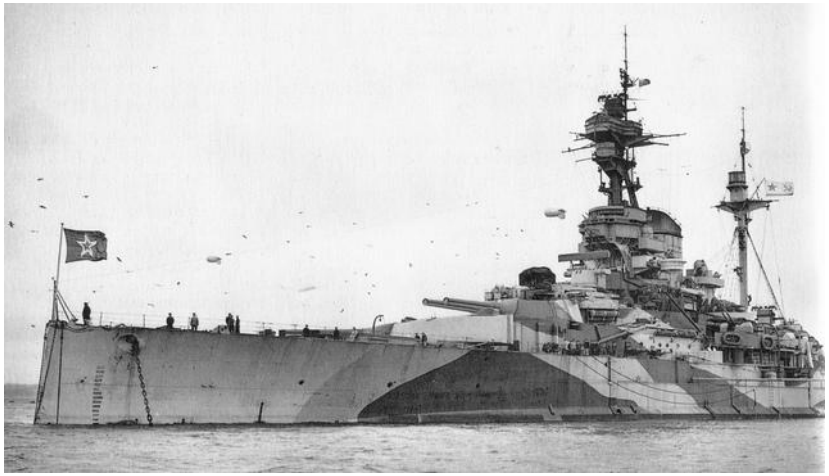
Optimizing Krylov Subspace Solvers on Graphics Processing Units

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

May 19, 2014

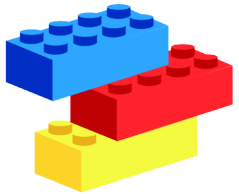
Krylov Methods

- 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

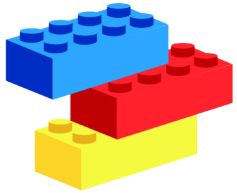
- NVIDIA provides a set of useful routines (cuBLAS/cuSPARSE)
- compose Krylov method out of these building blocks



is this efficient?

