GOING WIDE AND LOCAL

BLOCK KRYLOV SPACE SOLVERS AS A MEANS TO ACHIEVE BETTER COMPUTE RESOURCE UTILIZATION

Kate Clark, Alexei Strelchenko, Alejandro Vaquero
Mathias Wagner, Evan Weinberg

NVIDIA

HPC BEYOND MOORE'S LAW

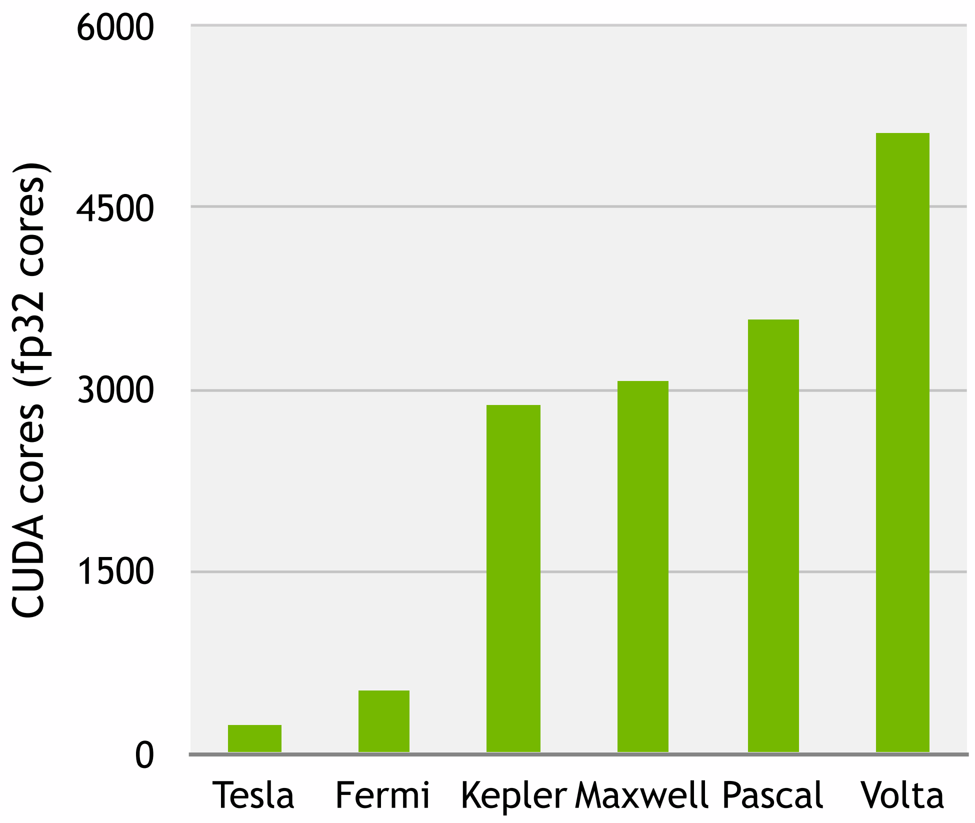
going wide

CPUs and GPUs becoming wider

increase in flops is driven by more cores

also applies to CPUs (server to mobile)

need sufficient amount of parallelism to fill architectures



HPC BEYOND MOORE'S LAW

going wide

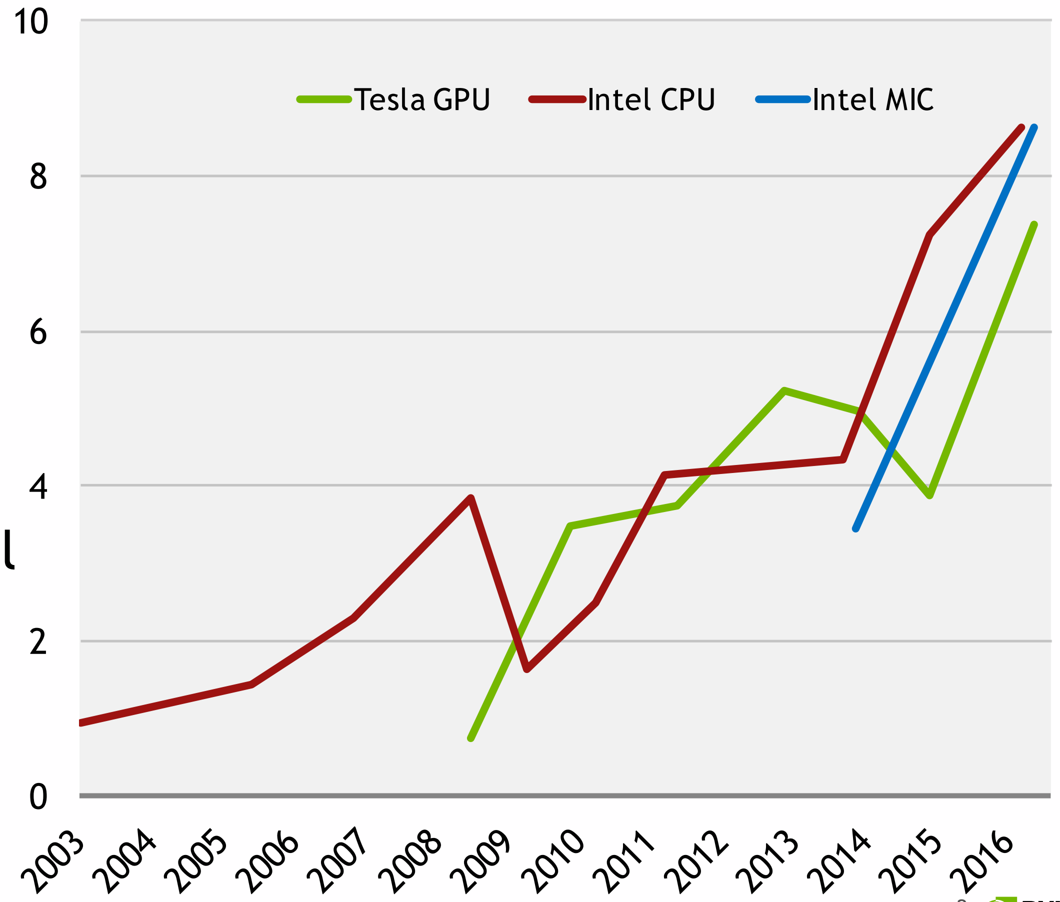
CPUs and GPUs becoming wider

increase in flops is driven by more cores

also applies to CPUs (server to mobile)

