Parallel SBR: a PLAPACK based PRISM kernel

Yuan-Jye Jason Wu
Christian H. Bischof
Argonne National Laboratory, IL

Philip A. Alpatov
Robert A. van de Geijn
The University of Texas at Austin

This work is part of the PRISM project and in part supported by DARPA grant P-95006.

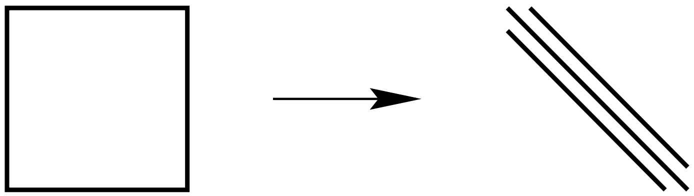
Presentation Outline

- Background about Successive Band Reduction (SBR)
- Full-to-Band Algorithm
- PLAPACK Based Full-to-Band Routines
- Performance
- Data Redistribution
- Conclusions and Future Work

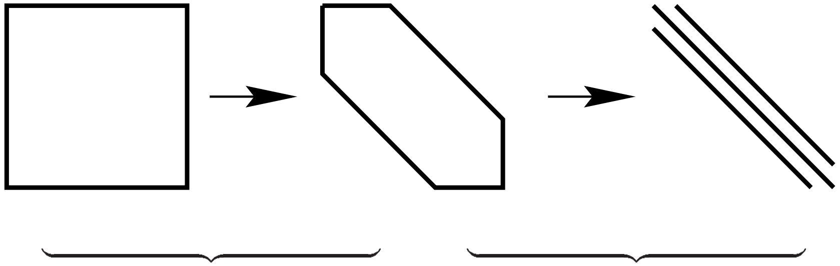
Background about SBR

Tridiagonalization:

- One key step in symmetric eigensolver
- Existing parallel packages: ScaLAPACK, HJS, SBNDRD
- Traditional tridiagonalization:



- (Two-steps) Successive band reduction: (Bischof and Sun, 1992)



SBNDRD TRIDRD

Data Redistribution

Observations on existing packages:

- Draw on low-level message-passing primitives and a serial BLAS library as buildings blocks
- Constraints on data layout or the topology of processors grid:
  - square processors grid (HJS and SBNDRD)
  - fixed physical data layout (HJS)
  - physical data layout block size = algorithmic block size (SBNDRD and ScaLAPACK)
  - final bandwidth = algorithmic block size (SBNDRD)
  - global matrix size = $k\times$ final bandwidth (SBNDRD)

Goal: Reimplement a full-to-band routine that did not suffer from these restrictions.

Householder transformation:

- Given a vector a, we can find a vector v,

$$v = a + \mathrm{sign}(a_1)\|a\|_2 e_1,$$

such that

$$(I - \frac{2vv^T}{v^Tv})a = -\mathrm{sign}\|a\|_2 e_1.$$

- Express the Householder transformation as

$$H = I + yw^T,$$

where

$$\begin{cases} y = v \\ w = -\frac{2v}{v^Tv} \end{cases}.$$

- Require further work to accumulate transformations for using BLAS-3 operations (block update).

WY representation: (Bischof and Van Loan, 1987)

- $H_i = I + y_i w_i^T$
- Let $Y_1 = y_1$ and $W_1 = w_1$,

$$
\begin{aligned}
H_2 H_1 &= (I + y_2 w_2^T)(I + y_1 w_1^T) \\
&= I + y_2 w_2^T + y_1 w_1^T + y_2 w_2^T y_1 w_1^T \\
&= I + [y_1\ y_2][w_1\ H_1 w_2]^T \\
&= I + Y_2 W_2^T.
\end{aligned}
$$

- Assume that $H_{k-1} \cdots H_1 = I + Y_{k-1} W_{k-1}^T$.
  Then we have

$$
\begin{aligned}
H_k(H_{k-1} \cdots H_1) &= (I + y_k w_k^T)(I + Y_{k-1} W_{k-1}^T) \\
&= I + Y_k W_k^T,
\end{aligned}
$$

where

$$
\left\{
\begin{aligned}
Y_k &= [Y_{k-1}\ y_k] \\
W_k &= [W_{k-1}\ \ (I + Y_{k-1} W_{k-1}^T) w_k]
\end{aligned}
\right. .
$$

Two-sided Householder transformation: (Dongarra, Hammarling, & Sorensen, 1989)

- $(I + y_1 w_1^T) A (I + w_1 y_1^T)$

$$\begin{aligned} &= A + y_1 w_1^T A + A w_1 y_1^T + \frac{1}{2} y_1 w_1^T A w_1 y_1^T + \frac{1}{2} y_1 w_1^T A w_1 y_1^T \\ &= A + y_1 z_1^T + z_1 y_1^T, \end{aligned}$$

  where $z_1 = A w_1 + \frac{1}{2} y_1 w_1^T A w_1$.