Porting Krylov Methods to GPUs

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

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

- **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

- **merge in one kernel**

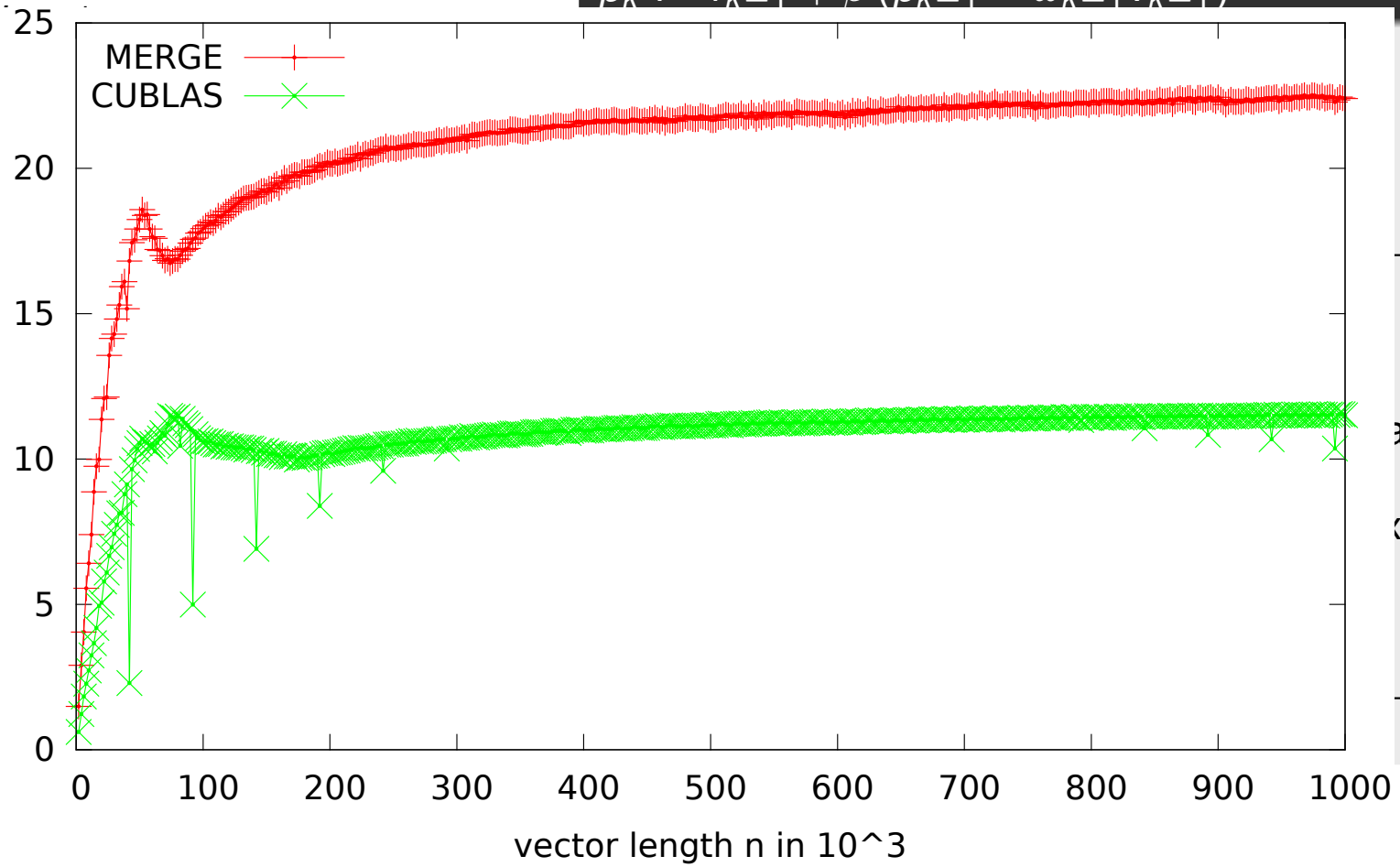
```
p_update( int n, double beta, double omega,
                double *v, double *r, double *p ){
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  if( i<n )
    p[i] = r[i] + beta * ( p[i]-omega*v[i] );
}
```

1 kernel - 3n reads, 1n writes

BiCGStab

1: **while** $(k < maxiter)$ && $(res > \tau)$
2: k
3: ρ_k
4: β_k
5: p_k
6: v_k
7: α_k
8: s_k
9: t_k
10: ω_k
11: x_k
12: r_k
13: re
14: **end**

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



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 );
    cublas_daxpy( n, 1.0, r, 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

```
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, rowA, 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, rowA, 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

- 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 );
    cublas_daxpy( n, 1.0, r, 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++;
}
```