need sufficient amount of parallelism to fill architectures

need to be able to feed the cores



AGENDA

QUDA

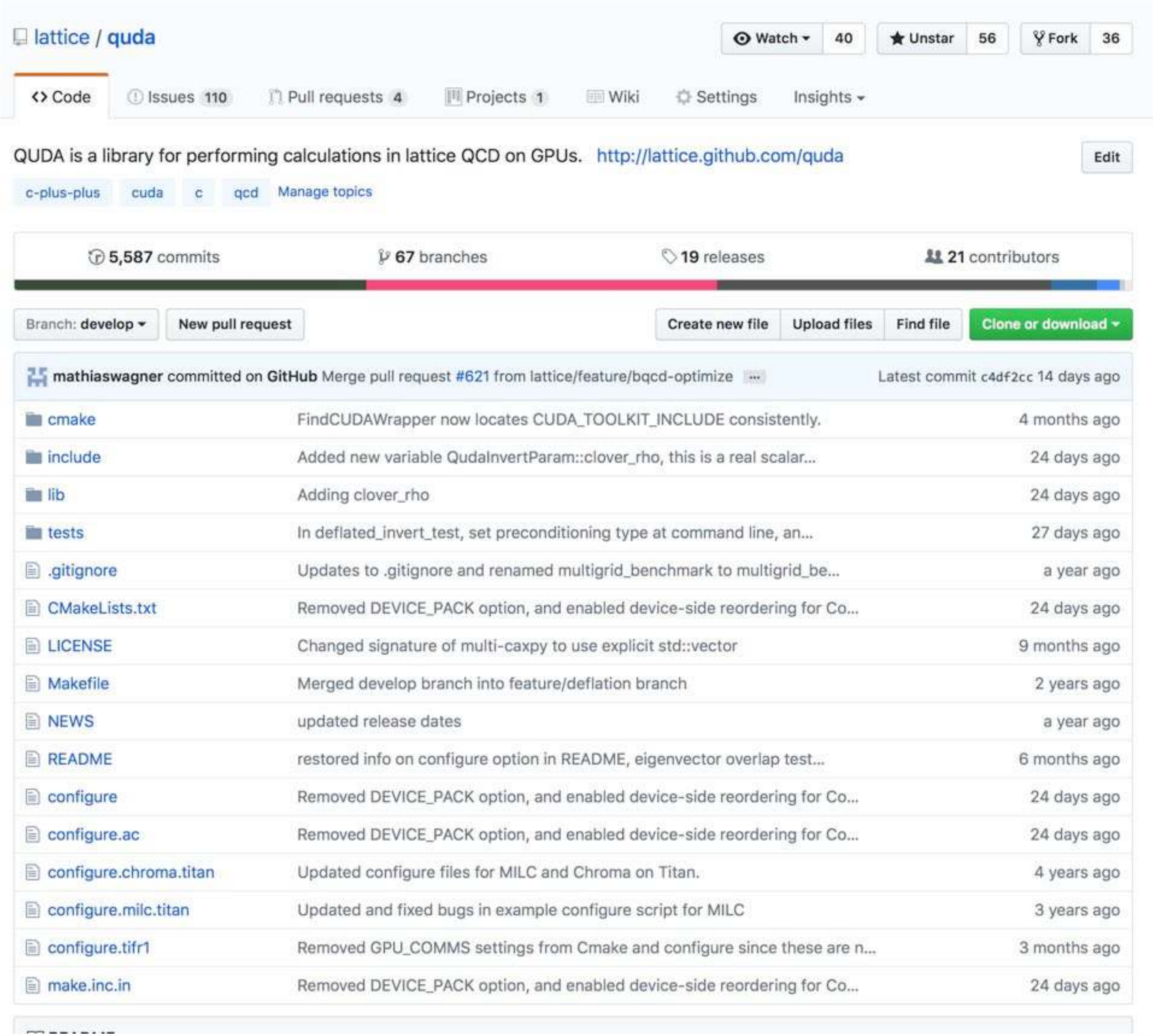
Block Krylov solvers

Efficient implementation

Time to solution

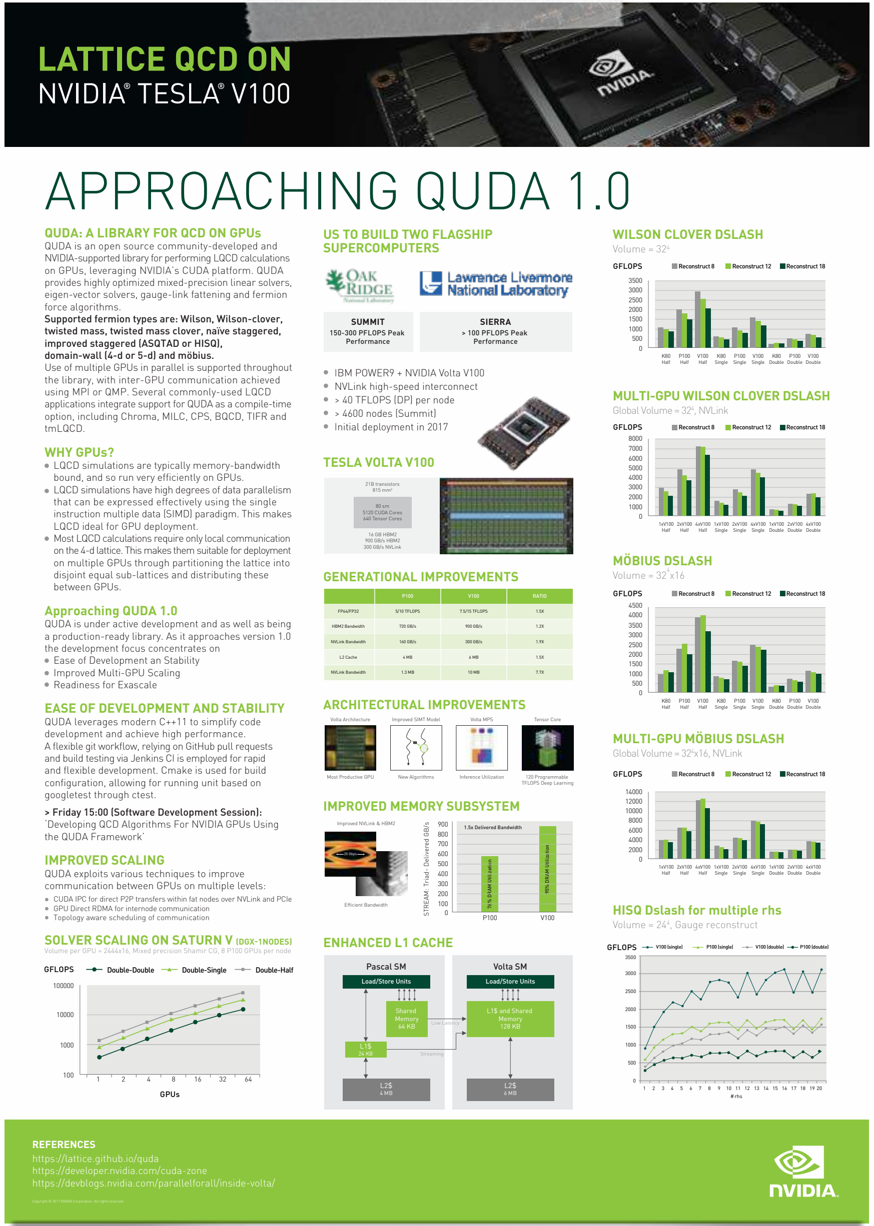
http://lattice.github.com/quda

QUDA
LATTICE QCD ON GPUS



POSTER SESSION

APPROACHING QUDA 1.0



LATTICE QCD ON NVIDIA® TESLA® V100

APPROACHING QUDA 1.0

QUDA: A LIBRARY FOR QCD ON GPUs

QUDA is an open source community-developed and NVIDIA-supported library for performing LQCD calculations on GPUs, leveraging NVIDIA's CUDA platform. QUDA provides highly optimized mixed-precision linear solvers, eigen-vector solvers, gauge-link fattening and fermion force algorithms.

Supported fermion types are: Wilson, Wilson-clover, twisted mass, twisted mass clover, naïve staggered, improved staggered (ASQTAD or HISQ), domain-wall (4-d or 5-d) and möbius.

Use of multiple GPUs in parallel is supported throughout the library, with inter-GPU communication achieved using MPI or QMP. Several commonly-used LQCD applications integrate support for QUDA as a compile-time option, including Chroma, MILC, CPS, BQCD, TIFR and tmLQCD.

WHY GPUs?

- LQCD simulations are typically memory-bandwidth bound, and so run very efficiently on GPUs.
- LQCD simulations have high degrees of data parallelism that can be expressed effectively using the single instruction multiple data (SIMD) paradigm. This makes LQCD ideal for GPU deployment.
- Most LQCD calculations require only local communication on the 4-d lattice. This makes them suitable for deployment on multiple GPUs through partitioning the lattice into disjoint equal sub-lattices and distributing these between GPUs.

Approaching QUDA 1.0

QUDA is under active development and as well as being a production-ready library. As it approaches version 1.0 the development focus concentrates on

- Ease of Development an Stability
- Improved Multi-GPU Scaling
- Readiness for Exascale

EASE OF DEVELOPMENT AND STABILITY

QUDA leverages modern C++11 to simplify code development and achieve high performance.

A flexible git workflow, relying on GitHub pull requests and build testing via Jenkins CI is employed for rapid and flexible development. Cmake is used for build configuration, allowing for running unit based on googletest through ctest.

> Friday 15:00 (Software Development Session):
'Developing QCD Algorithms For NVIDIA GPUs Using the QUDA Framework'

IMPROVED SCALING

QUDA exploits various techniques to improve communication between GPUs on multiple levels:

- CUDA IPC for direct P2P transfers within fat nodes over NVLink and PCIe
- GPU Direct RDMA for internode communication
- Topology aware scheduling of communication

SOLVER SCALING ON SATURN V (DGX-1NODES)

Volume per GPU = 2444x16, Mixed precision Shamir CG, 8 P100 GPUs per node

US TO BUILD TWO FLAGSHIP SUPERCOMPUTERS

SUMMIT	SIERRA
150-300 PFLOPS Peak Performance	> 100 PFLOPS Peak Performance

- IBM POWER9 + NVIDIA Volta V100
- NVLink high-speed interconnect
- > 40 TFLOPS (DP) per node
- > 4600 nodes (Summit)
- Initial deployment in 2017

TESLA VOLTA V100

21B transistors
815 mm²
80 sm
5120 CUDA Cores
640 Tensor Cores
16 GB HBM2
900 GB/s HBM2
300 GB/s NVLink

GENERATIONAL IMPROVEMENTS

	P100	V100	RATIO
FP64/FP32	5/10 TFLOPS	7.5/15 TFLOPS	1.5X
HBM2 Bandwidth	720 GB/s	900 GB/s	1.2X
NVLink Bandwidth	160 GB/s	300 GB/s	1.9X
L2 Cache	4 MB	6 MB	1.5X
NVLink Bandwidth	1.3 MB	10 MB	7.7X

ARCHITECTURAL IMPROVEMENTS

IMPROVED MEMORY SUBSYSTEM

ENHANCED L1 CACHE

WILSON CLOVER DSLASH

Volume = 32^4

MULTI-GPU WILSON CLOVER DSLASH

Global Volume = 32^4, NVLink

MÖBIUS DSLASH

Volume = 32^4x16

MULTI-GPU MÖBIUS DSLASH

Global Volume = 32^4x16, NVLink

HISQ Dslash for multiple rhs

Volume = 24^4, Gauge reconstruct

REFERENCES
https://lattice.github.io/quda
https://developer.nvidia.com/cuda-zone
https://devblogs.nvidia.com/parallelforall/inside-volta/

QUDA

driven by its developer community

Ron Babich (NVIDIA)
Simone Bacchio (Cyprus)
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 (NIH)
Joel Giedt (Rensselaer Polytechnic Institute)
Arjun Gambhir (William and Mary)
Steve Gottlieb (Indiana University)
Kyriakos Hadjiyiannakou (Cyprus)

Dean Howarth (Rensselaer Polytechnic Institute)
Bálint Joó (Jlab)
Hyung-Jin Kim (BNL -> Samsung)
Bartek Kostrzewa (Bonn)
Claudio Rebbi (Boston University)
Hauke Sandmeyer (Bielefeld)
Guochun Shi (NCSA -> Google)
Mario Schröck (INFN)
Alexei Strelchenko (FNAL)
Alejandro Vaquero (Utah University)
Mathias Wagner (NVIDIA)
Evan Weinberg (Boston University)
Frank Winter (Jlab)
Your Name Here

> Friday 15:00 (Software Development): *'Developing QCD Algorithms For NVIDIA GPUs Using the QUDA Framework'*

THE OLD WORK HORSE

CONJUGATE GRADIENT

procedure CG

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

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

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

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

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

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

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

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

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

end for

end procedure

matrix-vector operation dominates runtime

caxpy BLAS operations

Reductions

THE NEW TRACTOR

BLOCK KRYLOV SPACE SOLVERS

Share the Krylov space

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

retooled BlockCG by Dubrulle 2001

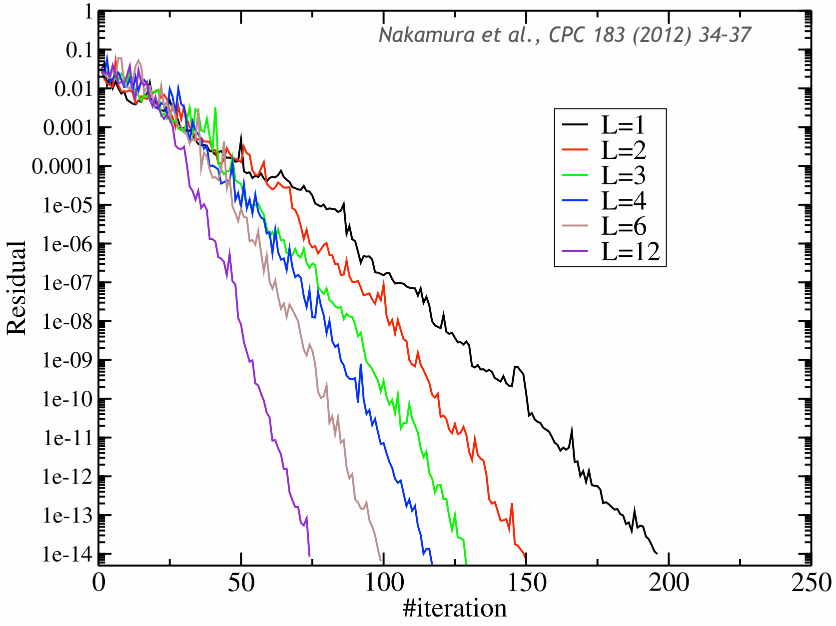
In exact precision converges in N / rhs iterations

Application in QCD:

Nakamura et. (modified block BiCGStab)

Birk and Frommer (block methods,

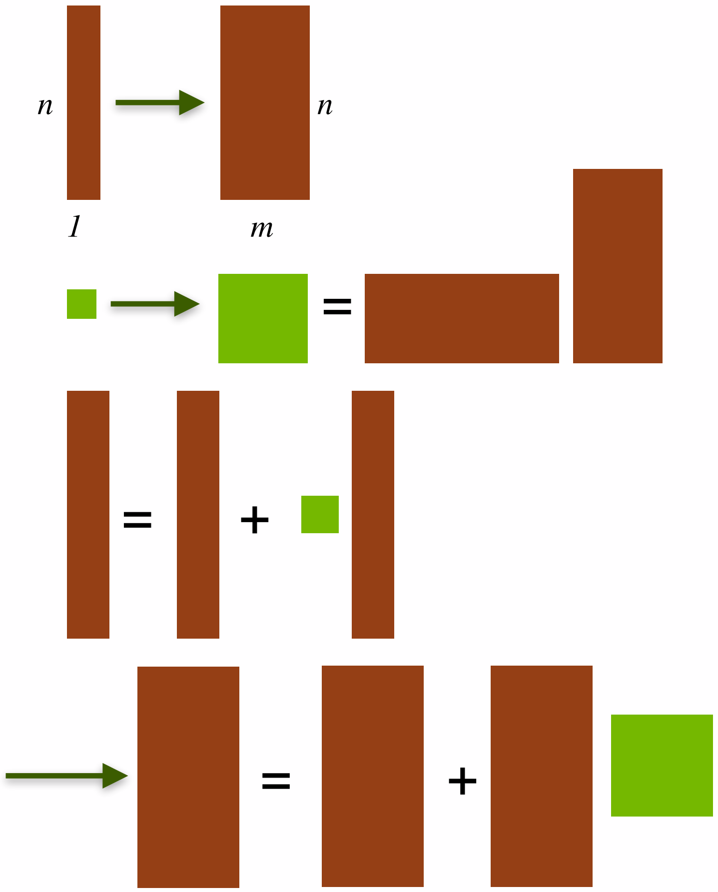
including block methods for multi shift)



