BLOCK SOLVERS FOR MULTIPLE RIGHT HAND SIDES ON NVIDIA GPUS

Mathias Wagner, Lattice 2016



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

□ Issues 107 □ Pull requests 2 □ Wiki Pulse □ Issues 107 □ Pull requests 2 □ Wiki Pulse □ Issues 107 □ Pull requests 2 □ Wiki Pulse □ Issues 107 □ Issues 107 □ Pull requests 2

QUDA is a library for performing calculations in lattice QCD on GPUs. http://lattice.github.com/quda — Edit

()

🕝 4,621 commits	پ 49 branches	♥ 19 releases	Le contributors
Branch: develop - New pull r	equest	Create new file Uplo	oad files Find file Clone or download -
Harding mathiaswagner committed on GitHub Merge pull request #487 from lattice/hotfix/checkerboard-reference Latest commit f3e2aa7 a day ago			
include	In ColorSpinorParam, if staggered fer	mions then set field dimension t	11 days ago
🖬 lib	Correctly set volumeCB for parity sub	oset references - need to check p.	a day ago
in tests	Requestingtest 1 with staggered_ds	slash_test now tests MdagM opera	ator 11 days ago
.gitignore	Updates to .gitignore and renamed m	ultigrid_benchmark to multigrid_b	e 3 months ago
CMakeLists.txt	added some comments to CMakeList	s.txt	3 months ago

5 💿 nvidia

+

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 Vaguero (INFN) Mathias Wagner (NVIDIA) Frank Winter (Jlab)

SOLVERS FOR MULTIPLE RIGHT HAND SIDES

CONJUGATE GRADIENT

just as a reminder

procedure CG $r^{(0)} = b - A x^{(0)}$ $p^{(0)} = r^{(0)}$ for $k = 1, 2, \ldots$ until converged **do** $z^{(k-1)} = Ap^{(k-1)}$ $\alpha^{(k-1)} = \frac{|r^{(k-1)}|^2}{\langle (p^{(k-1)}), z^{(k-1)} \rangle}$ $x^{(k)} = x(k-1) + \alpha^{(k-1)}p^{(k-1)}$ $r^{(k)} = r^{(k-1)} - \alpha^{(k-1)} z^{(k-1)}$ $\beta^{(k-1)} = \frac{|r^{(k-1)}|^2}{|r^{(k)}|^2}$ $p^{(k)} = r^{(k)} + \beta^{(k-1)} p^{(k-1)}$ end for end procedure

QCD is **memory bandwidth bound** Dslash arithmetic intensity for HISQ ~ 0.7

QCD is **memory bandwidth bound** Dslash arithmetic intensity for HISQ ~ 0.7

exploit SU(3) symmetry: reconstruct gauge field from 8/12 floats

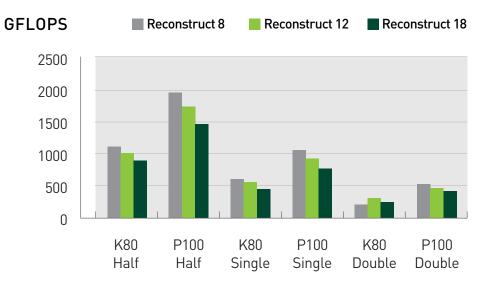


QCD is **memory bandwidth bound** Dslash arithmetic intensity for HISQ ~ 0.7

exploit SU(3) symmetry: reconstruct gauge field from 8/12 floats

WILSON CLOVER DSLASH

Volume = 32⁴



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 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

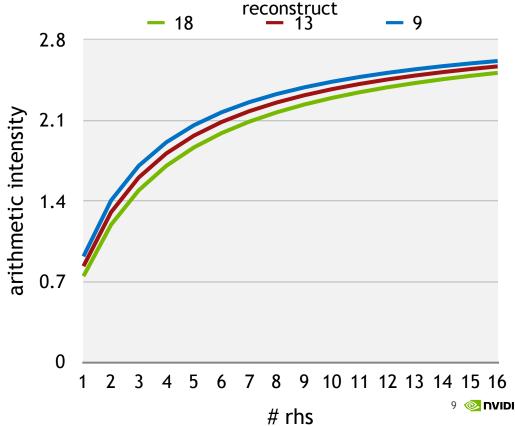
Reuse gauge field for multiple rhs

QCD 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

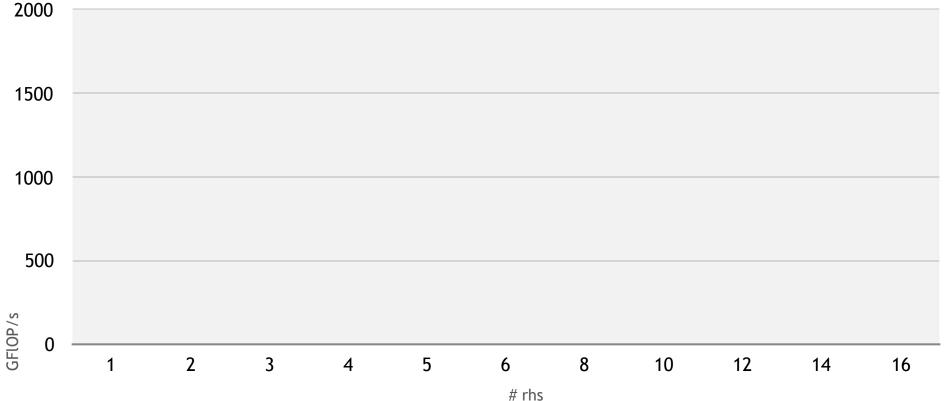
TITAN X

KITARY.