- Let $Y_1 = y_1$ and $Z_1 = z_1$. Assume that

$$\begin{aligned} A_{k-1} &= (I + y_{k-1} w_{k-1}^T) \cdots (I + y_1 w_1^T) A (I + w_1 y_1^T) \cdots (I + w_{k-1} y_{k-1}^T) \\ &= A + Y_{k-1} Z_{k-1}^T + Z_{k-1} Y_{k-1}^T. \end{aligned}$$

  Then we have

$$\begin{aligned} A_k &= (I + y_k w_k^T) A_{k-1} (I + w_k y_k^T) \\ &= A + Y_k Z_k^T + Z_k Y_k^T, \end{aligned}$$

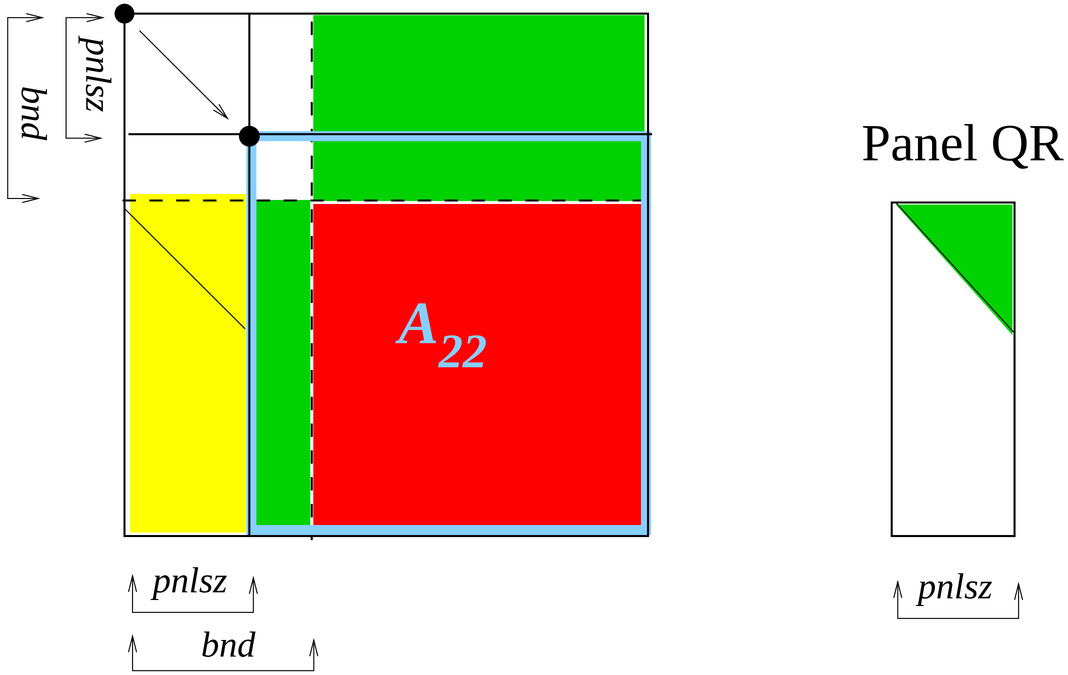
  where

$$\left\{ \begin{array}{lcl} Y_k & = & [Y_{k-1} \; y_k] \\ Z_k & = & [Z_{k-1} \; z_k] \\ z_k & = & A_{k-1} w_k + \frac{1}{2} y_k w_k^T A_{k-1} w_k \\ A_{k-1} w_k & = & A w_k + Y_{k-1} Z_{k-1}^T w_k + Z_{k-1} Y_{k-1}^T w_k \end{array} \right. .$$