 NVIDIA

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**

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

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

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

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

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

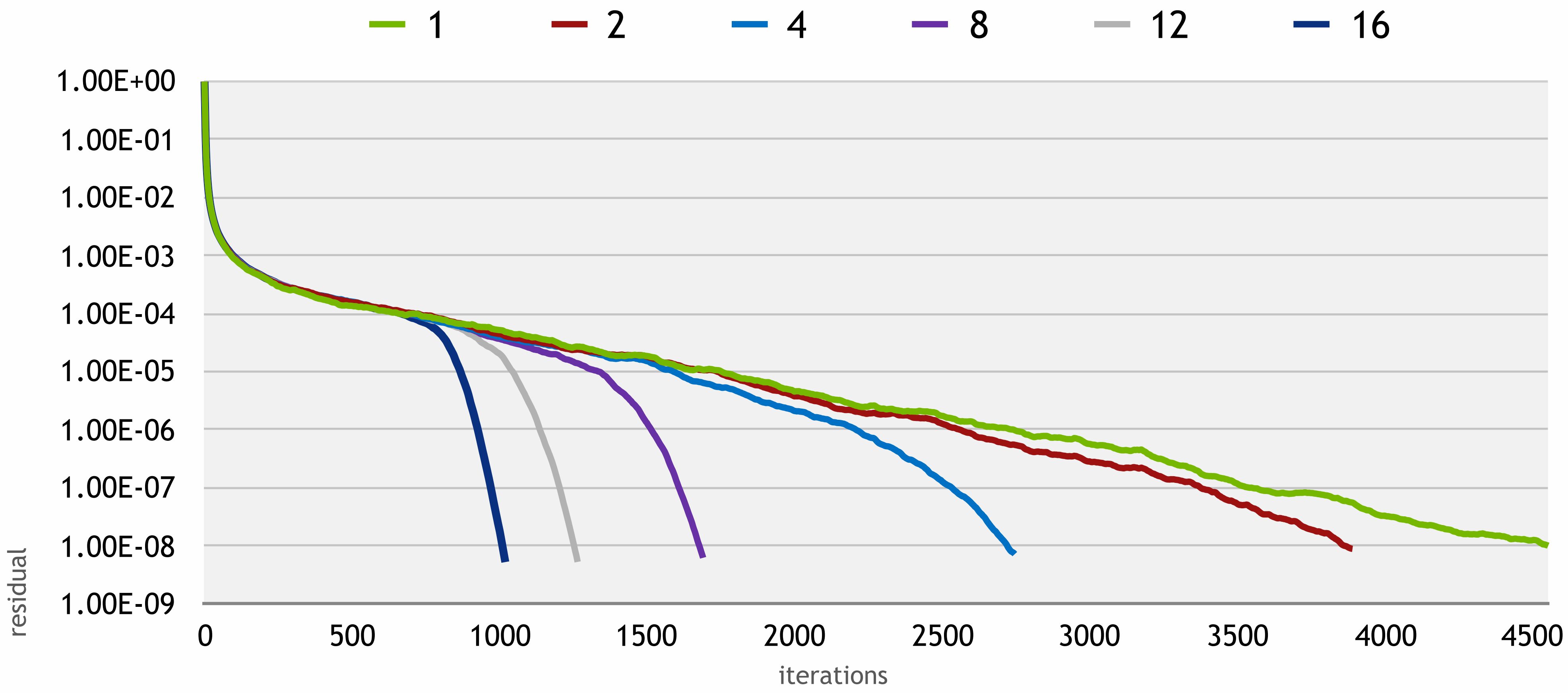
$P^{(k)} = R^{(k)} - P^{(k-1)}\beta^{(k-1)}$

end for

end procedure

REDUCED ITERATION COUNT

HISQ, 32^3x8, Gaussian random source



BLOCK CGRQ

Improve numerical behavior through orthogonalization

$A \in \mathcal{C}^{L \times L}; R, B, X, Q \in \mathcal{C}^{L \times N}; C, S, \beta \in \mathcal{C}^{N \times N}$

procedure BLOCKCGRQ($X^{L \times N}, B^{L \times N}$)

$R = B - AX$

$QC = R$ ▷ QR decomposition

$S = \mathbb{I}^{N \times N}$

$P = 0$

while not converged **do**

$P = Q + PS^\dagger$

$\beta = \left(P^\dagger AP\right)^{-1}$

$X = X + P\beta C$

$QS = Q - AP\beta$ ▷ QR decomposition

$C = SC$

end while

end procedure

ORTHOGONALIZATION

THIN QR

simple approach: (modified) Gram-Schmidt becomes prohibitively expensive

THIN QR

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 BLAS operations and reductions

Cholesksky Decompostion of small matrix efficient on the CPU (using Eigen)

