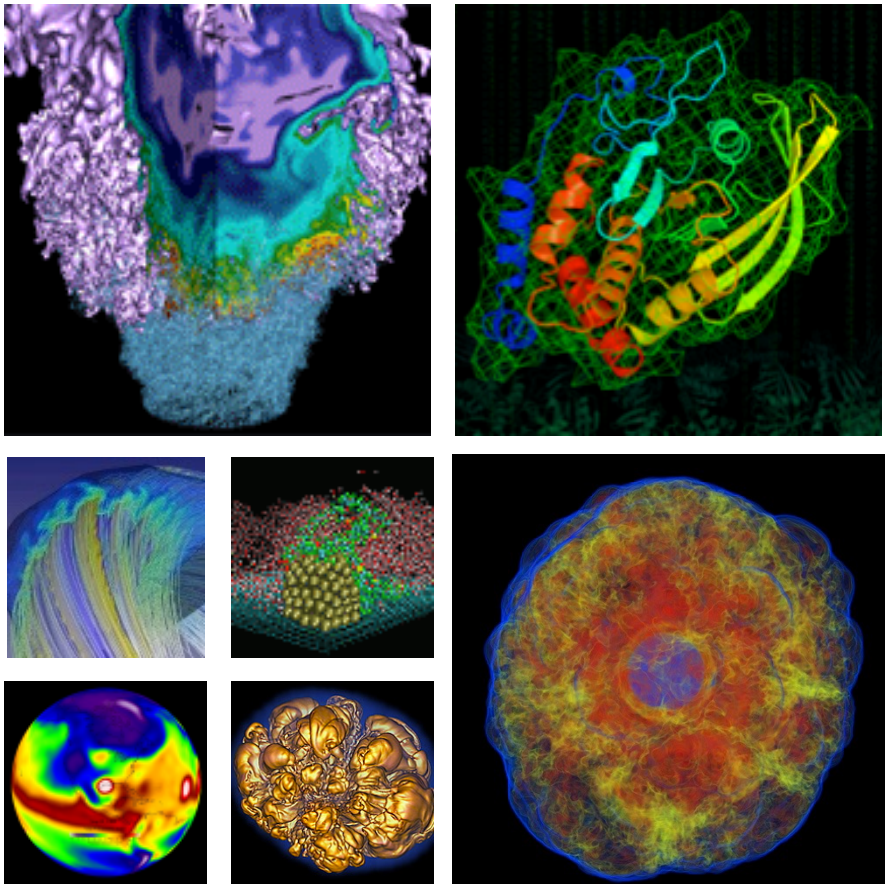
Optimizing SpMV and IDR Krylov solver in EMGeo for Intel KNL



Tareq Malas, Thorsten Kurth, Jack Deslippe

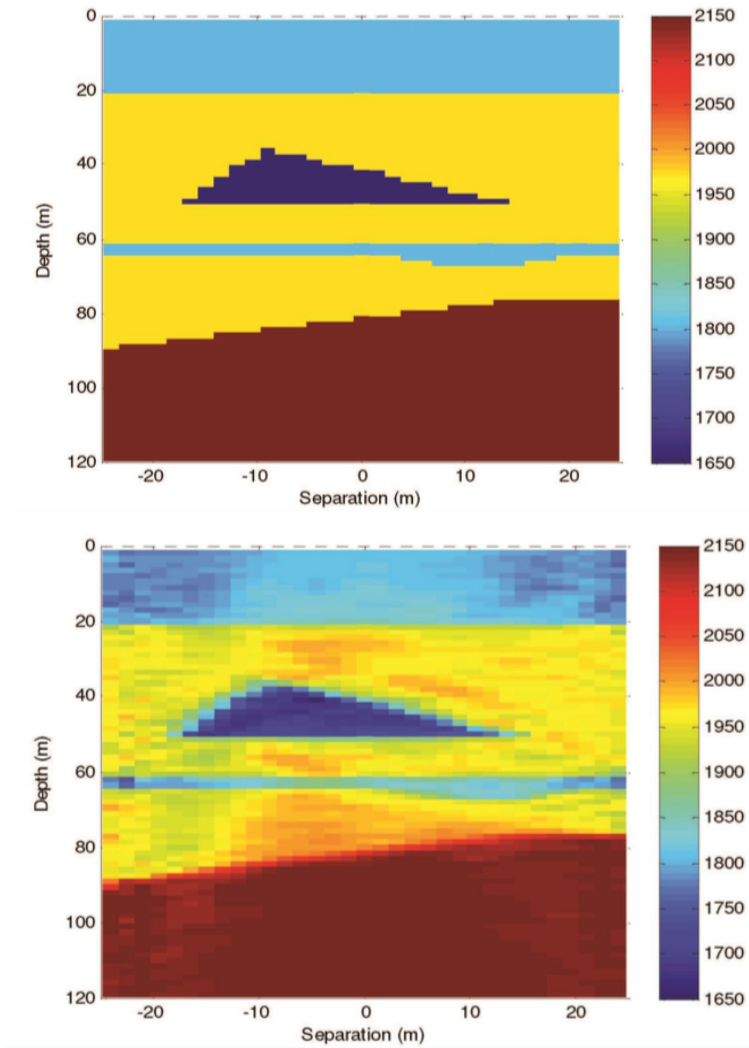
IXPUG 2016 Frankfurt, Germany
June 23, 2016

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

EMGeo - Introduction

- Geophysical tomography: measure composition of the ground by scattering seismic waves (inverse scattering problem)



Petrov and Newman 2014

EMGeo - Introduction

- Geophysical tomography: measure composition of the ground by scattering seismic waves (inverse scattering problem)
- Finite-Difference approximation of propagation kernel on grid with $N=N_x \cdot N_y \cdot N_z$ cells

$$\mathbf{K}_q \mathbf{v}_q = \mathbf{f}_q$$

$$\mathbf{K}_q \equiv \mathbf{I} - \langle b \rangle \mathbf{D}_\tau \cdot \left(\langle \mathbf{k}\mu \rangle \circ \mathbf{D}_v \right)$$

EMGeo - Introduction

- Geophysical tomography: measure composition of the ground by scattering seismic waves (inverse scattering problem)
- Finite-Difference approximation of propagation kernel on grid with $N=N_x \cdot N_y \cdot N_z$ cells
- Major matrices (double complex):
  - D_v (size 6Nx3N)
  - D_t (size 3Nx6N)

$$\mathbf{K}_q \mathbf{v}_q = \mathbf{f}_q$$

$$\mathbf{K}_q \equiv \mathbf{I} - \langle b \rangle \mathbf{D}_\tau \cdot \left(\langle \mathbf{k}\mu \rangle \circ \mathbf{D}_v \right)$$

$$\mathbf{D}_\tau = \begin{pmatrix} \tilde{D}_x & D_y & D_z & \tilde{D}_x & 0 & \tilde{D}_x \\ \tilde{D}_y & D_x & 0 & \tilde{D}_y & D_z & \tilde{D}_y \\ \tilde{D}_z & 0 & D_x & \tilde{D}_z & D_y & \tilde{D}_z \end{pmatrix}^T$$

$$\mathbf{D}_v = \begin{pmatrix} D_x & \tilde{D}_y & \tilde{D}_z & 0 & 0 & 0 \\ 0 & \tilde{D}_x & 0 & D_y & \tilde{D}_z & 0 \\ 0 & 0 & \tilde{D}_x & 0 & \tilde{D}_y & D_z \end{pmatrix}$$

EMGeo - Introduction

- Geophysical tomography: measure composition of the ground by scattering seismic waves (inverse scattering problem)
- Finite-Difference approximation of propagation kernel on grid with $N=N_x \cdot N_y \cdot N_z$ cells
- Major matrices (double complex):
  - D_v (size 6Nx3N)
  - D_t (size 3Nx6N)
- ELLPack storage format

$$\mathbf{K}_q \mathbf{v}_q = \mathbf{f}_q$$

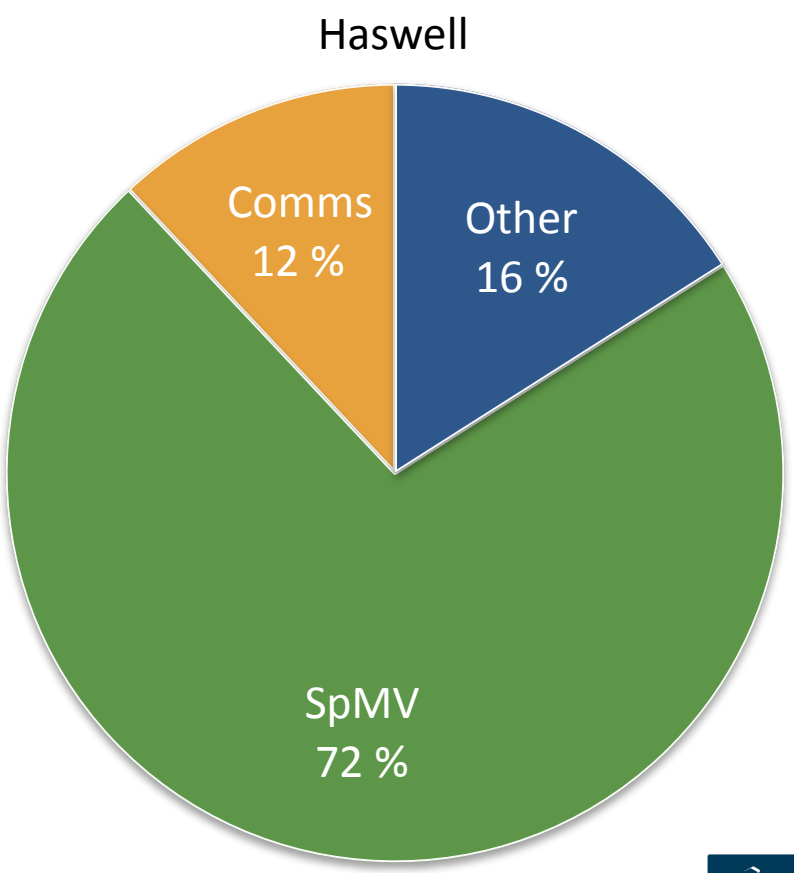
$$\mathbf{K}_q \equiv \mathbf{I} - \langle b \rangle \mathbf{D}_\tau \cdot \left(\langle \mathbf{k}\mu \rangle \circ \mathbf{D}_v \right)$$

$$\mathbf{D}_\tau = \begin{pmatrix} \tilde{D}_x & D_y & D_z & \tilde{D}_x & 0 & \tilde{D}_x \\ \tilde{D}_y & D_x & 0 & \tilde{D}_y & D_z & \tilde{D}_y \\ \tilde{D}_z & 0 & D_x & \tilde{D}_z & D_y & \tilde{D}_z \end{pmatrix}^T$$

$$\mathbf{D}_v = \begin{pmatrix} D_x & \tilde{D}_y & \tilde{D}_z & 0 & 0 & 0 \\ 0 & \tilde{D}_x & 0 & D_y & \tilde{D}_z & 0 \\ 0 & 0 & \tilde{D}_x & 0 & \tilde{D}_y & D_z \end{pmatrix}$$

