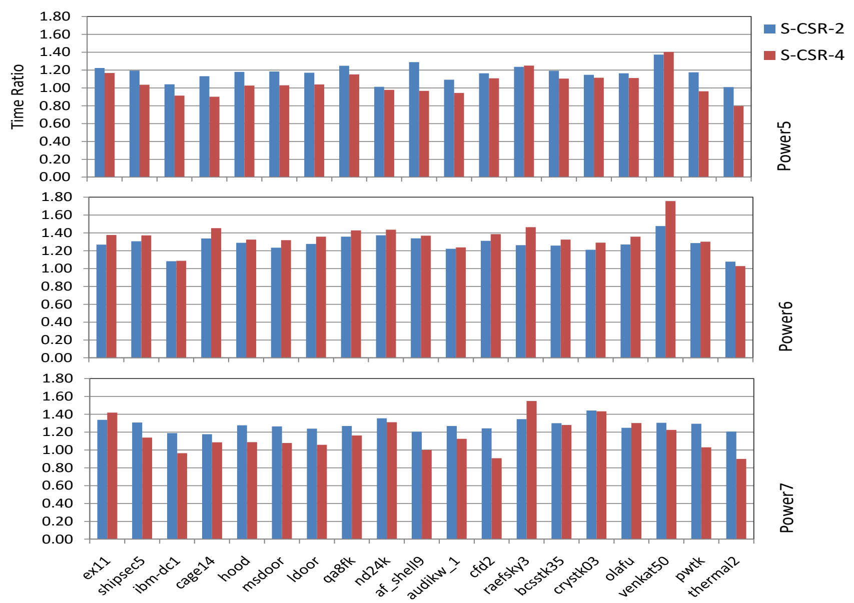
- The S-BCSR-4 format is significantly better than the BCSR-4 format on Power6 and Power7. The ratios can be over 1.80 on Power6, and 2.50 on Power7. On Power5, it is around 1.10 ~ 1.25 for about half of the matrices.