INCREASED COST PER ITERATION

DSLASH

apply Dslash to m rhs

naively scales linear

constant time per rhs

BLAS

couples m output vectors with m intput vectors via $m \times m$ matrix a

Quadratic scaling with m

used for orthogonalization

linear scaling of cost per rhs

REDUCTION

Instead of 1 dot product need to evaluate $m \times m$ dot products

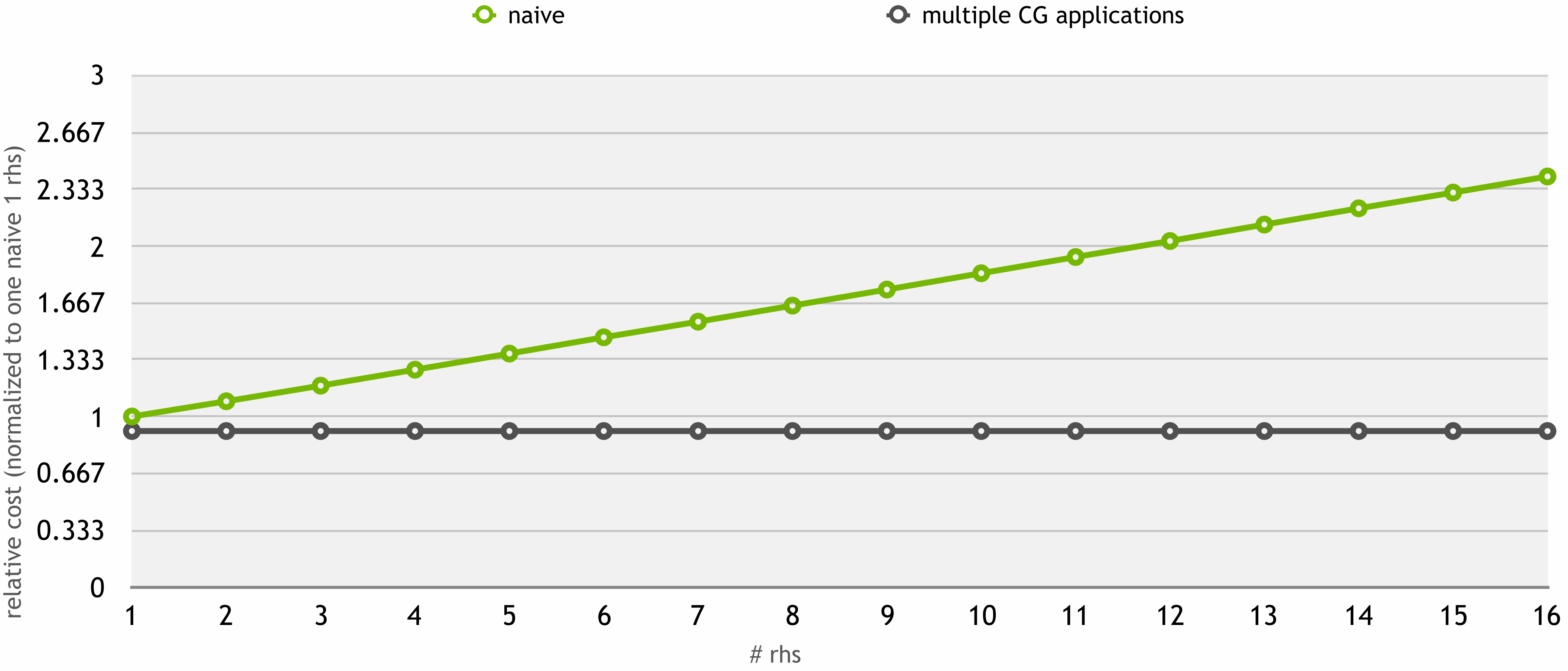
Quadratic scaling with m

used for orthogonalization

linear scaling of cost per rhs

COST PER ITERATION

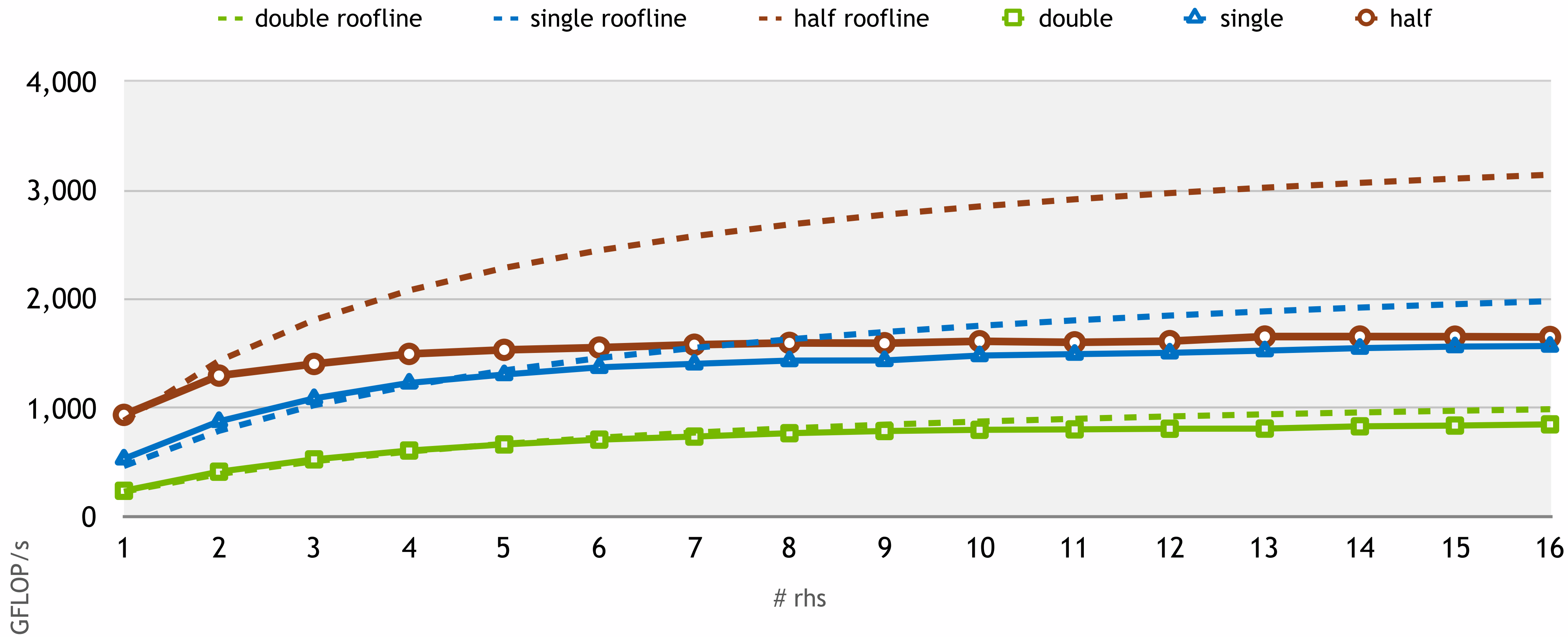
for one rhs



DSLASH FOR MULTIPLE RHS

MULTI-SRC DSLASH

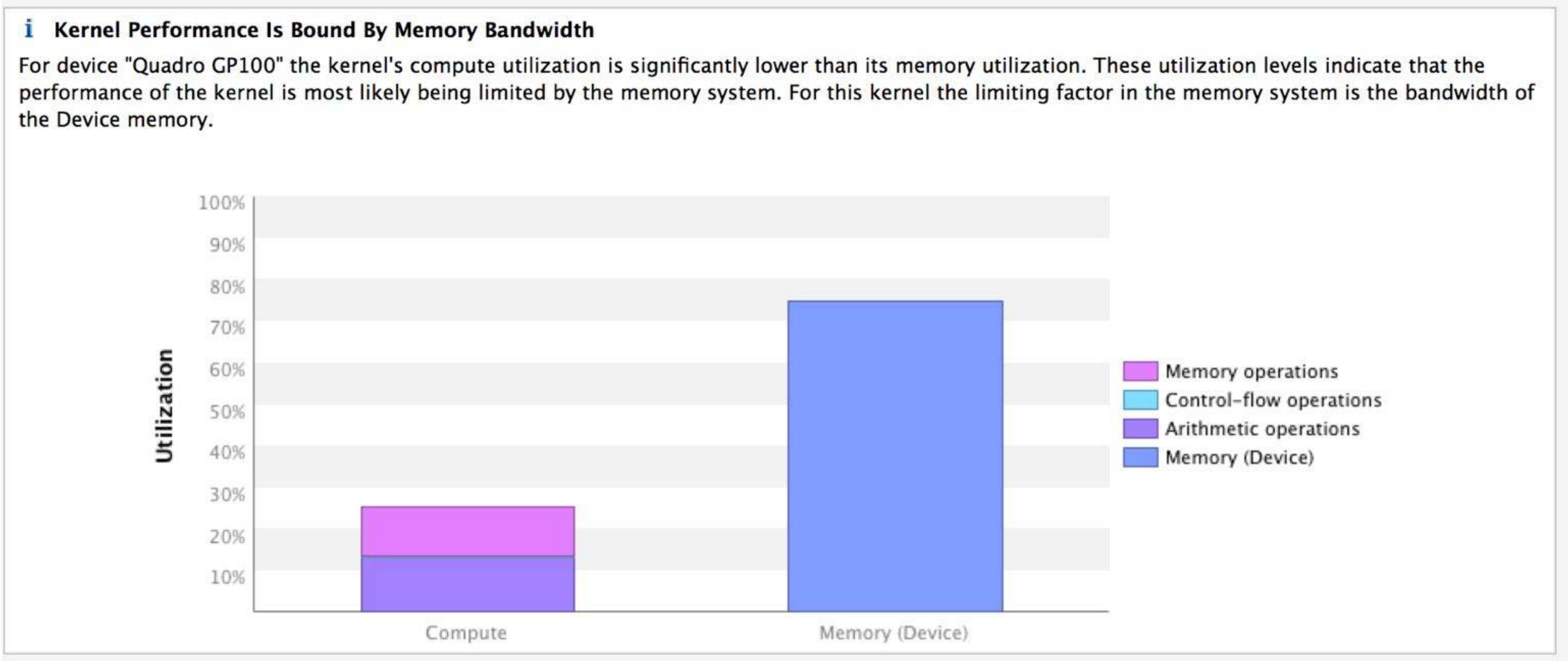
Volume 24^4, HISQ



roofline model: 100% cache reuse in x-direction, 50% in y-direction, Stream-triad bandwidth (550 GB/s for P100)

WHY DON'T WE SEE EXPECTED SCALING

1 rhs Dslash agrees with roofline



WHY DON'T WE SEE EXPECTED SCALING

1 rhs Dslash agrees with roofline

L2 Cache

Reads	6450241	669.848 GB/s	
Writes	124429	12.922 GB/s	
Total	6574670	682.77 GB/s	Idle Low Medium High Max

Unified Cache

Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Global Loads	0	0 B/s	
Global Stores	124416	12.92 GB/s	
Texture Reads	6635520	689.089 GB/s	
Unified Total	6759936	702.01 GB/s	Idle Low Medium High Max

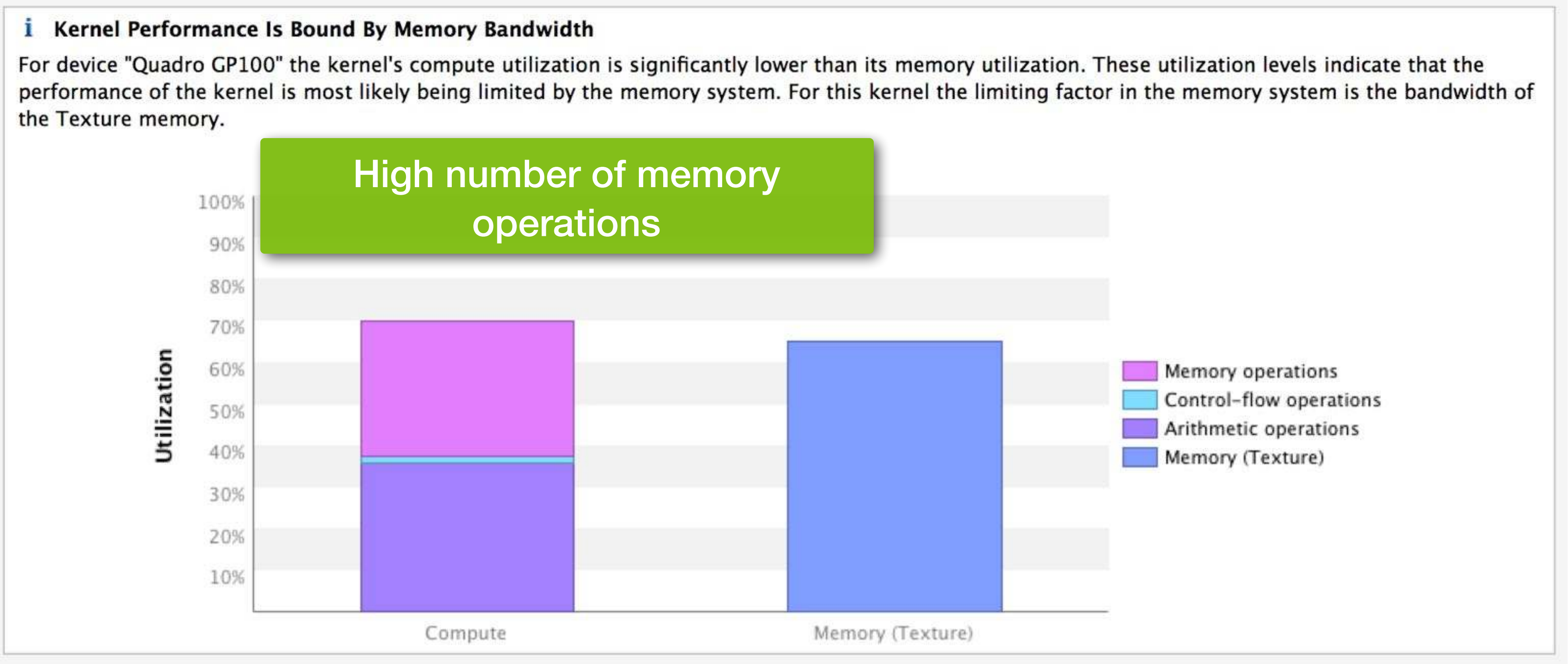
Device Memory

Reads	5014415	520.74 GB/s	
Writes	152277	15.814 GB/s	
Total	5166692	536.554 GB/s	Idle Low Medium High Max

Device memory bandwidth is performance limiter

CACHE BASED MULTI RHS DSLASH

Check extreme cases with 16 rhs



CACHE BASED MULTI RHS DSLASH

Check extreme cases with 16 rhs

L2 Cache

Reads	46681396	703.125 GB/s	
Writes	1990669	29.984 GB/s	
Total	48672065	733.109 GB/s	Idle Low Medium High Max

Unified Cache

Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Global Loads	0	0 B/s	
Global Stores	1990656	29.984 GB/s	
Texture Reads	106168320	1,599.13 GB/s	
Unified Total	108158976	1,629.113 GB/s	Idle Low Medium High Max

Unified Cache is limiting performance

Device Memory

Reads	16534840	249.051 GB/s	
Writes	2020709	30.436 GB/s	
Total	18555549	279.488 GB/s	Idle Low Medium High Max

REGISTER OPTIMIZATION

Hybrid implementation

Cache

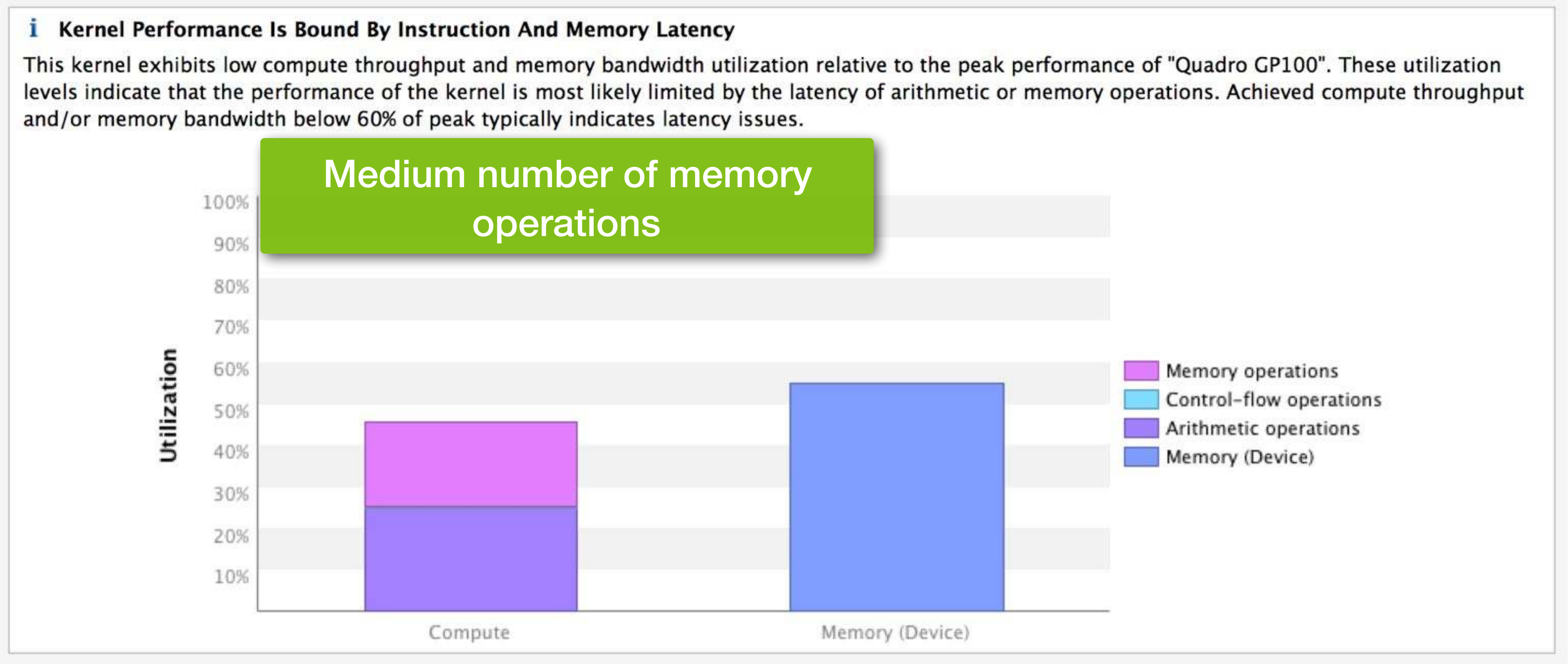
- Use y-dimension of CUDA blocks for rhs
- One lattice site is scheduled on the same SM for all rhs
- Each thread processes one rhs on one lattice site
- Necessary for cache reuse of gauge field

Register + Cache

- Use y -dimension of CUDA blocks for rhs
- One lattice site is schedule on the same SM for all rhs
- Each thread processes multiple rhs on one lattice site (reuse gauge field from registers)
- Reduces cache pressure

REGISTER REUSE DSLASH

Check extreme cases with 16 rhs



REGISTER REUSE DSLASH

Check extreme cases with 16 rhs

L2 Cache

Reads	36434573	772.596 GB/s	
Writes	1990669	42.212 GB/s	
Total	38425242	814.808 GB/s	Idle Low Medium High Max

Unified Cache

Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Global Loads	0	0 B/s	
Global Stores	1990656	42.212 GB/s	
Texture Reads	50429952	1,069.368 GB/s	
Unified Total	52420608	1,111.58 GB/s	Idle Low Medium High Max