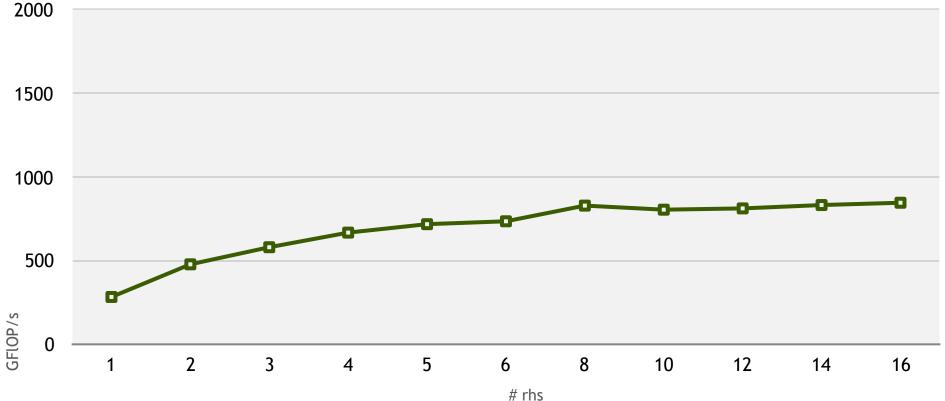
0

11 TF SP480 GB/sec Memory Bandwidth12 GB GDDR5XPascal architecture

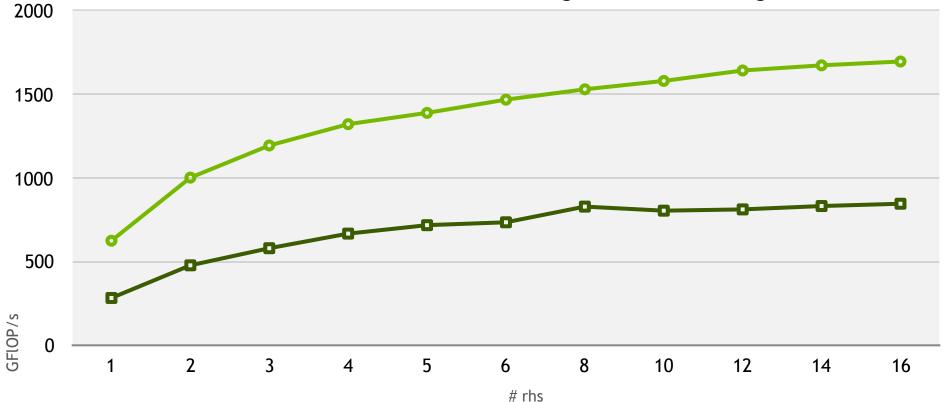
Volume 24⁴, HISQ, tuned gauge reconstruct



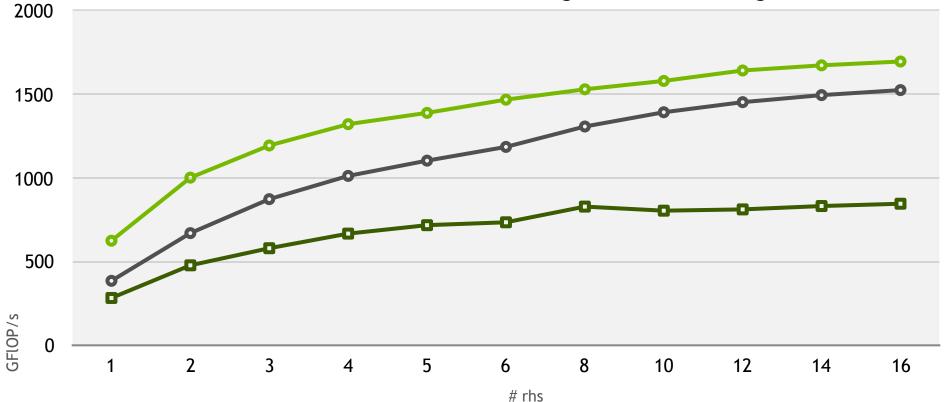
Volume 24⁴, HISQ, tuned gauge reconstruct



Volume 24⁴, HISQ, tuned gauge reconstruct



Volume 24⁴, HISQ, tuned gauge reconstruct



CONJUGATE GRADIENT

using multi-src Dslash

procedure CG WITH MULTI-SRC DSLASH

 $r_i = b_i - A x_i^{(0)}$ $p_i^{(0)} = r_i^{(0)}$ 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

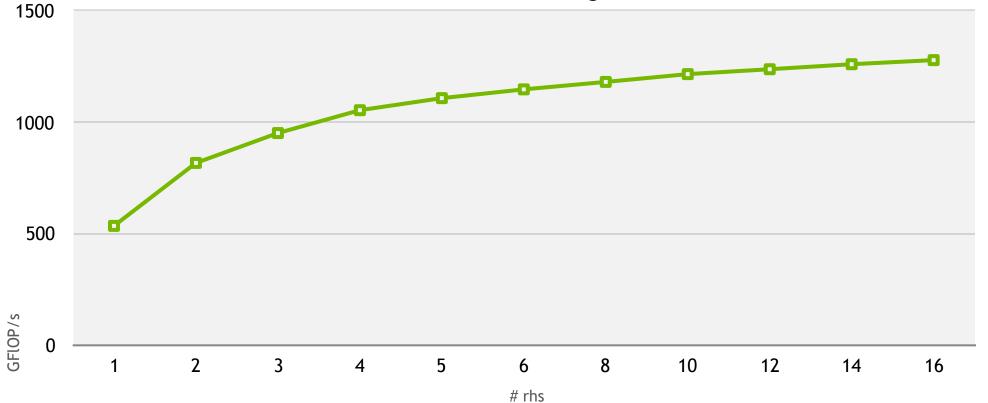
do all the linear algebra for each rhs

same iteration count as CG

MULTI-SRC CG ON PASCAL

Volume 24⁴, HISQ

P100 single



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



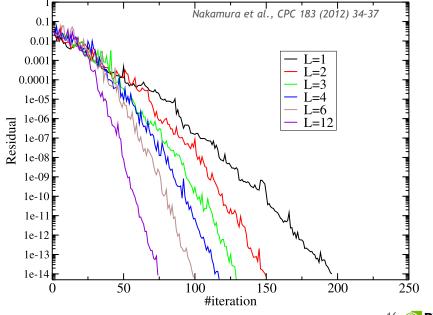
BLOCK KRYLOV SOLVERS

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)