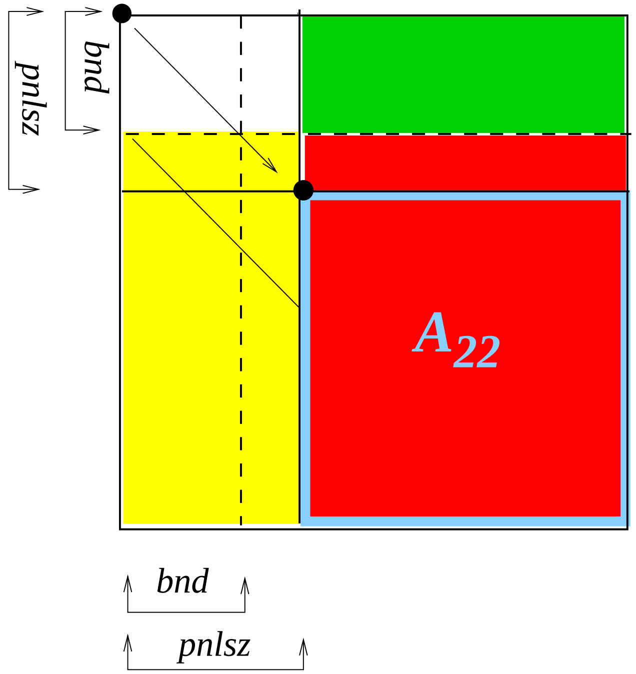
Full-to-Band Algorithm (SBNDRD)

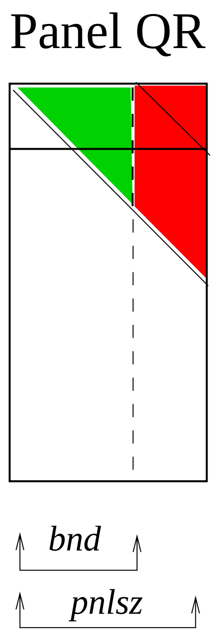
nb = the algorithmic block size
bnd = the final bandwidth
$pnlsz$ = the panelsize for QR ($<= nb$)

Case 1: $pnlsz <= bnd$



Case 2: $pnlsz > bnd$





Reduction procedure:

While ($pnlsz > 0$) {

Divide A to get panel and A_{22}

Panel QR (yellow)

Update A_{22} (green & red)

Replace A by A_{22} (black bullet) and compute next $pnlsz$

}

Transformations accumulation:

- Update on the fly

$$\underbrace{\underbrace{\underbrace{\underbrace{(I+W_1Y_1^T)}_{\texttt{step 1}}(I+W_2Y_2^T)}_{\texttt{step 2}}\cdots(I+W_{k-1}Y_{k-1}^T)}_{\texttt{step k}-\texttt{2}}(I+W_kY_k^T)}_{\texttt{step k}-\texttt{1}}$$

- Backward accumulation

$$\underbrace{(I+W_1Y_1^T)\underbrace{(I+W_2Y_2^T)\cdots\underbrace{(I+W_{k-1}Y_{k-1}^T)\underbrace{(I+W_kY_k^T)}_{\texttt{step 1}}}_{\texttt{step 2}}}_{\texttt{step k}-\texttt{2}}}_{\texttt{step k}-\texttt{1}}$$

PLAPACK based Full-to-Band Routines

Advantages of PLAPACK: (van de Geijn *et al*, 1996)

- Enable the rapid transformation of matrix algorithms into parallel codes
- Hide the complicated local index alignment and data movement issues
- Take care of the partial panel caused by uneven physical data layout

New SBNDRD routine:

- Allows all possible input options on the processors mesh, the algorithmic block size, the physical data layout, and the final band width
- Lacks explicit index transformations between the global matrix view and local processor index view

Code:

```
void prism_pla_sbndrd(PLA_Obj A, PLA_Obj Q, int nb, int bnd)
{
 /********   declarations and definitions  ***********/

 PLA_Matrix_global_all( A, &m, &n, &im, &in, &template);
 PLA_Matrix_create(MPI_DOUBLE, n-bnd, n-bnd, template,
                   im+bnd, in+bnd, &Y);
 PLA_Matrix_create(MPI_DOUBLE, n-bnd, n-bnd, template,
                   im+bnd, in+bnd, &W);
 PLA_Matrix_create(MPI_DOUBLE, n-bnd, n-bnd, template,
                   im+bnd, in+bnd, &Z);

 PLA_Obj_view_all(A, &Acur);

 m=n-bnd-1;
 for (k=0; k< m; k+=nb) {
  pnlsz = (m-k > nb)? nb: (m-k);
  PLA_Matrix_submatrix(W, n-bnd-k, pnlsz, k, k, &Wcur);
  PLA_Matrix_submatrix(Y, n-bnd-k, pnlsz, k, k, &Ycur);
  PLA_Matrix_submatrix(Z, n-bnd-k, pnlsz, k, k, &Zcur);
  /* panel QR */
  prism_pla_pnlqr(Acur, Wcur, Ycur, Zcur, bnd);
```

```
/* Update A22 */
PLA_Obj_split_4( Acur, pnlsz, pnlsz, &star, &star, &star, &A22);
if (pnlsz > bnd) {
```

