BLOCK SOLVERS FOR MULTIPLE RIGHT HAND SIDES ON NVIDIA GPUS

Mathias Wagner, Lattice 2016

NVIDIA

AGENDA

QUDA update

Dslash for multiple rhs

Block Krylov solvers

UPDATE ON QUDA

QUDA

“QCD on CUDA” – open source, BSD license

Effort started at Boston University in 2008, now in wide use as the GPU backend for BQCD, Chroma, CPS, MILC, TIFR, etc.

Latest release 0.8.0 (8th February 2016)

Provides:

- Various solvers for all major fermionic discretizations, with multi-GPU support
- Additional performance-critical routines needed for gauge-field generation

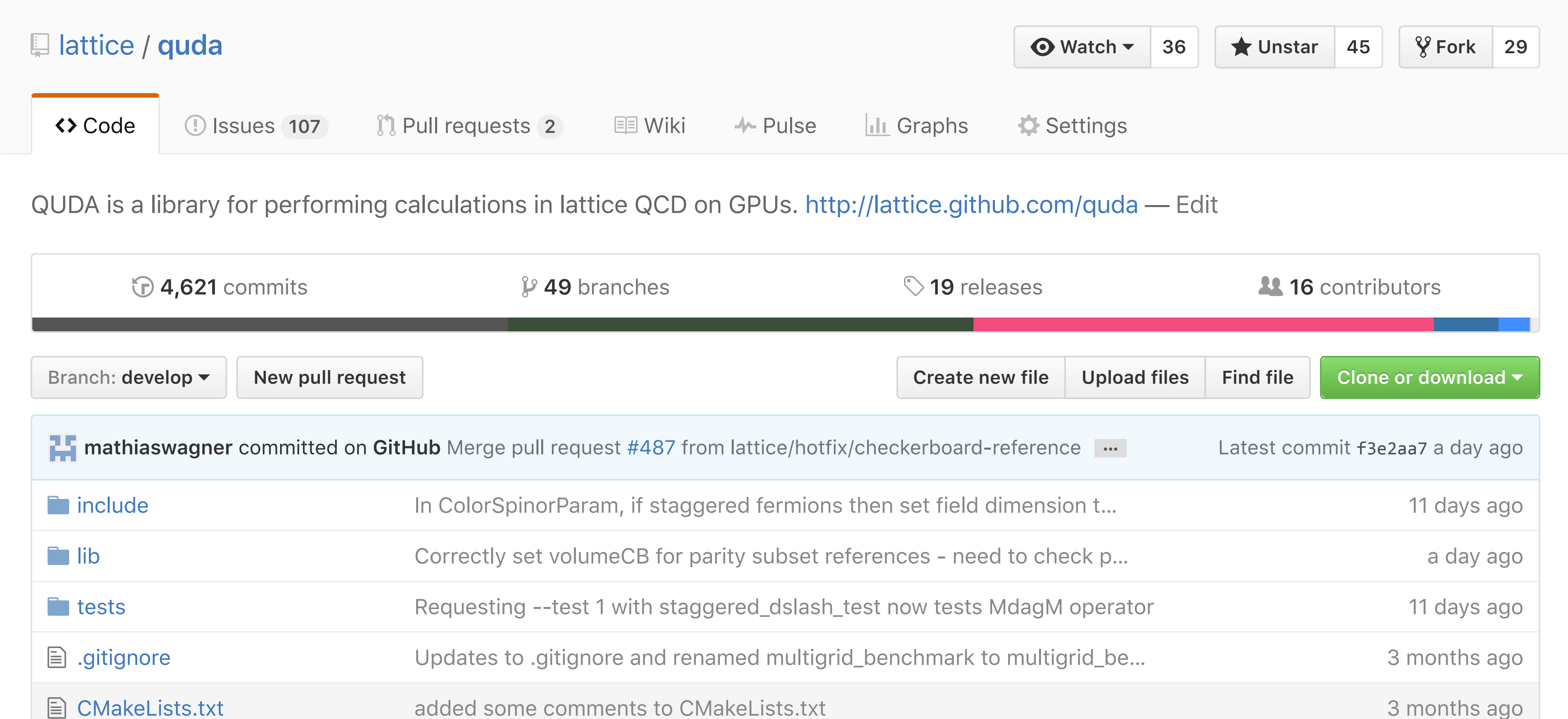
Maximize performance

- Exploit physical symmetries to minimize memory traffic
- Mixed-precision methods
- Autotuning for high performance on all CUDA-capable architectures
- Domain-decomposed (Schwarz) preconditioners for strong scaling
- Eigenvector and deflated solvers (Lanczos, EigCG, GMRES-DR)
- Multigrid solvers for optimal convergence

A research tool for how to reach the exascale

QUDA - LATTICE QCD ON GPUS

http://lattice.github.com/quda



QUDA AUTHORS

Ron Babich (NVIDIA)
Michael Baldhauf (Regensburg)
Kip Barros (LANL)
Rich Brower (Boston University)
Nuno Cardoso (Lisbon)
Kate Clark (NVIDIA)
Michael Cheng (Boston University)
Carleton DeTar (Utah University)
Justin Foley (Utah -> NIH)
Joel Giedt (Rensselaer Polytechnic Institute)
Arjun Gambhir (William and Mary)
Steve Gottlieb (Indiana University)
Dean Howarth (Rensselaer Polytechnic Institute)
Bálint Joó (Jlab)
Hyung-Jin Kim (BNL -> Samsung)
Claudio Rebbi (Boston University)
Guochun Shi (NCSA -> Google)
Mario Schröck (INFN)
Alexei Strelchenko (FNAL)
Alejandro Vaquero (INFN)
Mathias Wagner (NVIDIA)
Frank Winter (Jlab)

SOLVERS FOR MULTIPLE RIGHT HAND SIDES

CONJUGATE GRADIENT

just as a reminder

procedure CG

$r^{(0)} = b - Ax^{(0)}$

$p^{(0)} = r^{(0)}$

for $k = 1, 2, \ldots$ until converged **do**

$z^{(k-1)} = Ap^{(k-1)}$

$$\alpha^{(k-1)} = \frac{\left|r^{(k-1)}\right|^2}{\left\langle (p^{(k-1)}), z^{(k-1)} \right\rangle}$$

$x^{(k)} = x(k-1) + \alpha^{(k-1)} p^{(k-1)}$

$r^{(k)} = r^{(k-1)} - \alpha^{(k-1)} z^{(k-1)}$

$$\beta^{(k-1)} = \frac{\left|r^{(k-1)}\right|^2}{\left|r^{(k)}\right|^2}$$

$p^{(k)} = r^{(k)} + \beta^{(k-1)} p^{(k-1)}$

end for

end procedure

QCD PERFORMANCE LIMITERS

QCD is **memory bandwidth bound**

Dslash arithmetic intensity for HISQ ~ 0.7

QCD PERFORMANCE LIMITERS

QCD is memory bandwidth bound

Dslash arithmetic intensity for HISQ ~ 0.7

exploit SU(3) symmetry:

reconstruct gauge field from 8/12 floats

QCD PERFORMANCE LIMITERS

QCD is memory bandwidth bound

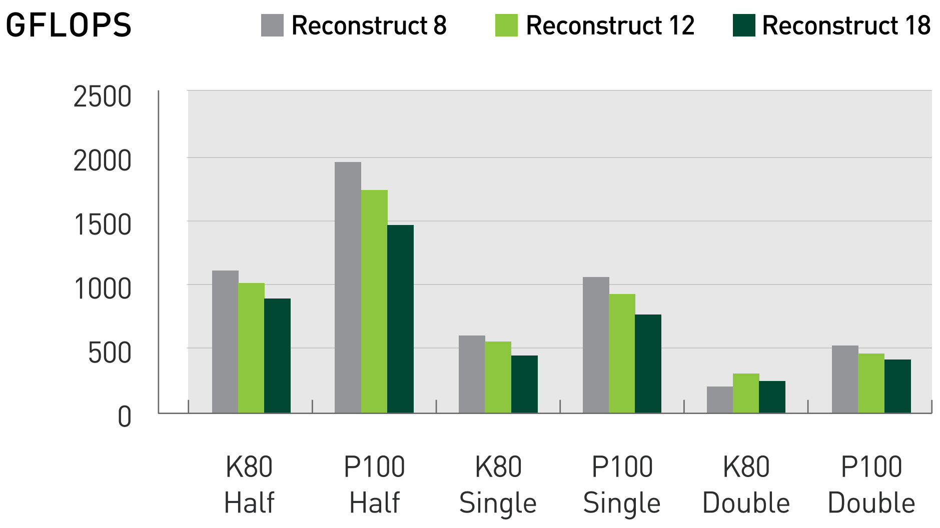
Dslash arithmetic intensity for HISQ ~ 0.7

exploit SU(3) symmetry:

reconstruct gauge field from 8/12 floats

WILSON CLOVER DSLASH

Volume = 32^4



QCD PERFORMANCE LIMITERS

QCD is **memory bandwidth bound**

Dslash arithmetic intensity for HISQ ~ 0.7

exploit SU(3) symmetry:

reconstruct gauge field from 8/12 floats

Smearing kills symmetry: stuck with 18 floats

QCD PERFORMANCE LIMITERS

QCD is **memory bandwidth bound**

Dslash arithmetic intensity for HISQ ~ 0.7

exploit SU(3) symmetry:

reconstruct gauge field from 8/12 floats

Smearing kills symmetry: stuck with 18 floats

Reuse gauge field for multiple rhs

QCD PERFORMANCE LIMITERS

QCD is memory bandwidth bound

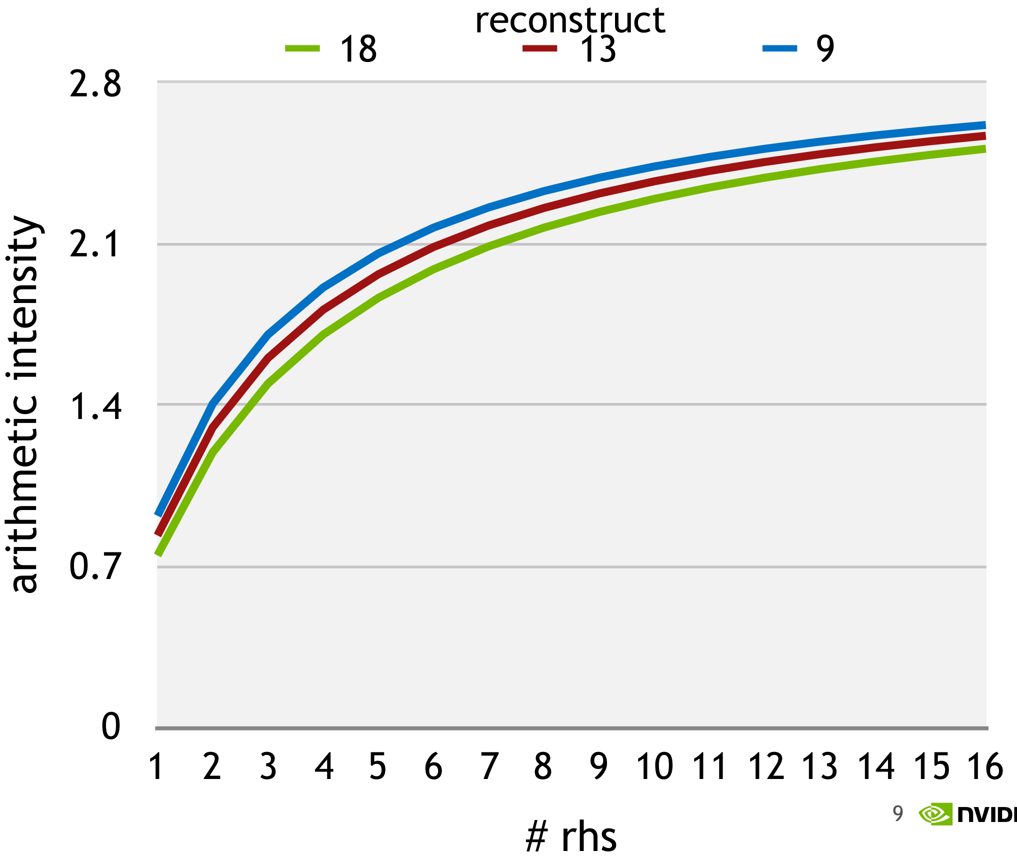
Dslash arithmetic intensity for HISQ ~ 0.7

exploit SU(3) symmetry:

reconstruct gauge field from 8/12 floats

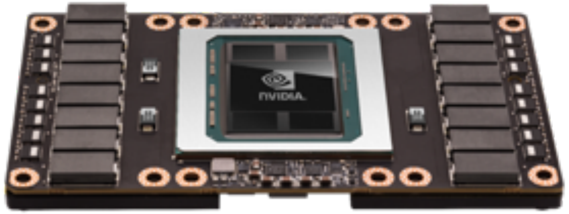
Smearing kills symmetry: stuck with 18 floats