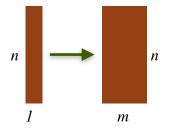


share Krylov space between multiple rhs

procedure BLOCKCG $R^{(0)} = B - AX^{(0)}$ $P^{(0)} = R^{(0)}$ for $k = 1, 2, \ldots$ until converged **do** $Z^{(k-1)} - A P^{(k-1)}$ $\alpha^{(k-1)} = \left[(P^{(k-1)})^H Z^{(k-1)} \right]^{-1} (R^{(k-1)})^H R^{(k-1)}$ $X^{(k)} = X^{(k-1)} + P^{(k-1)}\alpha^{(k-1)}$ $R^{(k)} = R^{(k-1)} - Z^{(k-1)} \alpha^{(k-1)}$ $\beta^{(k-1)} = \left[(R^{(k-1)})^H R^{(k-1)} \right]^{-1} (R^{(k)})^H R^{(k)}$ $P^{(k)} = R^{(k)} - P^{(k-1)}\beta^{(k-1)}$ end for

end procedure

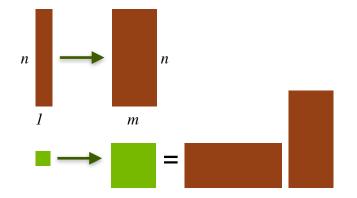
share Krylov space between multiple rhs



procedure BLOCKCG $R^{(0)} = B - AX^{(0)}$ $P^{(0)} = R^{(0)}$ for $k = 1, 2, \ldots$ until converged **do** $Z^{(k-1)} = A P^{(k-1)}$ $\alpha^{(k-1)} = \left[(P^{(k-1)})^H Z^{(k-1)} \right]^{-1} (R^{(k-1)})^H R^{(k-1)}$ $X^{(k)} = X^{(k-1)} + P^{(k-1)}\alpha^{(k-1)}$ $R^{(k)} = R^{(k-1)} - Z^{(k-1)} \alpha^{(k-1)}$ $\beta^{(k-1)} = \left[(R^{(k-1)})^H R^{(k-1)} \right]^{-1} (R^{(k)})^H R^{(k)}$ $P^{(k)} = R^{(k)} - P^{(k-1)}\beta^{(k-1)}$ end for

end procedure

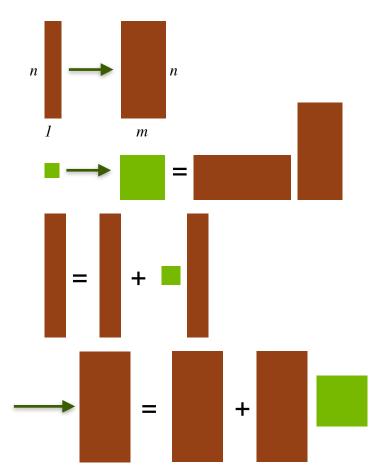
share Krylov space between multiple rhs



procedure BLOCKCG $R^{(0)} = B - AX^{(0)}$ $P^{(0)} = R^{(0)}$ for $k = 1, 2, \ldots$ until converged **do** $Z^{(k-1)} = A P^{(k-1)}$ $\alpha^{(k-1)} = \left[(P^{(k-1)})^H Z^{(k-1)} \right]^{-1} (R^{(k-1)})^H R^{(k-1)}$ $X^{(k)} = X^{(k-1)} + P^{(k-1)}\alpha^{(k-1)}$ $R^{(k)} = R^{(k-1)} - Z^{(k-1)} \alpha^{(k-1)}$ $\beta^{(k-1)} = \left[(R^{(k-1)})^H R^{(k-1)} \right]^{-1} (R^{(k)})^H R^{(k)}$ $P^{(k)} = R^{(k)} - P^{(k-1)}\beta^{(k-1)}$

end for end procedure

share Krylov space between multiple rhs

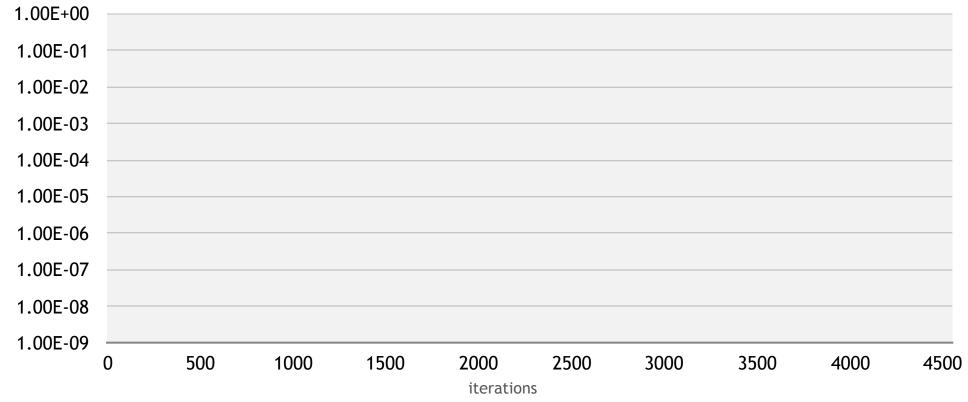


procedure BLOCKCG $R^{(0)} = B - AX^{(0)}$ $P^{(0)} = R^{(0)}$ for $k = 1, 2, \ldots$ until converged **do** $Z^{(k-1)} = A P^{(k-1)}$ $\alpha^{(k-1)} = \left[(P^{(k-1)})^H Z^{(k-1)} \right]^{-1} (R^{(k-1)})^H R^{(k-1)}$ $X^{(k)} = X^{(k-1)} + P^{(k-1)}\alpha^{(k-1)}$ $R^{(k)} = R^{(k-1)} - Z^{(k-1)} \alpha^{(k-1)}$ $\beta^{(k-1)} = \left[(R^{(k-1)})^H R^{(k-1)} \right]^{-1} (R^{(k)})^H R^{(k)}$ $P^{(k)} = R^{(k)} - P^{(k-1)}\beta^{(k-1)}$