two consecutive reductions

4nnz+29n reads + 11n writes

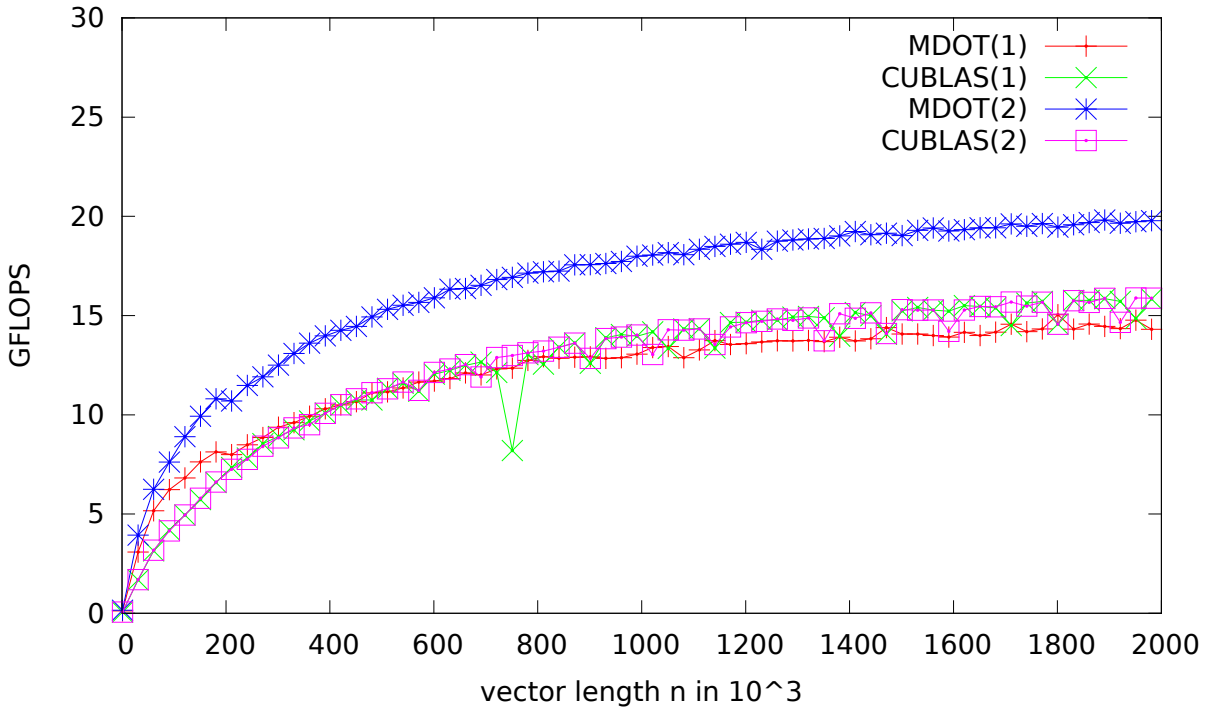
```
while( ( k < maxiter ) && ( res_host > epsilon ) ){
    magma_dbicgmerge_p_update<<<Gs, Bs, 0>>>
                ( n, skp, v, r, p );

    magma_dbicgmerge_spmv1<<<Gs, Bs, Ms1>>>
                ( n, valA, rowA, 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, rowA, 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