$A_{22} \leftarrow A_{22} + Z_{temp}Y_{temp}^T + Y_{temp}Z_{temp}^T$

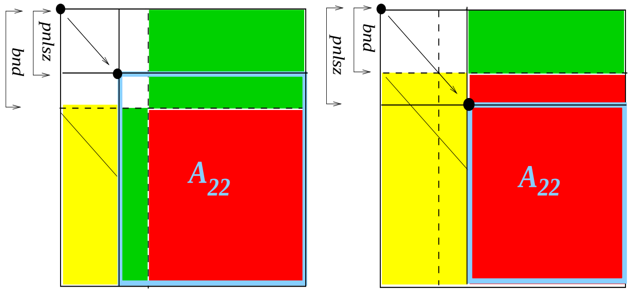
```
 PLA_Obj_horz_split_2( Ycur, bnd-pnlsz, &star, &Ytemp);
 PLA_Obj_horz_split_2( Zcur, bnd-pnlsz, &star, &Ztemp);
 PLA_Gemm(PLA_NO_TRANS,PLA_TRANS, PLA_ONE , Ztemp, Ytemp,PLA_ONE, A22);
 PLA_Gemm(PLA_NO_TRANS,PLA_TRANS, PLA_ONE , Ytemp, Ztemp,PLA_ONE, A22);
else {
```

$A_{temp} \leftarrow (I + Y_{cur}W_{cur}^T)A_{temp}$

```
 PLA_Obj_horz_split_2( A22, bnd-pnlsz, &star, &Atemp);
 PLA_Matrix_submatrix(Z, pnlsz, n-pnlsz-k, k, k+pnlsz, &work);
 PLA_Gemm(PLA_TRANS,PLA_NO_TRANS, PLA_ONE, Wcur, Atemp, PLA_ZERO, work);
 PLA_Gemm(PLA_NO_TRANS,PLA_NO_TRANS, PLA_ONE , Ycur, work, PLA_ONE, Atemp);
```

$A_{temp} \leftarrow A_{temp}(I + W_{cur}Y_{cur}^T)$

```
 PLA_Obj_vert_split_2( A22, bnd-pnlsz, &star, &Atemp);
 PLA_Matrix_submatrix(Z, n-pnlsz-k, pnlsz, k+pnlsz, k, &work);
 PLA_Gemm(PLA_NO_TRANS,PLA_NO_TRANS, PLA_ONE, Atemp, Wcur, PLA_ZERO, work);
 PLA_Gemm(PLA_NO_TRANS,PLA_TRANS, PLA_ONE , work, Ycur, PLA_ONE, Atemp);
}
PLA_Obj_view_swap(&Acur, &A22);
} /* end of A-update */
```



```
m=n-pnlsz-1;
for (k=m; k>=bnd; k-=nb) {
 PLA_Obj_split_4(Q, k, k, &star, &star, &star, &Q2);
 PLA_Matrix_submatrix(W, n-k, pnlsz, k, k-bnd, &Wtemp);
 PLA_Matrix_submatrix(Y, n-k, pnlsz, k, k-bnd, &Ytemp);

 if (k==m) {
  PLA_Gemm(PLA_NO_TRANS,PLA_TRANS, PLA_ONE, Wtemp, Ytemp, PLA_ONE, Q2);
 }
 else {
  PLA_Matrix_submatrix(Z, pnlsz, n-k, k-bnd, k, &Ztemp);
  if (nb>=1) {
   PLA_Gemm(PLA_TRANS,PLA_NO_TRANS, PLA_ONE, Ytemp, Q2, PLA_ZERO, Ztemp);
   PLA_Gemm(PLA_NO_TRANS,PLA_NO_TRANS, PLA_ONE, Wtemp, Ztemp, PLA_ONE, Q2);
  }
 }
pnlsz = nb;
} /* end of Q-update */

 /********  free objects **********/

} /* end of prism_pla_sbndrd */
```

$$\underbrace{(I + W_1Y_1^T)\underbrace{(I + W_2Y_2^T)\cdots\underbrace{(I + W_{k-1}Y_{k-1}^T)(I + W_kY_k^T)}}_{\cdots}}$$