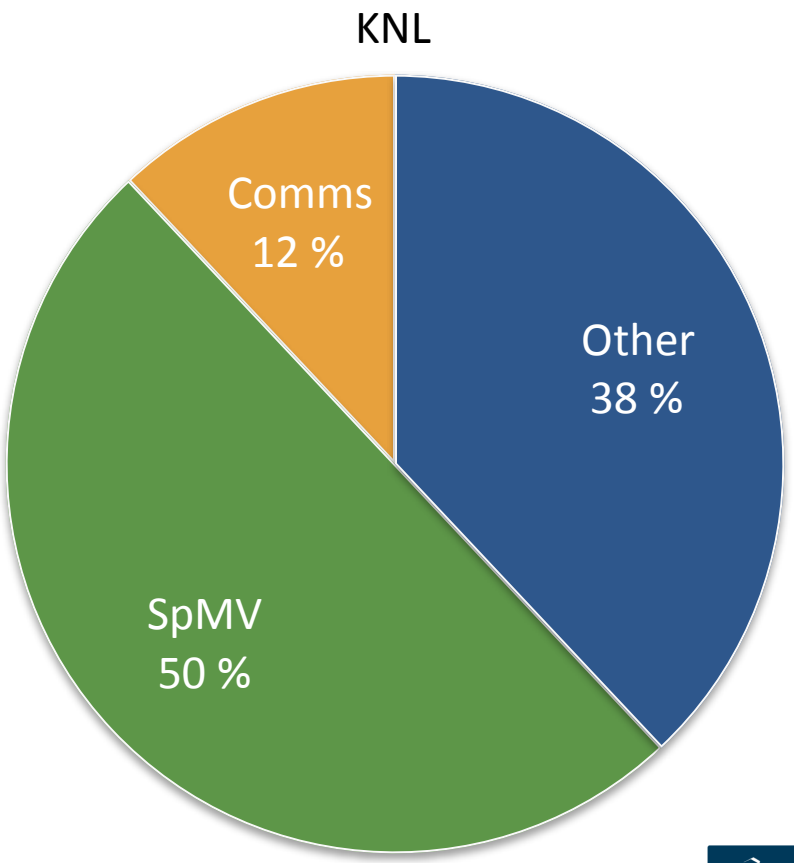
EMGeo - what we started from

- focus on forward pass: SpMV and linear solves
- code uses IDR (Krylov) solver for iterative solution of non-symmetric linear system
- where does the wall time go?
- Test on NERSC Cori Phase I
  - Single Haswell Node
  - Grid size 100x50x50
  - 32 Right Hand Sides, max. iterations 500
  - 2x4x4 MPI ranks w/o multithreading
- Other: (bi-)orthogonalization, control-flow, building IDR space



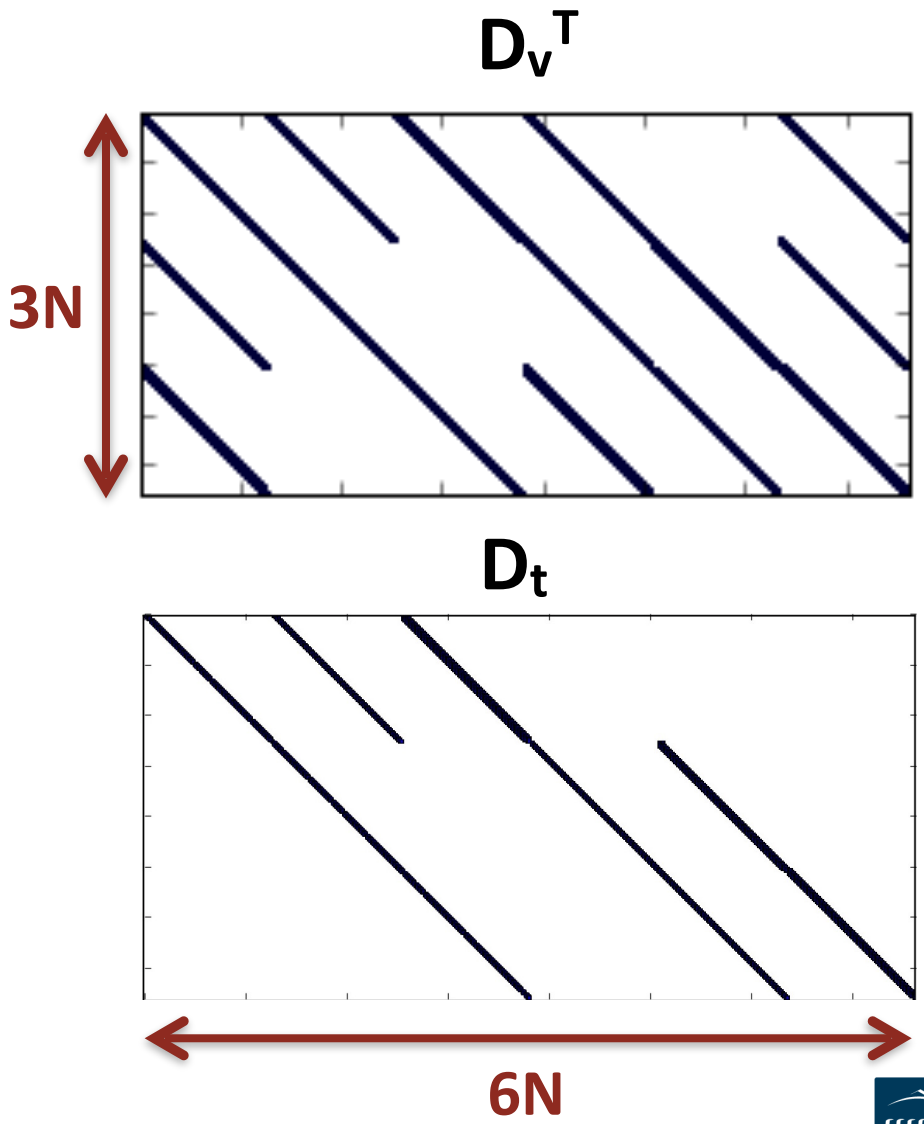
EMGeo - what we started from

- focus on forward pass: SpMV and linear solves
- code uses IDR (Krylov) solver for iterative solution of non-symmetric linear system
- where does the wall time go?
- Test on NERSC Cori Phase I
  - Single Haswell Node
  - Grid size 100x50x50
  - 32 Right Hand Sides, max. iterations 500
  - 2x4x4 MPI ranks w/o multithreading
- Other: (bi-)orthogonalization, control-flow, building IDR space



Matrix sparsity pattern

- D_v:
  - 50%: 8 nnz/row
  - 50%: 12 nnz/row
- D_t: 12 nnz/row
- $D_t \cdot D_v$:
  - up to 51 nnz/row $\Longrightarrow$ requires more memory and 30% more operations



ELLPack vs. SELLPack

```
do i=1,Nrows
  z(i) = DCMPLX(0.0D0)
  do j=1,ndiag
    z(i) = z(i) + A(j,i)*v(i_A(j,i))
  enddo
enddo
```

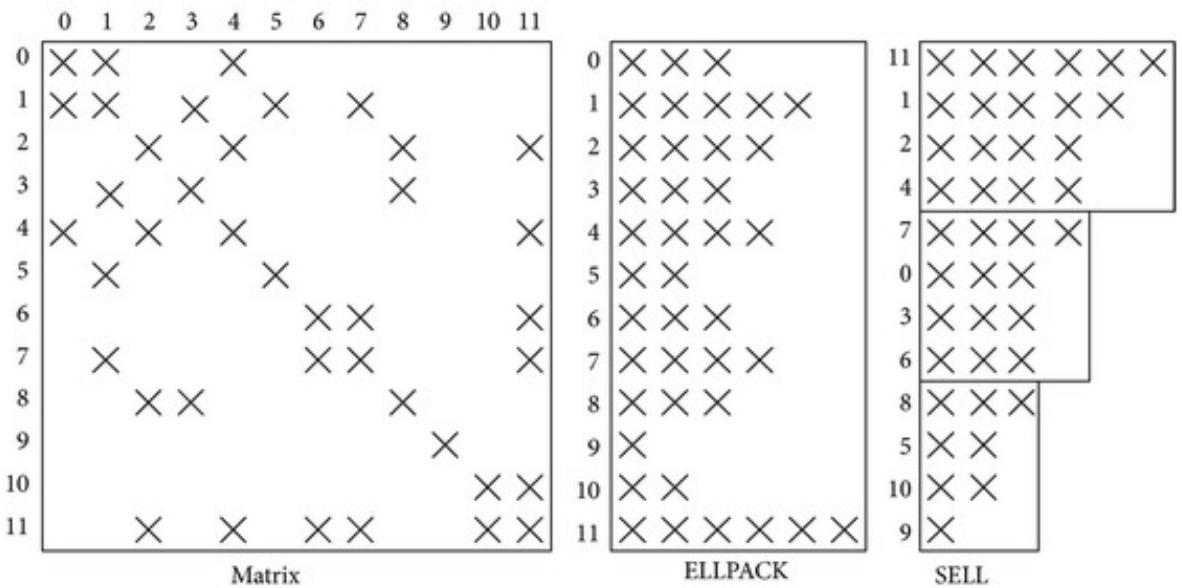
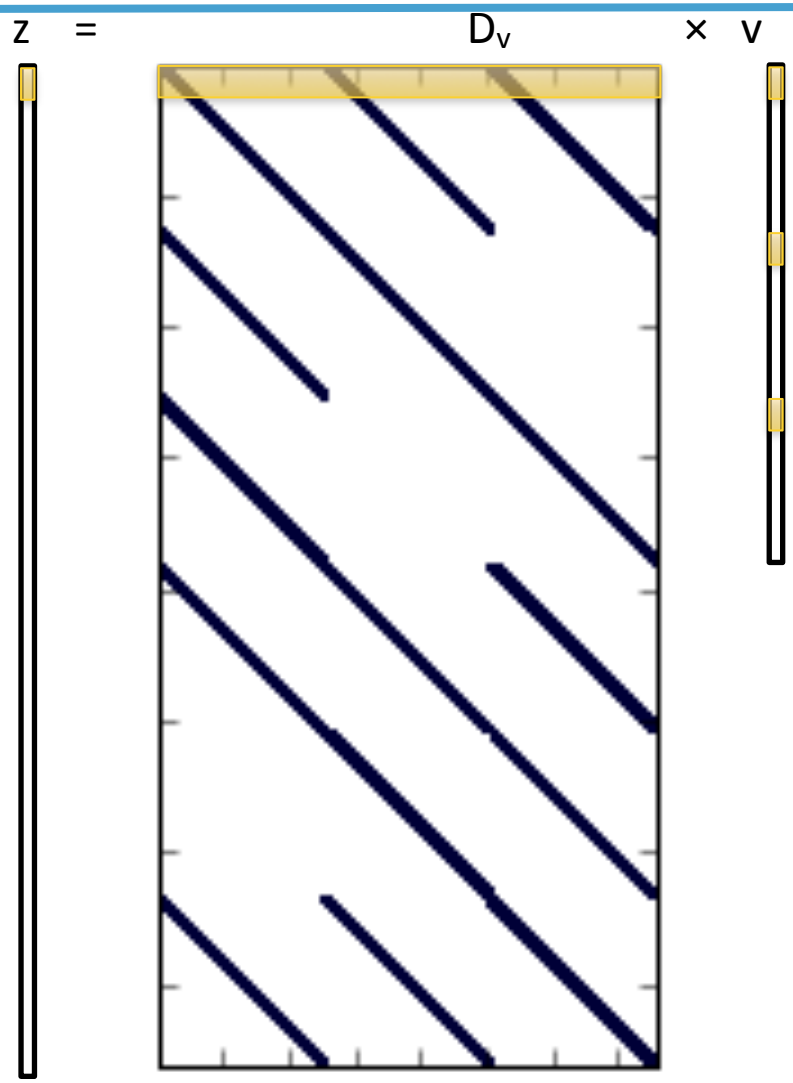


Fig. from: J. Zhang and L. Zhang, “Efficient CUDA Polynomial Preconditioned Conjugate Gradient Solver for Finite Element Computation of Elasticity Problems,” Math. Prob. in Eng., vol. 2013.