end for end procedure

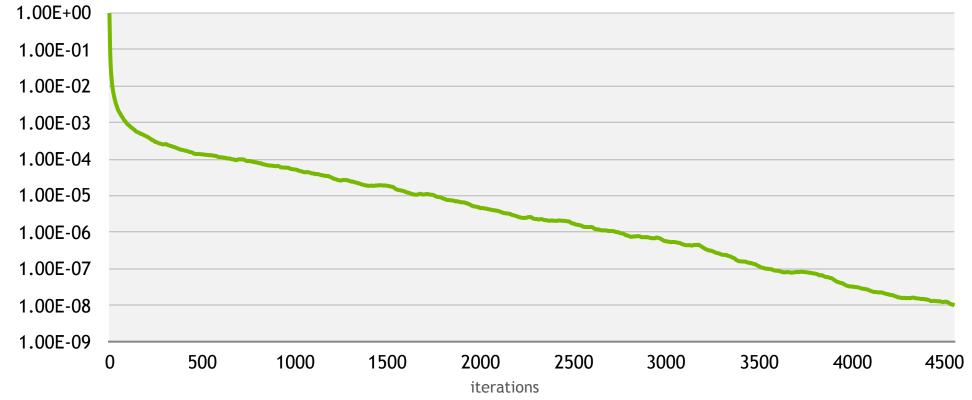
HISQ, 32³x8, Gaussian random source

$$-1$$
 -2 -4 -8 -12 -16



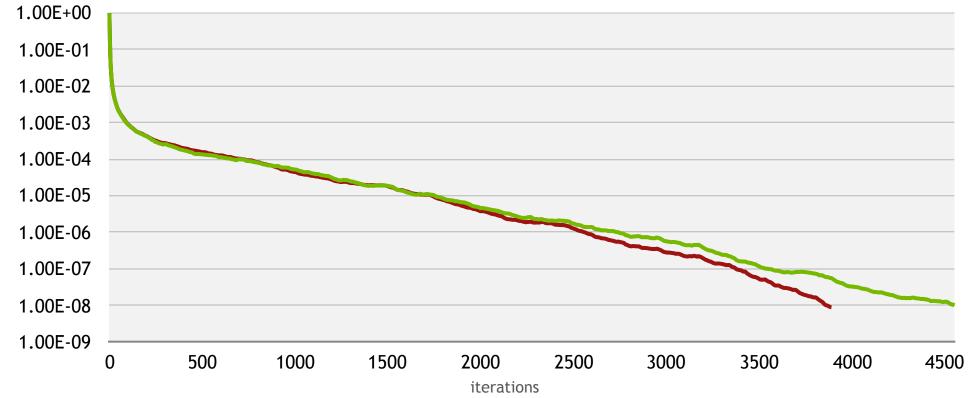
HISQ, 32³x8, Gaussian random source

$$-1$$
 -2 -4 -8 -12 -16



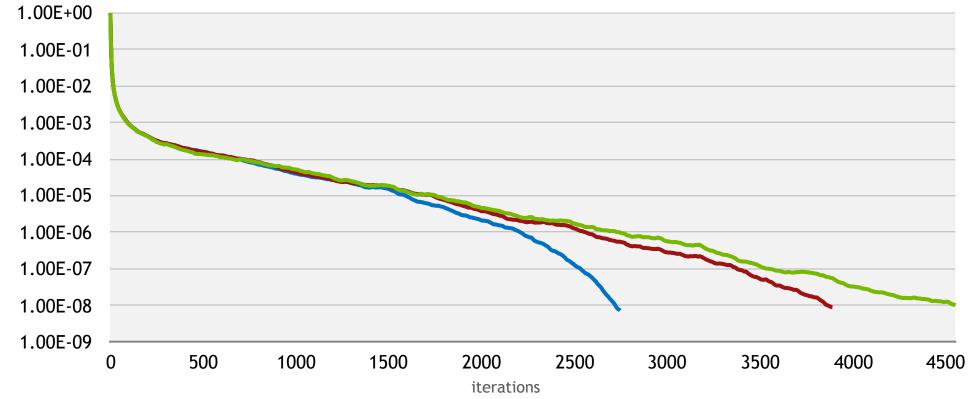
HISQ, 32³x8, Gaussian random source

$$-1$$
 -2 -4 -8 -12 -16



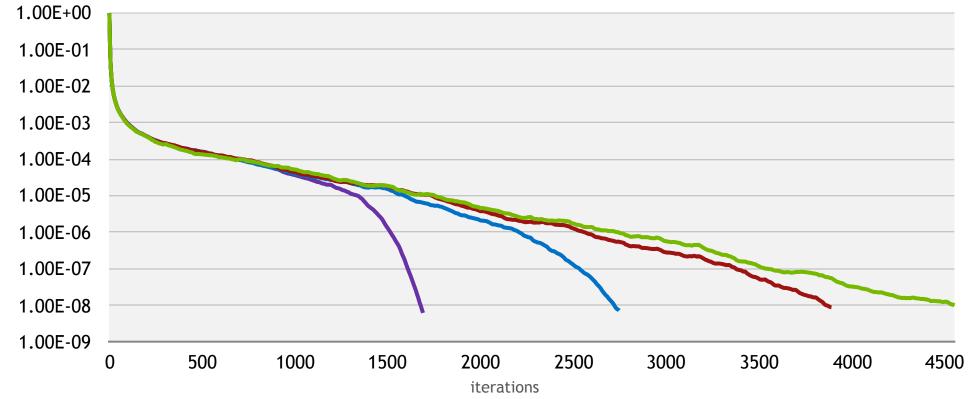
HISQ, 32³x8, Gaussian random source

$$-1$$
 -2 -4 -8 -12 -16



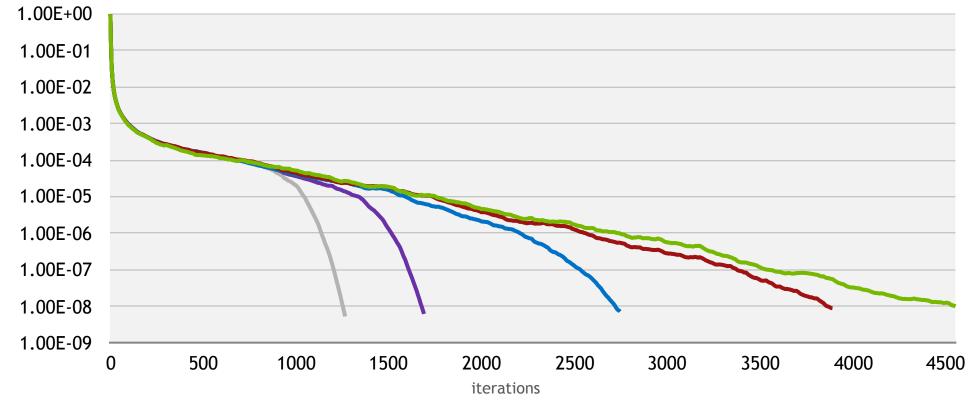
HISQ, 32³x8, Gaussian random source

$$-1$$
 -2 -4 -8 -12 -16



HISQ, 32³x8, Gaussian random source

$$-1$$
 -2 -4 -8 -12 -16

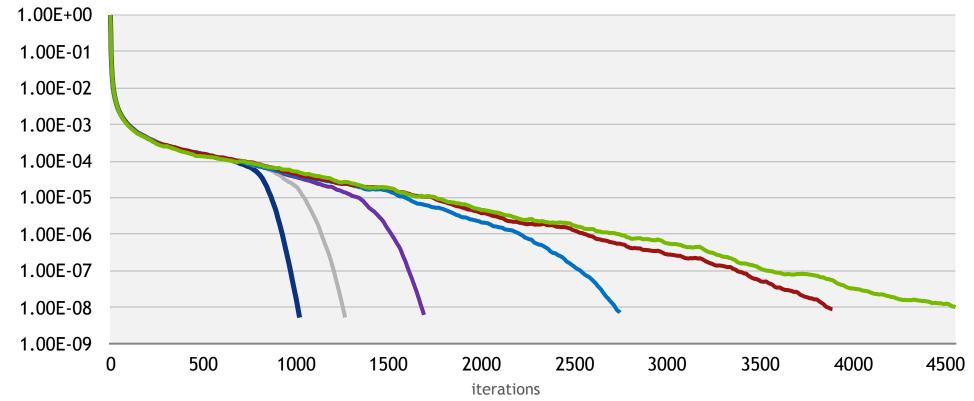


residual

18 📀 nvidia

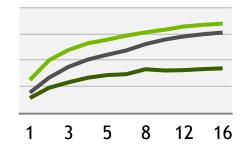
HISQ, 32³x8, Gaussian random source

$$-1$$
 -2 -4 -8 -12 -16



SCALING

Dslash exploits reuse of gauge field



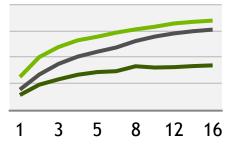


SCALING

Dslash exploits reuse of gauge field

Linear Algebra number of dot products scales quadratically number of axpy calls scales quadratically

$$\alpha_{ij} = \langle x_i, x_j \rangle$$
$$y_i = \sum a_{ij} x_j + y_i$$

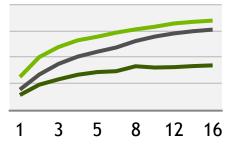


SCALING

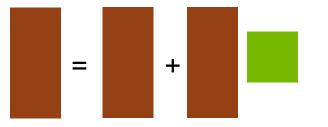
Dslash exploits reuse of gauge field

Linear Algebra number of dot products scales quadratically number of axpy calls scales quadratically