Unified Cache is no longer limiting performance

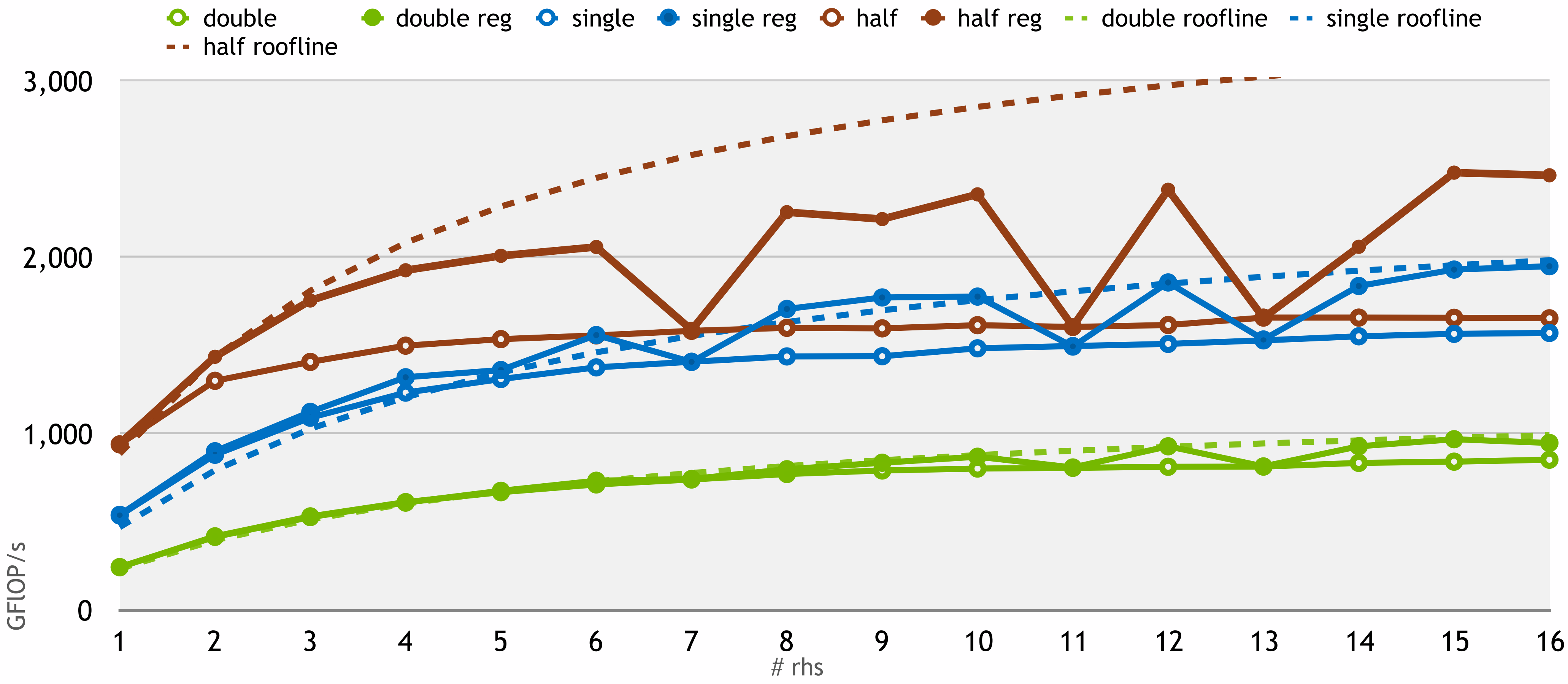
Device Memory

Reads	16840673	357.107 GB/s	
Writes	2013116	42.688 GB/s	
Total	18853789	399.795 GB/s	Idle Low Medium High Max

System Memory [PCIe configuration: Gen3 x16, 8 Gbit/s]

MULTI-SRC DSLASH ON PASCAL

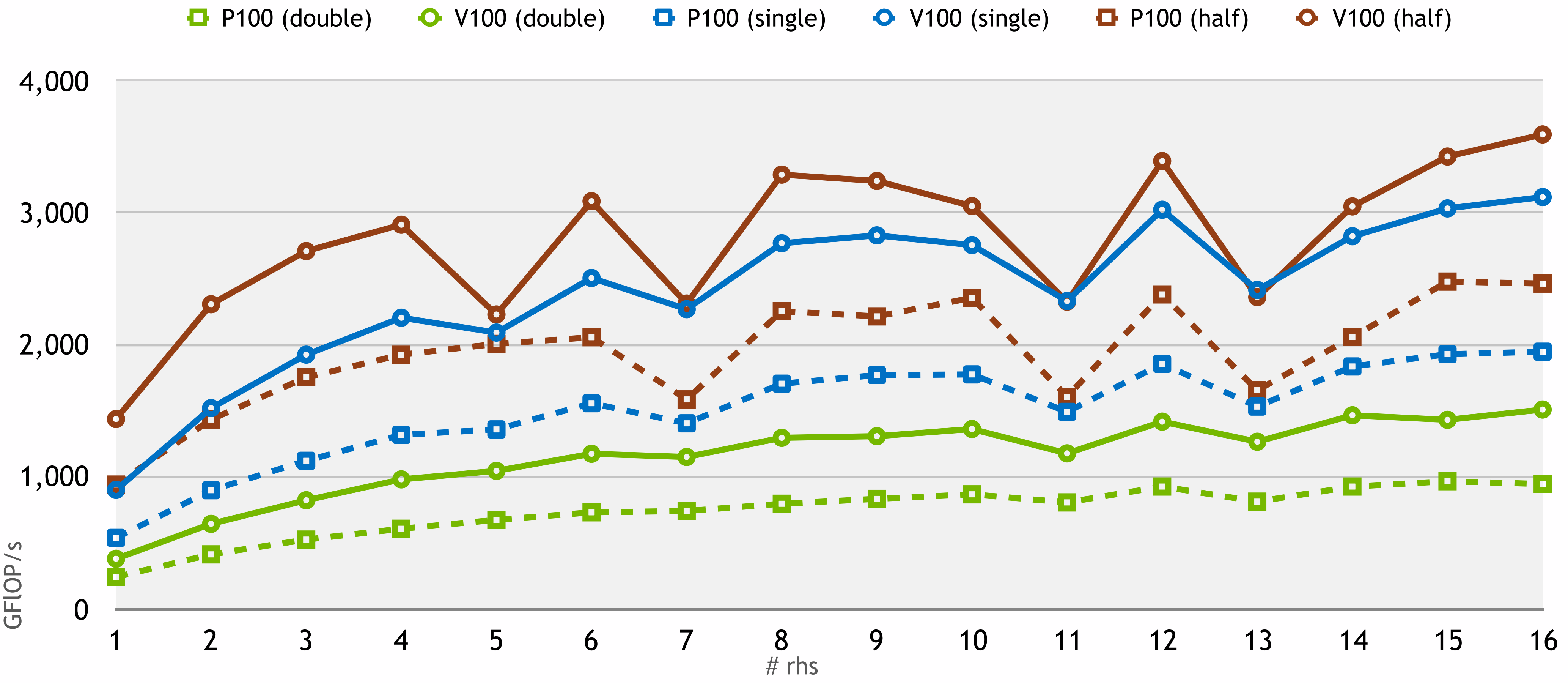
Volume 24^4, HISQ, tuned gauge reconstruct



NVIDIA

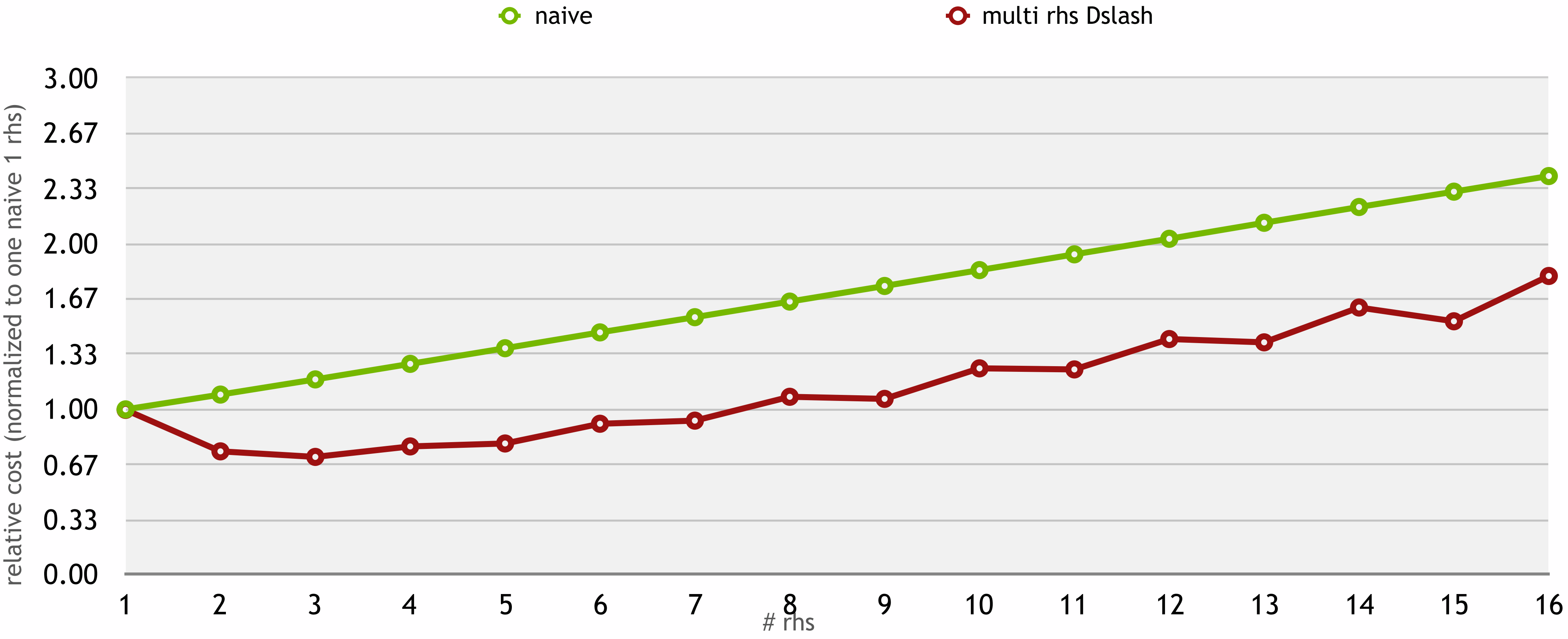
MULTI-SRC DSLASH ON VOLTA

Volume 24^4, HISQ, tuned gauge reconstruct



COST PER ITERATION

for one rhs

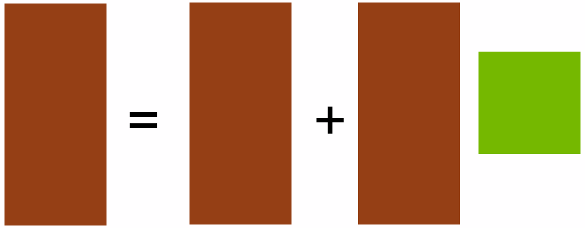


OPTIMIZING LINEAR ALGEBRA

EXPLOIT GPU ARCHITECTURE

to overcome quadratically scaling

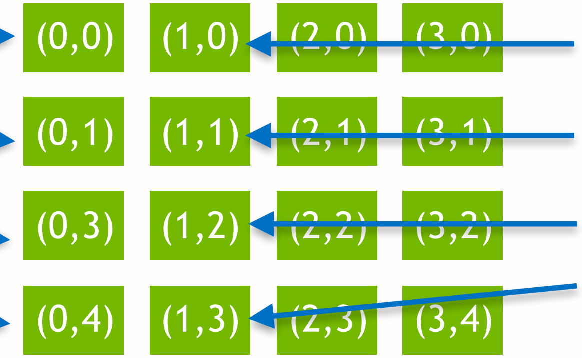
$$y_i = \sum a_{ij} x_j + y_i$$



CUDA supports two dimensional grid blocks:

easy to exploit locality for texture cache / shared memory

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



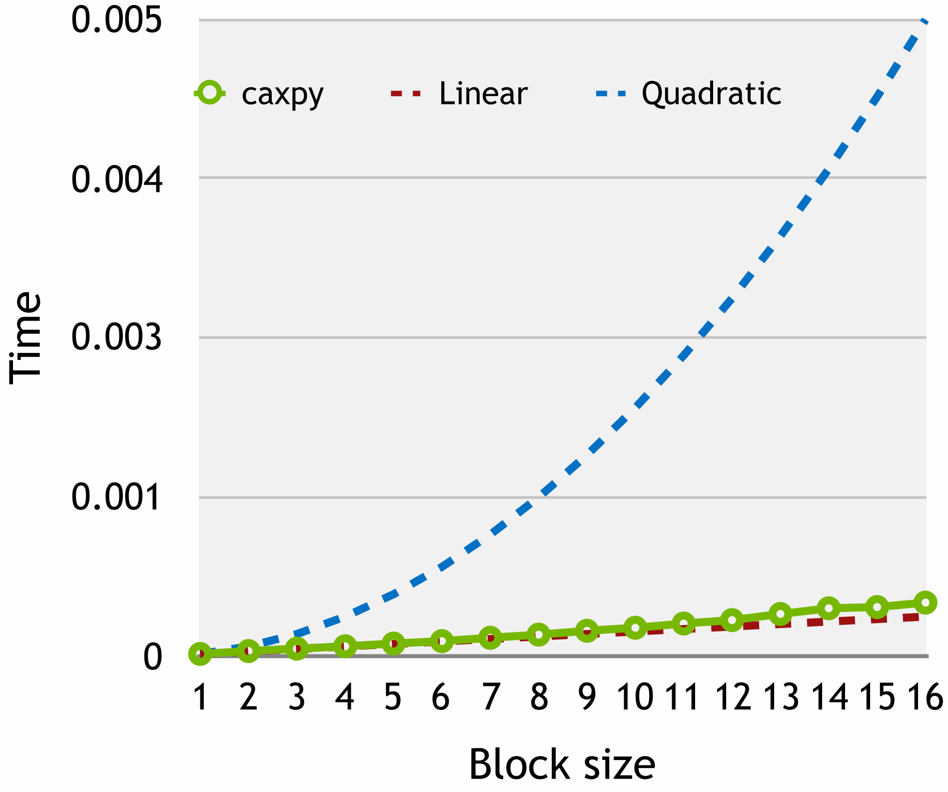
$$y_0(1) = a_{00}x$$
$$y_1(1) = a_{10}x$$
$$y_2(1) = a_{20}x$$
$$y_3(1) = a_{30}x$$