Reuse gauge field for multiple rhs



TESLA PASCAL P100

Tesla P100
for NVLink-enabled Servers

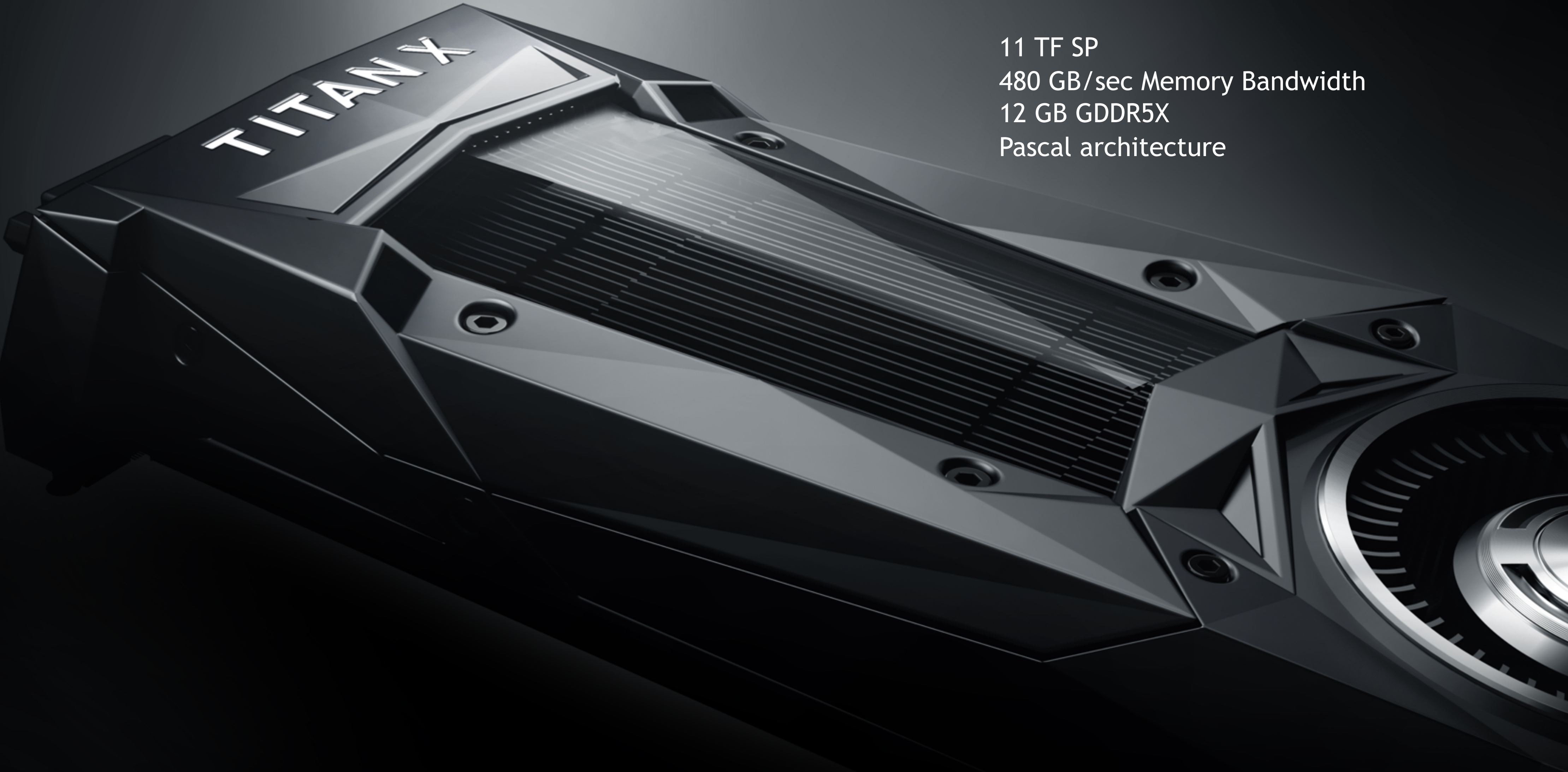


5.3 TF DP · 10.6 TF SP · 21 TF HP
720 GB/sec Memory Bandwidth
16 GB HBM2

Tesla P100
for PCIe-Based Servers

4.7 TF DP · 9.3 TF SP · 18.7 TF HP
Config 1: 16 GB (HBM2), 720 GB/sec
Config 2: 12 GB (HBM2), 540 GB/sec

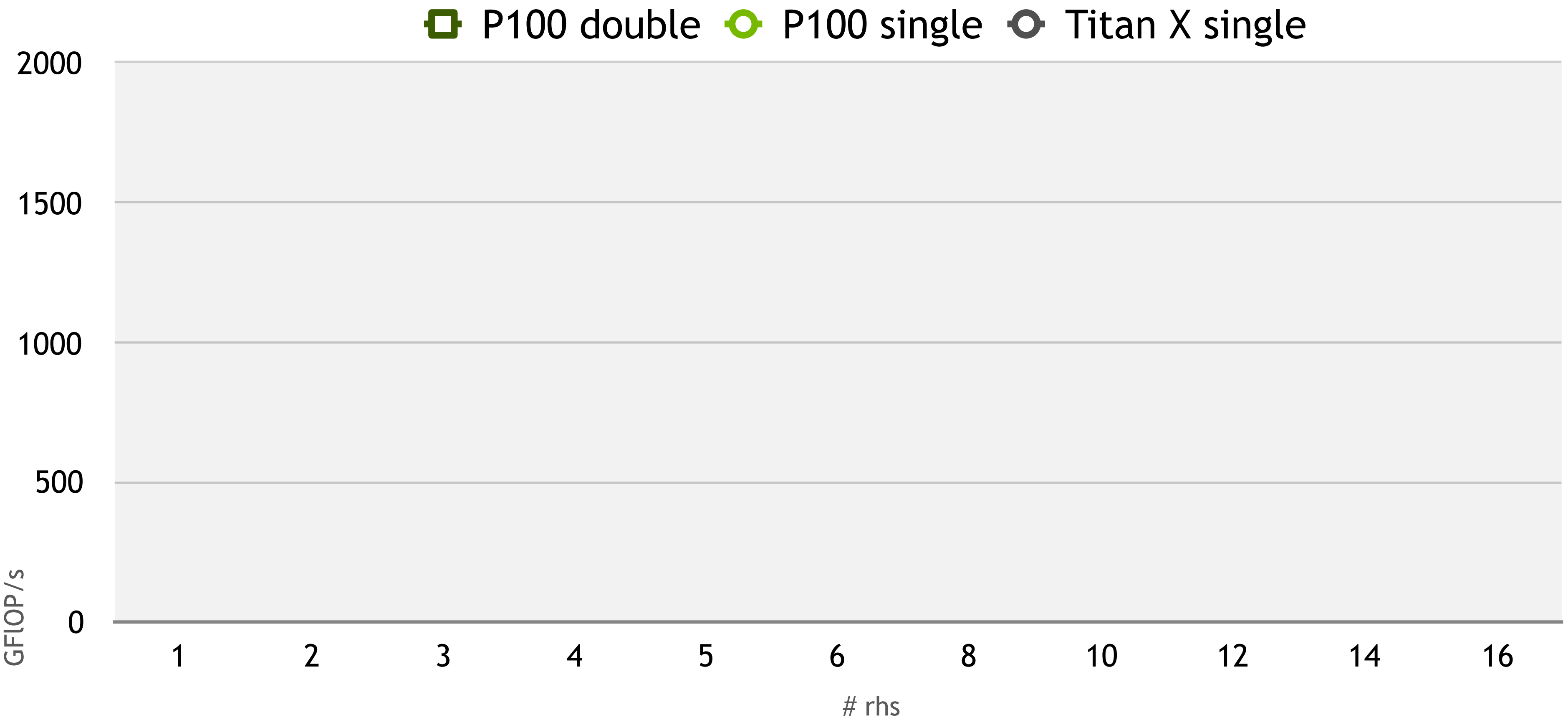
TITAN X



11 TF SP
480 GB/sec Memory Bandwidth
12 GB GDDR5X
Pascal architecture

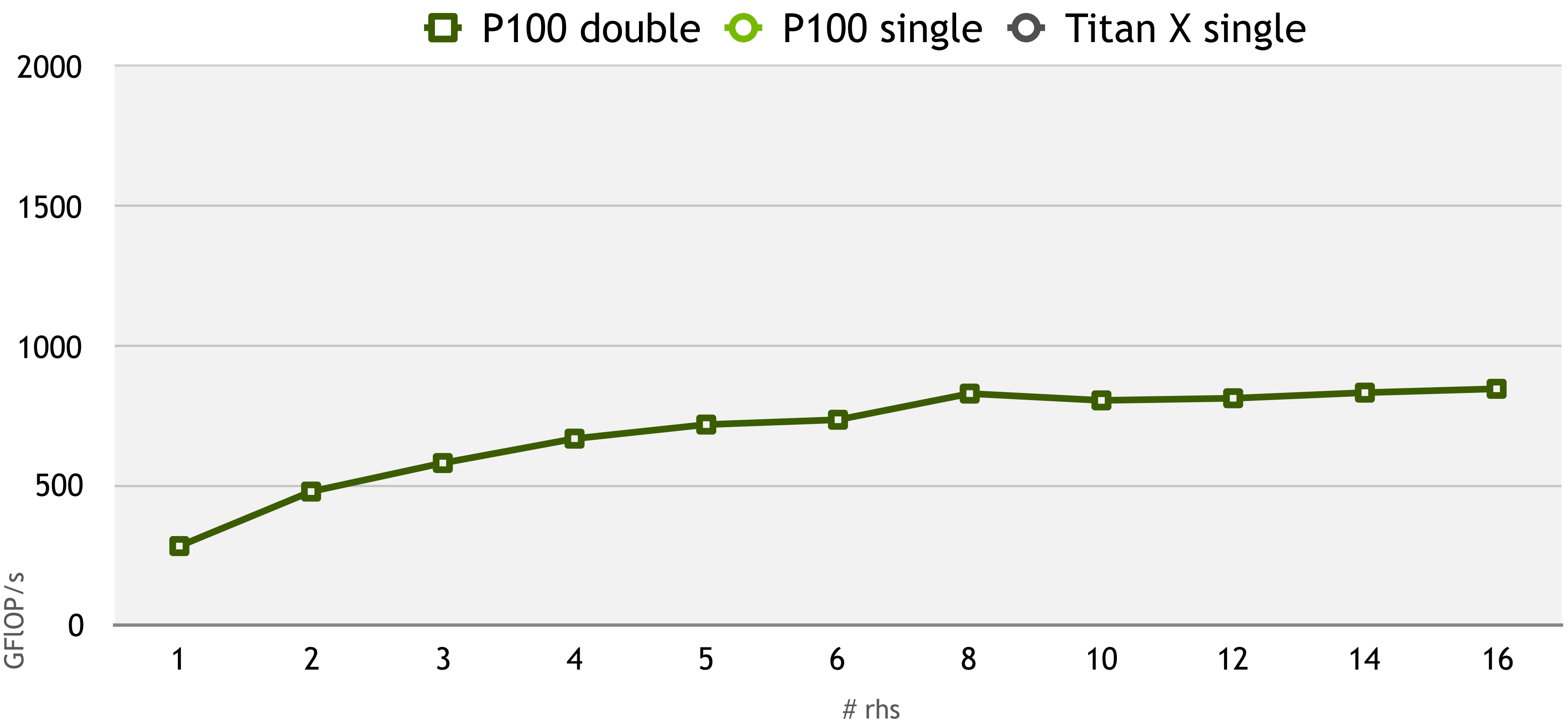
MULTI-SRC DSLASH ON PASCAL

Volume 24^4, HISQ, tuned gauge reconstruct



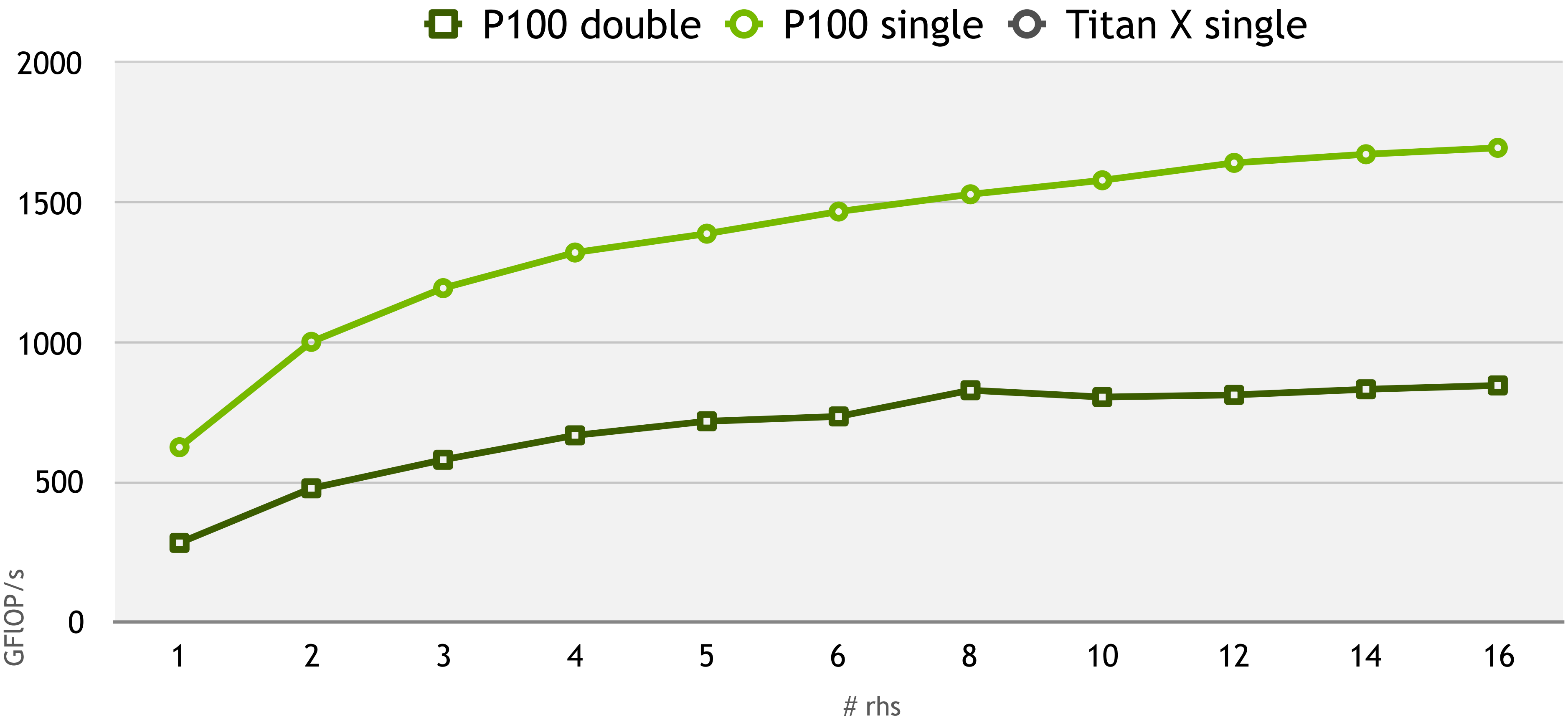
MULTI-SRC DSLASH ON PASCAL

Volume 24^4, HISQ, tuned gauge reconstruct



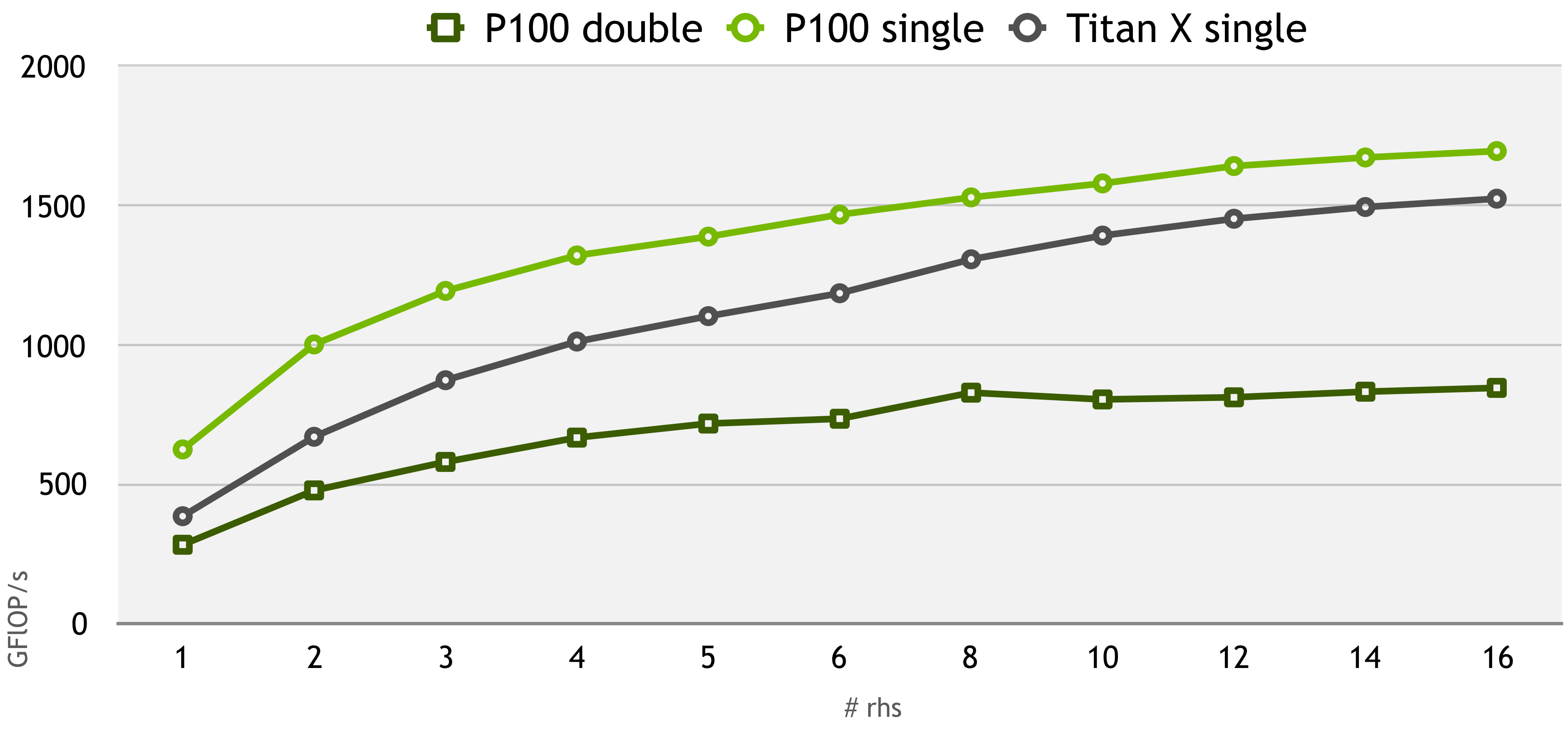
MULTI-SRC DSLASH ON PASCAL

Volume 24^4, HISQ, tuned gauge reconstruct



MULTI-SRC DSLASH ON PASCAL

Volume 24^4, HISQ, tuned gauge reconstruct



CONJUGATE GRADIENT

using multi-src Dslash

procedure CG WITH MULTI-SRC DSLASH

$r_i = b_i - Ax_i^{(0)}$

$p_i^{(0)} = r_i^{(0)}$

for $k = 1, 2, \ldots$ until converged **do**

$\left\{z_i^{(k-1)}\right\} = A\left\{p^{(k-1)}\right\}$

$\alpha_i^{(k-1)} = |r_i^{(k-1)}|^2 / \langle (p_i^{(k-1)}), z_i^{(k-1)} \rangle$

