Optimizing Krylov Subspace Solvers on Graphics Processing Units

Hartwig Anzt, W. Sawyer, S. Tomov, P. Luszczek, I. Yamazaki, J. Dongarra

May 19, 2014
Krylov Methods

- Alexei Nikolajewitsch Krylov (1863-1945)
- iterative solvers for linear systems
- searching for solution approximation in subspace
- matrix-vector product generates (Krylov-) subspace
- target: sparse systems
- examples: CG, BiCGStab, GMRES...
Porting Krylov Methods to GPUs

- NVIDIA provides a set of useful routines (cuBLAS/cuSPARSE)
- compose Krylov method out of these building blocks

*is this efficient?*
Porting Krylov Methods to GPUs

- provides a set of useful routines (cuBLAS/cuSPARSE)
- compose Krylov method out of these building blocks

_is this efficient?_

- Krylov methods are usually memory-bound
- every kernel requires accessing all needed data in main memory
- high number of kernels in iteration loop
- idea: replace cuBLAS/cuSPARSE by algorithm-specific kernels
  - reduced memory traffic
  - reduced kernel launch overhead
BiCGStab

1: while \((k < \text{maxiter}) \&\& (\text{res} > \tau)\) 
2: \(k := k + 1\) 
3: \(\rho_k := \hat{r}_0^T r_{k-1}\) 
4: \(\beta_{k+1} := \frac{\rho_k}{\rho_{k-1}} \frac{\alpha_{k-1}}{\omega_{k-1}}\) 
5: \(p_k := r_{k-1} + \beta (\rho_{k-1} - \omega_{k-1} v_{k-1})\) 
6: \(v_k := Ap_k\) 
7: \(\alpha_k := \frac{\rho_k}{\hat{r}_0^T v}\) 
8: \(s_k := r_{k-1} - \alpha v_k\) 
9: \(t_k := As_k\) 
10: \(\omega_k := \frac{s_k^T t_k}{t_k^T t_k}\) 
11: \(x_{k+1} := x_k + \alpha_k p_k + \omega_k s_k\) 
12: \(r_k := s_k - \omega t_k\) 
13: \(\text{res} = r_k^T r_k\) 
14: end
BiCGStab

1: while \((k < maxiter) \&\& (res > \tau)\) 
2: \(k := k + 1\) 
3: \(\rho_k := \hat{r}_0^T r_{k-1}\) 
4: \(\beta_{k+1} := \frac{\rho_k}{\rho_k-1} \frac{\alpha_{k-1}}{\omega_{k-1}}\) 
5: \(p_k := r_{k-1} + \beta (p_{k-1} - \omega_{k-1} v_{k-1})\) 
6: \(v_k := Ap_k\) 
7: \(\alpha_k := \frac{\rho_k}{\hat{r}_0^T v}\) 
8: \(s_k := r_{k-1} - \alpha v_k\) 
9: \(t_k := As_k\) 
10: \(\omega_k := \frac{s_k^T t_k}{t_k^T t_k}\) 
11: \(x_{k+1} := x_k + \alpha_k p_k + \omega_k s_k\) 
12: \(r_k := s_k - \omega t_k\) 
13: \(res = r_k^T r_k\) 
14: end

\[p_k := r_{k-1} + \beta (p_{k-1} - \omega_k v_{k-1})\]

- **cuBLAS**
  - cublasDscal( n, beta, p, 1 );
  - cublasDaxpy( n, omega * beta, v, 1, p, 1 );
  - cublasDaxpy( n, 1.0, r, 1, p, 1 );

  3 kernels - 5n reads, 3n writes

- **merge in one kernel**

  p_update( int n, double beta, double omega,
            double *v, double *r, double *p ){
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if( i<n )
      p[i] = r[i] + beta * ( p[i]-omega*v[i] );
  }

  1 kernel - 3n reads, 1n writes
BiCGStab

1: while \((k < \text{maxiter}) \&\& (\text{res} > \tau)\)
2: 
3: \(k := k + 1\)
4: \(p_k := r_k - 1 + \beta (p_{k-1} - \omega_{k-1} v_{k-1})\)

```
p_k := r_{k-1} + \beta (p_{k-1} - \omega_{k-1} v_{k-1})
```

```
cuBLAS
  cublasDscal(n, beta, p, 1);
cublasDaxpy(n, omega * beta, v, 1, p, 1);
cublasDaxpy(n, 1.0, r, 1, p, 1);
```