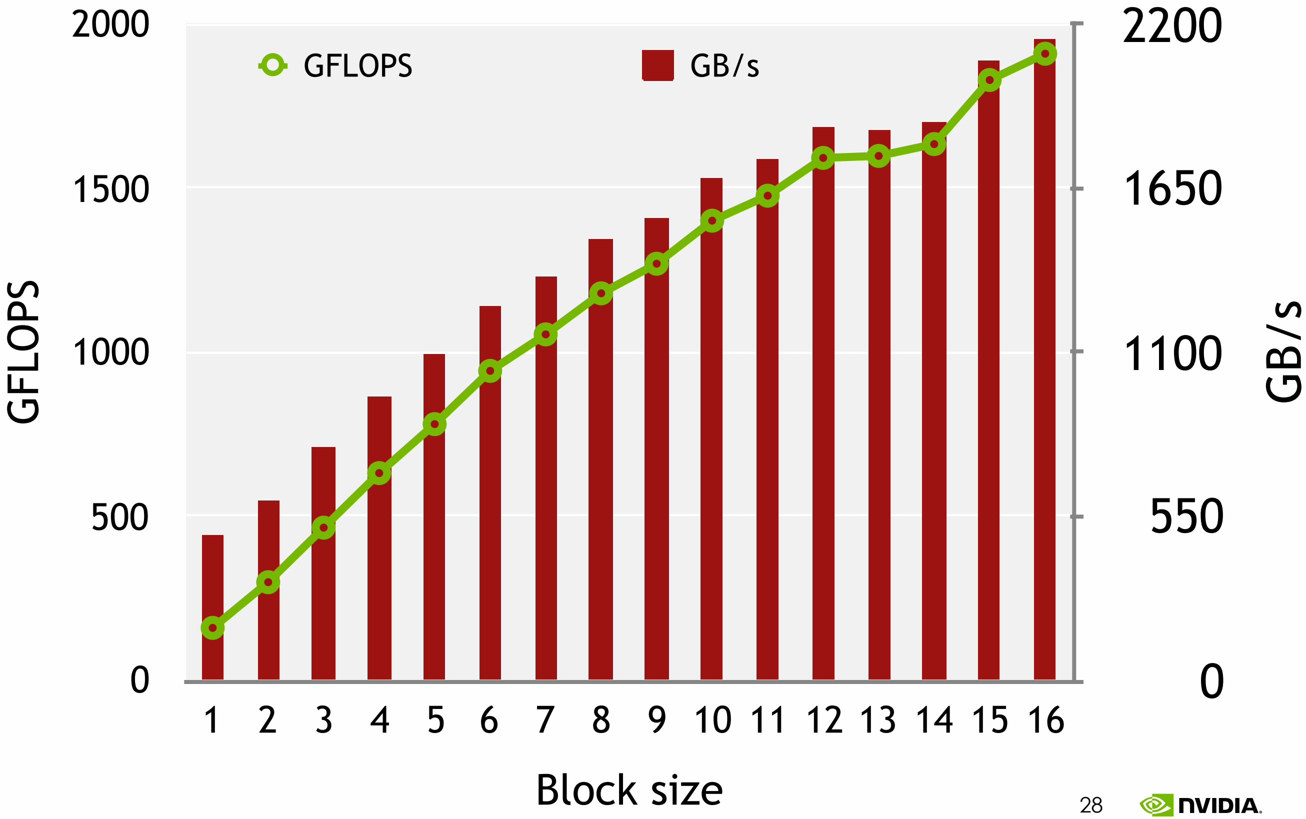
cache reuse

NVIDIA

MULTI-BLAS

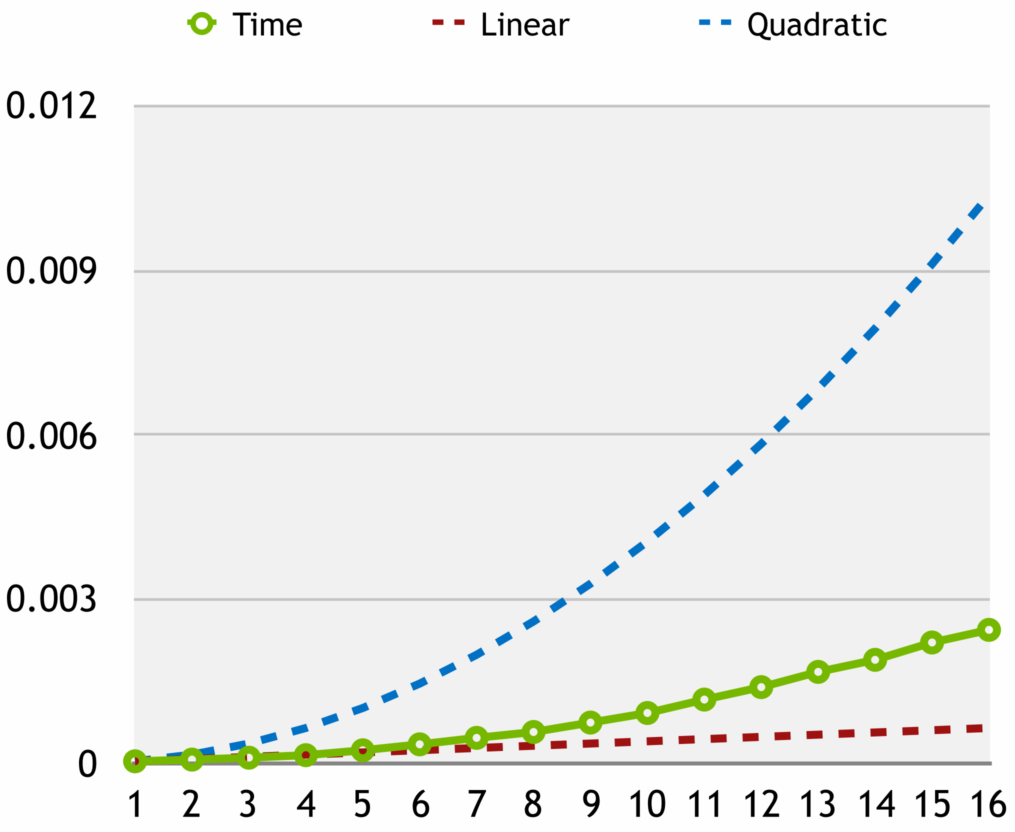
caxpy

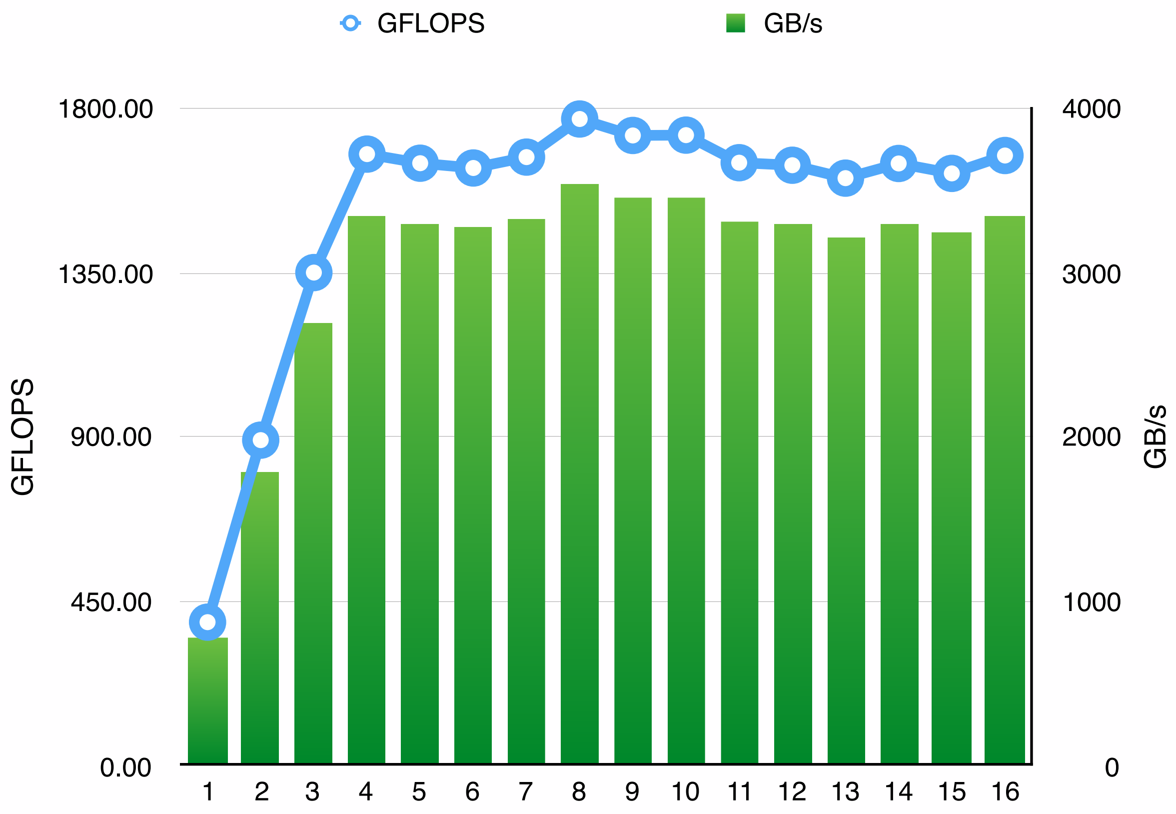




MULTI-REDUCTION

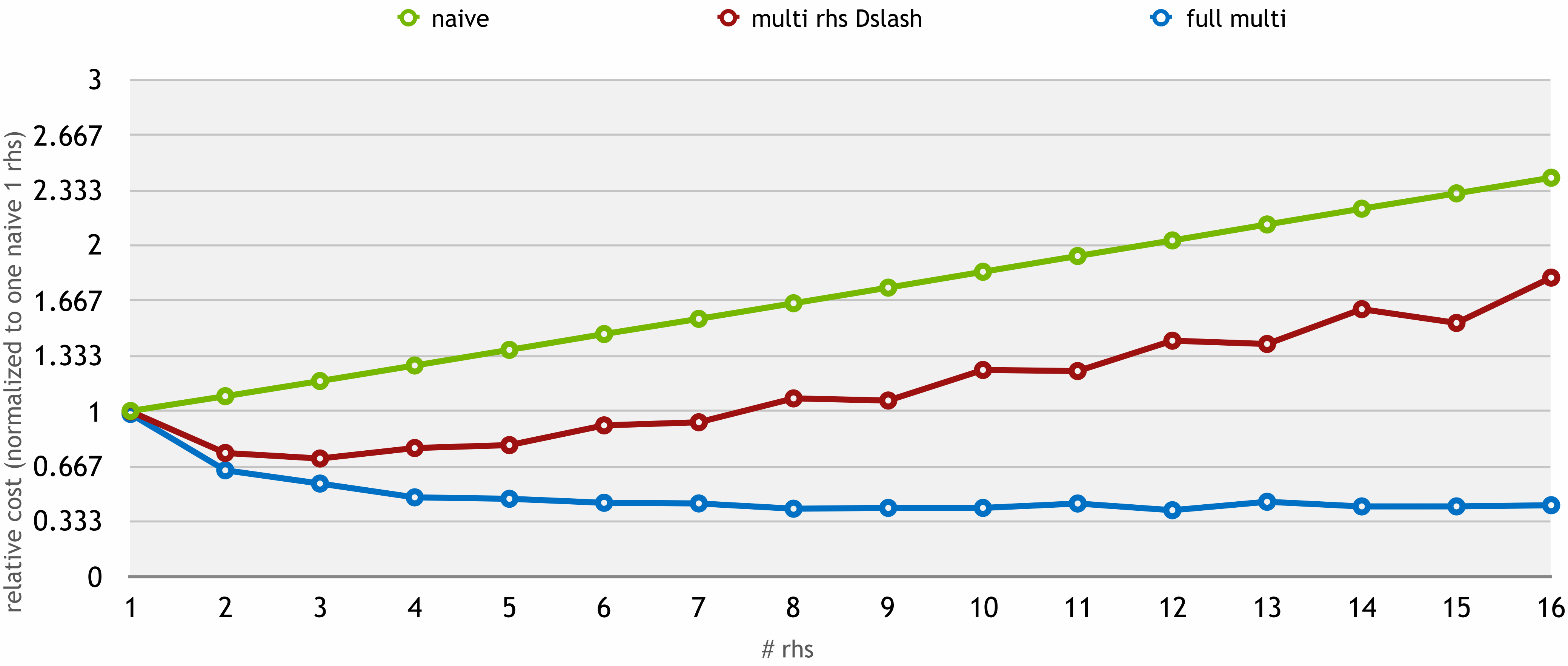
cdot





COST PER ITERATION

for one rhs



BLOCKCGRQ IN THE WILD

TIME TO SOLUTION

solver setup from MILC G-2 project

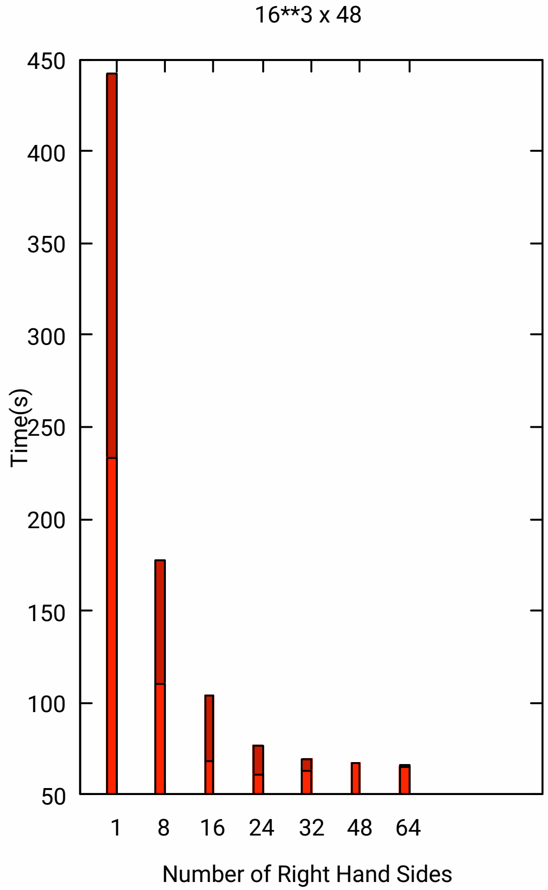
Calculations inherently needs to solve multiples of 8 rhs

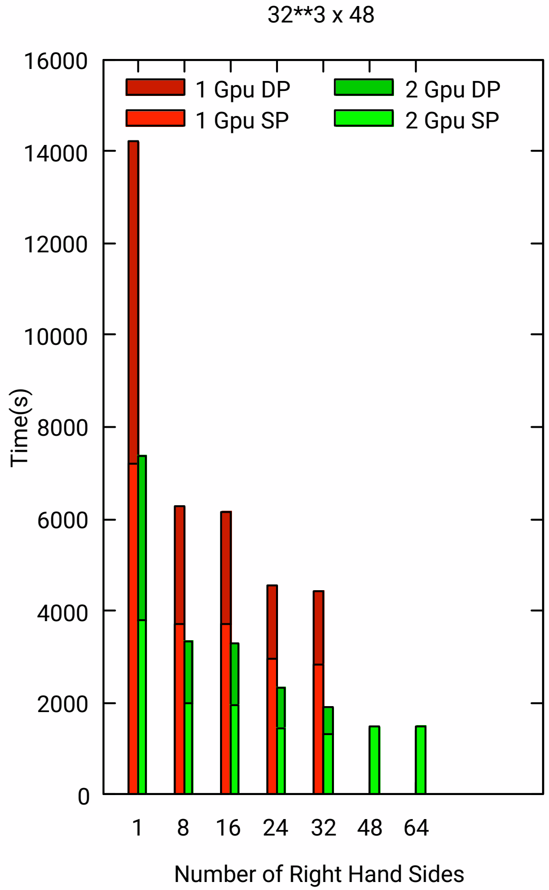
Use a two-step solution process (sloppy solve + refinement step)

→ might not be ideal as it discards shared Krylov space

mixed precision using reliable updates

(wip - still some cases with significant iteration count increase)





TIME TO SOLUTION

solver setup from MILC G-2 project

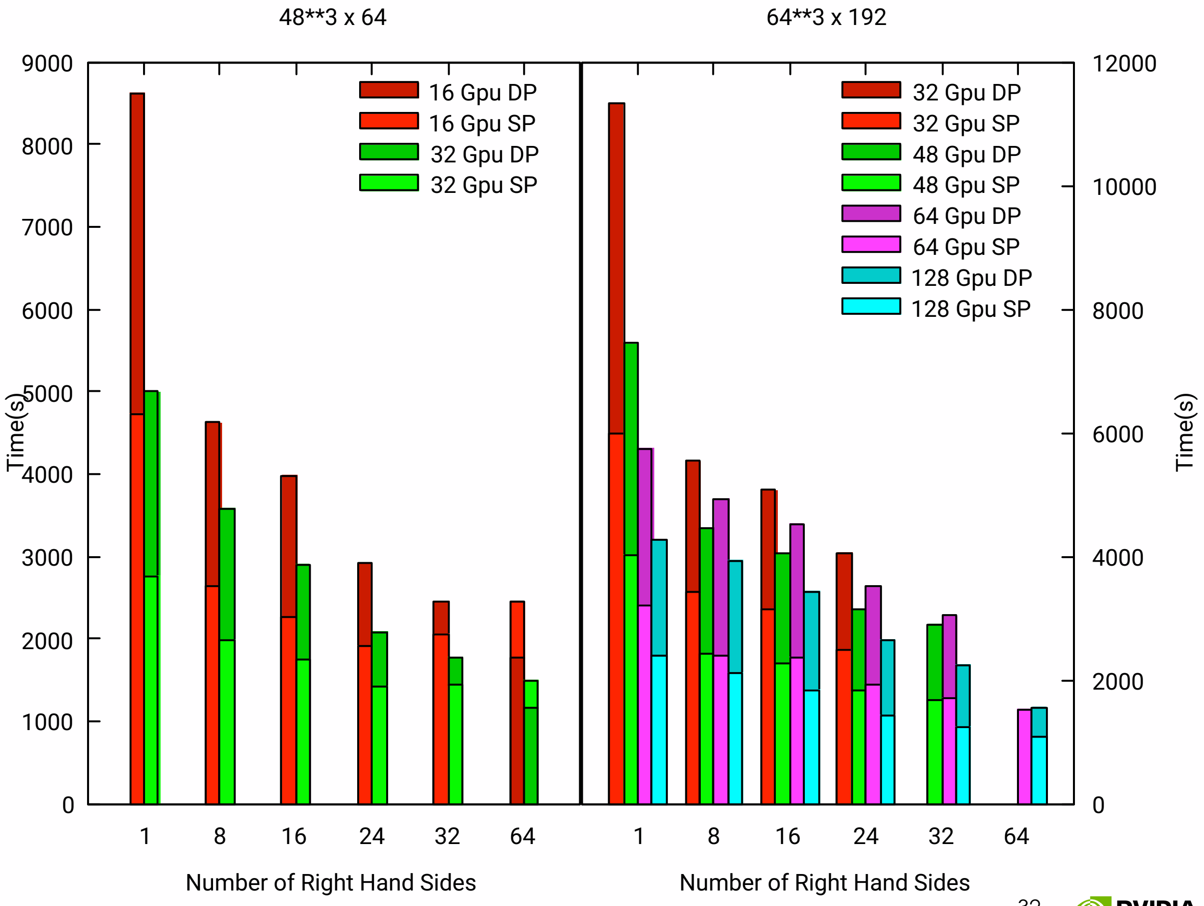
Calculations inherently needs to solve multiples of 8 rhs

Use a two-step solution process (sloppy solve + refinement step)