$x_i^{(k)} = x_i^{(k-1)} + \alpha_i^{(k-1)} p_i^{(k-1)}$

$r_i^{(k)} = r_i^{(k-1)} - \alpha_i^{(k-1)} z_i^{(k-1)}$

$\beta_i^{(k-1)} = |r_i^{(k-1)}|^2 / |r_i^{(k)}|^2$

$p_i^{(k)} = r_i^{(k)} + \beta_i^{(k-1)} p_i^{(k-1)}$

end for

end procedure

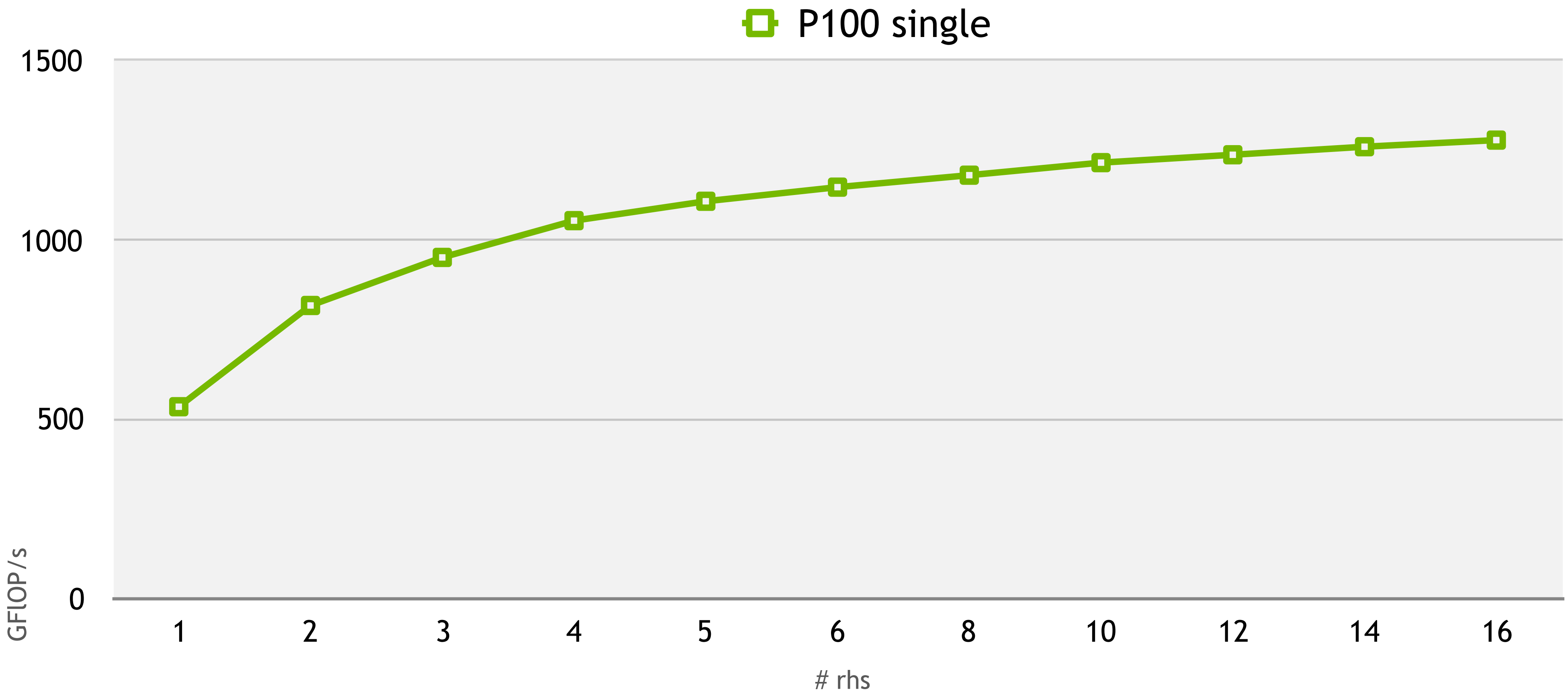
exploit multi-src Dslash performance

do all the linear algebra for each rhs

same iteration count as CG

MULTI-SRC CG ON PASCAL

Volume 24^4, HISQ



BLOCK KRYLOV SPACE SOLVERS

BLOCK KRYLOV SOLVERS

Share the Krylov space

BlockCG solver suggested by O'Leary in early 80's

retooled BlockCG by Dubrulle 2001

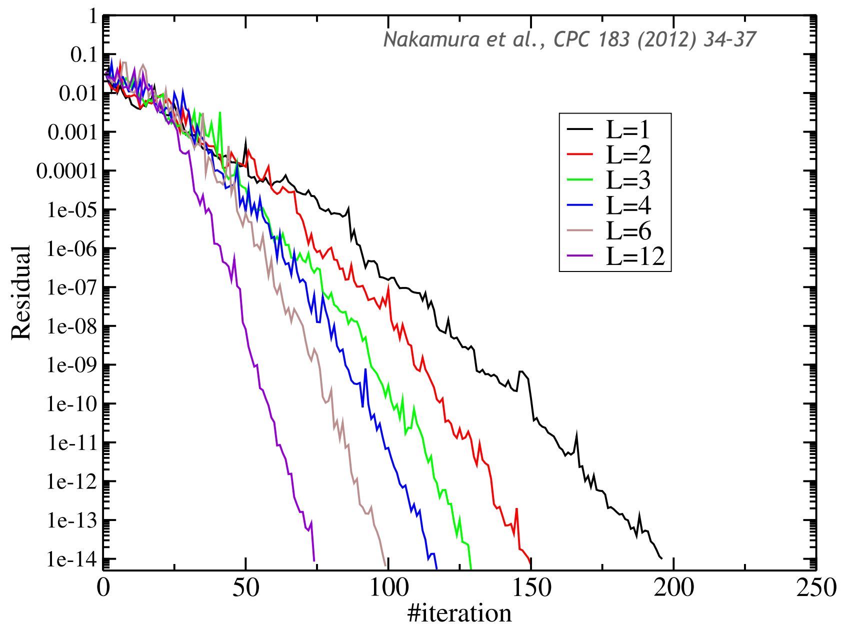
In exact precision converges in N / rhs iterations

BLOCK KRYLOV SOLVERS

Share the Krylov space

BlockCG solver suggested by O'Leary in early 80's
retooled BlockCG by Dubrulle 2001
In exact precision converges in N / rhs iterations

Application in QCD:
Nakamura et. (modified block BiCGStab)
Birk and Frommer (block methods, including block methods for multi shift)



BLOCK CG

share Krylov space between multiple rhs

procedure BlockCG

$$R^{(0)} = B - AX^{(0)}$$

$$P^{(0)} = R^{(0)}$$

for $k = 1, 2, \ldots$ until converged **do**

$$Z^{(k-1)} = AP^{(k-1)}$$

$$\alpha^{(k-1)} = \left[(P^{(k-1)})^H Z^{(k-1)} \right]^{-1} (R^{(k-1)})^H R^{(k-1)}$$

$$X^{(k)} = X^{(k-1)} + P^{(k-1)} \alpha^{(k-1)}$$

$$R^{(k)} = R^{(k-1)} - Z^{(k-1)} \alpha^{(k-1)}$$

$$\beta^{(k-1)} = \left[(R^{(k-1)})^H R^{(k-1)} \right]^{-1} (R^{(k)})^H R^{(k)}$$

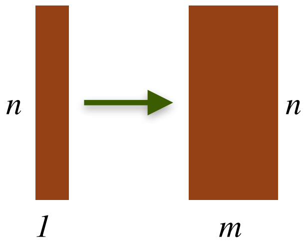
$$P^{(k)} = R^{(k)} - P^{(k-1)} \beta^{(k-1)}$$

end for

end procedure

BLOCK CG

share Krylov space between multiple rhs



procedure BlockCG

$$R^{(0)} = B - AX^{(0)}$$

$$P^{(0)} = R^{(0)}$$

for $k = 1, 2, \ldots$ until converged **do**

$$Z^{(k-1)} = AP^{(k-1)}$$

$$\alpha^{(k-1)} = \left[(P^{(k-1)})^H Z^{(k-1)} \right]^{-1} (R^{(k-1)})^H R^{(k-1)}$$

$$X^{(k)} = X^{(k-1)} + P^{(k-1)} \alpha^{(k-1)}$$

$$R^{(k)} = R^{(k-1)} - Z^{(k-1)} \alpha^{(k-1)}$$

$$\beta^{(k-1)} = \left[(R^{(k-1)})^H R^{(k-1)} \right]^{-1} (R^{(k)})^H R^{(k)}$$

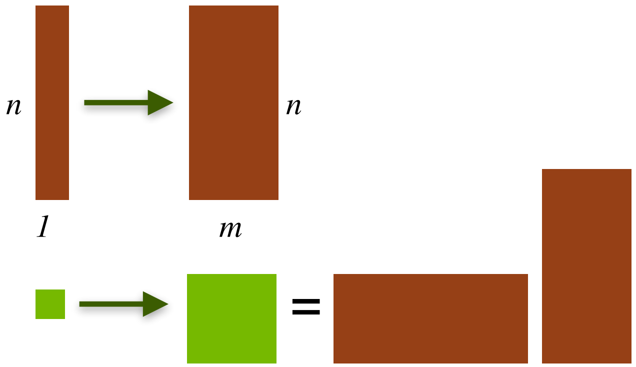
$$P^{(k)} = R^{(k)} - P^{(k-1)} \beta^{(k-1)}$$

end for

end procedure

BLOCK CG

share Krylov space between multiple rhs



procedure BlockCG

$$R^{(0)} = B - AX^{(0)}$$
$$P^{(0)} = R^{(0)}$$

for $k = 1, 2, \ldots$ until converged **do**

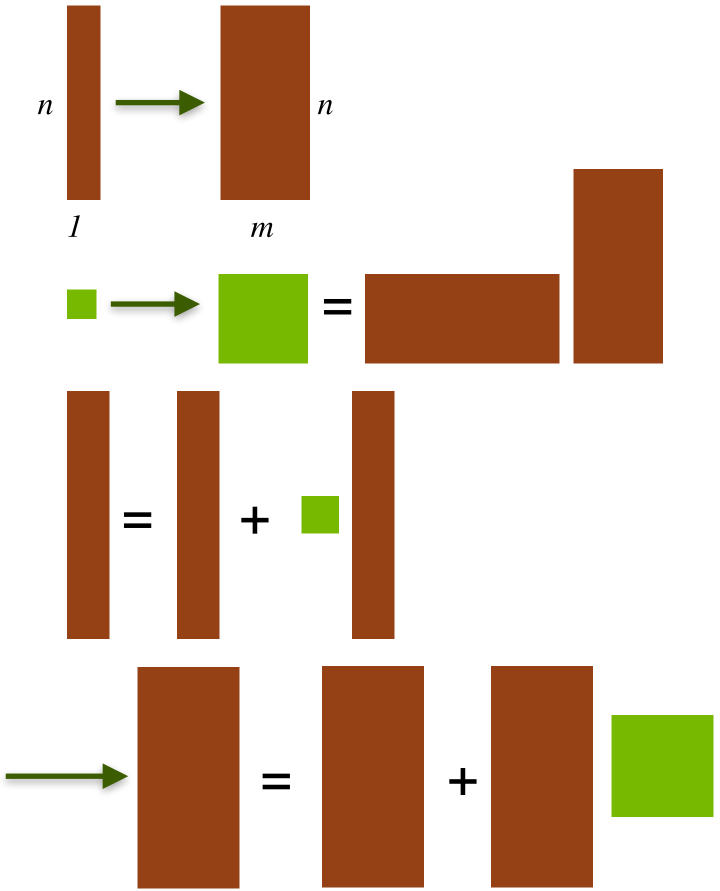
$$Z^{(k-1)} = AP^{(k-1)}$$
$$\alpha^{(k-1)} = \left[(P^{(k-1)})^H Z^{(k-1)} \right]^{-1} (R^{(k-1)})^H R^{(k-1)}$$
$$X^{(k)} = X^{(k-1)} + P^{(k-1)} \alpha^{(k-1)}$$
$$R^{(k)} = R^{(k-1)} - Z^{(k-1)} \alpha^{(k-1)}$$
$$\beta^{(k-1)} = \left[(R^{(k-1)})^H R^{(k-1)} \right]^{-1} (R^{(k)})^H R^{(k)}$$
$$P^{(k)} = R^{(k)} - P^{(k-1)} \beta^{(k-1)}$$

end for

end procedure

BLOCK CG

share Krylov space between multiple rhs



procedure BlockCG

$$R^{(0)} = B - AX^{(0)}$$

$$P^{(0)} = R^{(0)}$$

for $k = 1, 2, \ldots$ until converged **do**

$$Z^{(k-1)} = AP^{(k-1)}$$

$$\alpha^{(k-1)} = \left[(P^{(k-1)})^H Z^{(k-1)}\right]^{-1} (R^{(k-1)})^H R^{(k-1)}$$

$$X^{(k)} = X^{(k-1)} + P^{(k-1)}\alpha^{(k-1)}$$

$$R^{(k)} = R^{(k-1)} - Z^{(k-1)}\alpha^{(k-1)}$$

$$\beta^{(k-1)} = \left[(R^{(k-1)})^H R^{(k-1)}\right]^{-1} (R^{(k)})^H R^{(k)}$$