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

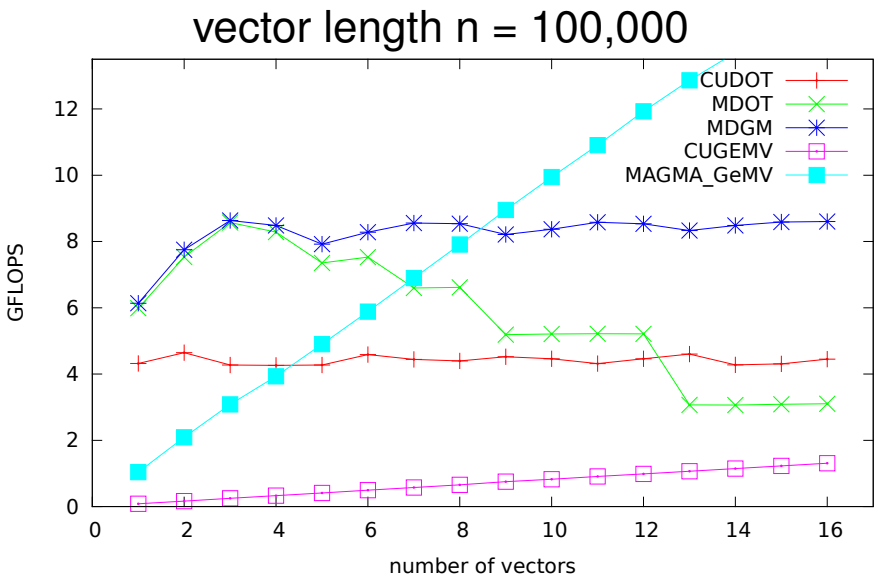
merging multiple dot-products (MDOT)

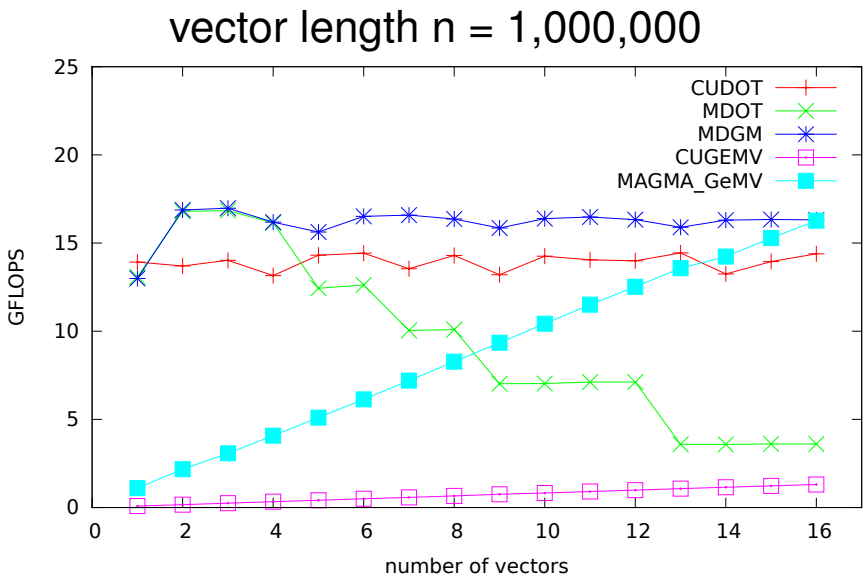
- consecutive dot products sharing one common vector can be merged
- reduced memory reads improve performance

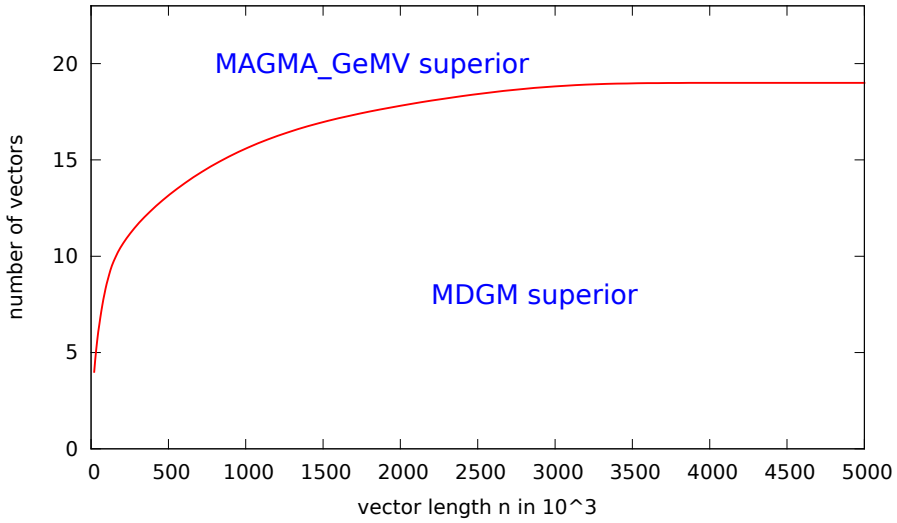


where is the line between merged dot-product and matrix-vector product?

merged dot vs. matrix vector product