$$\alpha_{ij} = \langle x_i, x_j \rangle$$
$$y_i = \sum a_{ij} x_j + y_i$$



to overcome quadratically scaling GPIOP Pacel Whitepaper



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

CUDA supports two dimensional grid blocks: easy to exploit locality for texture cache / shared memory



00 GPU Hardware Architecture

Note: Kepler GK110 had a 3:1 ratio of SP units to DP units.

to overcome quadratically scaling GPIOD Pascal Whitepaper



CUDA supports two dimensional grid blocks: easy to exploit locality for texture cache / shared memory

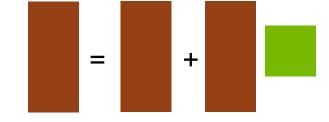
 $y_i = \sum a_{ij} x_j + y_i$



00 GPU Hardware Architecture



to overcome quadratically scaling GPIOD Pascal Whitepaper



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

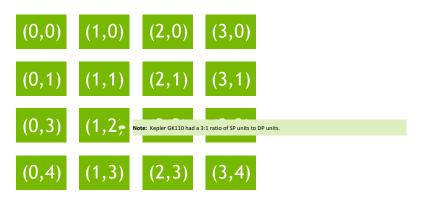
CUDA supports two dimensional grid blocks: easy to exploit locality for texture cache / shared memory

$$y_0(0) = a_{00}x_0(0) + a_{01}x_1(0) + \dots$$

$$y_1(0) = a_{10}x_0(0) + a_{11}x_1(0) + \dots$$

$$y_2(0) = a_{20}x_0(0) + a_{21}x_1(0) + \dots$$

$$y_3(0) = a_{30}x_0(0) + a_{31}x_1(0) + \dots$$





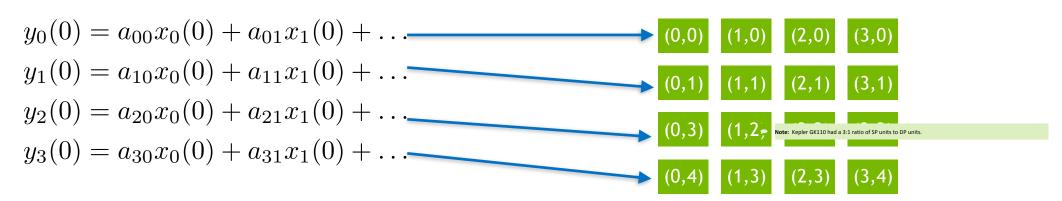
100 GPU Hardware Architecture Ir

to overcome quadratically scaling GPIDO Pascal Whitepaper



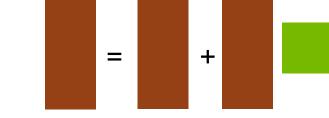
CUDA supports two dimensional grid blocks:

easy to exploit locality for texture cache / shared memory



SM																	
							Instruct	ion Cache									
			nstructi	-	Б		Instruction Buffer										
	Warp Scheduler									Warp Scheduler							
Dispatch Unit Dispatch Unit							Dispatch Unit Dispatch Unit										
Register File (32,768 x 32-bit)							Register File (32,768 x 32-bit)										
Core	Core	DP Unit	Core	Core	DP Unit		SFU	Core	Core	DP Unit	Core	Core	DP Unit		SFU		
Core	Core	DP Unit	Core	Core	DP Unit		SFU	Core	Core	DP Unit	Core	Core	DP Unit		SFU		
Core	Core	DP Unit	Core	Core	DP Unit		SFU	Core	Core	DP Unit	Core	Core	DP Unit		SFU		
Core	Core	DP Unit	Core	Core	DP Unit		SFU	Core	Core	DP Unit	Core	Core	DP Unit		SFU		
Core	Core	DP Unit	Core	Core	DP Unit		SFU	Core	Core	DP Unit	Core	Core	DP Unit		SFU		
Core	Core	DP Unit	Core	Core	DP Unit		SFU	Core	Core	DP Unit	Core	Core	DP Unit		SFU		
Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU		
Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU		
Texture / L1 Cache																	
	Tex Tex						Tex Tex										
64KB Shared Memory																	

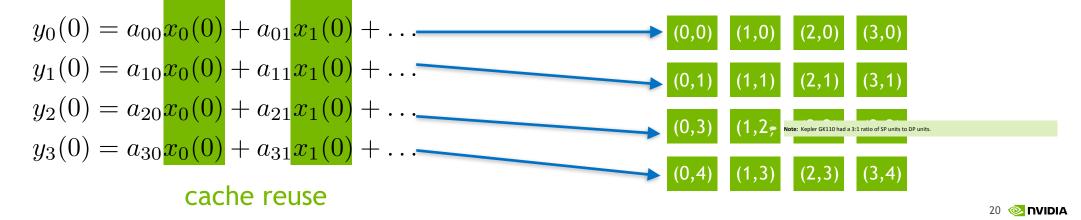
to overcome quadratically scaling GPIOP Pacel Whitepaper



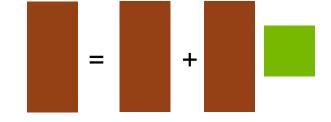
 $y_i = \sum a_{ij} x_j + y_i$

CUDA supports two dimensional grid blocks: easy to exploit locality for texture cache / shared memory



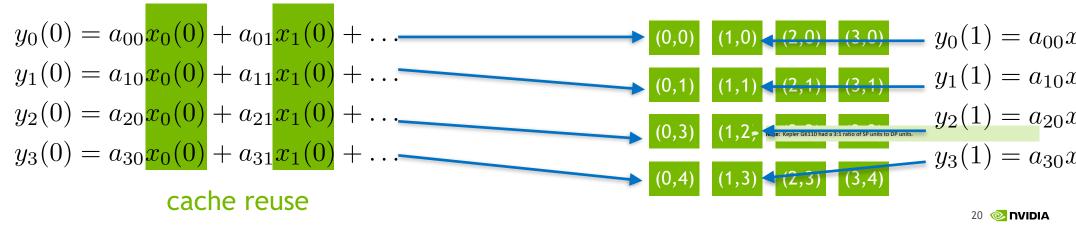


to overcome quadratically scaling GPIOP Pacel Whitepaper



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

CUDA supports two dimensional grid blocks: easy to exploit locality for texture cache / shared memory



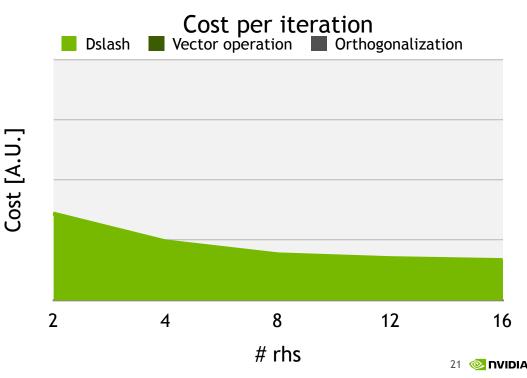
							Instructi	on Cache									
Instruction Buffer									Instruction Buffer								
			Warp So	heduler							Warp So	heduler					
Dispatch Unit Dispatch Unit								Dispatch Unit Dispatch Unit									
		Regist	er File (:	32,768 x	32-bit)					Regist	er File (:	32,768 x	32-bit)				
Core	Core	DP Unit	Core	Core	DP Unit		SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU		
Core	Core	DP Unit	Core	Core	DP Unit		SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU		
Core	Core	DP Unit	Core	Core	DP Unit		SFU	Core	Core	DP Unit	Core	Core	DP Unit	LDIST	SFU		
Core	Core	DP Unit	Core	Core	DP Unit		SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU		
Core	Core	DP Unit	Core	Core	DP Unit		SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU		
Core	Core	DP Unit	Core	Core	DP Unit		SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU		
Core	Core	DP Unit	Core	Core	DP Unit		SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU		
Core	Core	DP Unit	Core	Core	DP Unit		SFU	Core	Core	DP Unit	Core	Core	DP Unit	LD/ST	SFU		
							Texture	L1 Cache	(