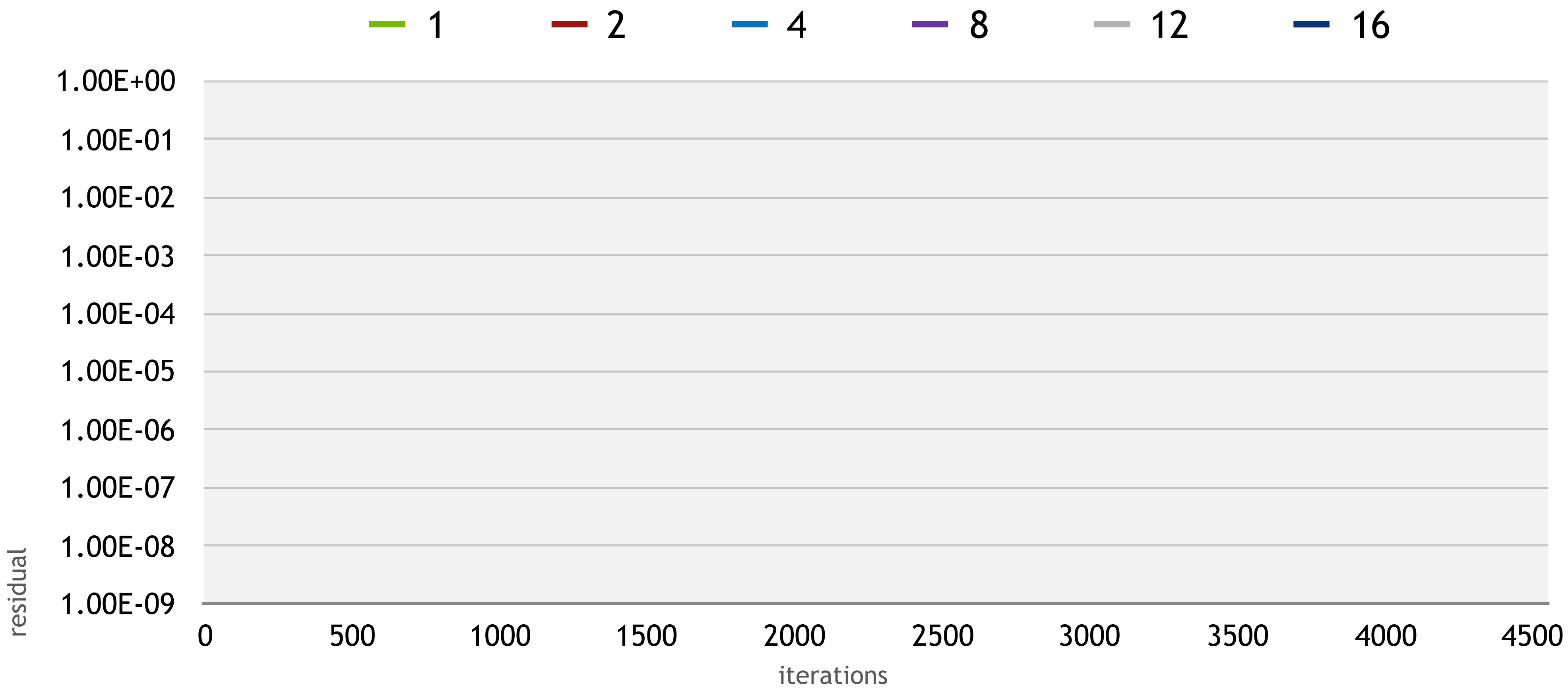
$$P^{(k)} = R^{(k)} - P^{(k-1)}\beta^{(k-1)}$$