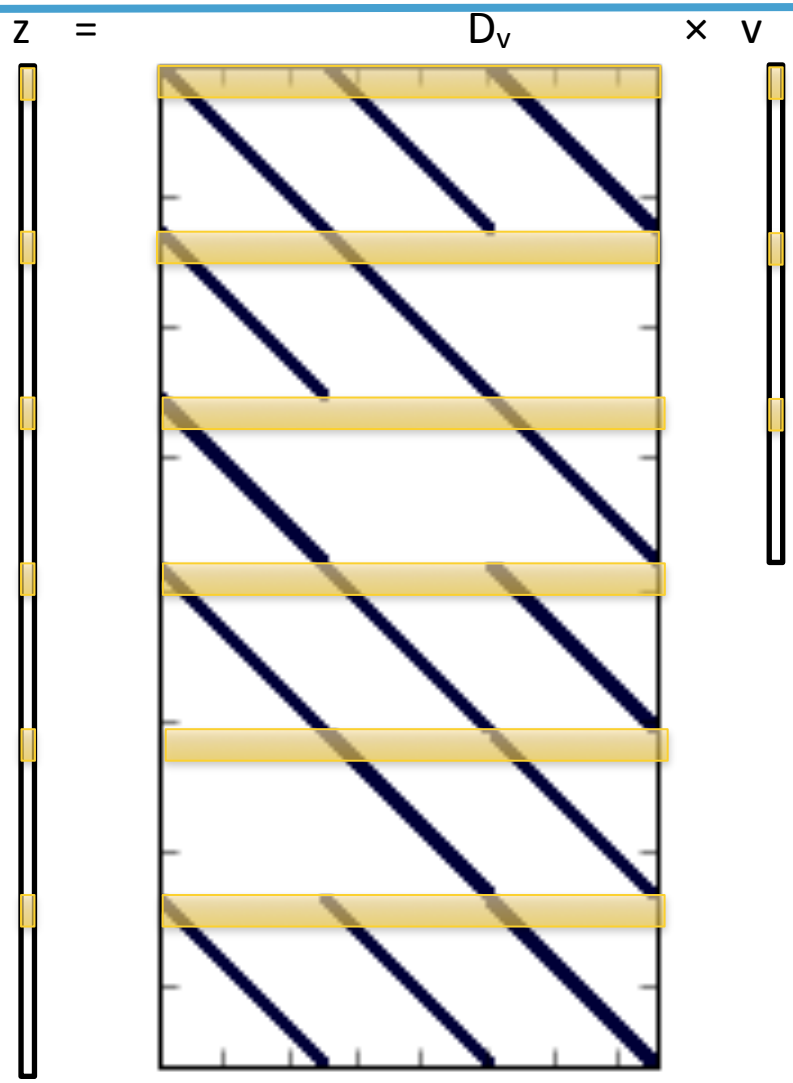
Spatial blocking vector data transfer model

- Block rows, threads working on single block $\Longrightarrow$ read data once
- block size needs to be tuned to cache size



Spatial blocking vector data transfer model

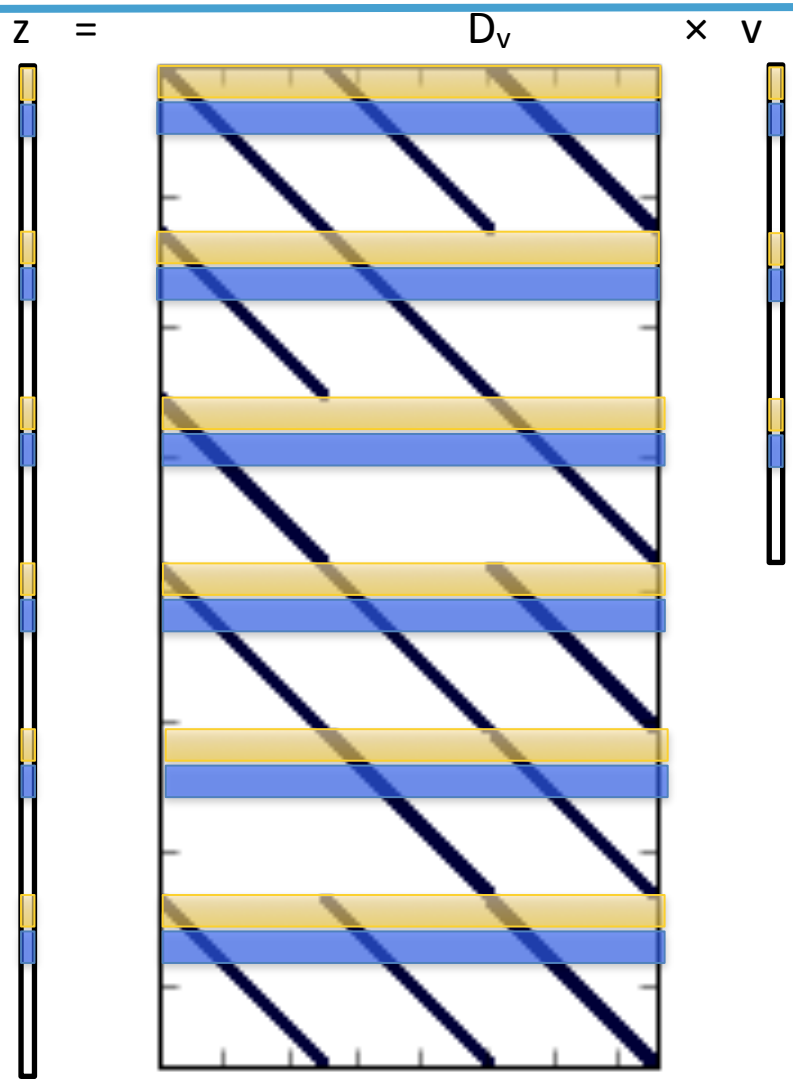
- Block rows, threads working on single block $\Longrightarrow$ read data once
- block size needs to be tuned to cache size



Spatial blocking vector data transfer model

- Block rows, threads working on single block $\Longrightarrow$ read data once
- block size needs to be tuned to cache size

	Total bytes	Read (words)	Write (no SS) (words)	Trasfer Improv.
D_v	240N	3N	$6N \cdot 2$	1.8x
D_t	192N	6N	$3N \cdot 2$	1.25x

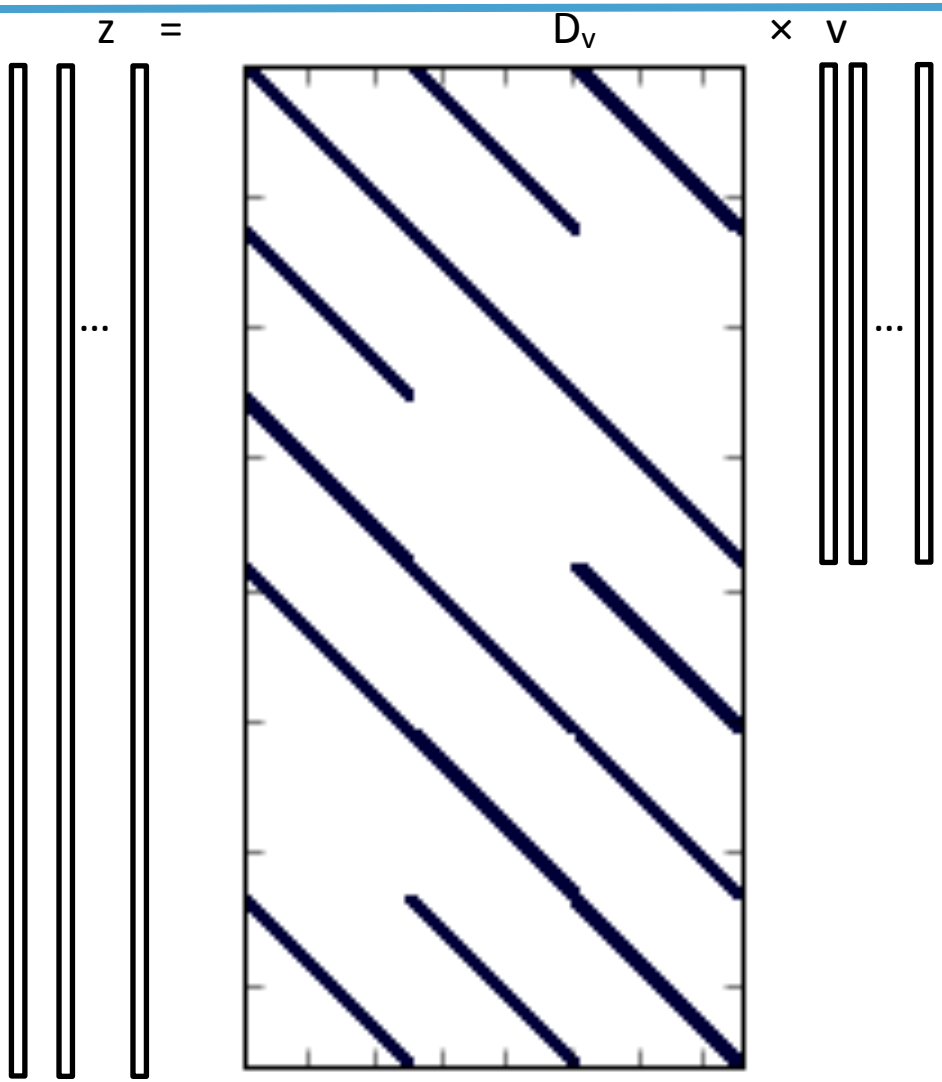


Arithmetic Intensities for $D_t \cdot Dv$

	Flops	Bytes	AI	Data improvement
Original	648N	2832N	0.23	-
SELL	576N	2592N	0.22	9%
Spt. Blk.	648N	2592N	0.25	9%
SELL + Spt. Blk.	576N	2352N	0.25	20%

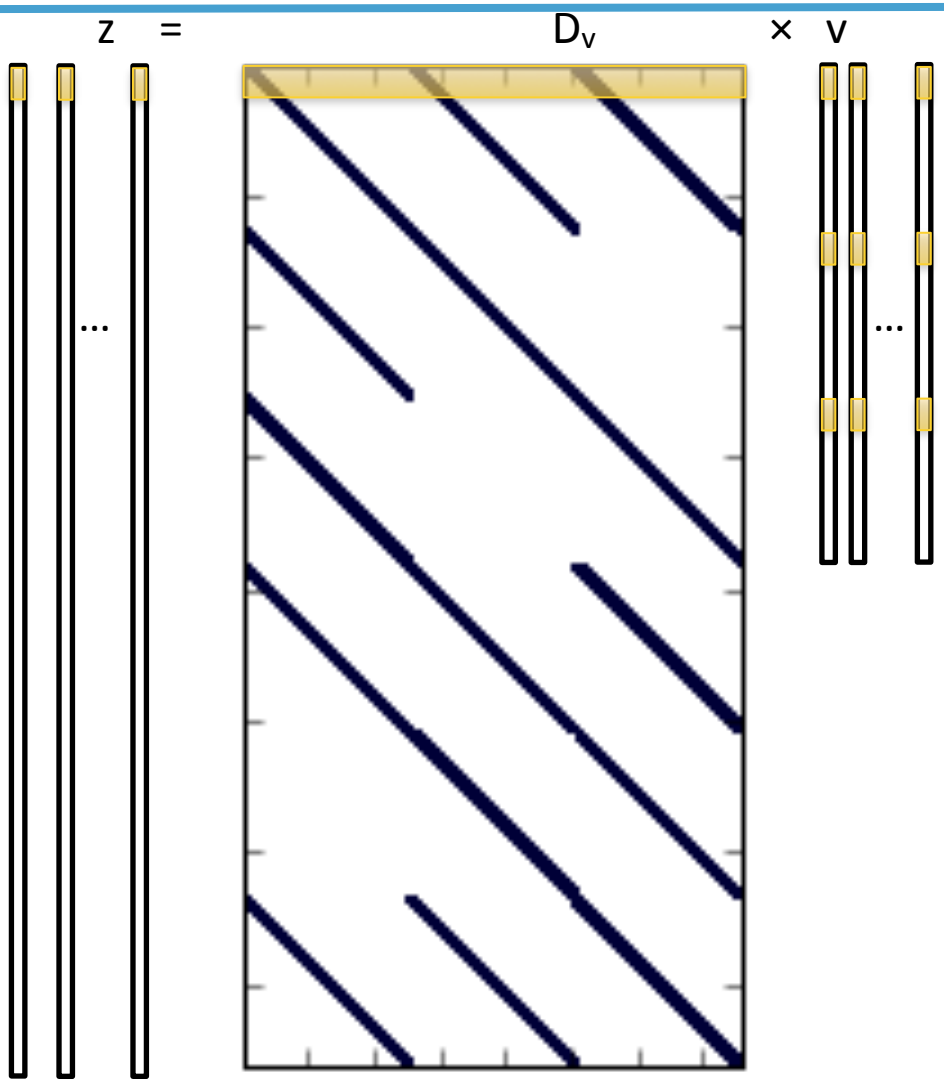
Multiple Right Hand Side (mRHS) blocking