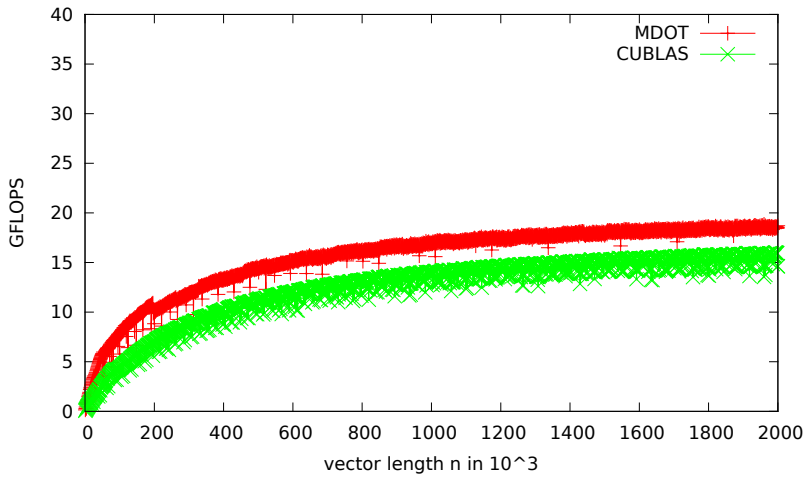


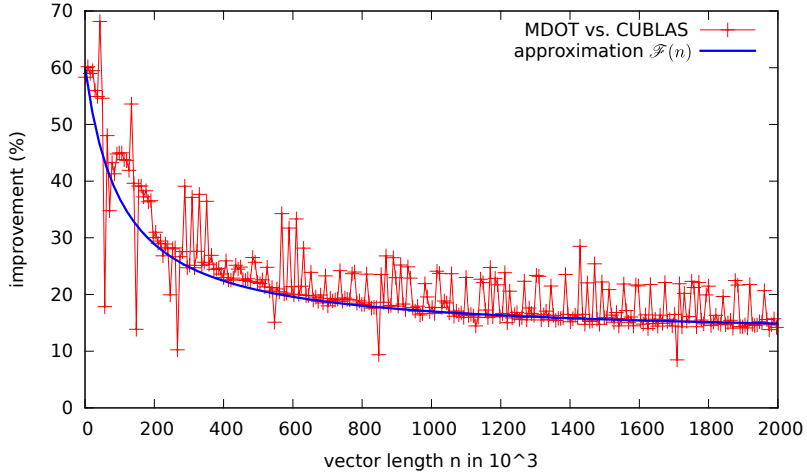




BiCGStab

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





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

merged BiCGStab performance model

is it worth the effort?

$$P_{\text{memory}} = \underbrace{(1 - SpMV) \times \eta_{\text{memory}}}_{P_{\text{memory}}}$$

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

merged BiCGStab performance model