end for

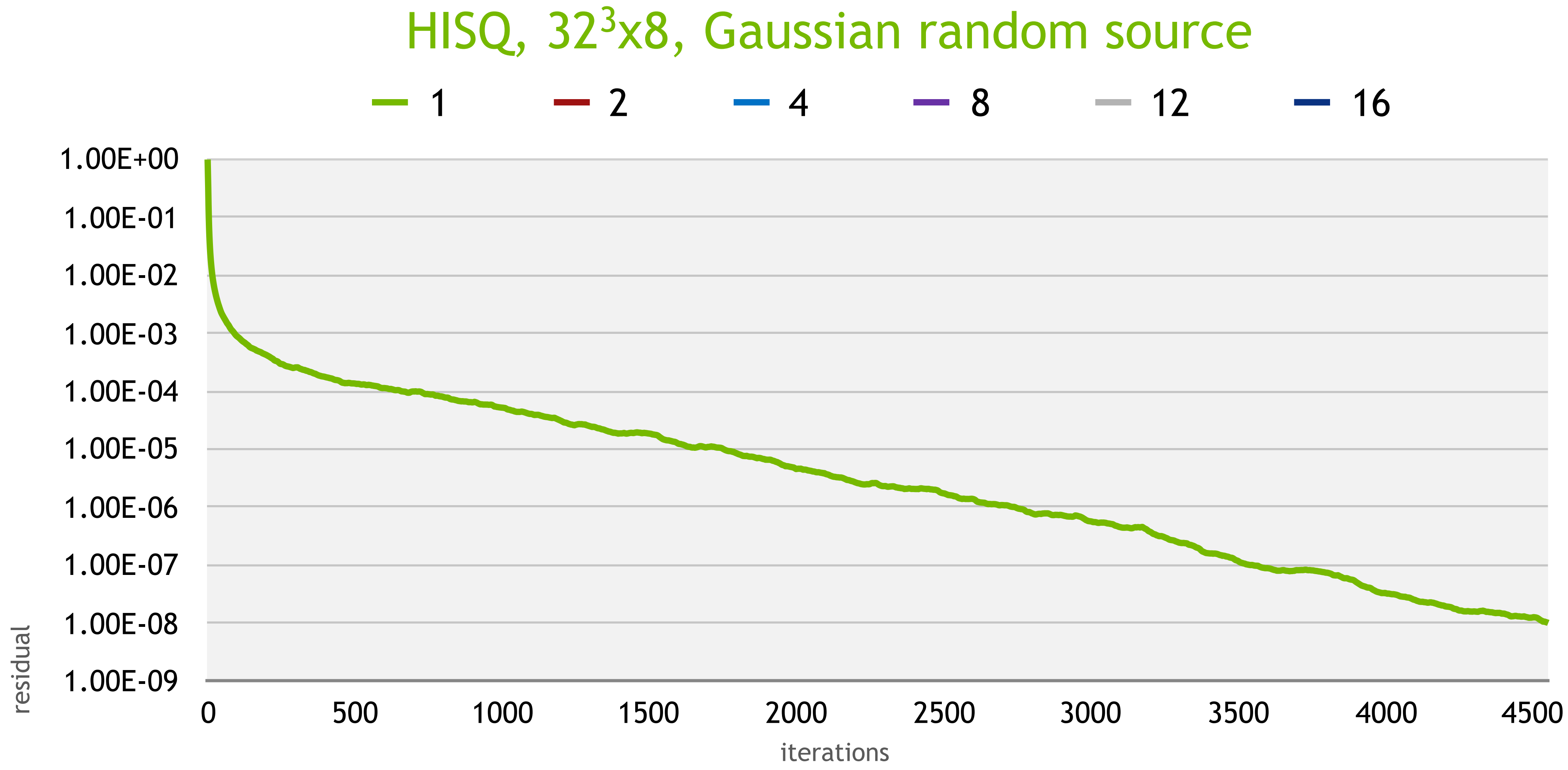
end procedure

REDUCED ITERATION COUNT

HISQ, 32^3x8, Gaussian random source

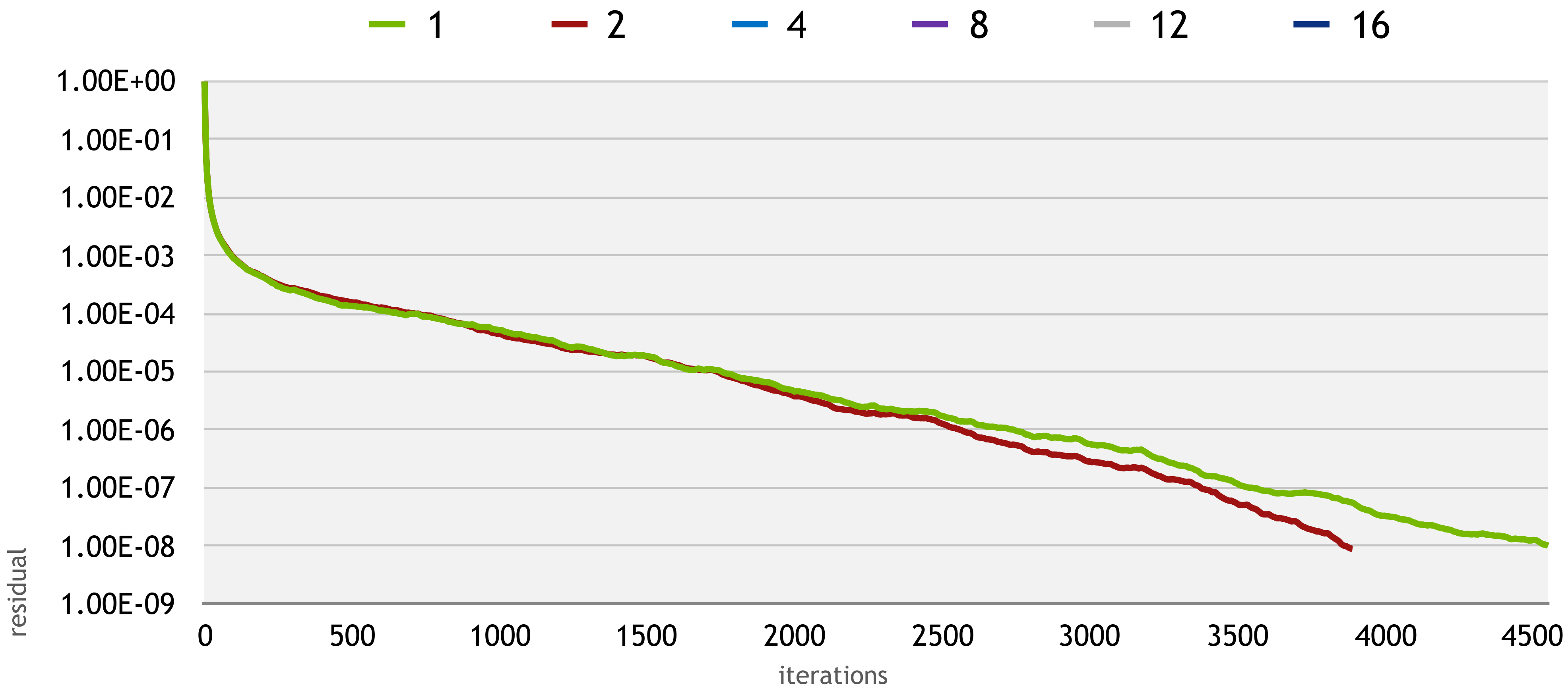


REDUCED ITERATION COUNT



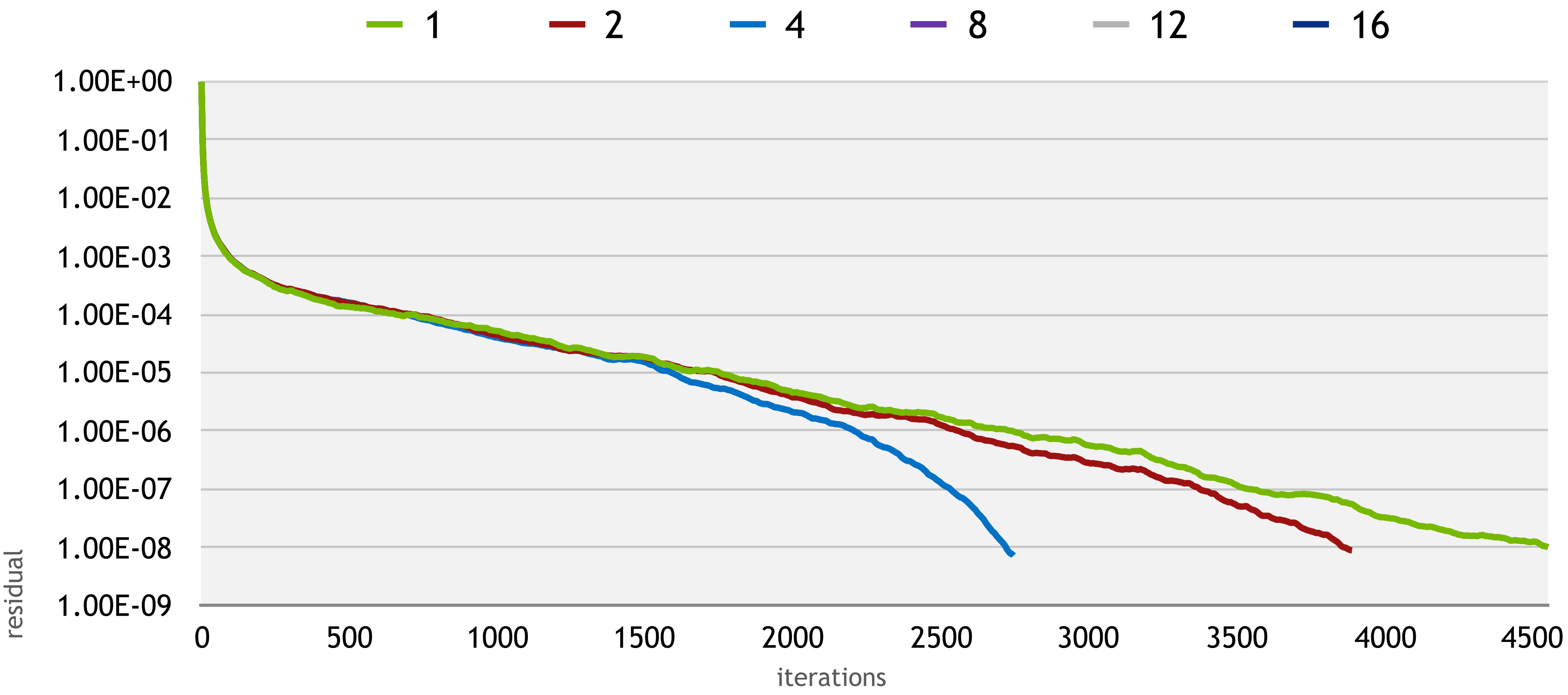
REDUCED ITERATION COUNT

HISQ, 32^3x8, Gaussian random source



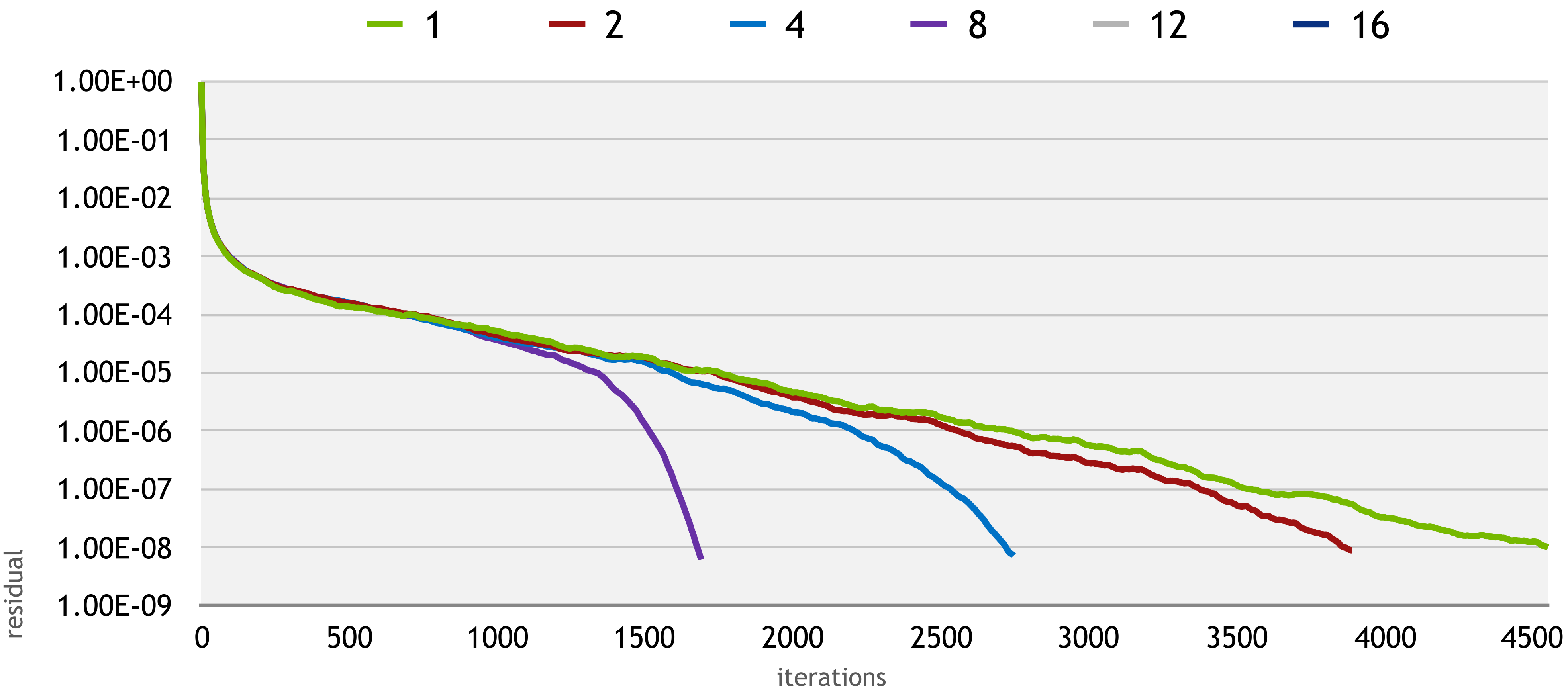
REDUCED ITERATION COUNT

HISQ, 32^3x8, Gaussian random source



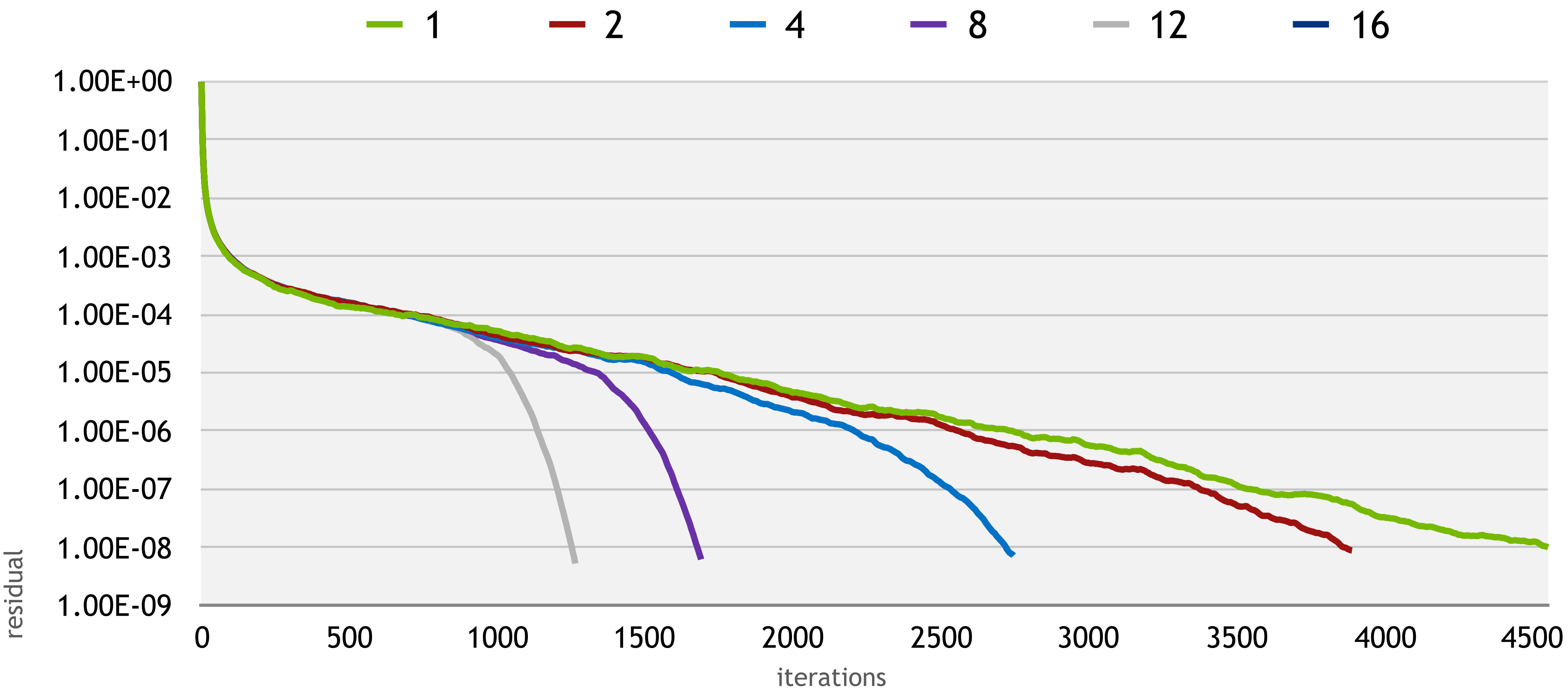
REDUCED ITERATION COUNT

HISQ, 32^3x8, Gaussian random source



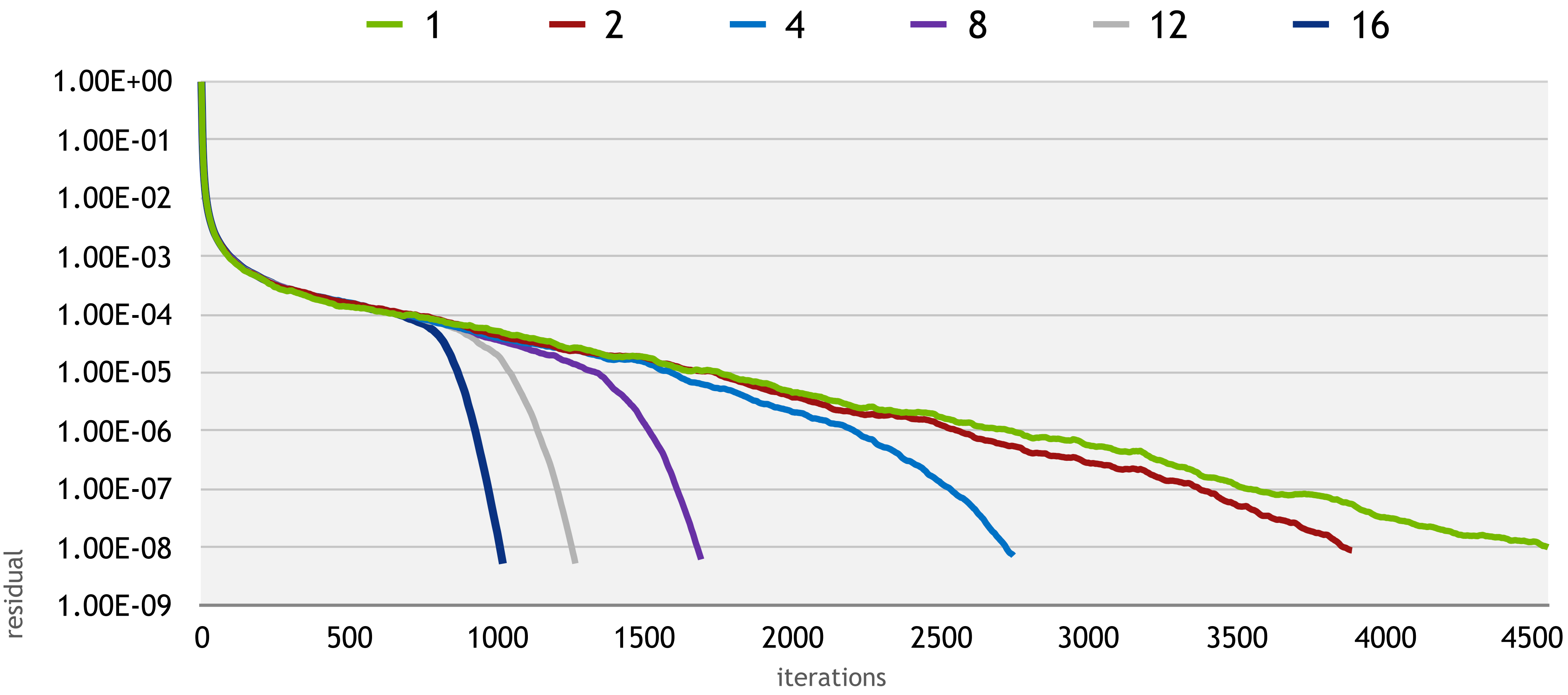
REDUCED ITERATION COUNT

HISQ, 32^3x8, Gaussian random source



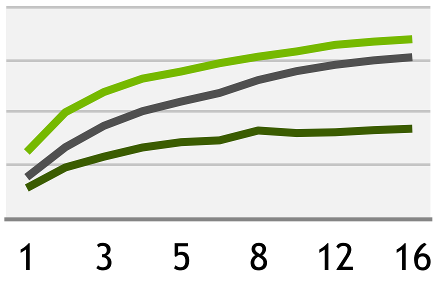
REDUCED ITERATION COUNT

HISQ, 32^3x8, Gaussian random source

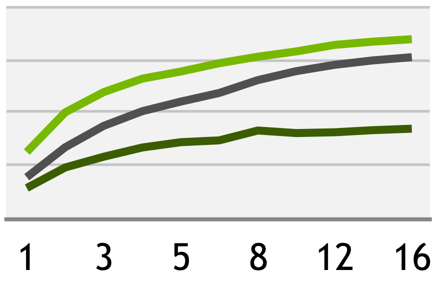


SCALING

Dslash exploits reuse of gauge field



SCALING



Dslash exploits reuse of gauge field

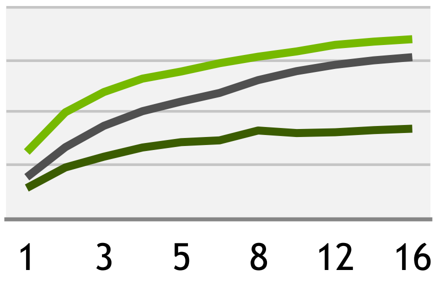
Linear Algebra

number of dot products scales quadratically $\alpha_{ij} = \langle x_i, x_j \rangle$

number of axpy calls scales quadratically $y_i = \sum a_{ij} x_j + y_i$

SCALING

Dslash exploits reuse of gauge field



Linear Algebra

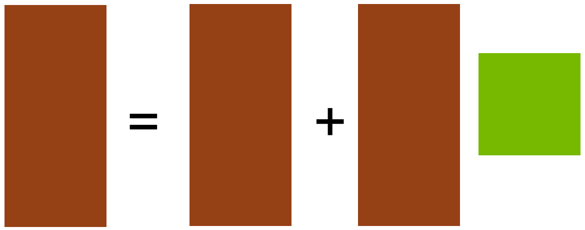
number of dot products scales quadratically $\alpha_{ij} = \langle x_i, x_j \rangle$

number of axpy calls scales quadratically $y_i = \sum a_{ij} x_j + y_i$

EXPLOIT GPU ARCHITECTURE

to overcome quadratically scaling

$$y_i = \sum a_{ij} x_j + y_i$$



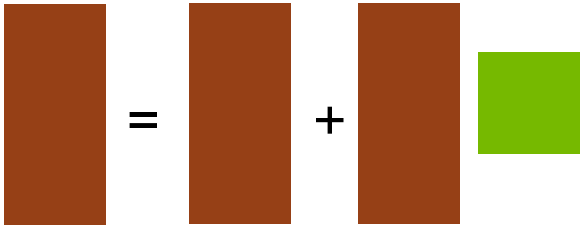
CUDA supports two dimensional grid blocks:

easy to exploit locality for texture cache / shared memory

EXPLOIT GPU ARCHITECTURE

to overcome quadratically scaling

$$y_i = \sum a_{ij} x_j + y_i$$



CUDA supports two dimensional grid blocks:

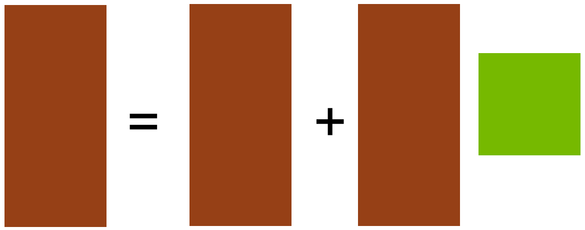
easy to exploit locality for texture cache / shared memory

(0,0)	(1,0)	(2,0)	(3,0)
(0,1)	(1,1)	(2,1)	(3,1)
(0,3)	(1,2)	(2,2)	(3,2)
(0,4)	(1,3)	(2,3)	(3,4)

EXPLOIT GPU ARCHITECTURE

to overcome quadratically scaling

$$y_i = \sum a_{ij} x_j + y_i$$



CUDA supports two dimensional grid blocks:

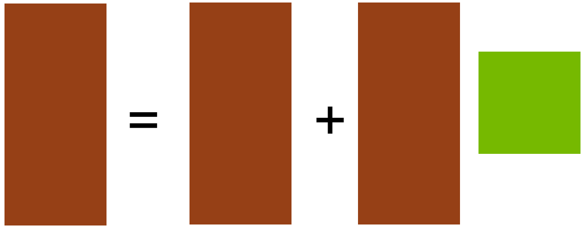
easy to exploit locality for texture cache / shared memory

$$y_0(0) = a_{00}x_0(0) + a_{01}x_1(0) + \dots$$
$$y_1(0) = a_{10}x_0(0) + a_{11}x_1(0) + \dots$$
$$y_2(0) = a_{20}x_0(0) + a_{21}x_1(0) + \dots$$
$$y_3(0) = a_{30}x_0(0) + a_{31}x_1(0) + \dots$$