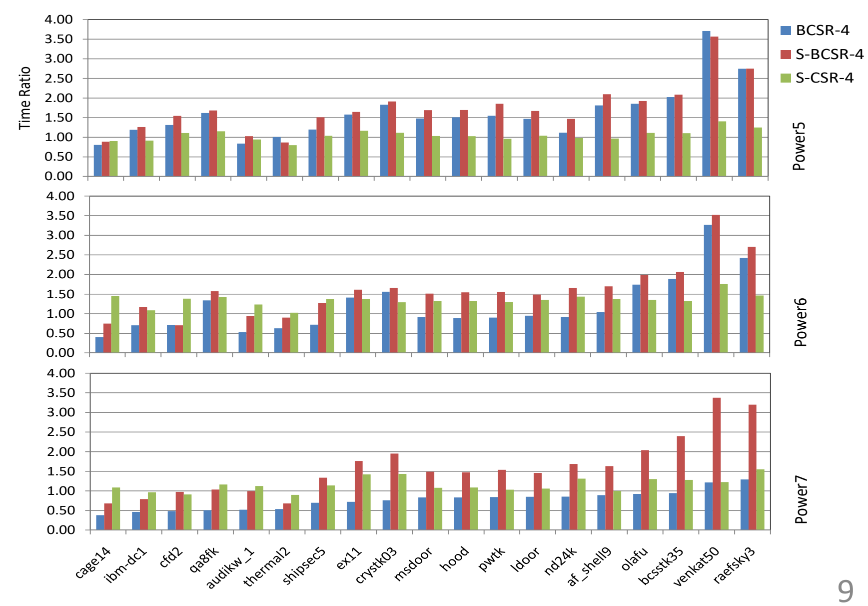




LU Triangular Solver Test Results

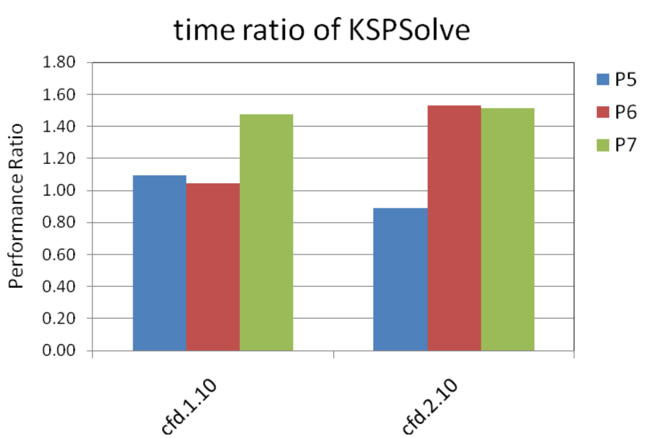
- Either the S-CSR-2 or the S-CSR-4 format is better than the CSR format, the time ratio between them can achieve at most 1.40 on Power5, 1.76 on Power6, and 1.55 on Power7.
- The S-BCSR-4 format is significantly better than the BCSR-4 format on Power6 and Power7. The ratios between them can be over 1.85 on Power6, and 2.78 on Power7. On Power5, it is around 1.10 ~ 1.25 for about half of the matrices.

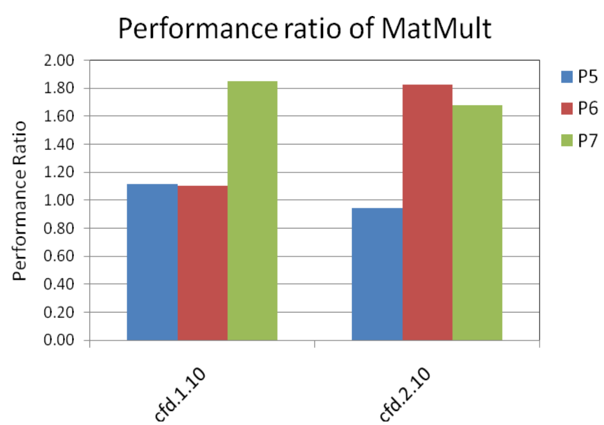


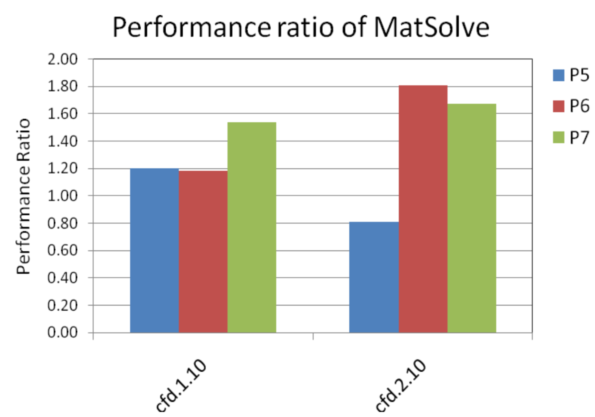


The streamed format with PETSc

- PETSc is a popular scientific computing package in the HPC community, developed at ANL
- The streamed format are implemented for MatMult, MatSolve, etc.
- Two matrices, cfd.1.10 (small) and cfd.2.10 (large), are tested on P5, P6 and P7 chips.
- The streamed format results in significant performance benefit on P6 and P7, although it doesn't achieve much on P5.