3 kernels - 5n reads, 3n writes
better merge in one kernel
```
p_update(int n, double beta, double omega, double *v, double *r, double *p) {
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  if (i < n)
    p[i] = r[i] + beta * (p[i] - omega * v[i]);
```
1 kernel - 3n reads, 1n writes

```
MERGE
```

```
CUBLAS
```

- vector length n in 10^3
- GFLOPS
- vector length n in 10^3
- GFLOPS

7 May 19, 2014 Hartwig Anzt - Optimizing Krylov Subspace Solvers on GPUs
while( ( k < maxiter ) && ( res > epsilon ) ){
  rho_new = cublas_dot( n, r_hat, 1, r, 1 );
  beta = rho_new/rho_old * alpha/omega;  
  cublas_dscal( n, beta, p, 1 );
  cublas_daxpy( n, omega * beta, v, 1 , p, 1 );
  dSpMV <<<Gs,Bs>>> ( n, rowA, colA, valA, p, v );
  alpha = rho_new / cublas_dot( n, r_hat, 1, v, 1 );
  cublas_dcopy( n, r, 1 , s, 1 );
  cublas_daxpy( n, -1.0 * alpha, v, 1 , s, 1 );
  dSpMV <<<Gs,Bs>>> ( n, rowA, colA, valA, s, t );
  omega = cublas_dot( n, t, 1, s, 1 )
    / cublas_dot( n, t, 1, t, 1 );
  cublas_daxpy( dofs, alpha, p, 1 , x, 1 );
  cublas_daxpy( dofs, omega, s, 1 , x, 1 );
  cublas_dcopy( n, s, 1 , r, 1 );
  cublas_daxpy( n, -1.0 * omega, t, 1 , r, 1 );
  res = cublas_dnrm2( n, r, 1 );
  rho_old = rho_new;  
  k++;  }

4nnz+29n reads + 11n writes 4nnz+16n reads + 6n writes

- small variances possible due to recursive reduction (depending on n)
- savings about 35% when neglecting SpMV
merged BiCGStab

while( ( k < maxiter ) && ( res > epsilon ) ){
    rho_new = cublas_dot( n, r_hat, 1, r, 1 );
    beta = rho_new/rho_old * alpha/omega;
    cublas_dscal( n, beta, p, 1 );
    cublas_daxpy( n, omega * beta, v, 1 , p, 1 );
    dSpMV <<<Gs,Bs>>> ( n, rowA, colA, valA, p, v );
    alpha = rho_new / cublas_dot( n, r_hat, 1, v, 1 );
    cublas_dcopy( n, r, 1 , s, 1 );
    cublas_daxpy( n, -1.0 * alpha, v, 1 , s, 1 );
    dSpMV <<<Gs,Bs>>> ( n, rowA, colA, valA, s, t );
    omega = cublas_dot( n, t, 1, s, 1 )
        / cublas_dot( n, t, 1, t, 1 );
    cublas_daxpy( dofs, alpha, p, 1 , x, 1 );
    cublas_daxpy( dofs, omega, s, 1 , x, 1 );
    cublas_dcopy( n, s, 1 , r, 1 );
    cublas_daxpy( n, -1.0 * omega, t, 1 , r, 1 );
    res = cublas_dnrm2( n, r, 1 );
    rho_old = rho_new;
    k++;
}

4nnz+29n reads + 11n writes

- small variances possible due to recursive reduction (depending on n)
- savings about 35 % when neglecting SpMV

while( ( k < maxiter ) && ( res_host > epsilon ) ){
    magma_dbicgmerge_p_update<<<Gs, Bs, 0>>>
        ( n, skp, v, r, p );
    magma_dbicgmerge_spmv1<<<Gs, Bs, Ms1>>> ( n, valA, colA, p, r, v, d1 );
    magma_zbicgmerge_reduce1( n, Gs, Bs, d1, d2, skp );
    magma_dbicgmerge_s_update<<<Gs, Bs, 0>>>
        ( n, skp, r, v, s );
    magma_dbicgmerge_spMV2<<<Gs, Bs, Ms2>>> ( n, valA, colA, s, t, d1 );
    magma_dbicgmerge_reduce2( n, Gs, Bs, d1, d2, skp );
    magma_dbicgmerge_xr_update<<<Gs, Bs, 0>>> ( n, skp, r_hat, r, p, s, t, x, d1);
    magma_dbicgmerge_reduce3( n, Gs, Bs, d1, d2, skp);
    magma_memcopy( 1, skp+5, res_host );
    k++;
}