Programming tips:

- Design a global block algorithm first
- Minimize local control
- Have a clear picture of global alignments of PLA_Objs
- Use PLA_BLAS to get correct answer first

Remarks:

- Less local indices
- No communication functions
- Easy to maintain or add new functions (skipping for PRISM)
- One global index k
  (To eliminate k: `for` loop $\Longrightarrow$ `while` loop + `PLA_Obj_view_shift`)

Performance

The first result:
matrix size $n = 2048$, processors grid 4×4,
template $tnb = 24$, algorithmic block size $nb = 32$
IBM SP

	bnd			
time (sec)	4	8	16	32
PLA_SBNDRD	824.48	719.59	642.31	618.84
OLD_SBNDRD	65.07	51.68	45.00	41.98
ratio	12.67	13.92	14.23	14.74

Slow, but the answers are correct!!

Optimization:

- Set PLA_Obj orientation:

$$A_{temp} \leftarrow (I + Y_{cur}W_{cur}^T)A_{temp}$$

```
PLA_Matrix_submatrix(Z, pnlsz, n-pnlsz-k, k, k+pnlsz, &work);
PLA_Obj_set_orientation(work, PLA_PROJ_ONTO_ROW);
PLA_Gemm(PLA_TRANS,PLA_NO_TRANS, PLA_ONE, Wcur, Atemp, PLA_ZERO, work);
PLA_Gemm(PLA_NO_TRANS,PLA_NO_TRANS, PLA_ONE, Ycur, work, PLA_ONE, Atemp);
```

- Create "conformal to" PLA_Obj:

$$a \leftarrow (I + YW^T)a$$

```
PLA_Pmvector_create_conf_to(W,PLA_PROJ_ONTO_ROW,PLA_ALL_ROWS,1,&work);
PLA_Gemv(PLA_TRANS, PLA_ONE, W, a, PLA_ZERO, work);
PLA_Gemv(PLA_NO_TRANS, PLA_ONE, Y, work, PLA_ONE, a);
```

Progress: (time unit = second)

- Intel Paragon: $n = 750$; $tnb = 30$; $bnd = 30$; $nb = 30$; mesh=3×3

Routine	Panel QR	A_{22} Update	Q Update	Total (sec)
Original	103.6150	62.2850	11.6312	177.75
Optimized	13.4601	5.6688	2.1606	21.49

- IBM SP1: $n = 2048$; mesh=4×4

Parameters			PLA_SBNDRD				OLD_ SBNDRD	NEW/OLD Ratio
nb	bnd	tnb	pnlqr	A22	Q	Total		
32	32	16	22.55 (0.75)	28.17 (50.01)	11.39 (61.87)	62.74	41.98	1.49
1	1	32	15.10 (0.035)	165.00 (10.84)	65.62 (13.63)	247.75	134.15	1.85
32	1	32	85.71 (4.48)	11.87 (62.58)	12.30 (59.97)	110.44	134.15	0.82
The first results								
32	32	24				618.84	41.98	14.74

($\star\star\star$): Mflops per second per node

More timing results: (time unit = second)
IBM SP1

matrix (mesh)	PLA_SBNDRD b=32	OLD_SBNDRD b=1	ScaLAPACK b=1	ScaLAPACK b=5	ScaLAPACK b=10
2000 (4x4)	116.90	126.08 *0.92*	118.94 *0.98*	56.51 *2.07*	49.56 *2.36*
3000 (5x5)	220.29	261.58 *0.84*	222.22 *0.99*	120.60 *1.83*	102.25 *2.15*

ratio = (PLA_SBNDRD) / (current time)

Generality and Flexibility: IBM SP2

All entrywise relative error $\in$ [8.5E-16 , 2.5E-15]; Matrix size=2000 $\times$ 2000