- Forward solve in EMGeo is performed on many RHS (O(100))
- Load the matrix once for many RHSs
- Amortizes the cost of loading the matrix



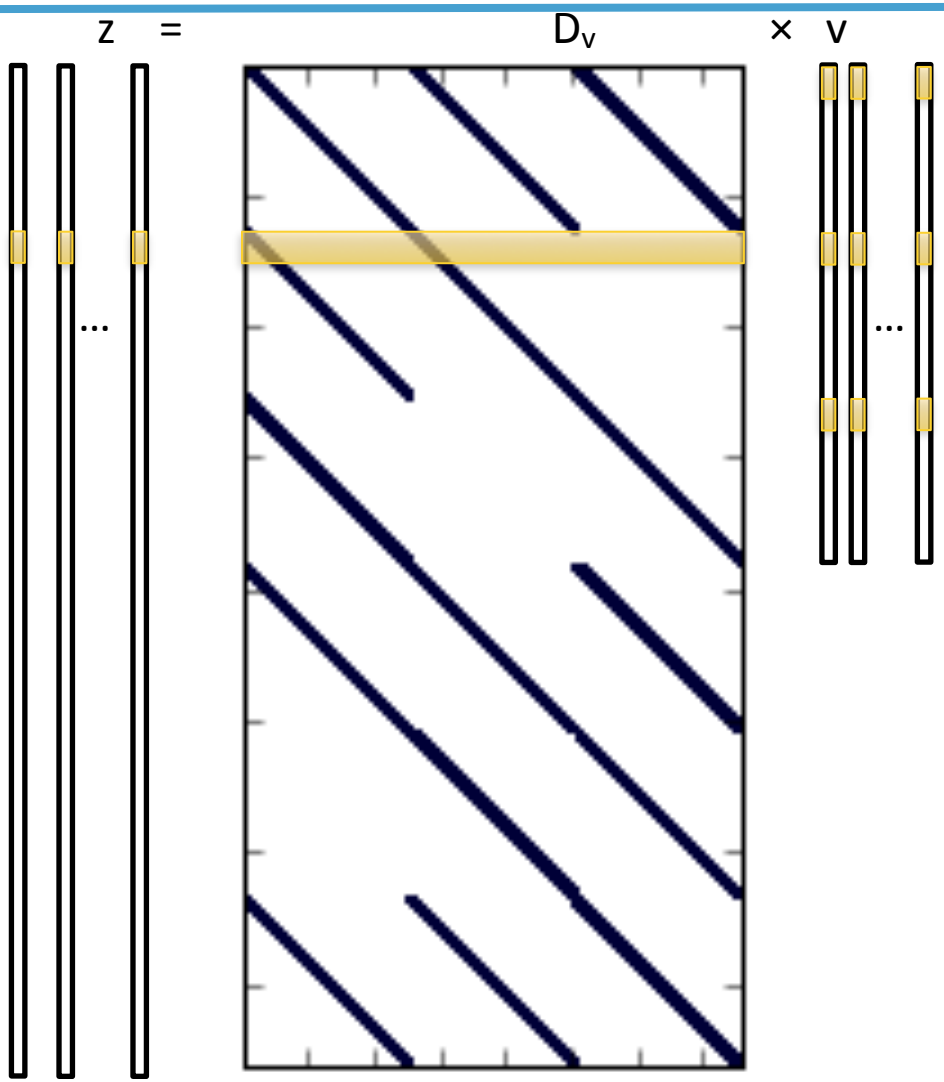
Multiple Right Hand Side (mRHS) blocking

- Forward solve in EMGeo is performed on many RHS (O(100))
- Load the matrix once for many RHSs
- Amortizes the cost of loading the matrix



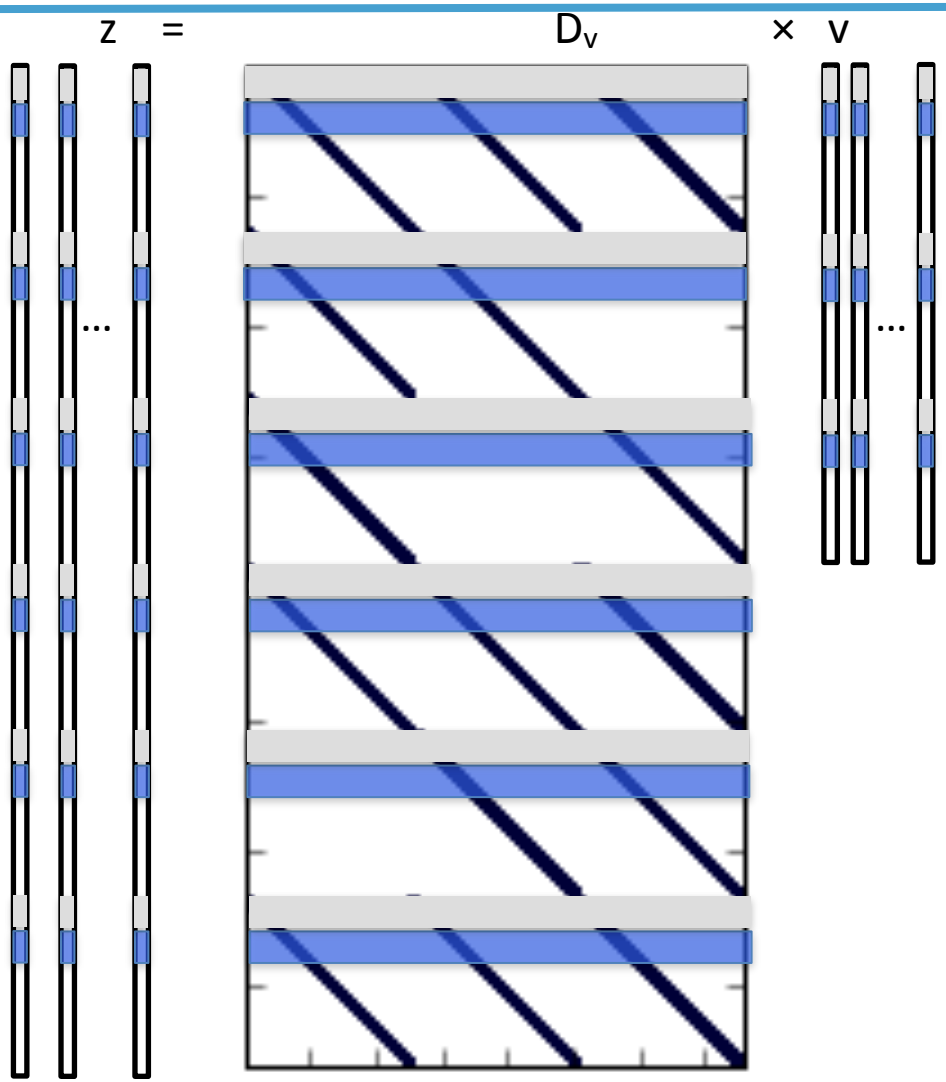
Multiple Right Hand Side (mRHS) blocking

- Forward solve in EMGeo is performed on many RHS (O(100))
- Load the matrix once for many RHSs
- Amortizes the cost of loading the matrix



Multiple Right Hand Side (mRHS) blocking

- Forward solve in EMGeo is performed on many RHS (O(100))
- Load the matrix once for many RHSs
- Amortizes the cost of loading the matrix



Kernels for D_V

D_V-Kernel w/o SB

```
!dir$ ASSUME_ALIGNED mat416:64, mat523:64, ind416:64, ind523:64, z:64, x:64
 !$omp parallel do private(c, i, rhs, ztmp, j)
 do im = 1, s
   do c = 0,2 ! loop over the matrix components
     i = im + c*s
     do rhs = 1,nRHS ! loop over RHS

       ! SELLPack slice 1
       ztmp =  (0.0d0, 0.0d0)
       do j = 1, 12
         ztmp = ztmp + mat416(j,i) * x(rhs, ind416(j,i))
       end do
       z(rhs, iorig416(i)) = ztmp

       ! SELLPack slice 2
       ztmp =  (0.0d0, 0.0d0)
       do j = 1, 8
         ztmp = ztmp + mat523(j,i) * x(rhs, ind523(j,i))
       end do
       z(rhs, iorig523(i)) = ztmp

     end do  ! RHS loop
   end do
 end do  ! loop over the matrix vector blocks
```

Kernels for D_V

D_V-Kernel with SB

```
!$omp parallel private(iblk_b, iblk_e, rhs, cb, c, ib, ie, ibt, iet, \
nth, tid, r, q, i, j, ztmp)
  do iblk_b = 1, s, blk_size !spatial blocking loop
    iblk_e = min(iblk_b + blk_size, s)
    do cb = 0,5 !
      ... ! compute the thread loop bounds explicitly
      if(cb .lt. 3) then ! first SELLPack slice
        do i = ibt, iet
          do rhs = 1,nRHS
            ztmp =  (0.0d0, 0.0d0)
            do j = 1, 12
              ztmp = ztmp + mat416(j,i) * x(rhs, ind416(j,i))
            end do
            z(rhs, iorig416(i)) = ztmp
          end do  ! RHS loop
        end do
      else ! Second SELLPack slice
        do i = ibt, iet
          do rhs = 1,nRHS
            ztmp =  (0.0d0, 0.0d0)
            do j = 1, 8
              ztmp = ztmp + mat523(j,i) * x(rhs, ind523(j,i))
            end do
            z(rhs, iorig523(i)) = ztmp
          end do  ! RHS loop
        end do
      endif ! SELLPack slices
    end do  !
  end do ! Spatial blocking loop
```

Kernels for D_V

D_V-Kernel with SB

```
!$omp parallel private(iblk_b, iblk_e, rhs, cb, c, ib, ie, ibt, iet, \
nth, tid, r, q, i, j, ztmp)
  do iblk_b = 1, s, blk_size !spatial blocking loop
    iblk_e = min(iblk_b + blk_size, s)
    do cb = 0,5 !
      ... ! compute the thread loop bounds explicitly
      if(cb .lt. 3) then ! first SELLPack slice
        do i = ibt, iet
          do rhs = 1,nRHS
            ztmp =  (0.0d0, 0.0d0)
            do j = 1, 12
              ztmp = ztmp + mat416(j,i) * x(rhs, ind416(j,i))
            end do
            z(rhs, iorig416(i)) = ztmp
          end do  ! RHS loop
        end do
      else ! Second SELLPack slice
        do i = ibt, iet
          do rhs = 1,nRHS
            ztmp =  (0.0d0, 0.0d0)
            do j = 1, 8
              ztmp = ztmp + mat523(j,i) * x(rhs, ind523(j,i))
            end do
            z(rhs, iorig523(i)) = ztmp
          end do  ! RHS loop
        end do
      endif ! SELLPack slices
    end do  !
  end do ! Spatial blocking loop
```