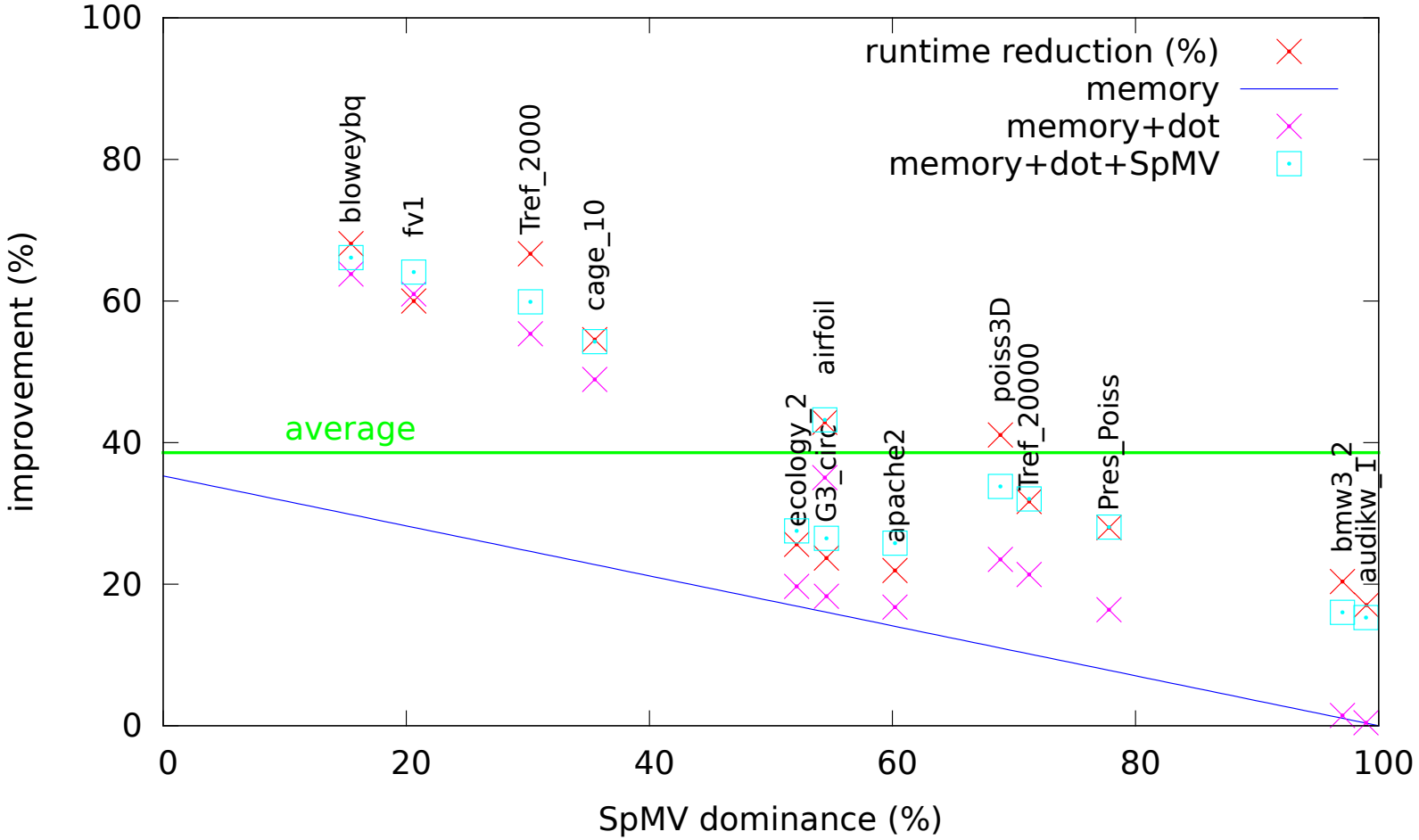
is it worth the effort?

$$P_{\text{memory+dot}} = \underbrace{(1 - SpMV) \times \eta_{\text{memory}}}_{P_{\text{memory}}} + \underbrace{(1 - \mathcal{F}(n)) \times \text{dot}}_{P_{\text{dot}}}$$

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

$$\mathcal{F}(n) = \frac{1}{100}\left(\frac{1}{2\times10^{-7}\times n+0.021} + 12.5\right)$$

merged BiCGStab performance