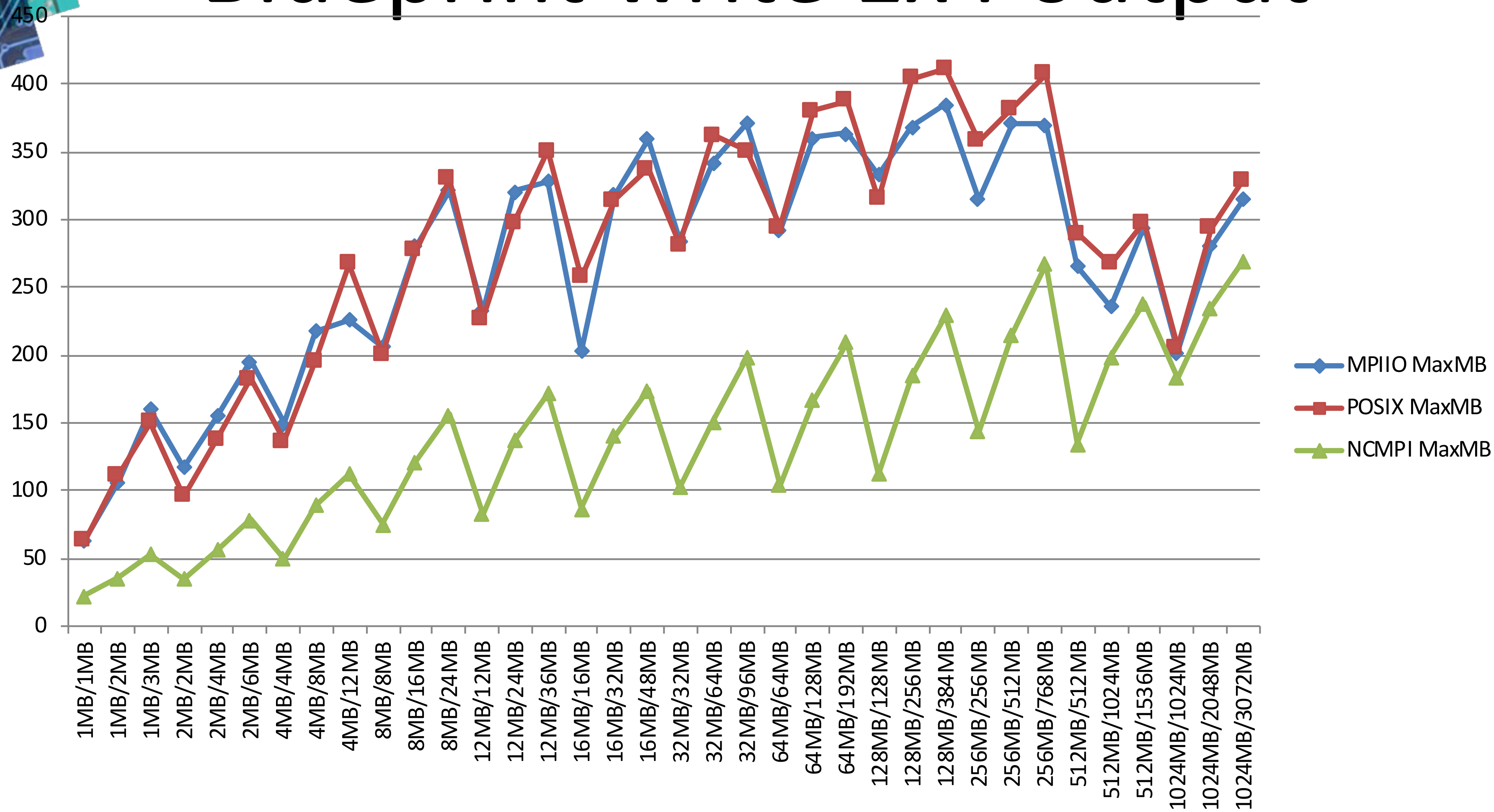




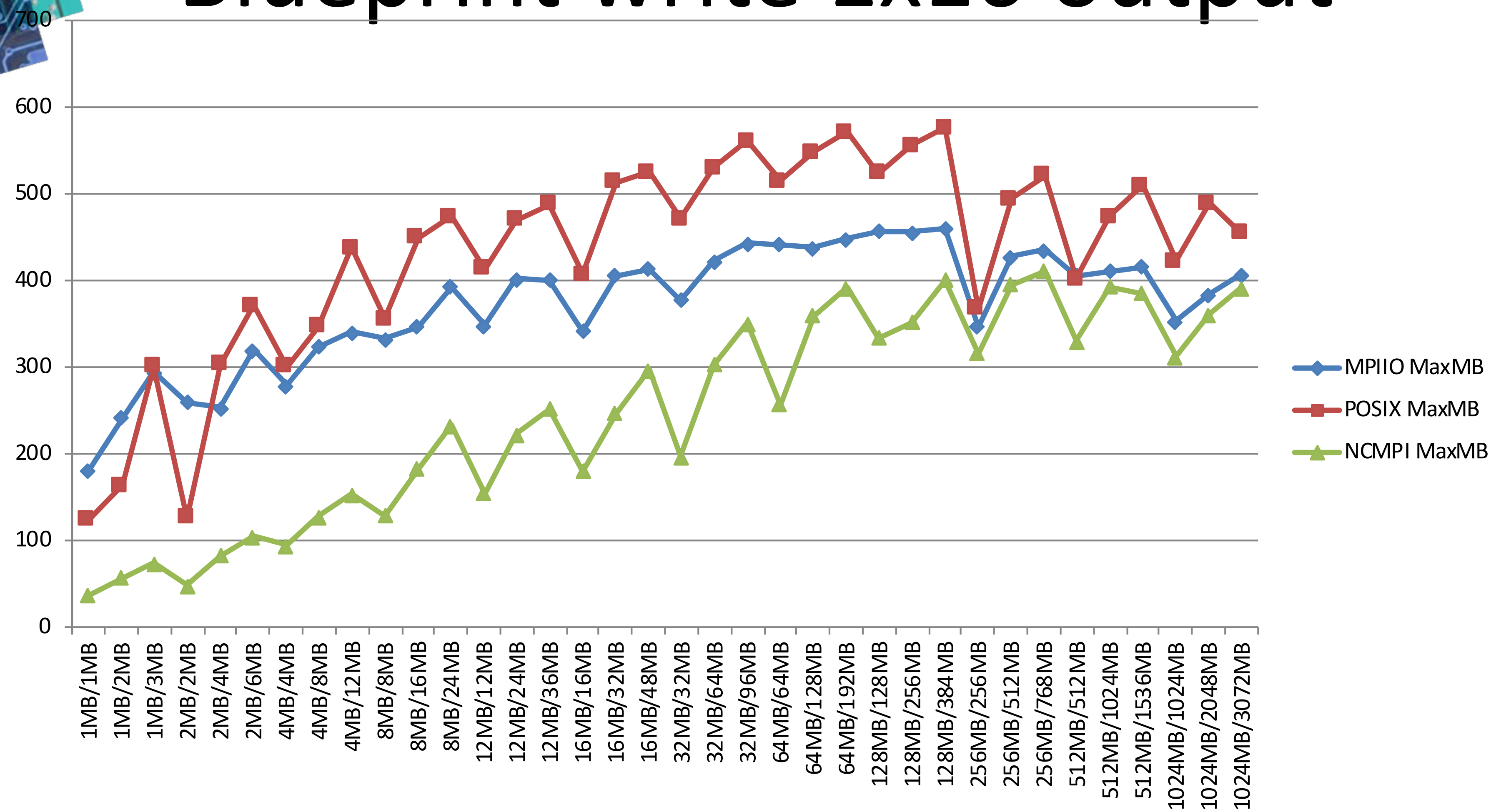
I/O Library Tuning

- I/O performance is a key problem for some applications, particularly at scale
- Applications are using HDF5 and pnetCDF, as well as their own
- Currently evaluating the performance of Parallel NetCDF and Parallel HDF5 vs MPI-IO and POSIX, using the standard IOR (interleaved or random) tool version 2.10.2.
- Note both pnetCDF and HDF5 use MPI-IO; pnetCDF is a thin layer over MPI-IO
  - pnetCDF performance should be very close to MPI-IO
  - MPI-IO semantics less constraining than POSIX implies performance should be no worse
  - Not what we see on our Power5 (Blueprint) system
  - Indicates implementation inefficiencies in libraries
  - These sorts of tests are needed in any serious library development