Reduction into temporary variable helps compiler to vectorize

Kernels for D_t

D_t-Kernel with SB and reduction

```
!$omp parallel private(iblk_b, iblk_e, rhs, c, ib, ie, ix, j, ii, vtmp)
  do iblk_b = 1, s, blk_size !spatial blocking loop
    iblk_e = min(iblk_b + blk_size, s)
    do c = 0,2 ! loop over the matrix vector blocks
      ib = iblk_b + c*s
      ie = iblk_e + c*s

      !$omp do reduction(+:normr)
      do ix = ib, ie
        do rhs = 1,nRHS
          vtmp = DCMPLX(0.0D0)
          do j=1,ndiag
              vtmp = vtmp + DT(j,ix)*z(rhs, i_DT(j,ix))
          enddo
          v(rhs, ix) = vtmp
          ii=orig_V(ix)
          G(rhs, ix)=U(rhs, ii)-v(rhs, ix)
          normr(rhs) = normr(rhs) + G(rhs, ix)*conjg(G(rhs, ix))
        end do  ! RHS loop
      end do
      !$omp end do nowait
    end do  ! loop over the matrix vector blocks
  end do ! blocking loop
!$omp end parallel
```

Kernels for D_t

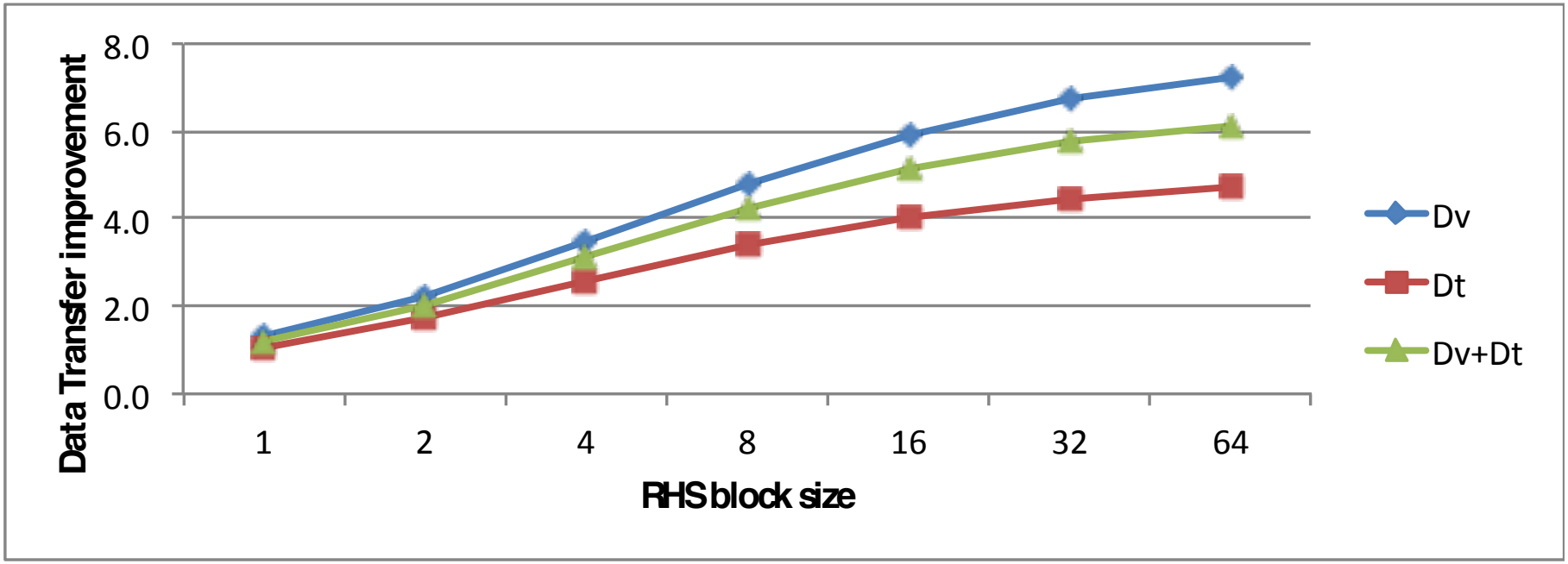
D_t-Kernel with SB and reduction

```
!$omp parallel private(iblk_b, iblk_e, rhs, c, ib, ie, ix, j, ii, vtmp)
  do iblk_b = 1, s, blk_size !spatial blocking loop
    iblk_e = min(iblk_b + blk_size, s)
    do c = 0,2 ! loop over the matrix vector blocks
      ib = iblk_b + c*s
      ie = iblk_e + c*s

      !$omp do reduction(+:normr)
      do ix = ib, ie
        do rhs = 1,nRHS
          vtmp = DCMPLX(0.0D0)
          do j=1,ndiag
              vtmp = vtmp + DT(j,ix)*z(rhs, i_DT(j,ix))
          enddo
          v(rhs, ix) = vtmp
          ii=orig_V(ix)
          G(rhs, ix)=U(rhs, ii)-v(rhs, ix)
          normr(rhs) = normr(rhs) + G(rhs, ix)*conjg(G(rhs, ix))
        end do  ! RHS loop
      end do
      !$omp end do nowait
    end do  ! loop over the matrix vector blocks
  end do ! blocking loop
!$omp end parallel
```

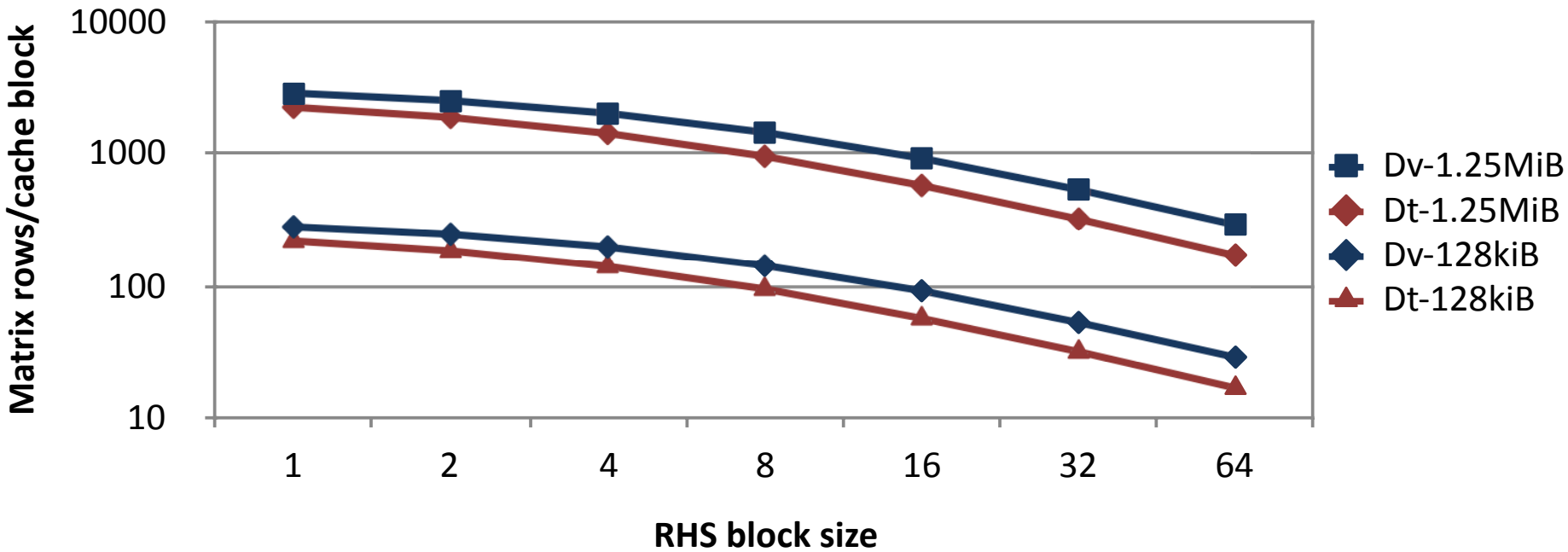
Maximize cache reuse by pooling BLAS-like ops

mRHS data transfer improvement model



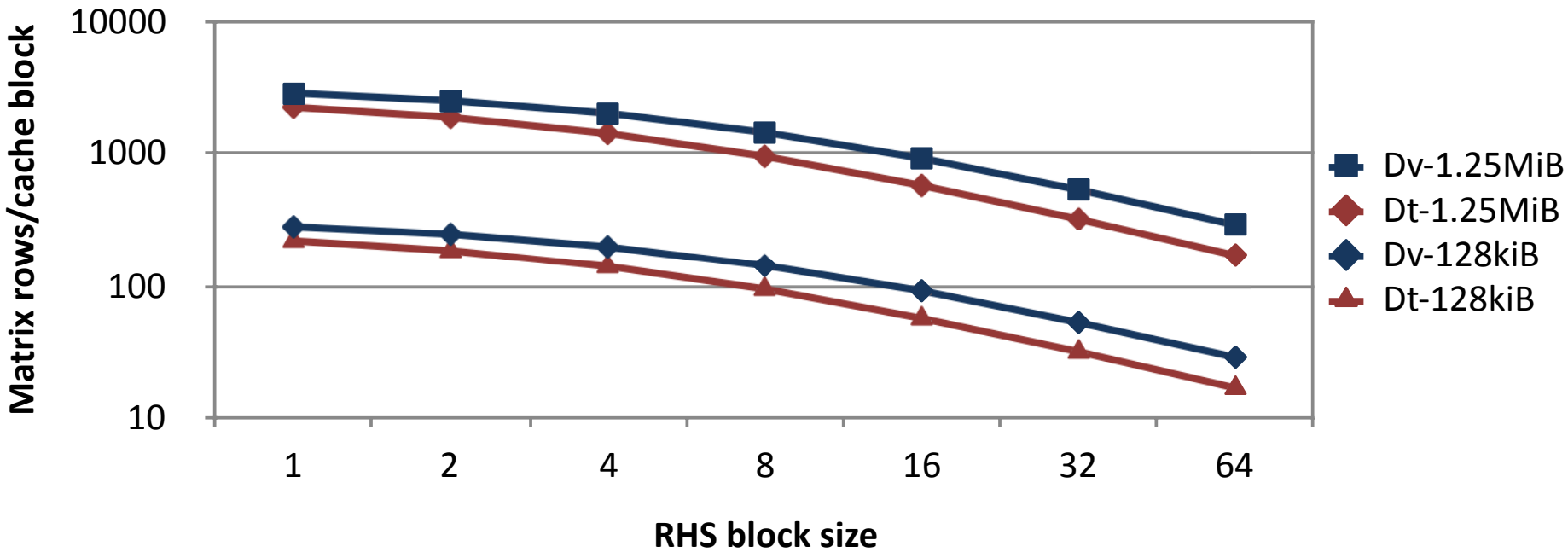
$$\text{Improvement Factor} = \frac{M_{\text{RHS}}(M_b + V_b)}{M_b + V_b M_{\text{RHS}}}$$

mRHS Cache block size requirements



$$\text{rows per block} = \frac{\text{cache size}}{\frac{vector\ components}{\text{row}} \cdot M_{\text{RHS}} \cdot 16 + 2 \cdot \frac{\text{NNZ}}{\text{row}} \cdot (16 + 4)}$$