$mesh = 4 \times 4$

bnd (columns), *tnb* (rows), $nb = 21$

tnb \ bnd	11	22	33
10	142.36	68.81	73.03
20	120.32	59.50	54.71
30	110.11	46.71	46.97

nb (columns), *tnb* (rows), $bnd = 23$

tnb \ nb	11	22	33
10	74.49	66.29	153.51
20	56.03	53.36	106.96
30	52.09	49.06	97.41

$mesh = 4 \times 5$

bnd (columns), *tnb* (rows), $nb = 21$

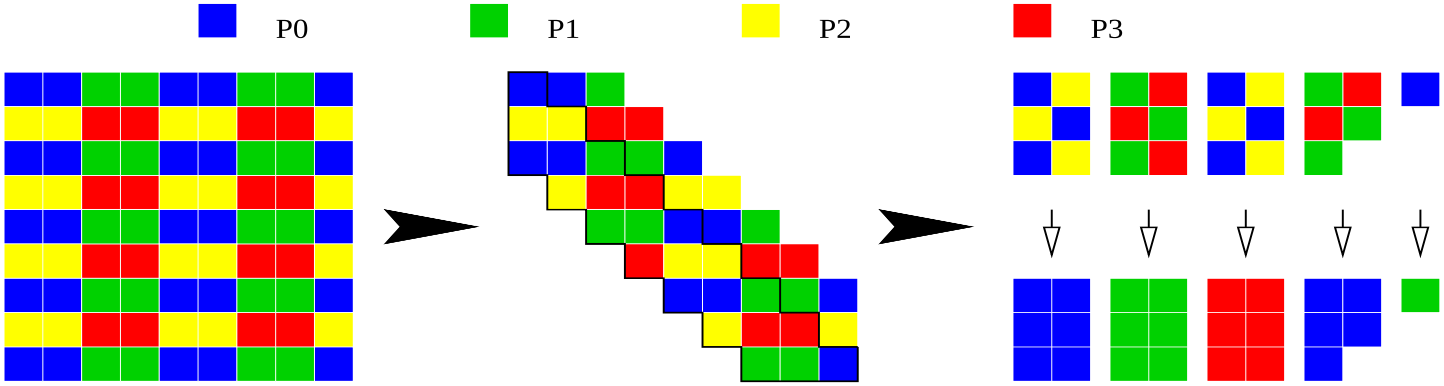
tnb \ bnd	11	22	33
10	144.98	67.78	68.14
20	111.27	53.44	53.69
30	97.11	46.44	46.86

nb (columns), *tnb* (rows), $bnd = 23$

tnb \ nb	11	22	33
10	70.93	63.61	139.63
20	51.66	50.74	104.31
30	50.13	44.91	97.39

Data Redistribution

Example: 2D full → 1D packed, 4 procs → 3 procs



Some timing results:

$bnd = 50, tnb = 32, np =$ no. of processors for packed matrix

case	2000 (4x4)			3000 (5x5)		
np	3	4	5	4	5	6
time(sec)	3.67	2.66	2.09	4.72	3.31	2.88

Code: (lines ratio OLD:PLA = 120:1)

```
/* A = PLA_Matrix, b_A = packed matrix, me = my rank,
   mycols = columns of b_A I own, np = no. nodes for b_A */
PLA_API_begin();
PLA_Obj_API_open( A);
if ( me < np ) {
  if ( mycols != 0) {
    b_A=(double *) malloc( (2*bnd+1)*mycols*sizeof(double) );
    local_b_A=b_A;
  }
}

for ( k=0; k<n; k++ ) {
  if ( (int)(k/bnd)%np == me ) {
    j=(n-k<=bnd)? n-k: bnd+1;
    PLA_API_axpy_global_to_matrix( j, 1, &r_one, A, k, k,
                                   local_b_A, 2*bnd+1 );
    local_b_A += (2*bnd+1);
  }
}

PLA_Obj_API_close( A);
PLA_API_end();
```

Conclusions and Future Work

Conclusions:

- PLAPACK based routines are simple and clean.
- The portability and simplicity of our code does not impact performance.
- The codes are easy to document, maintain, and extend.

Future work:

- Optimize the current PLAPACK based SBNDRD
- Convert the current TRIDRD to use PLAPACK
- Built PLAPACK based PRISM eigensolver

PRISM (Parallel Research on Invariant Subspace Methods) project:
http://www.mcs.anl.gov/Projects/PRISM/