GP100 GPU Hardware Architecture In-Dep

BlockCG is not always numerically stable

simple approach: Gram-Schmidt or modified Gram-Schmidt

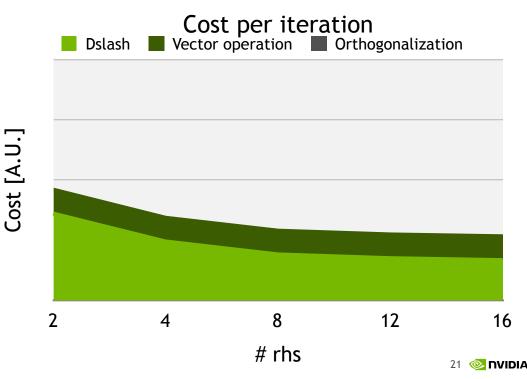
becomes prohibitively expensive



BlockCG is not always numerically stable

simple approach: Gram-Schmidt or modified Gram-Schmidt

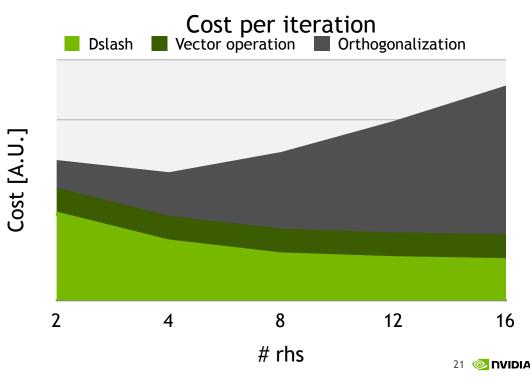
becomes prohibitively expensive



BlockCG is not always numerically stable

simple approach: Gram-Schmidt or modified Gram-Schmidt

becomes prohibitively expensive

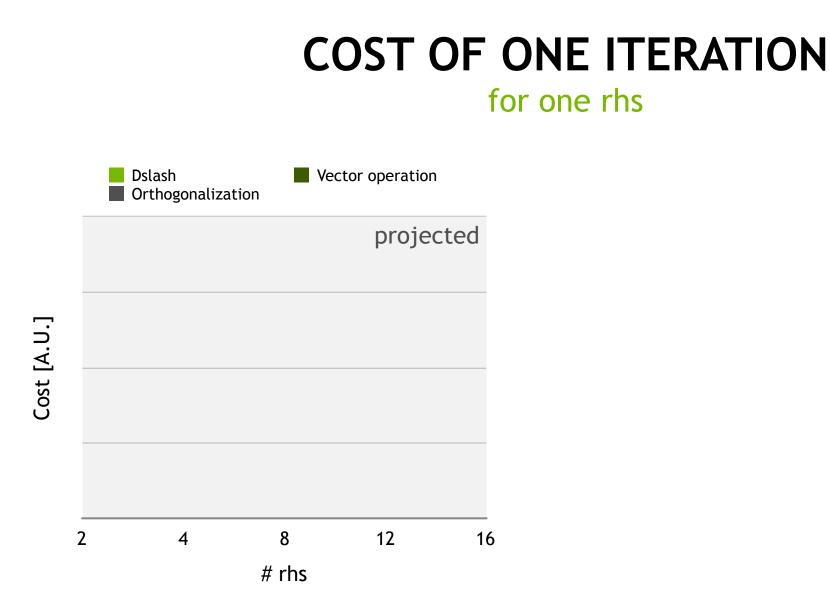


simple approach: Gram-Schmidt or modified Gram-Schmidt becomes prohibitively expensive

CholQR

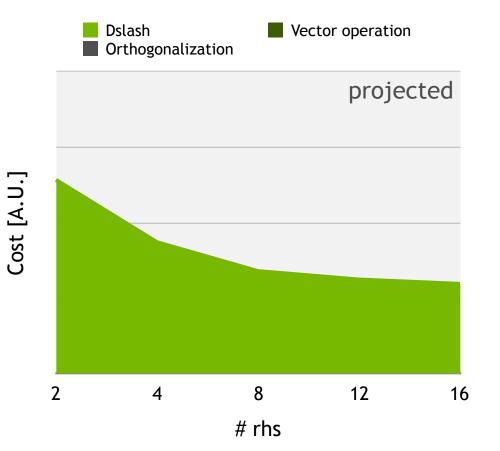
Gram-Matrix:	$B = R^H R$	m imes m dot products of length n
Cholesky Decomposition	$S^H S = B$	of $m imes m$ matrix
apply to vectors	$Q = RS^{-1}$	axpy $m imes m$ (output, input)

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



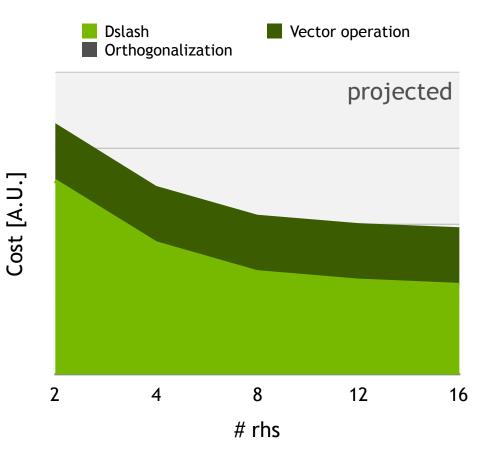
23 📀 nvidia





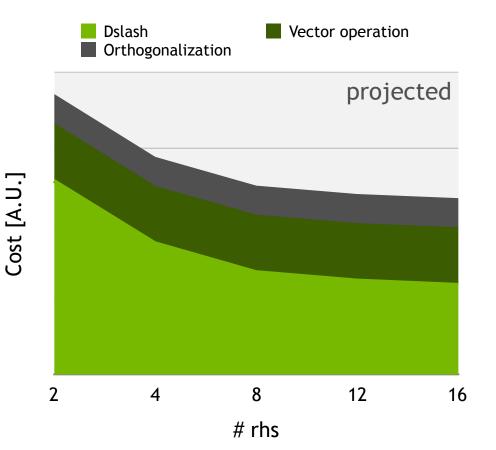
large benefits from multi-src Dslash

COST OF ONE ITERATION for one rhs



large benefits from multi-src Dslash

COST OF ONE ITERATION for one rhs

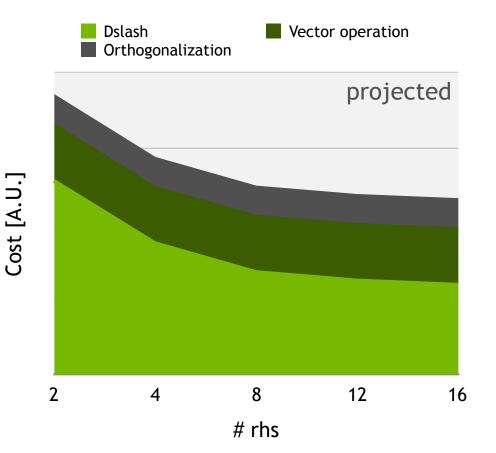


large benefits from multi-src Dslash

linear algebra and orthogonalization stay constant

23 💿 nvidia

COST OF ONE ITERATION for one rhs



large benefits from multi-src Dslash

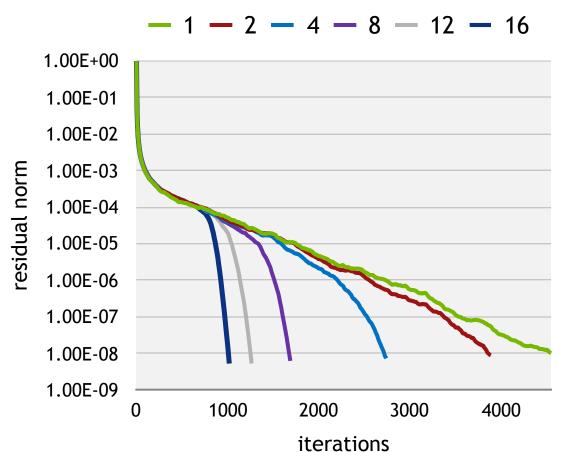
linear algebra and orthogonalization stay constant