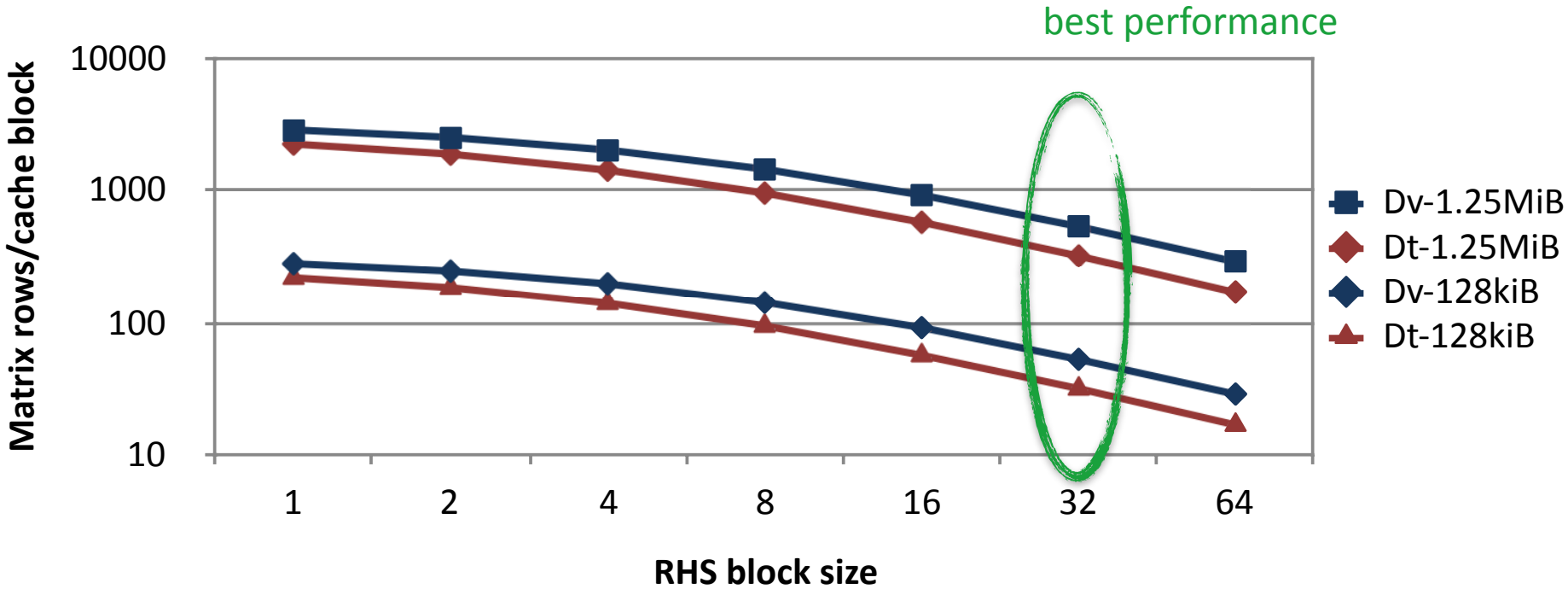
mRHS Cache block size requirements



$$\text{rows per block} = \frac{\text{cache size}}{\frac{vector\ components}{\text{row}} \cdot M_{\text{RHS}} \cdot 16 + 2 \cdot \frac{\text{NNZ}}{\text{row}} \cdot (16 + 4)}$$

safety factor for cache optimization

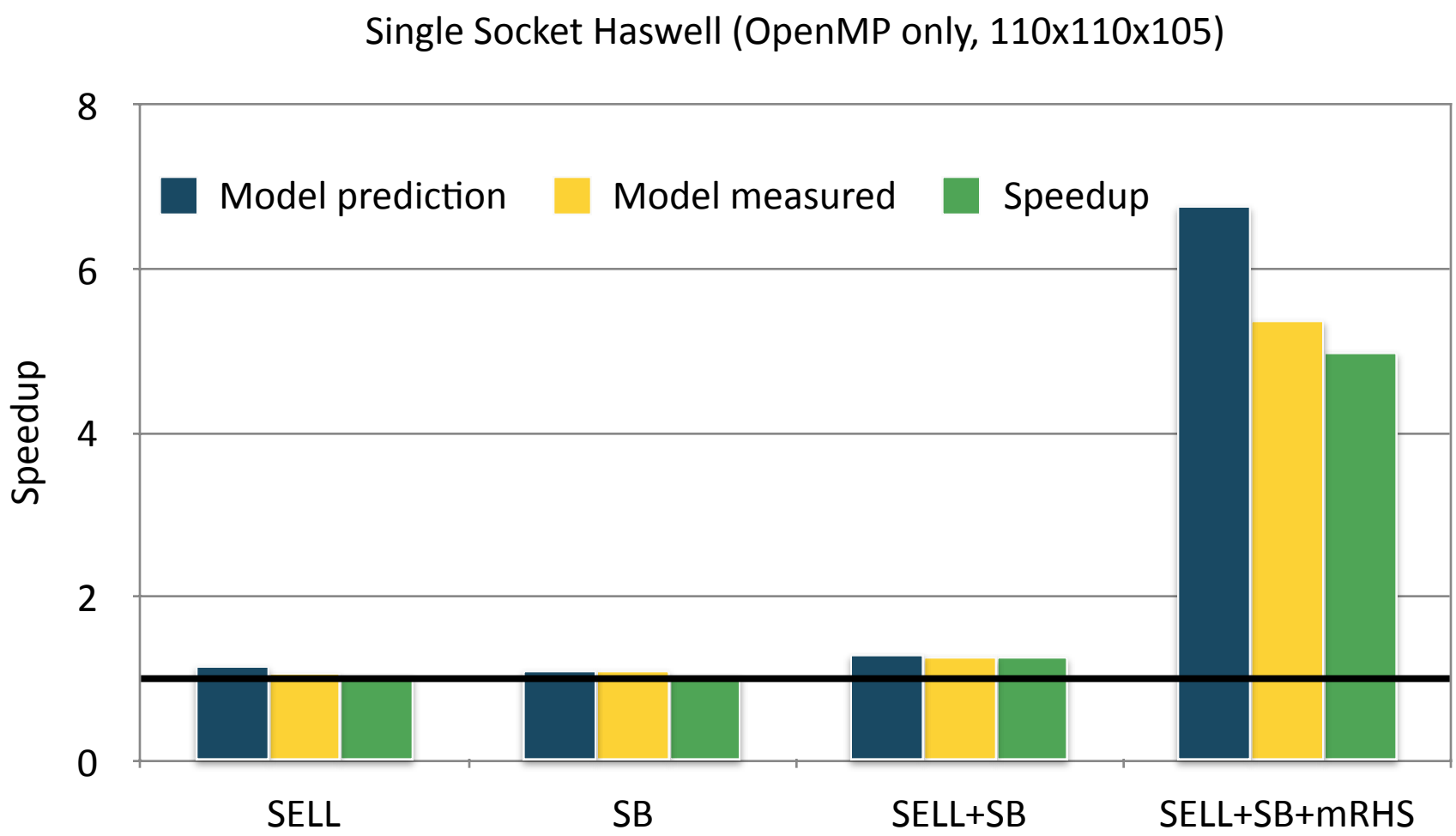
mRHS Cache block size requirements



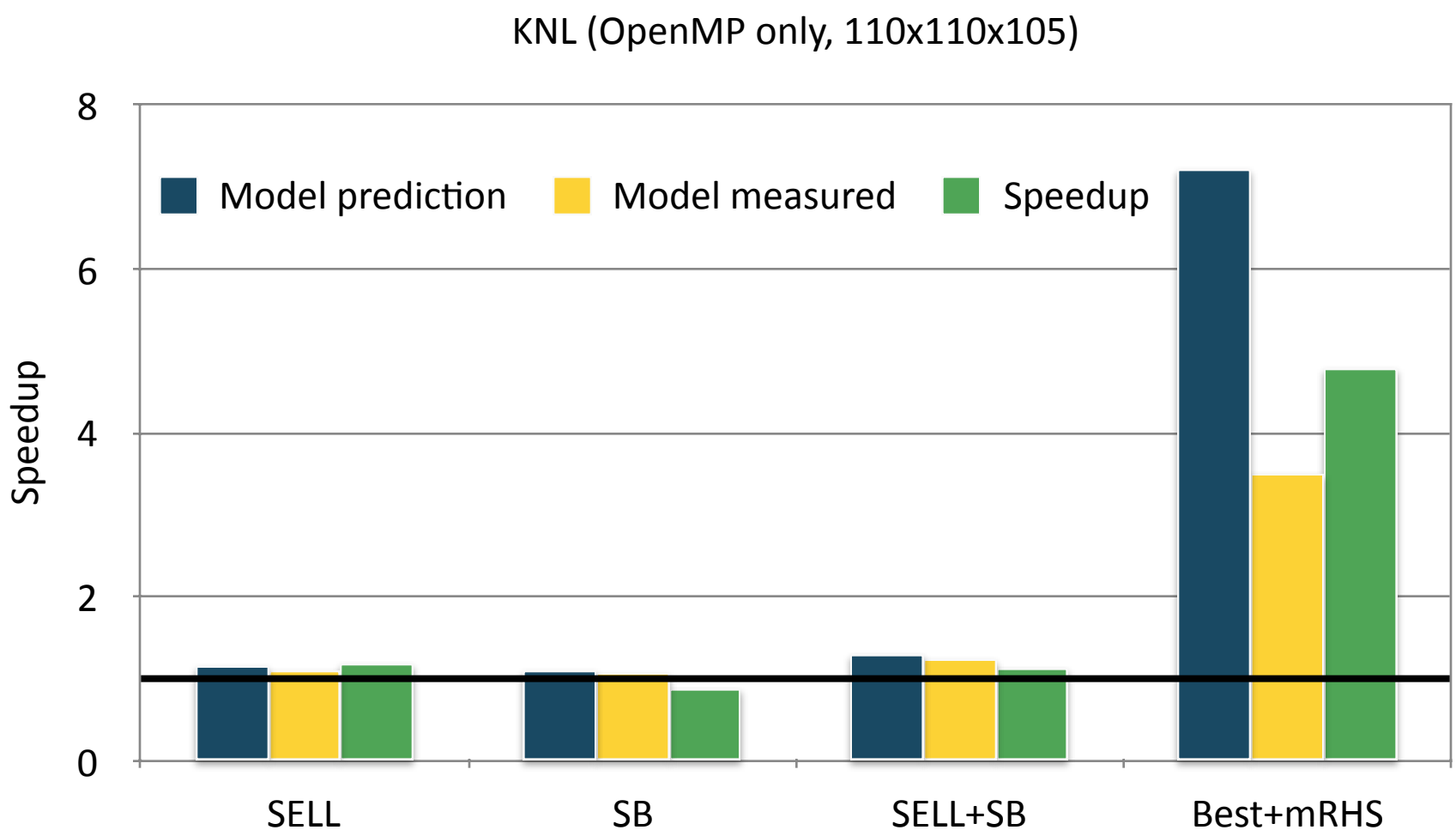
$$\text{rows per block} = \frac{\text{cache size}}{\frac{vector\ components}{\text{row}} \cdot M_{\text{RHS}} \cdot 16 + 2 \cdot \frac{\text{NNZ}}{\text{row}} \cdot (16 + 4)}$$