EXPLOIT GPU ARCHITECTURE

to overcome quadratically scaling

$$y_i = \sum a_{ij} x_j + y_i$$



CUDA supports two dimensional grid blocks:

easy to exploit locality for texture cache / shared memory

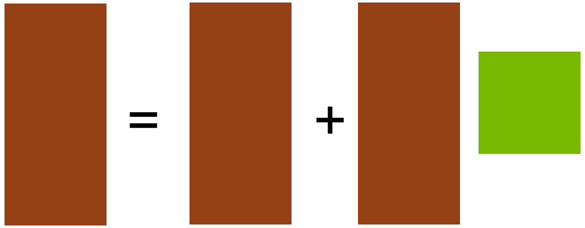
$$y_0(0) = a_{00}x_0(0) + a_{01}x_1(0) + \ldots$$
$$y_1(0) = a_{10}x_0(0) + a_{11}x_1(0) + \ldots$$
$$y_2(0) = a_{20}x_0(0) + a_{21}x_1(0) + \ldots$$
$$y_3(0) = a_{30}x_0(0) + a_{31}x_1(0) + \ldots$$

(0,0)	(1,0)	(2,0)	(3,0)
(0,1)	(1,1)	(2,1)	(3,1)
(0,3)	(1,2)	(2,2)	(3,2)
(0,4)	(1,3)	(2,3)	(3,4)

EXPLOIT GPU ARCHITECTURE

to overcome quadratically scaling

$$y_i = \sum a_{ij} x_j + y_i$$



CUDA supports two dimensional grid blocks:

easy to exploit locality for texture cache / shared memory

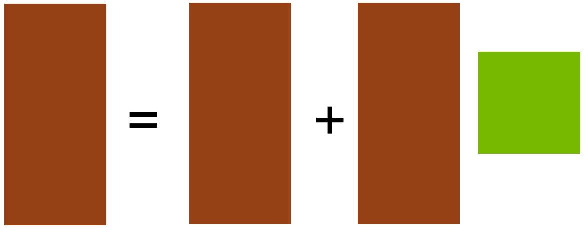
$$y_0(0) = a_{00} x_0(0) + a_{01} x_1(0) + \ldots$$
$$y_1(0) = a_{10} x_0(0) + a_{11} x_1(0) + \ldots$$
$$y_2(0) = a_{20} x_0(0) + a_{21} x_1(0) + \ldots$$
$$y_3(0) = a_{30} x_0(0) + a_{31} x_1(0) + \ldots$$

cache reuse

EXPLOIT GPU ARCHITECTURE

to overcome quadratically scaling

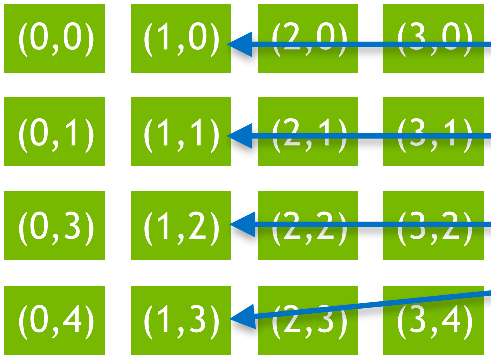
$$y_i = \sum a_{ij} x_j + y_i$$



CUDA supports two dimensional grid blocks:

easy to exploit locality for texture cache / shared memory

$$y_0(0) = a_{00}x_0(0) + a_{01}x_1(0) + \ldots$$
$$y_1(0) = a_{10}x_0(0) + a_{11}x_1(0) + \ldots$$
$$y_2(0) = a_{20}x_0(0) + a_{21}x_1(0) + \ldots$$
$$y_3(0) = a_{30}x_0(0) + a_{31}x_1(0) + \ldots$$



$$y_0(1) = a_{00}x$$
$$y_1(1) = a_{10}x$$
$$y_2(1) = a_{20}x$$
$$y_3(1) = a_{30}x$$

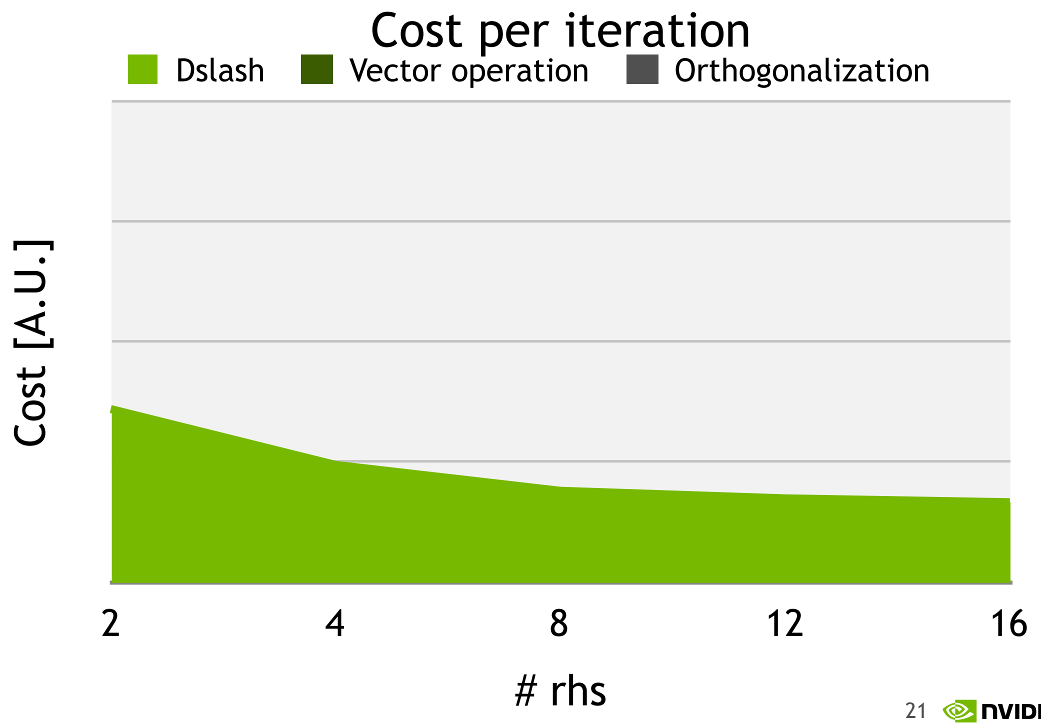
cache reuse

ORTHOGONALIZATION

BlockCG is not always numerically stable

simple approach: Gram-Schmidt or modified Gram-Schmidt

becomes prohibitively expensive

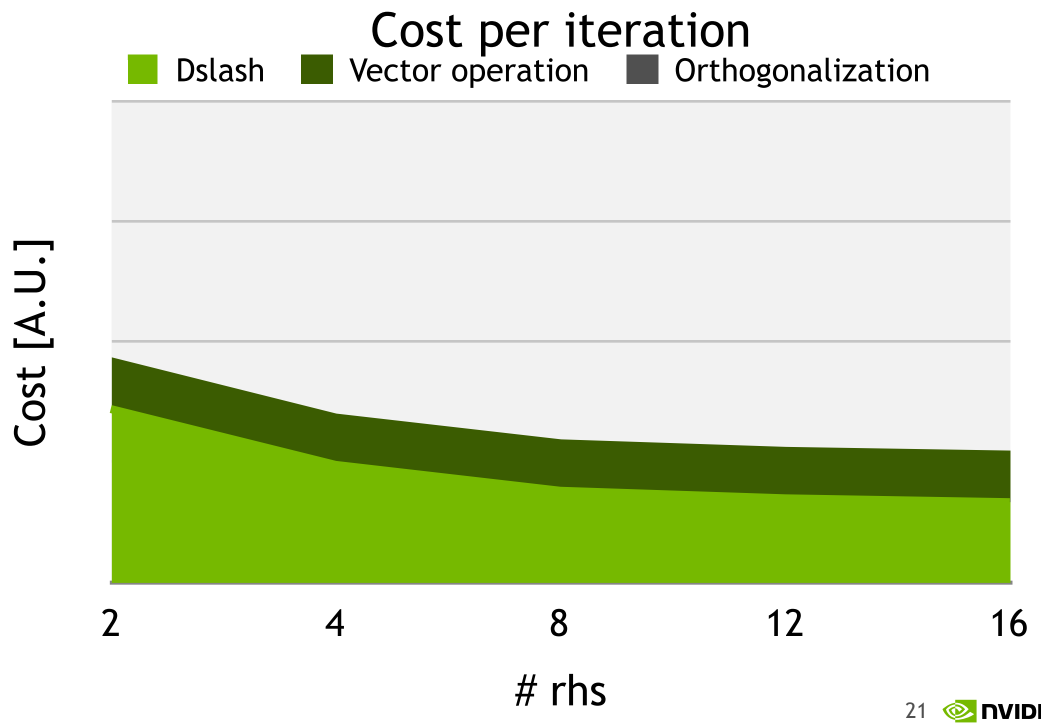


ORTHOGONALIZATION

BlockCG is not always numerically stable

simple approach: Gram-Schmidt or modified Gram-Schmidt

becomes prohibitively expensive

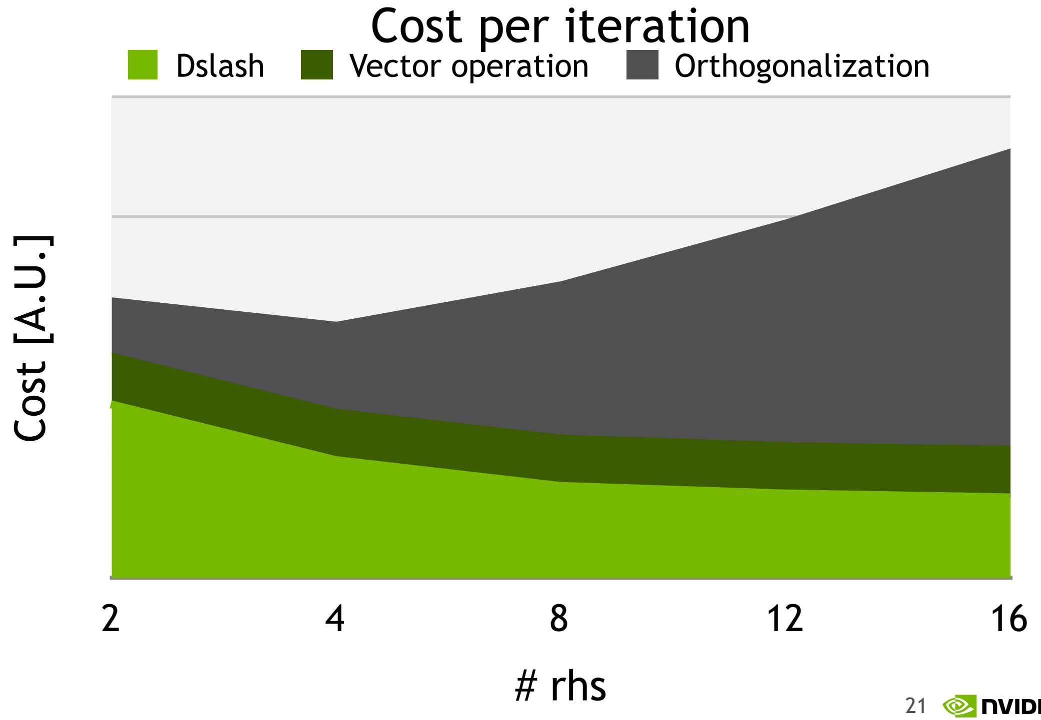


ORTHOGONALIZATION

BlockCG is not always numerically stable

simple approach: Gram-Schmidt or modified Gram-Schmidt

becomes prohibitively expensive



ORTHOGONALIZATION

simple approach: Gram-Schmidt or modified Gram-Schmidt

becomes prohibitively expensive

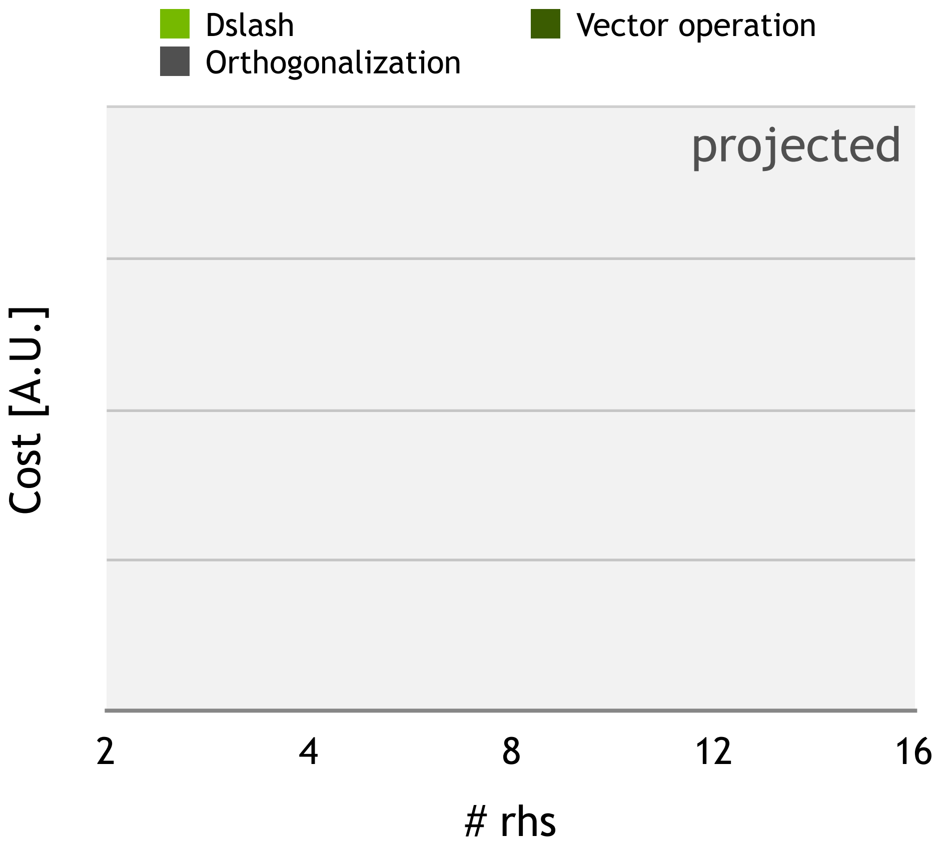
CholQR

Gram-Matrix:	$B = R^H R$	$m \times m$ dot products of length n
Cholesky Decomposition	$S^H S = B$	of $m \times m$ matrix
apply to vectors	$Q = RS^{-1}$	axpy $m \times m$ (output, input)