relative importance of Dslash reduces



WORK TO BE DONE

your milage may vary



stability needs real world testing orthogonalization might be necessary

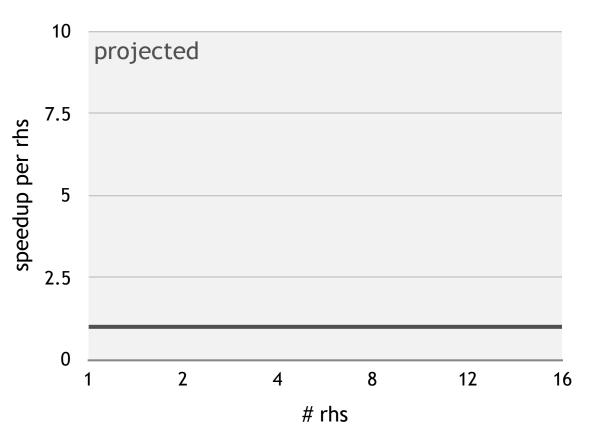
iteration count improvement may depend on gauge field and sources

need to finish up implementation

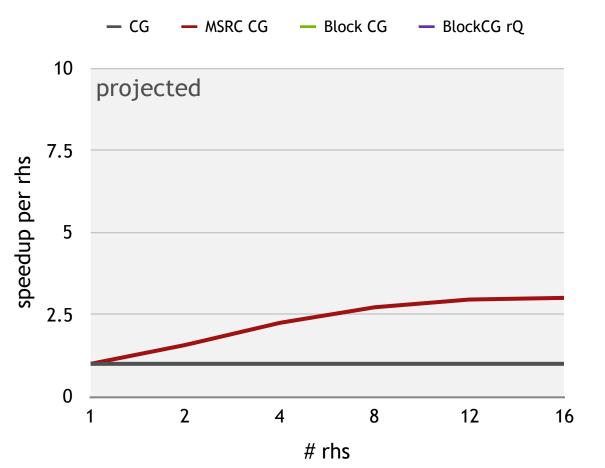
add mixed precision

multi-rhs Block Solvers provide an easy drop in

— CG — MSRC CG — Block CG — BlockCG rQ



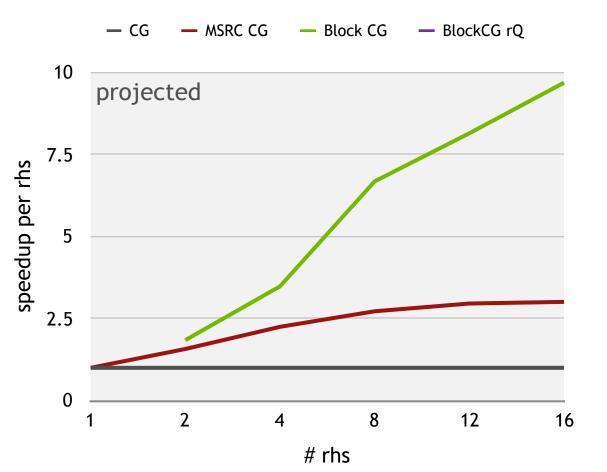
multi-rhs Block Solvers provide an easy drop in



reuse gauge field for Dslash

26 📀 nvidia

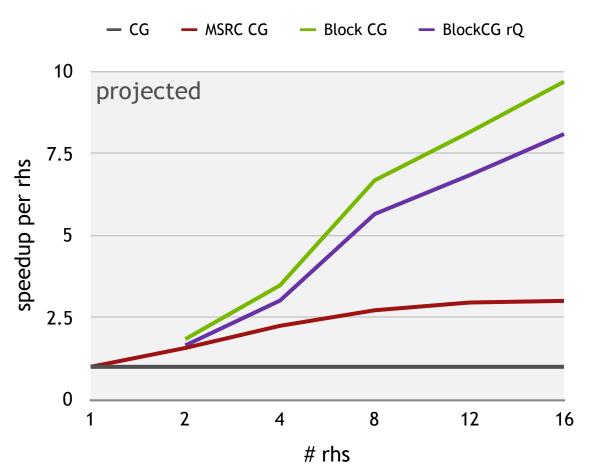
multi-rhs Block Solvers provide an easy drop in



reuse gauge field for Dslash

reduced iteration count

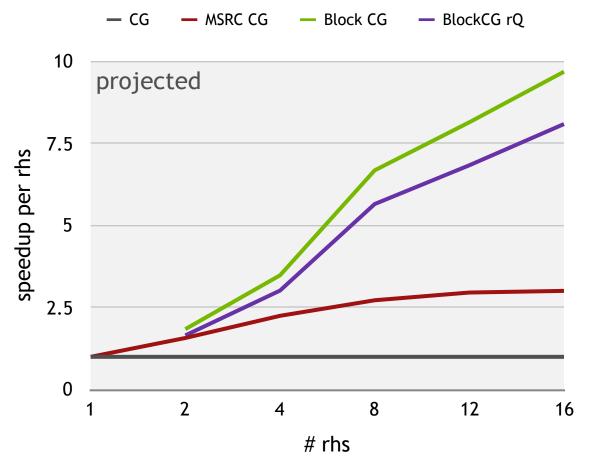
multi-rhs Block Solvers provide an easy drop in



reuse gauge field for Dslash

reduced iteration count

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