→ might not be ideal as it discards shared Krylov space

mixed precision using reliable updates

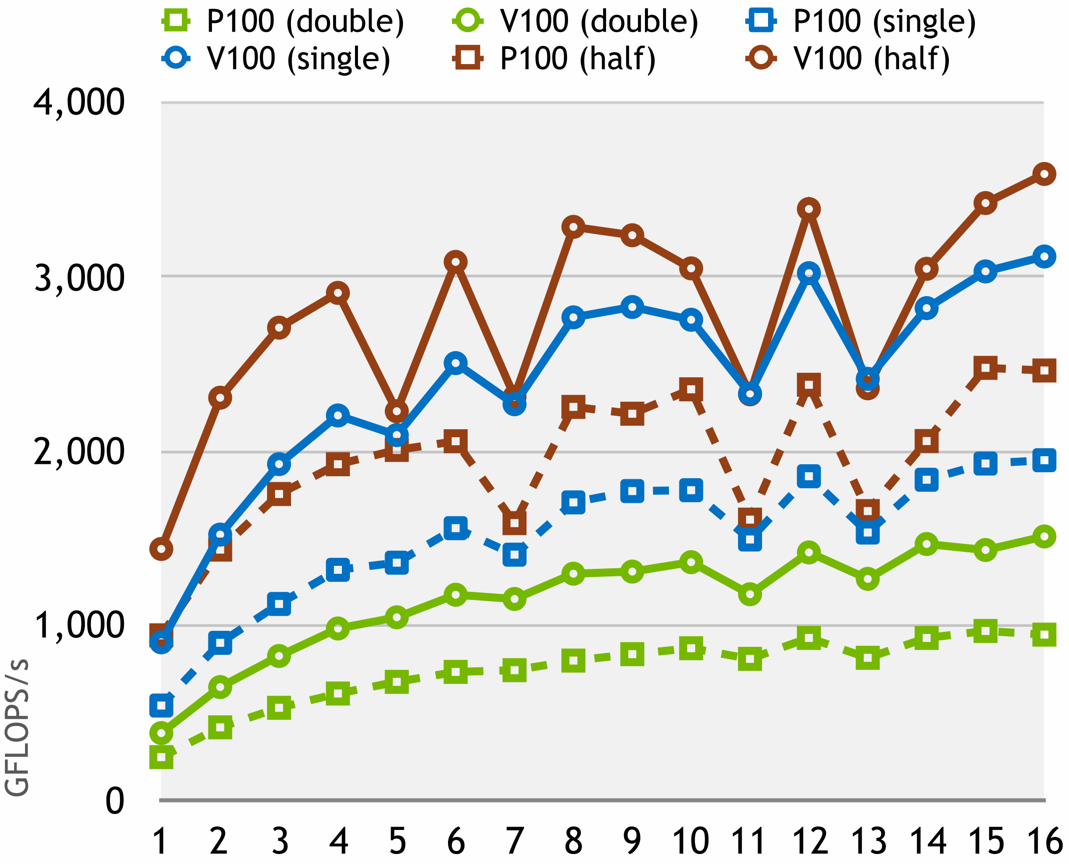
(wip - still some cases with significant iteration count increase)



SUMMARY

PUSHING MORE FLOPS

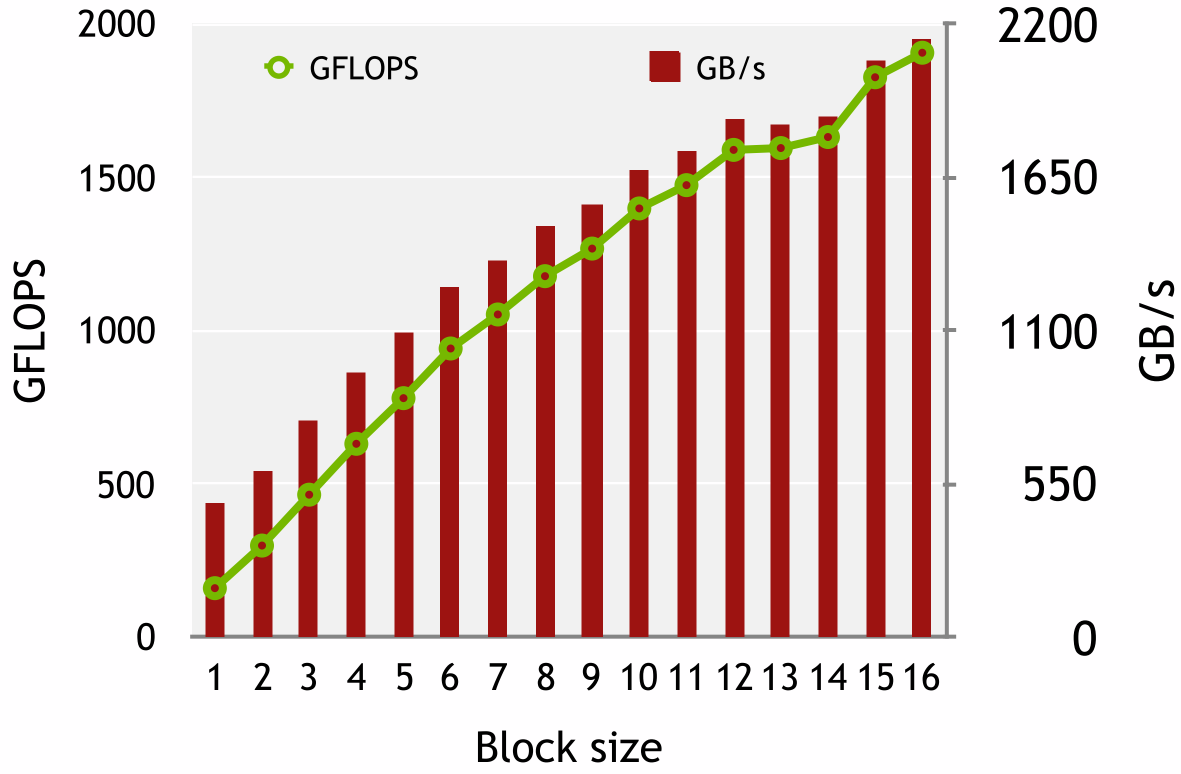
more parallelism and lots of locality to exploit



Reuse gauge field for Dslash

PUSHING MORE FLOPS

more parallelism and lots of locality to exploit



Reuse gauge field for Dslash

Reuse vectors in BLAS and reductions

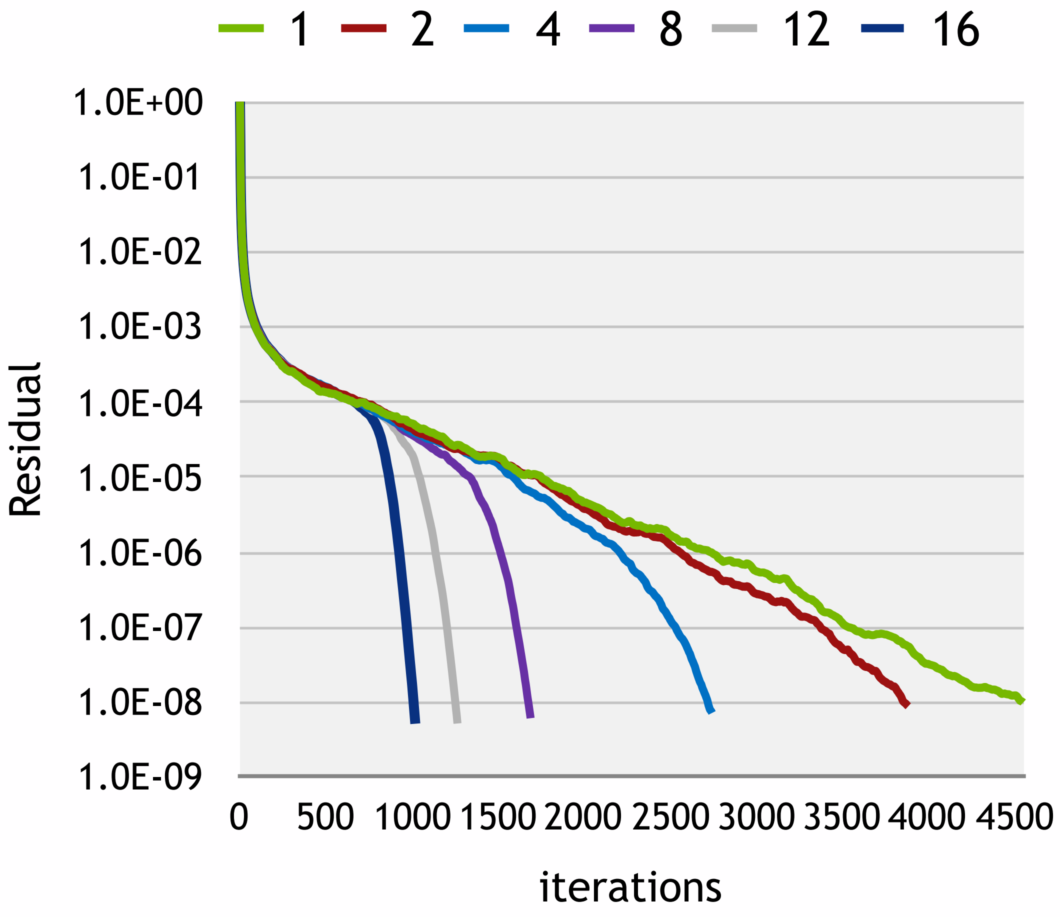
avoid quadratical scaling in linear algebra and orthogonalization

more parallelism to saturate wider architectures

squeezes out ~4x more FLOPS out of a P100

TIME TO SOLUTION

Combined effect of reduced iteration count and cost per iteration



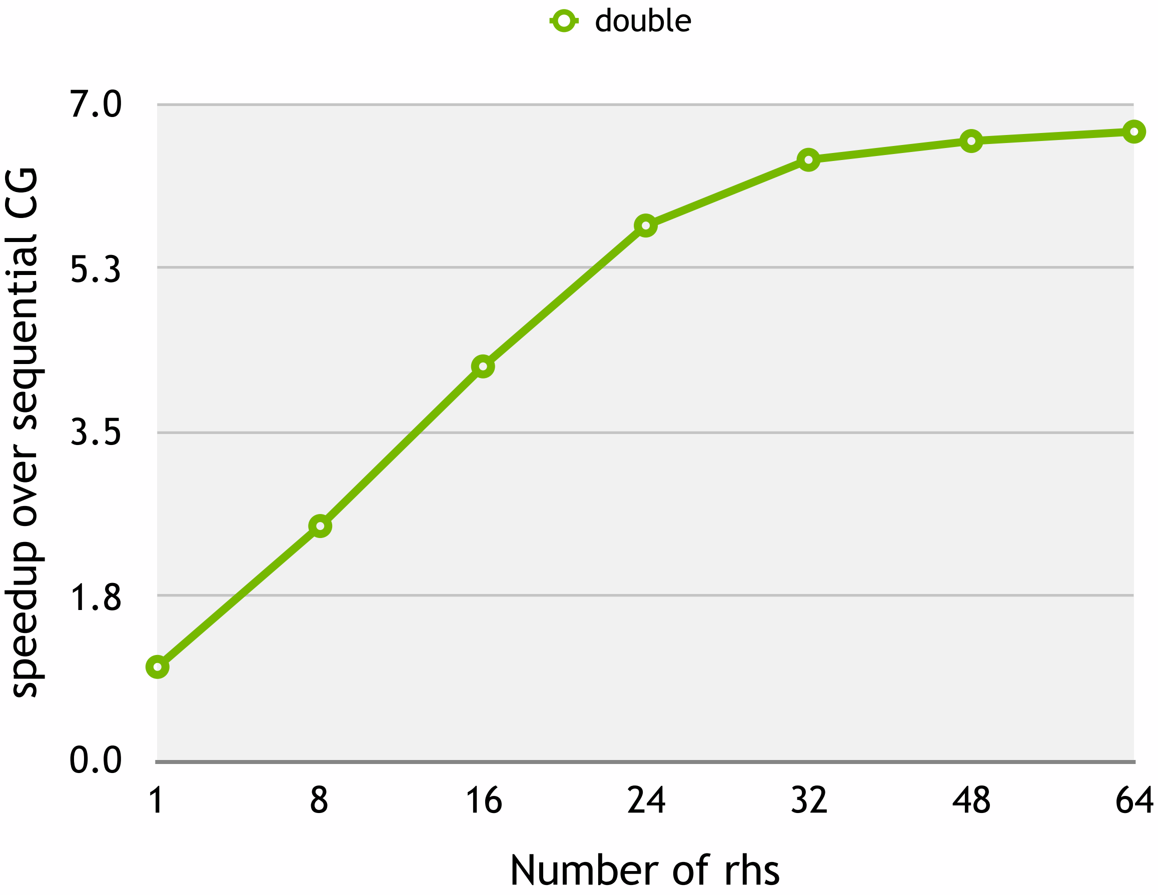
~ factor 3 improvement in cost/iter

additional (almost free) flops in BLAS/ reduction drive reduced iteration count

reduced iteration count depends on rhs, target residual and matrix condition

TIME TO SOLUTION

Combined effect of reduced iteration count and cost per iteration



~ factor 3 improvement in cost/iter

additional (almost free) flops in BLAS/ reduction drive reduced iteration count

reduced iteration count depends on rhs, target residual and matrix condition

Immediate drop in for CG solver (works for staggered)

No setup costs: immediate returns

NVIDIA