4nnz+16n reads + 6n writes
merging multiple dot-products (MDOT)

- consecutive dot products sharing one common vector can be merged
- reduced memory reads improve performance

where is the line between merged dot-product and matrix-vector product?
merged dot vs. matrix vector product

vector length $n = 100,000$

vector length $n = 1,000,000$

MDGM superior

MAGMA_GeMV superior

Hartwig Anzt - Optimizing Krylov Subspace Solvers on GPUs
BiCGStab

1: while (k < maxiter) && (res > τ) 
2: \[ k := k + 1 \]
3: \[ \rho_k := \hat{r}_0^T r_{k-1} \]
4: \[ \beta_{k+1} := \frac{\rho_k}{\rho_{k-1}} \frac{\alpha_{k-1}}{\omega_{k-1}} \]
5: \[ p_k := r_{k-1} + \beta (p_{k-1} - \omega_{k-1} v_{k-1}) \]
6: \[ v_k := A p_k \]
7: \[ \alpha_k := \frac{\rho_k}{r_0^T v} \]
8: \[ s_k := r_{k-1} - \alpha v_k \]
9: \[ t_k := A s_k \]
10: \[ \omega_k := \frac{s_k^T t_k}{t_k^T t_k} \]
11: \[ x_{k+1} := x_k + \alpha_k p_k + \omega_k s_k \]
12: \[ r_k := s_k - \omega t_k \]
13: \[ res = r_k^T r_k \]
14: end

\[ F(n) = \frac{1}{100} \left( \frac{1}{2 \times 10^{-7} \times n + 0.021} + 12.5 \right) \]

12 May 19, 2014 Hartwig Anzt - Optimizing Krylov Subspace Solvers on GPUs
is it worth the effort?

\[ P_{\text{memory}} = \left( 1 - \text{SpMV} \right) \times \eta_{\text{memory}} \]

\[ \eta_{\text{memory}} = \frac{13n + 5n - 6n}{29n + 11n - 6n} \approx 35\% \]
is it worth the effort?

\[ P_{\text{memory+dot}} = (1 - \text{SpMV}) \times \eta_{\text{memory}} + (1 - \mathcal{F}(n)) \times \text{dot} \]

\[ \eta_{\text{memory}} = \frac{13n+5n-6n}{29n+11n-6n} \approx 35\% \]

\[ \mathcal{F}(n) = \frac{1}{100} \left( \frac{1}{2 \times 10^{-7} \times n + 0.021} + 12.5 \right) \]
merged BiCGStab performance

![Graph showing the relationship between SpMV dominance and runtime reduction for different benchmarks. The x-axis represents SpMV dominance (%) ranging from 0 to 100, and the y-axis represents improvement (%) ranging from 0 to 100. Various benchmarks such as bloweybq, fv1, Tref_2000, cage_10, ecology_2, airfoil, poiss3D, Tref_20000, Pres_Poiss, bmw3_2, audikw_1 are plotted on the graph. The graph shows a linear trend line indicating the average performance improvement.]

15 May 19, 2014 Hartwig Anzt - Optimizing Krylov Subspace Solvers on GPUs
merged BiCGStab performance

counter-intuitive kernel performance on K20c

<table>
<thead>
<tr>
<th>kernel1</th>
<th>kernel2</th>
</tr>
</thead>
</table>
| 1: $y = Ax \ (\text{SpMV})$ | 1: $y = Ax \ (\text{SpMV})$  
2: $y = y + x \ (\text{axpy})$ |

- kernel2 compiles with more registers
- execution time of kernel2 is shorter
- effect not reproducible on Fermi, K40 . . .
merged BiCGStab performance

runtime reduction (%)
- memory
- memory + dot
- memory + dot + SpMV

improvement (%)

SpMV dominance (%)
Summary & Take Home Message

- reduced GPU memory access
- accelerated dot product
- model predicting performance benefits

**cuSPARSE** LEGO is nice but we need it open source!