merged BiCGStab performance

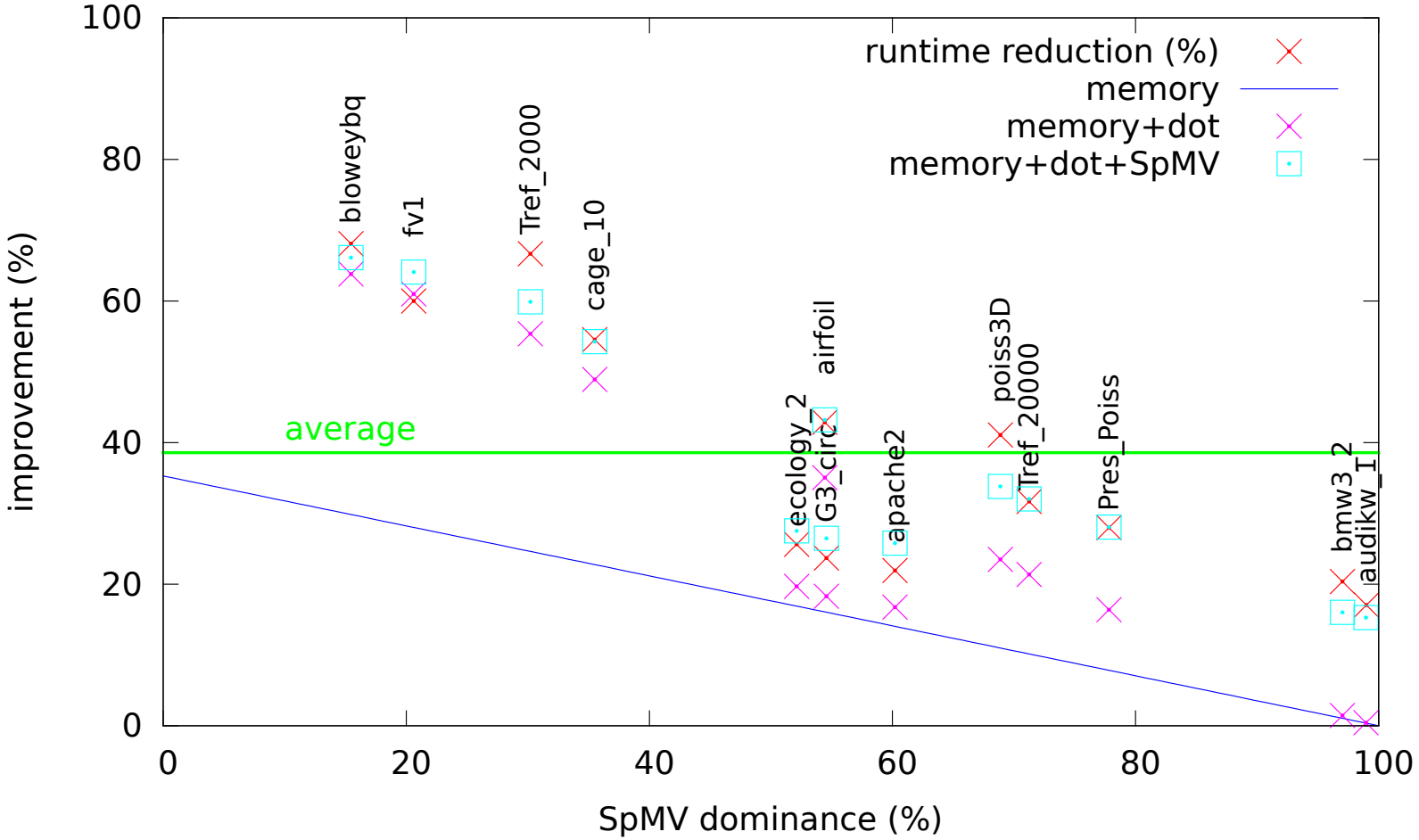
counter-intuitive kernel performance on K20c

kernel1
1: $y = Ax$ (SpMV)

kernel2
1: $y = Ax$ (SpMV) 2: $y = y + x$ (axpy)

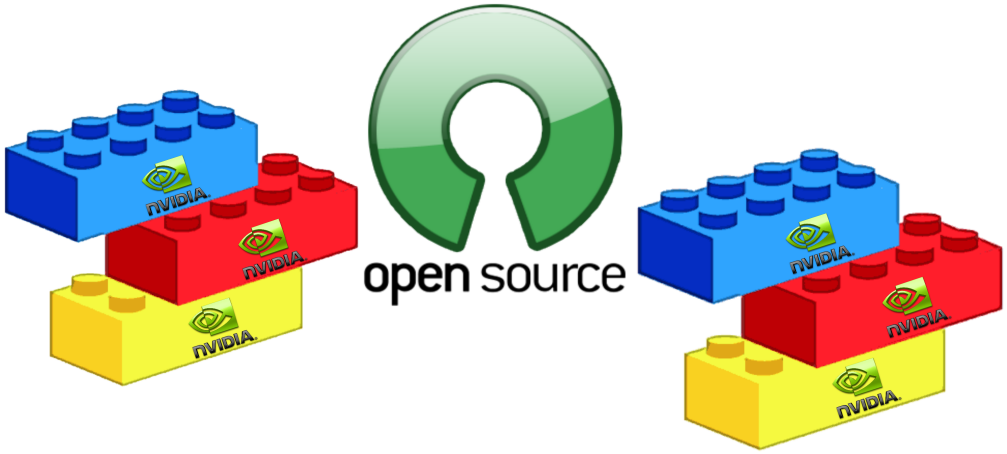
- kernel2 compiles with more registers
- execution time of kernel2 is shorter
- effect not reproducible on Fermi, K40 . . .

merged BiCGStab performance



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!