relies on the same kernel as vector operations: can get linear scaling

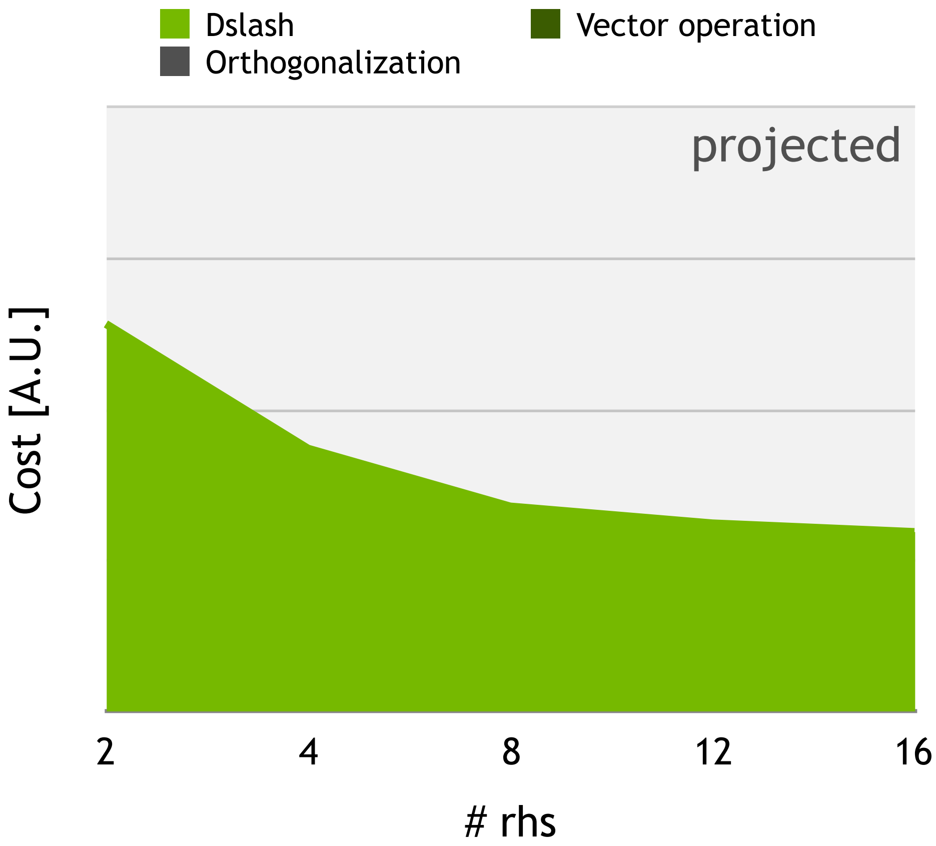
COST OF ONE ITERATION

for one rhs



COST OF ONE ITERATION

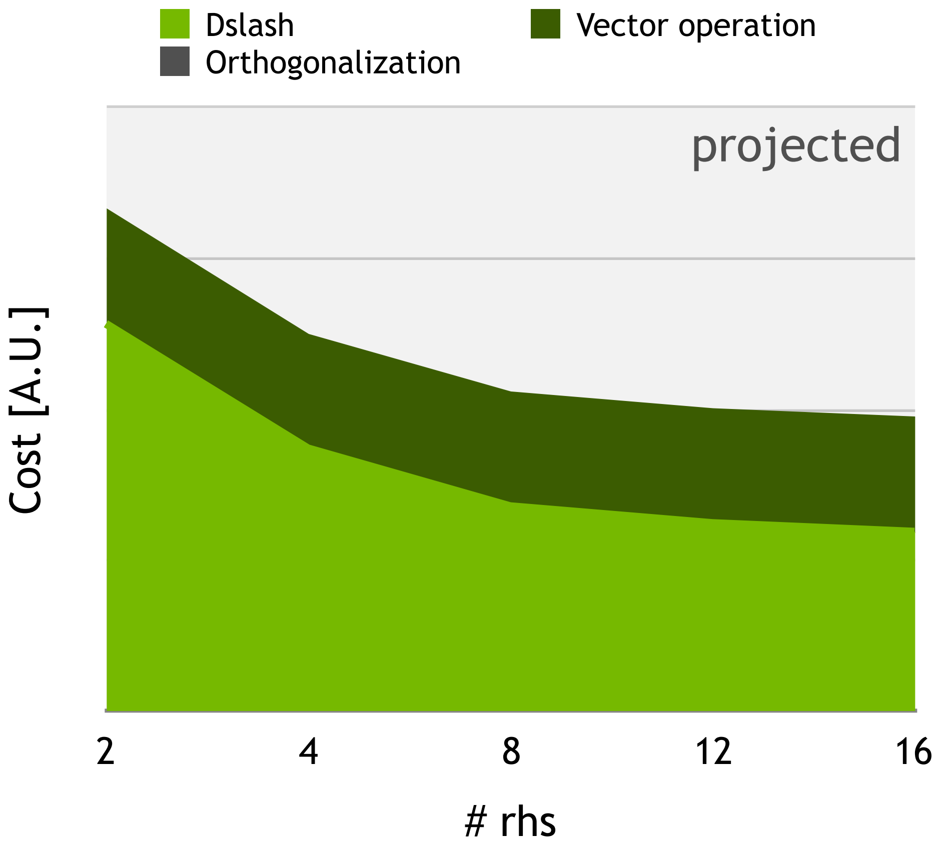
for one rhs



large benefits from multi-src Dslash

COST OF ONE ITERATION

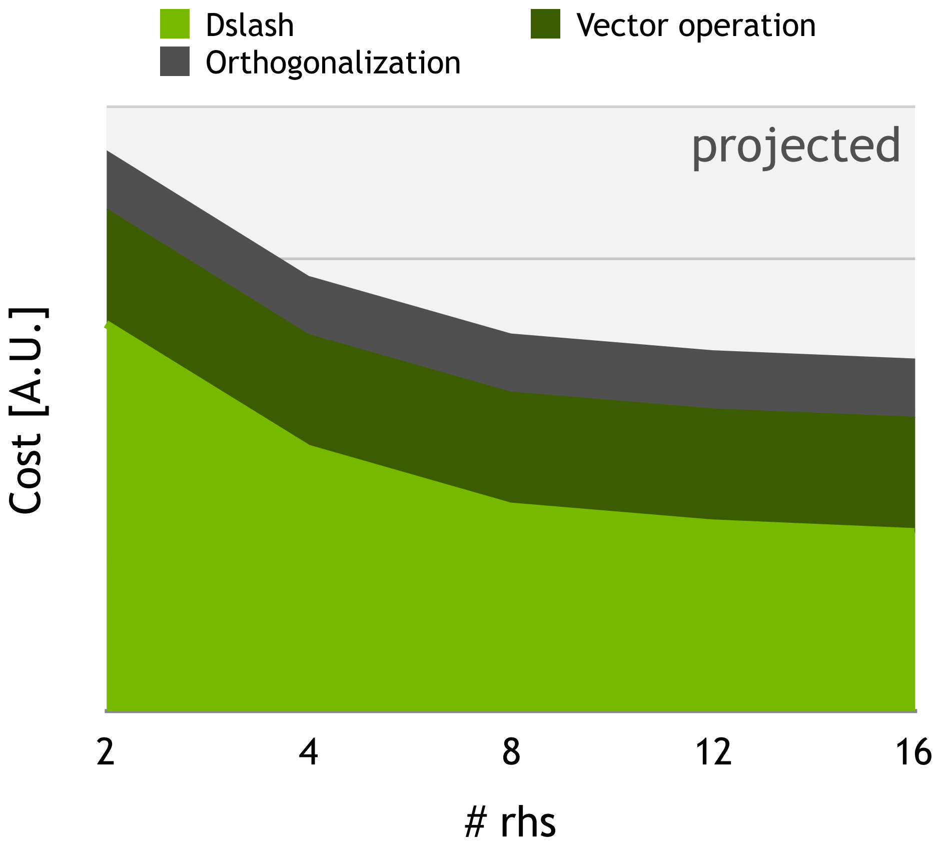
for one rhs



large benefits from multi-src Dslash

COST OF ONE ITERATION

for one rhs

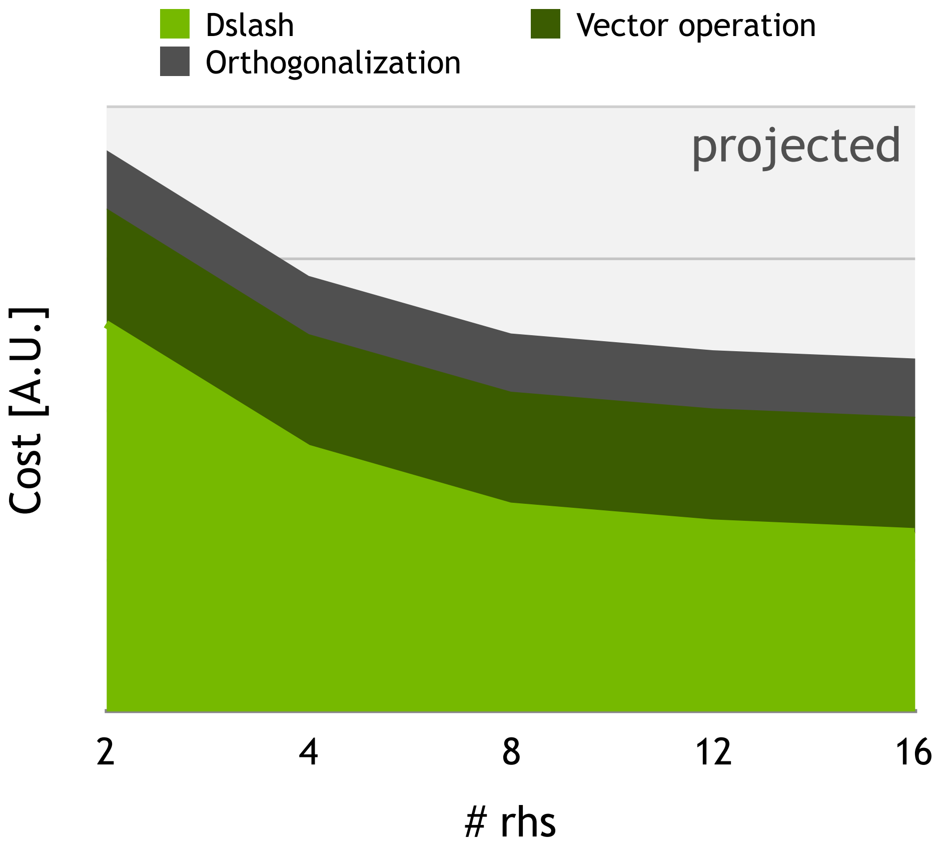


large benefits from multi-src Dslash

linear algebra and orthogonalization stay constant

COST OF ONE ITERATION

for one rhs



large benefits from multi-src Dslash

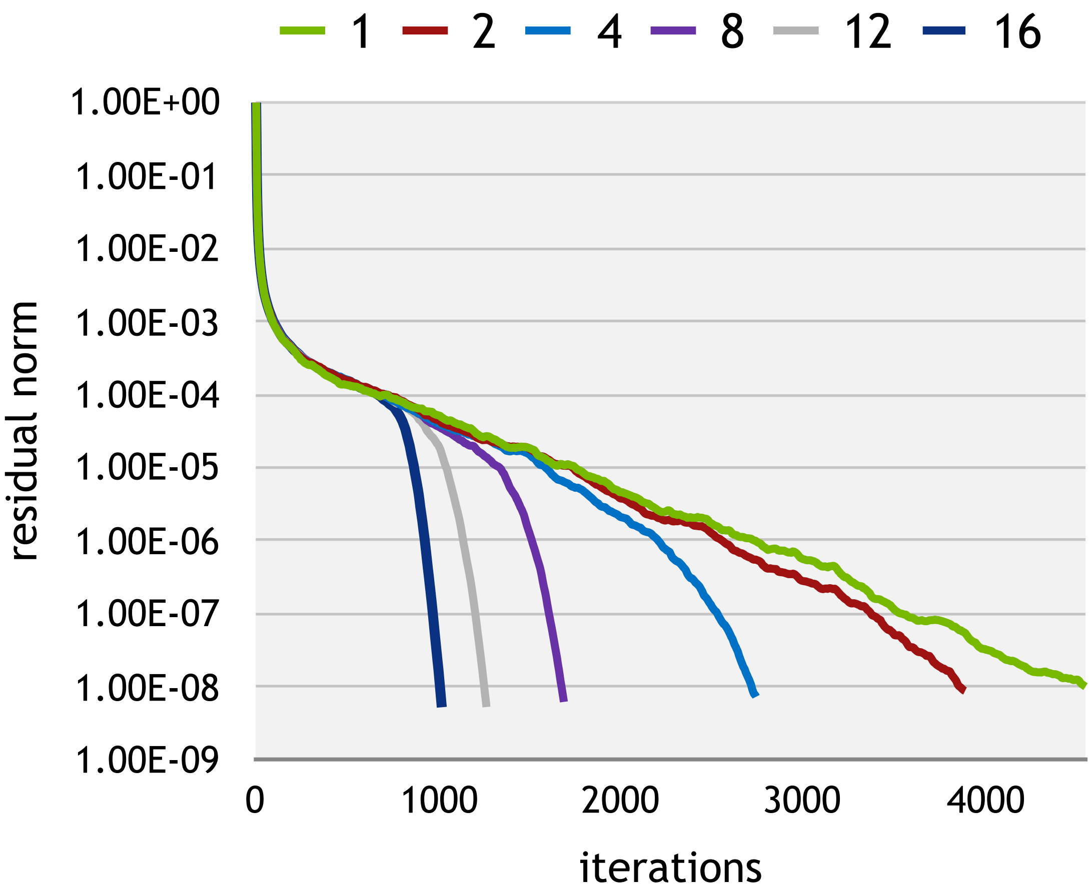
linear algebra and orthogonalization stay constant

relative importance of Dslash reduces

SUMMARY

WORK TO BE DONE

your milage may vary



stability needs real world testing
orthogonalization might be necessary

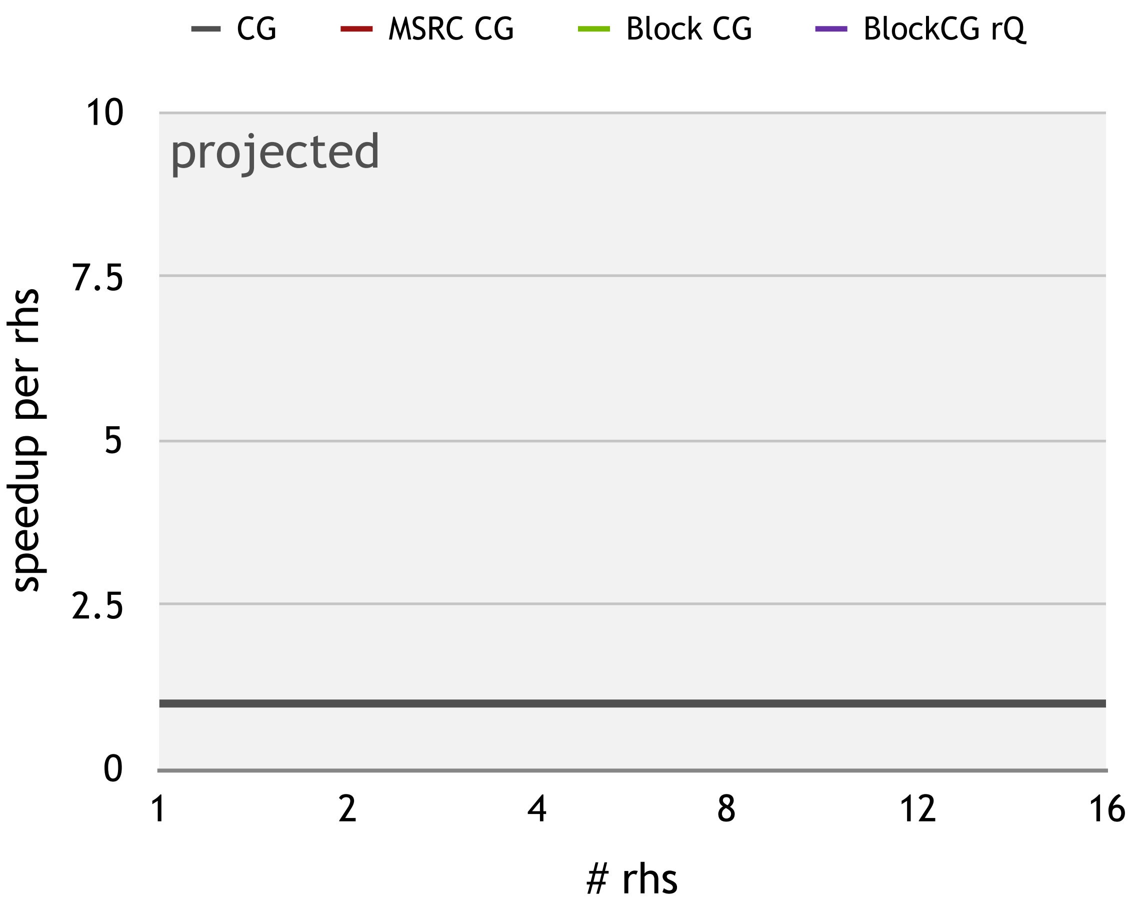
iteration count improvement may
depend on gauge field and sources