safety factor for cache optimization

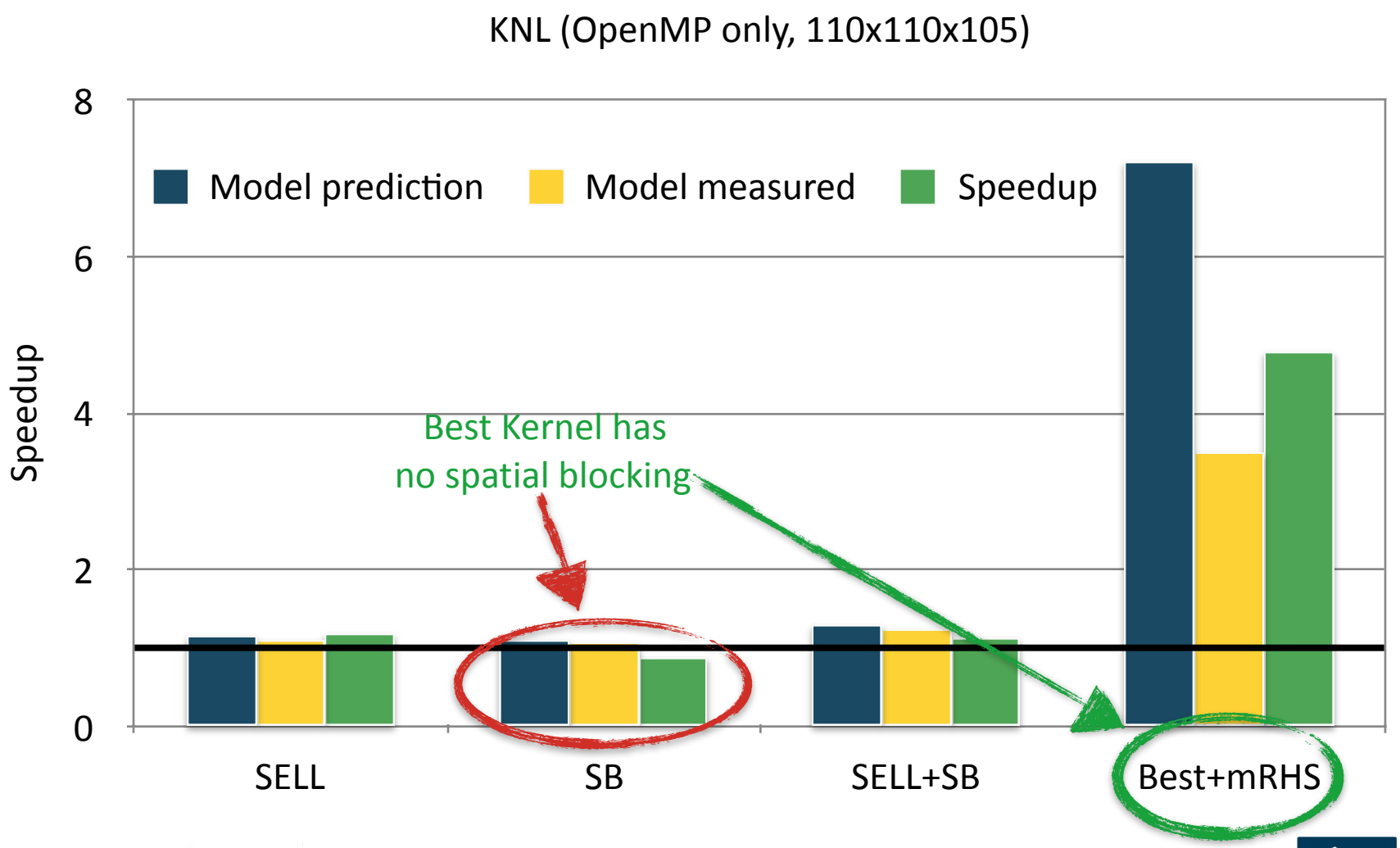
Impact of our optimizations on D_v



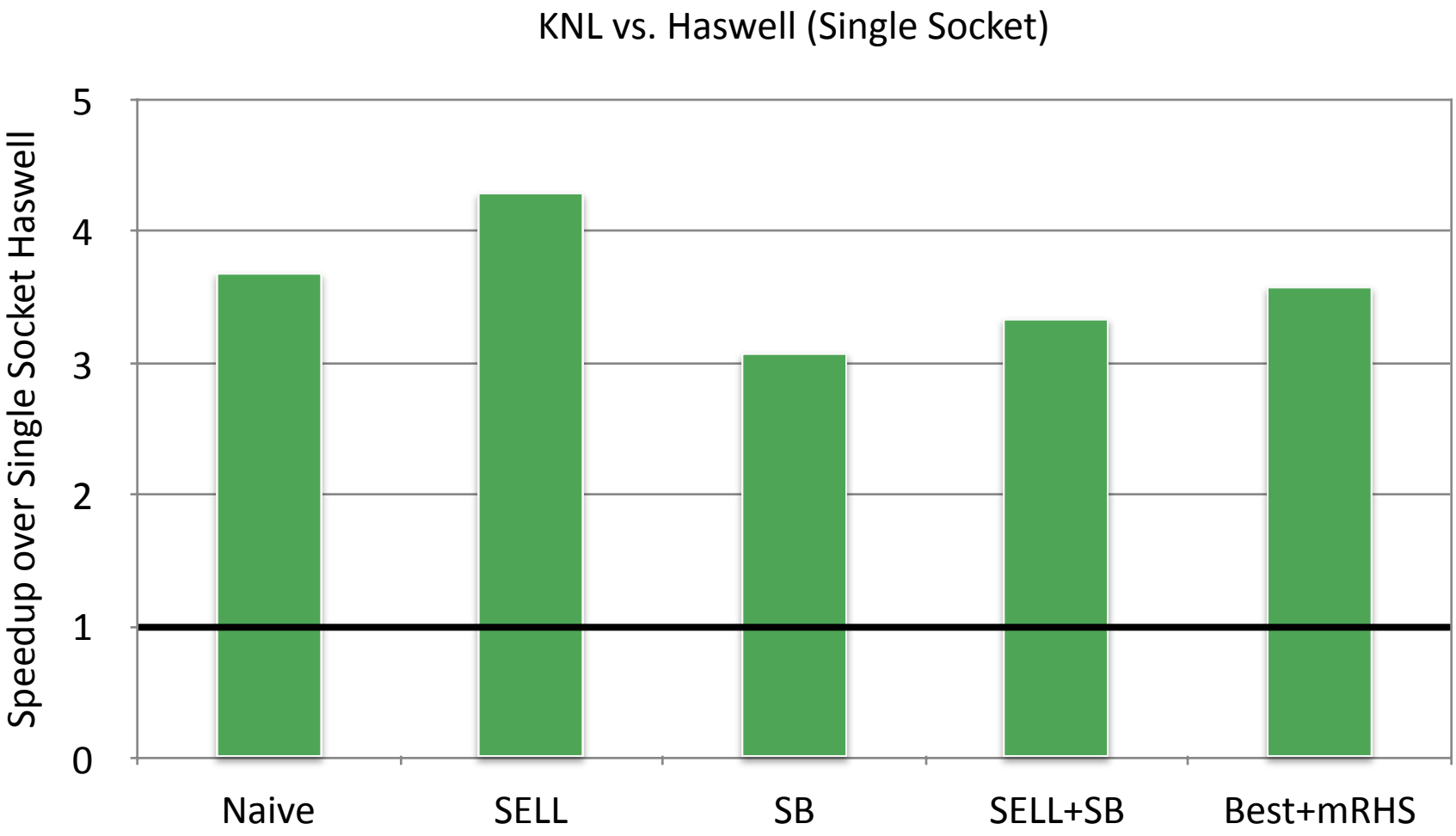
Impact of our optimizations on D_v



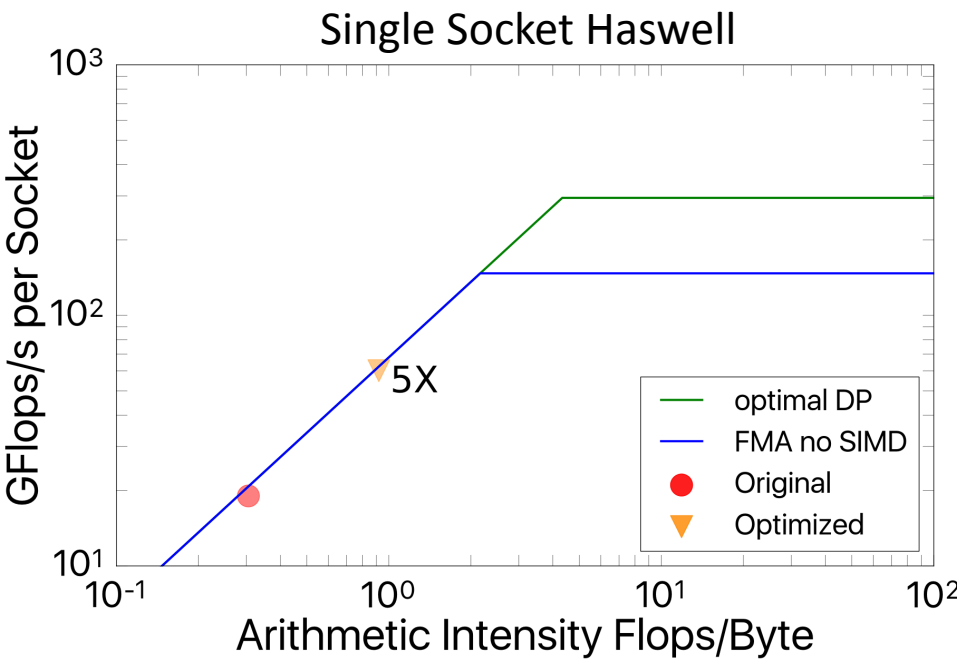
Impact of our optimizations on D_v

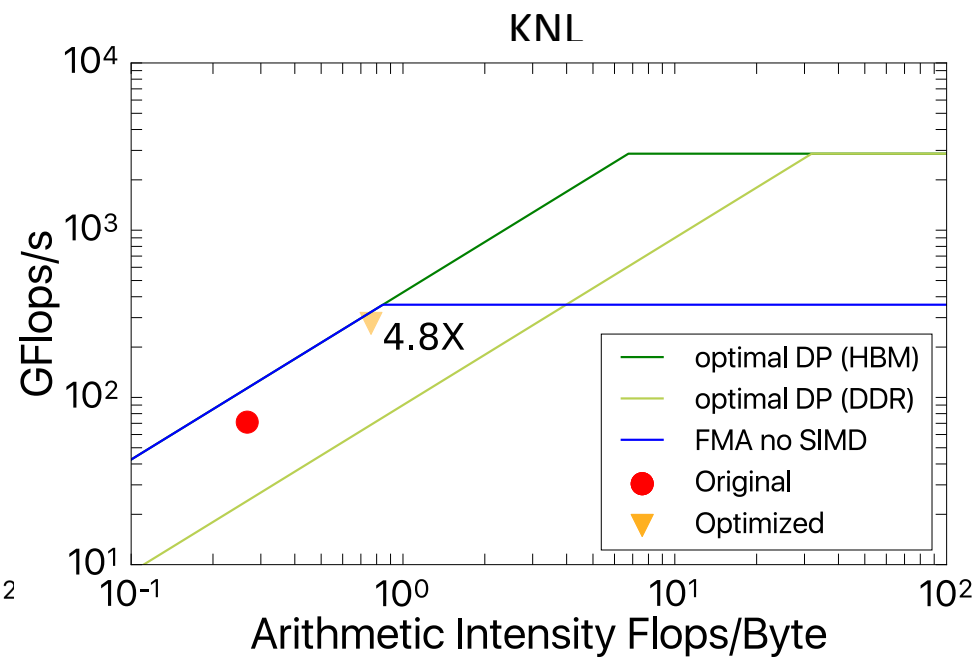


Impact of our optimizations on D_v



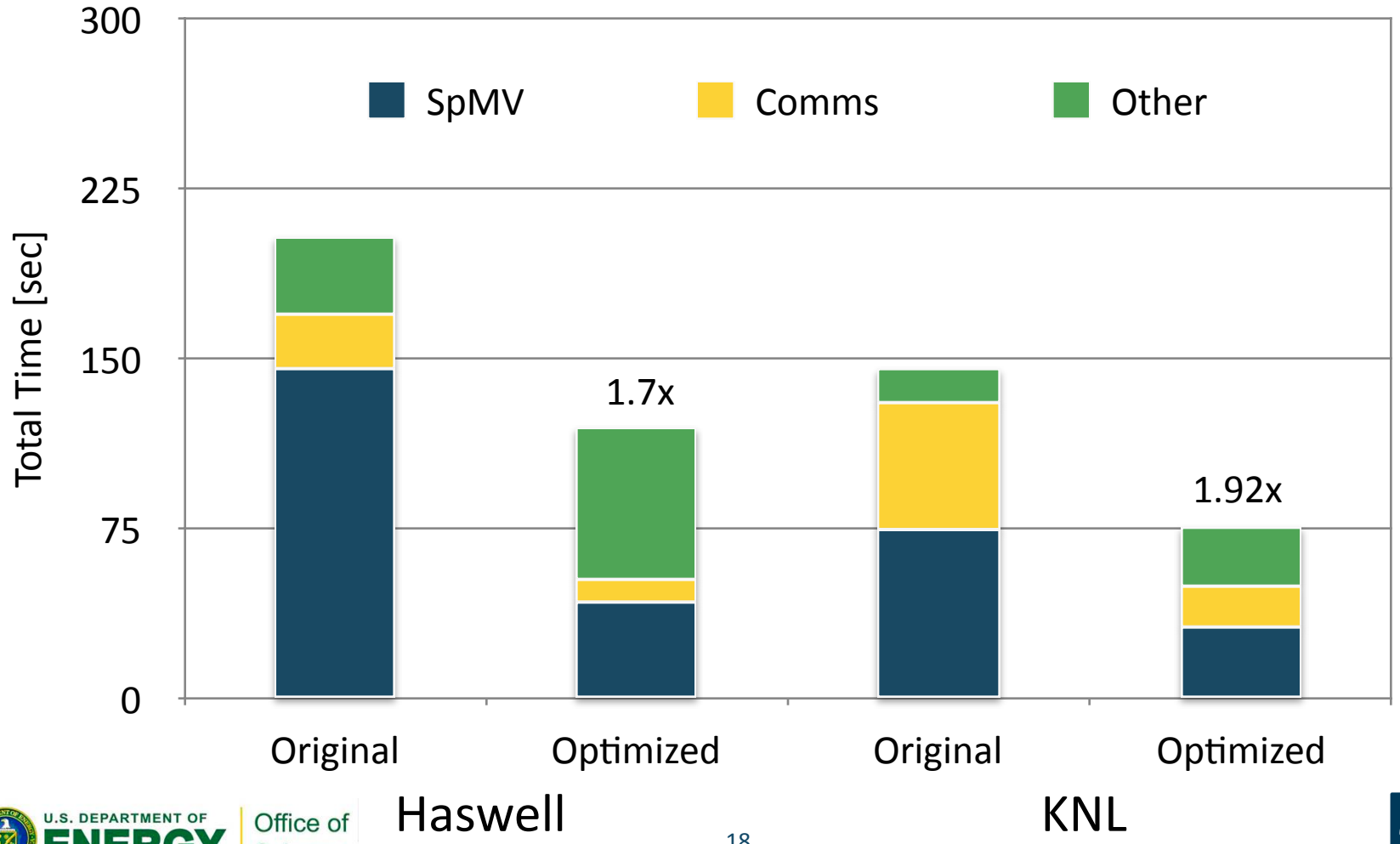
D_V-Kernel in the Roofline Model



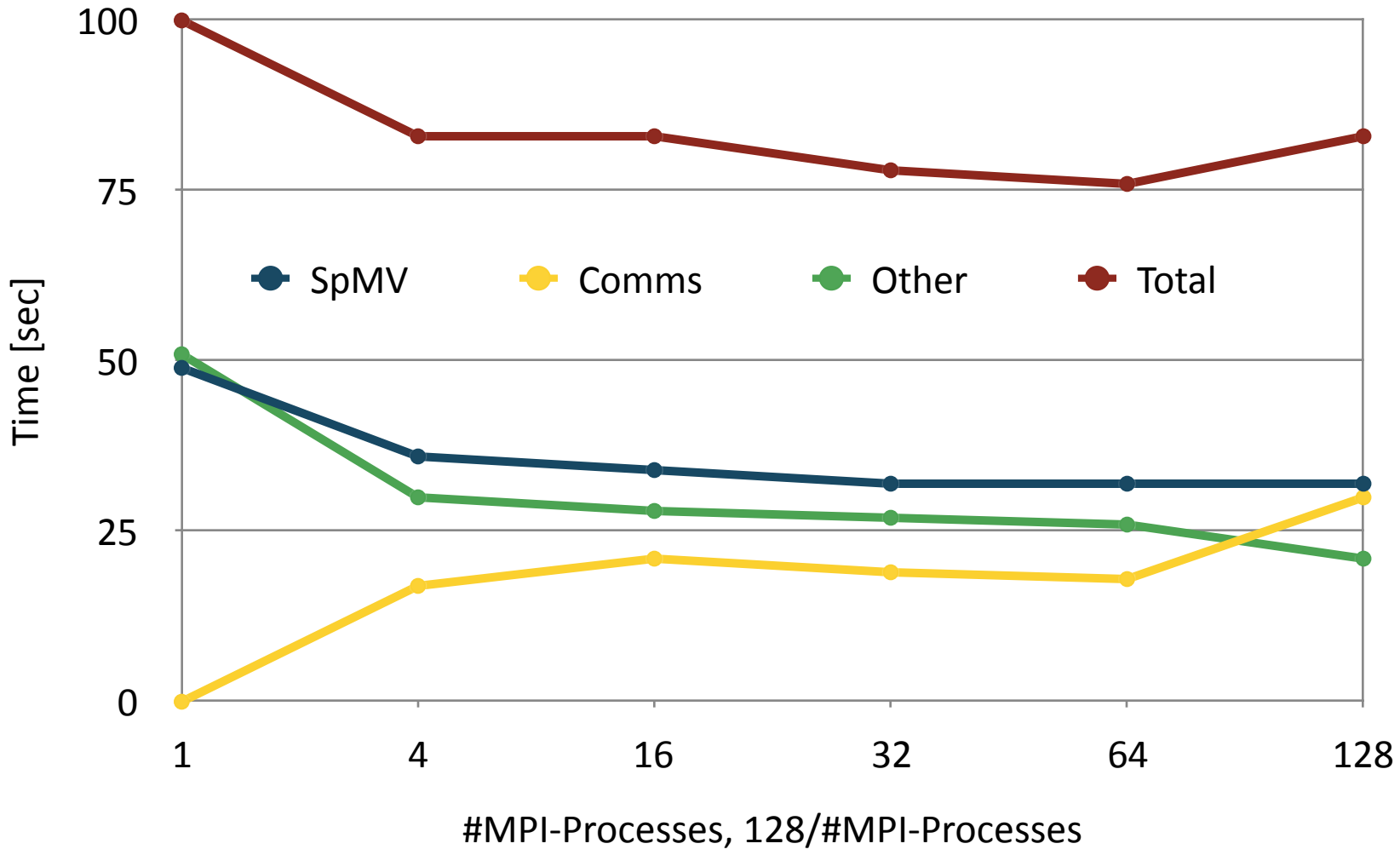


EMGeo IDR improvements – Single node

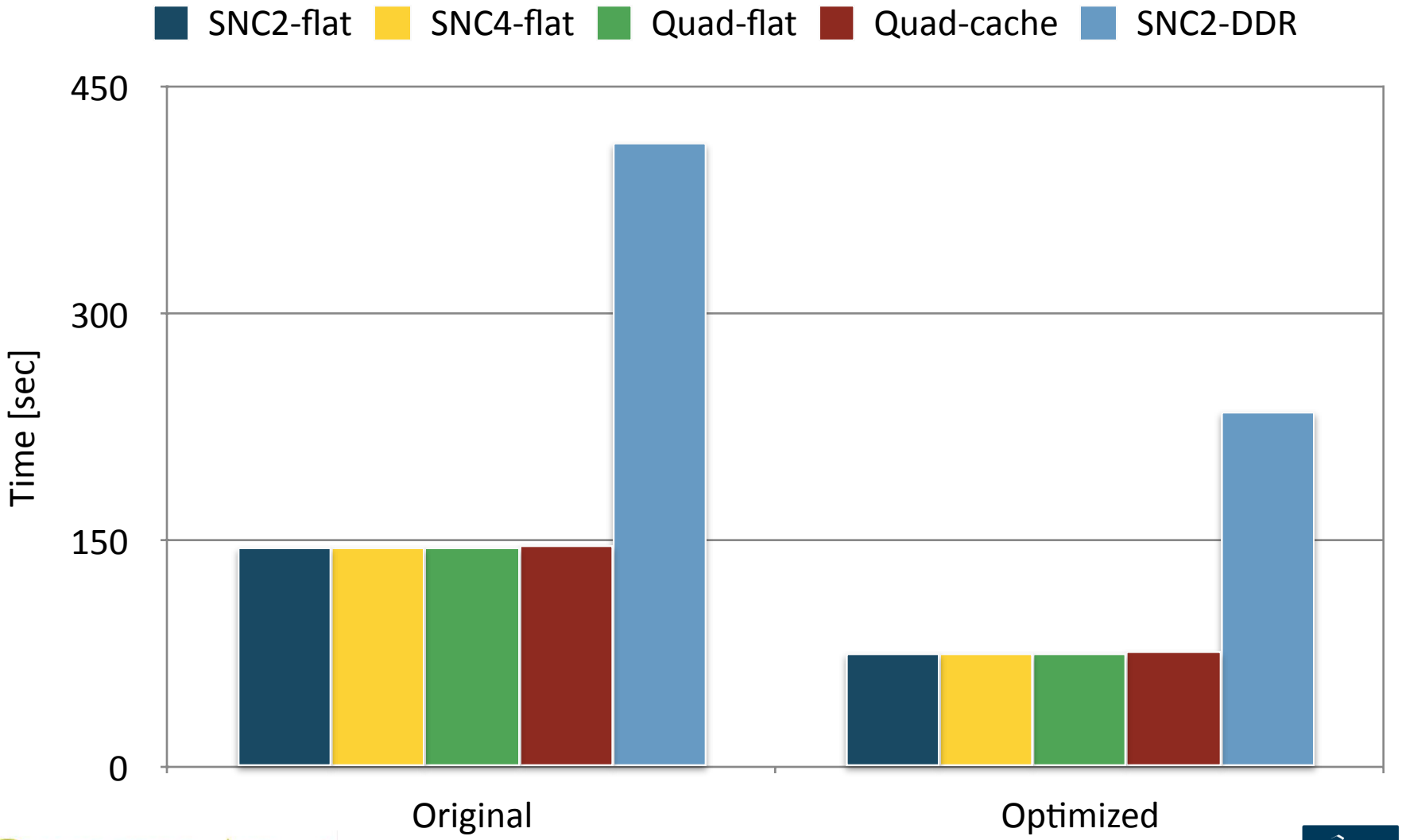
- 32 RHS, max. iter. 500, and grid size 100x50x50



KNL MPI vs. OpenMP



KNL modes and MCDRAM vs. DDR4



Conclusion

- Used performance modeling to
  - identify relevant optimizations and
  - understand the optimization quality and problems
- Spatial blocking works on Haswell, but not on KNL
- RHS blocking provides significant performance improvements and prepares the code for:
  - blocked solvers
  - overlapping computations with communications

Current/Future work

- Overlap the comp/comm of independent RHS
- Explore blocked solvers
- Consider using the combined matrix $D_t \cdot D_v$ with mRHS blocking

Thank you