Blueprint write 1x4 output



Blueprint write 1x16 output



Conjugate Gradient Alternatives

- Problem statement
  - Conjugate Gradient requires that computation of a dot product whose result is used in the immediately following step
  - In parallel computing, this introduces a costly synchronizing operation
- Relationship to MILC
  - A CG solve is a key part of MILC; performance estimates suggest that the dot product will be a significant cost at the scale of Blue Waters
- Example solution
  - Rearrange the computation of the dot product to allow it to overlap the matrix products (at the cost of extra floating point work but no communication)

The Conjugate Gradient Algorithm

```
while norm(r)/norm(b) >tol && iter<maxiters
    iter = iter+1;
    Ap = A*p;
    alpha = rtr / (r'*Ap);
    x = x + alpha*p;
    r = r - alpha*Ap;
    rtr_old = rtr;
    rtr = r'*r;
    beta = rtr/rtr_old; %% <rnew,rnew> / <r,r>
    p = r + beta*p;
end
```

Stability of the Conjugate Gradient

- There are different ways to implement the CG (conjugate gradient) method
- These ways depend on properties of the algorithm and formula derivations, and will affect the stability of the algorithm: *accuracy is lost due to rounding error accumulation when orthogonalizing at each step of the iteration*
- The following three of different implementations of CG differ in the computation of α, or the improvement at the step

Stability of the Conjugate Gradient (cont.)

- Method 1:
  - The residual, r, and the search direction, p, are orthogonal to each other
  - Then, the improvement at the step is:

$$\alpha = \frac{< r, r >}{< p, Ap >}$$

where <,> stands for the inner product of two vectors

Stability of the Conjugate Gradient (cont.)

- Method 2:
  - The residuals are orthogonal to each other
  - This is also presented in Saad's book *Iterative methods for sparse linear systems (2nd edition)*

$$\alpha = \frac{< r, r >}{< r, Ap >}$$

  - This is also the most stable method

Stability of the Conjugate Gradient (cont.)

- Method 3:
  - Like in Method 1, except further simplifications are done to reduce the improvement at this step to the following form:

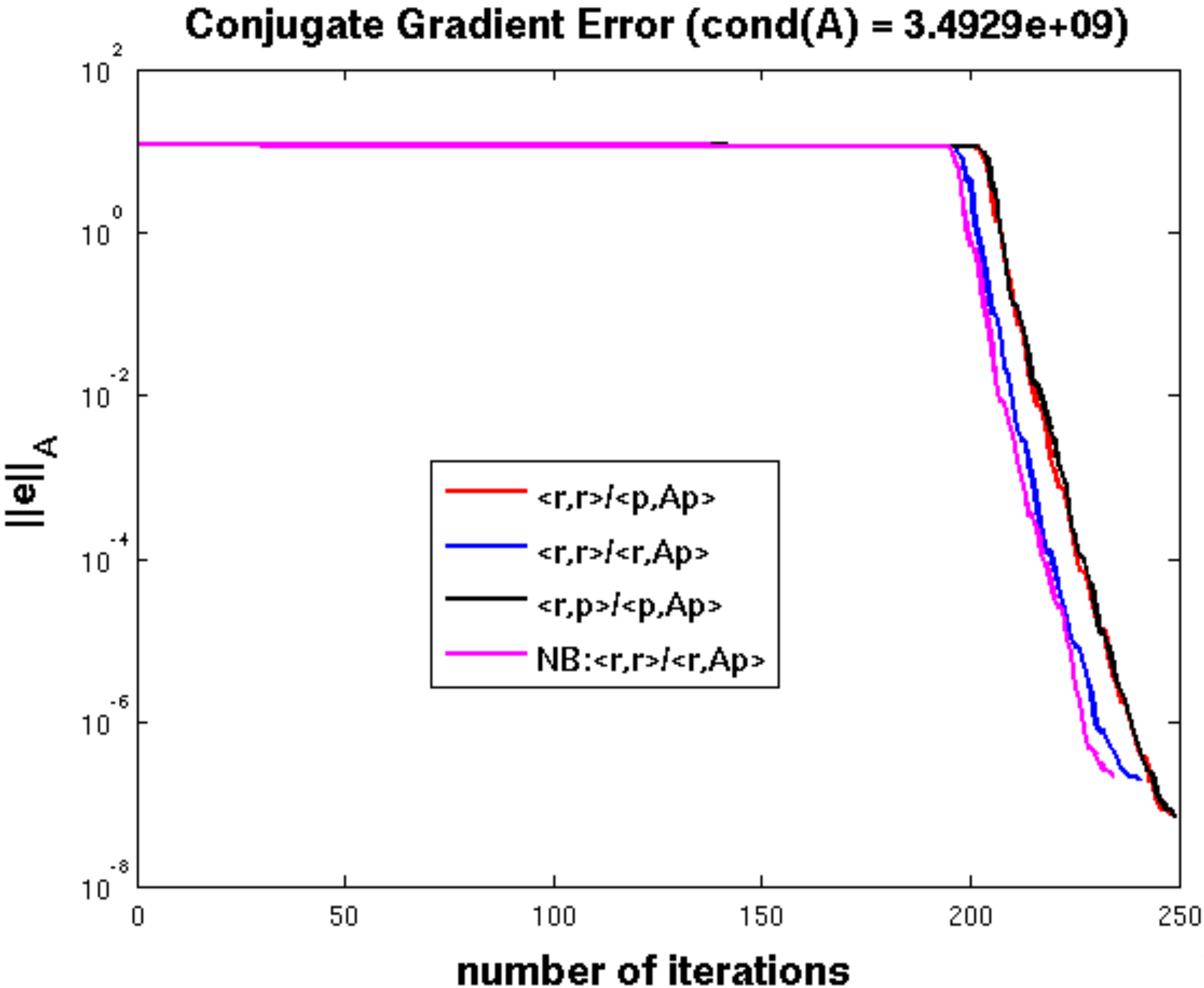
$$\alpha = \frac{< p, r >}{< p, Ap >}$$

  - Slightly better than Method 1, but worse than Method 2

A non-blocking version of Method 2

```
while norm(r)/norm(b) >tol && iter<maxiters
  iter = iter+1;
  Ap = A*p;
  s = Z + beta*s;    % Could start t=r'*s here
  S = M * s; % M is the identity in this example
  t = r'*s; % the inner product would call MPI_Iallreduce; the local work would then be completed
                    %and an MPI_Wait call would wait for the communication to finish
  alpha = rtr / t;
  x = x + alpha*p;
  r = r - alpha*s; %r - alpha*Ap;
  z = z - alpha*S; % Could start  rtr = r'*z here
  Z = A*z;
  rtr_old = rtr;
  rtr = r'*z;
  beta = rtr/rtr_old;
  p = z + beta*p;
end
```

This approach requires additional daxpy computations, but overlaps dot products with other work and communication


Conjugate Gradient Error (cond(A) = 3.4929e+09)
10^2
10^0
10^-2
10^-4
10^-6
10^-8
||e||_A
0
50
100
150
200
250
number of iterations
<r,r>/<p,Ap>
<r,r>/<r,Ap>
<r,p>/<p,Ap>
NB:<r,r>/<r,Ap>

Summary

- Existing algorithms can be re-arranged to better exploit memory hierarchy
- Consistency tests reveal performance problems in library implementations
  - Can also be applied *within* a library
- Trading arithmetic operations for more communication overlap *may* provide better scaling and performance