need to finish up implementation

add mixed precision

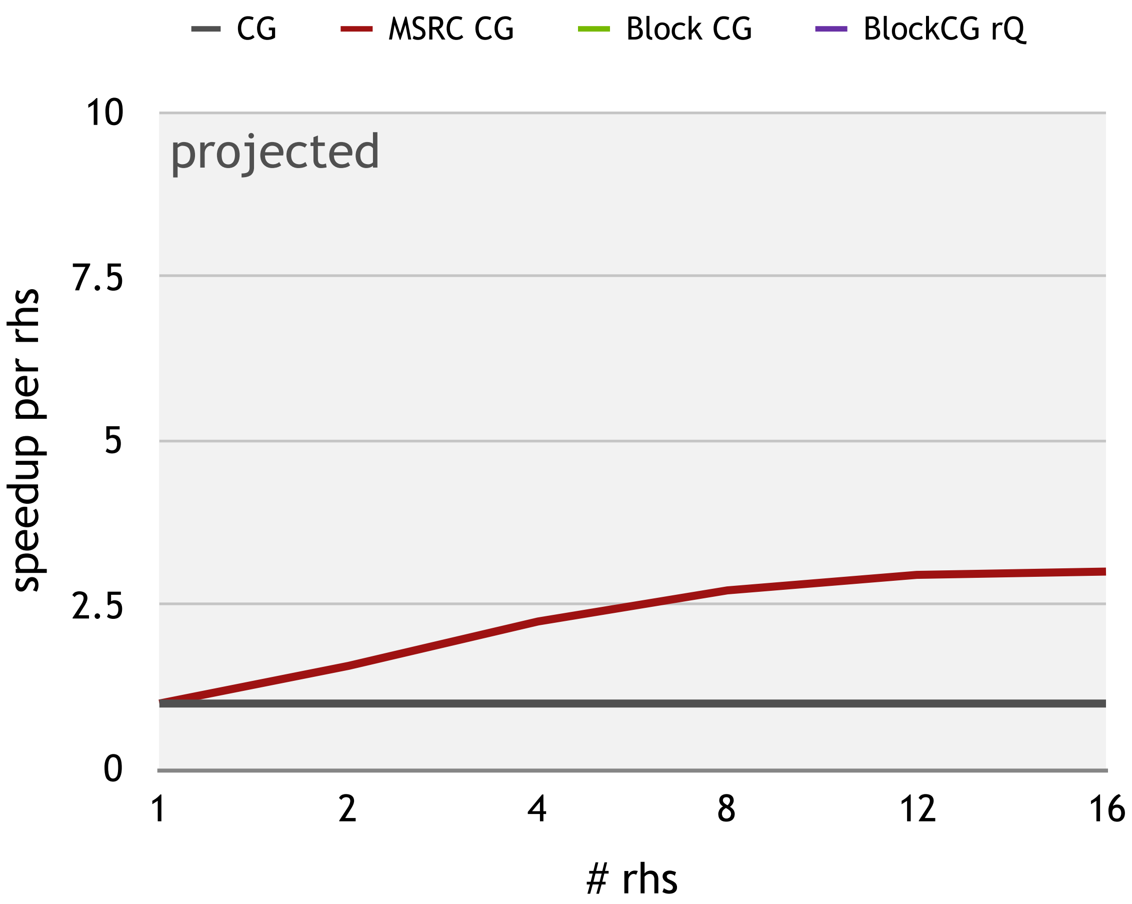
SPEEDUP OVER CG

multi-rhs Block Solvers provide an easy drop in



SPEEDUP OVER CG

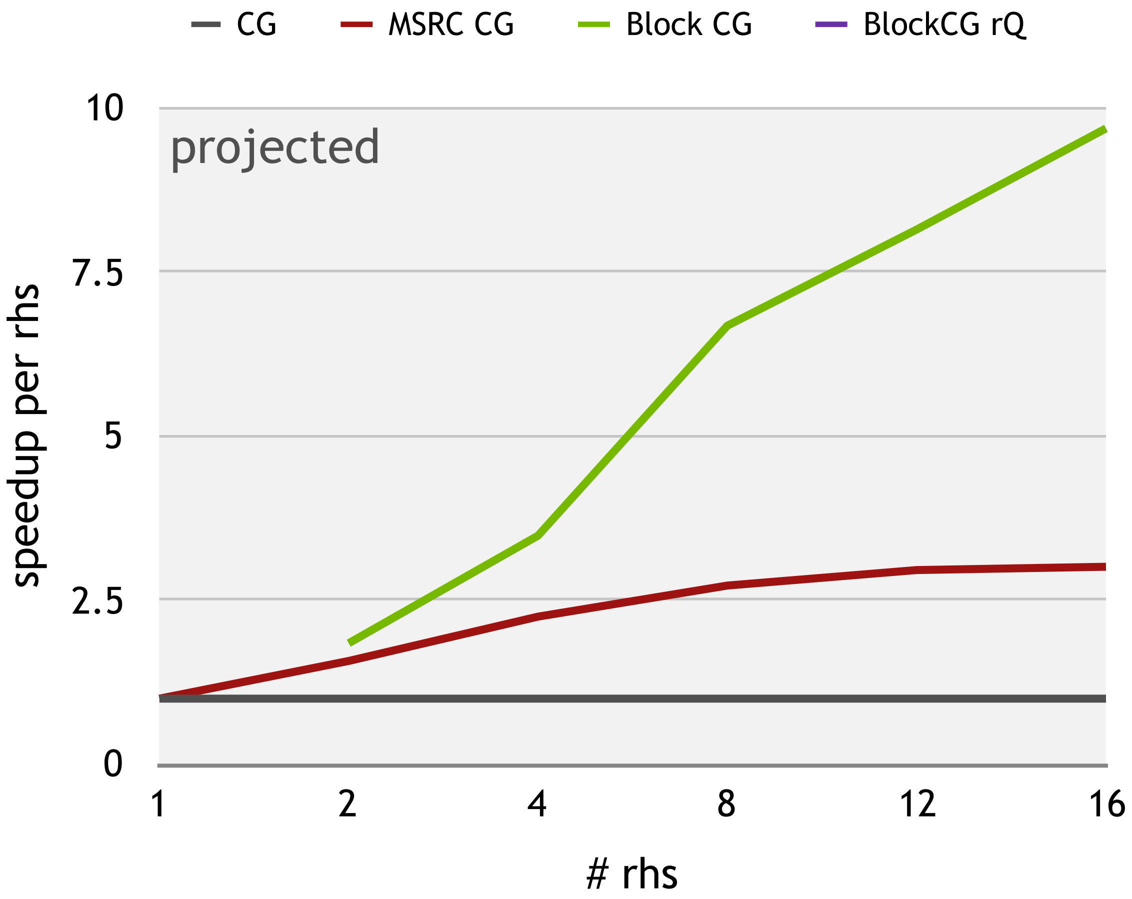
multi-rhs Block Solvers provide an easy drop in



reuse gauge field for Dslash

SPEEDUP OVER CG

multi-rhs Block Solvers provide an easy drop in

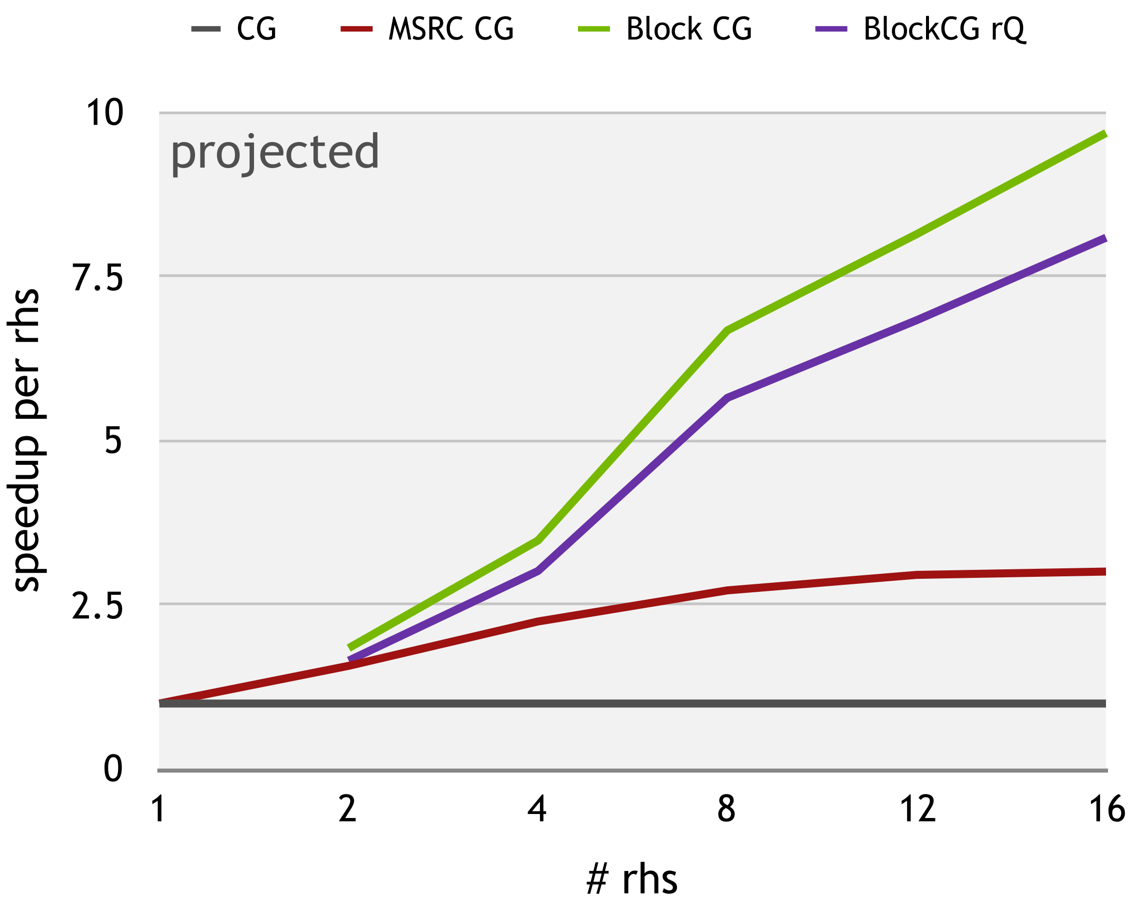


reuse gauge field for Dslash

reduced iteration count

SPEEDUP OVER CG

multi-rhs Block Solvers provide an easy drop in

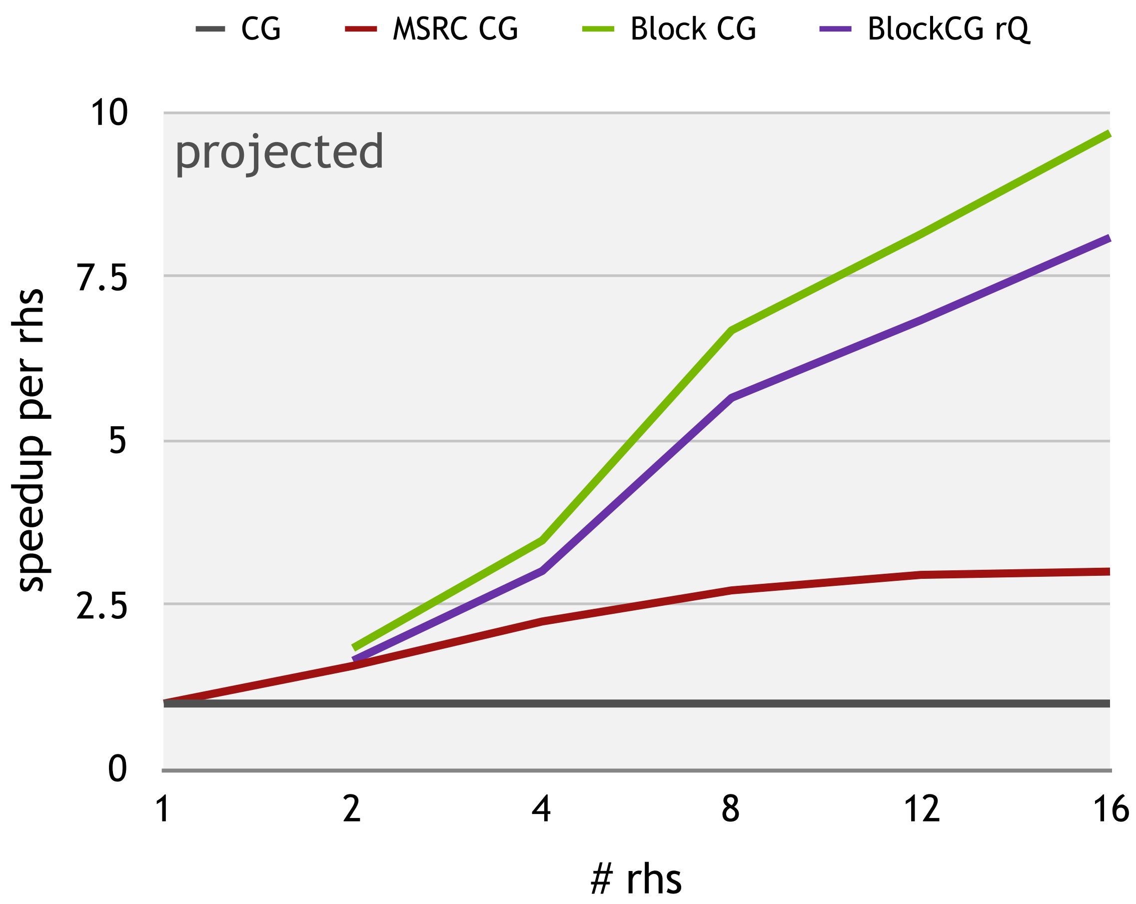


reuse gauge field for Dslash

reduced iteration count

SPEEDUP OVER CG

multi-rhs Block Solvers provide an easy drop in



reuse gauge field for Dslash

reduced iteration count

avoid quadratical scaling in linear algebra and orthogonalization

no memory overhead / setup cost

speedups up to 10x

NVIDIA