A performance study of Quantum ESPRESSO's PWscf code on multi-core and GPU systems

Josh Romero, Everett Phillips, Gregory Ruetsch, Massimiliano Fatica - NVIDIA
Filippo Spiga - University of Cambridge (UK)
Paolo Giannozzi - University of Udine (IT)

Outline

- Quantum ESPRESSO/PWscf

- GPU implementation in CUDA Fortran

- Benchmarking and Results

- Conclusions

Quantum ESPRESSO/PWscf

Quantum ESPRESSO (QE)

- Integrated suite of open-source software for simulations of materials based on density-functional theory

- Complete distribution contains approximately 520,000 lines of Fortran 95 source code

- Popular package widely used within academia and industry

Plane-Wave Self-Consistent Field (PWscf)

- One of the main programs distributed with QE

- Computes the Kohn-Sham (KS) orbitals and energies of material systems

- Uses an iterative method that seeks self-consistent input and output charge densities

- See *Giannozzi et al. J. Phys 2009 Appendix A.2* for details

Plane-Wave Self-Consistent Field (PWscf)

- Each iteration requires:
    - Diagonalization of the Hamiltonian operator H_{KS}
        - done iteratively using a block Davidson method
        - performed for each KS orbital (***k*-point**) across *bands*

    - Computation of output charge density using diagonalization results

- Repeated until self-consistency is obtained within a desired tolerance

Parallelization Options

- PWscf has a number of parallelization options available. Options used in this study:
  - *k*-point parallelization using `-npool`:
    - Distributes *k*-points into N_K pools of processes.
    - Enables parallel execution of the iterative diagonalizations.
  - Linear algebra parallelization using `-ndiag`:
    - Distributes the dense diagonalization, needed by the block Davidson algorithm, among N_D processes.
    - Enables use of distributed eigensolver like ScaLAPACK

GPU Implementation in CUDA Fortran

CUDA Fortran

- Since baseline CPU code is written in Fortran, natural choice for GPU port is CUDA Fortran.

- *CUDA Fortran for Scientists and Engineers* by Ruetsch and Fatica is a good starting reference.

- Requires PGI compilers, free community editions now available at *www.pgroup.com*

CUDA Fortran

- Benefits:
  - More control than OpenACC:
    - Explicit GPU kernels written natively in Fortran are supported
    - Full control of host/device data movement

  - Directive-based programming available via CUF kernels

  - Easier to maintain than mixed CUDA C and Fortran approaches

Profiling with NVPROF + NVVP + NVTX

- When porting programs, profiling (and profiling often) is very important:
  - Identify and focus efforts on performance-critical sections of the program

  - Understand interactions between CPU and GPU:
    - Am I getting expected H2D/D2H BW over PCIe or NVLink?
    - Can I hide this data movement behind GPU computation?

  - Understand library behavior:
    - How is my linked MPI library handling communication between GPUs?
    - Is the CPU being used in any library computations?

Profiling with NVPROF + NVVP + NVTX

- NVPROF:
  - Powerful profiler provided in every CUDA toolkit installation
  - Can be used to gather detailed kernel properties and timing information

- NVIDIA Visual Profiler (NVVP):
  - Graphical interface to visualize and analyze NVPROF generated profiles
  - Does not show CPU activity out of the box

- NVIDIA Tools EXtension (NVTX) markers:
  - Enables annotation with labeled ranges within program
  - Useful for categorizing parts of profile to put activity into context
  - Can be used to visualize normally hidden CPU activity (e.g. MPI communication)
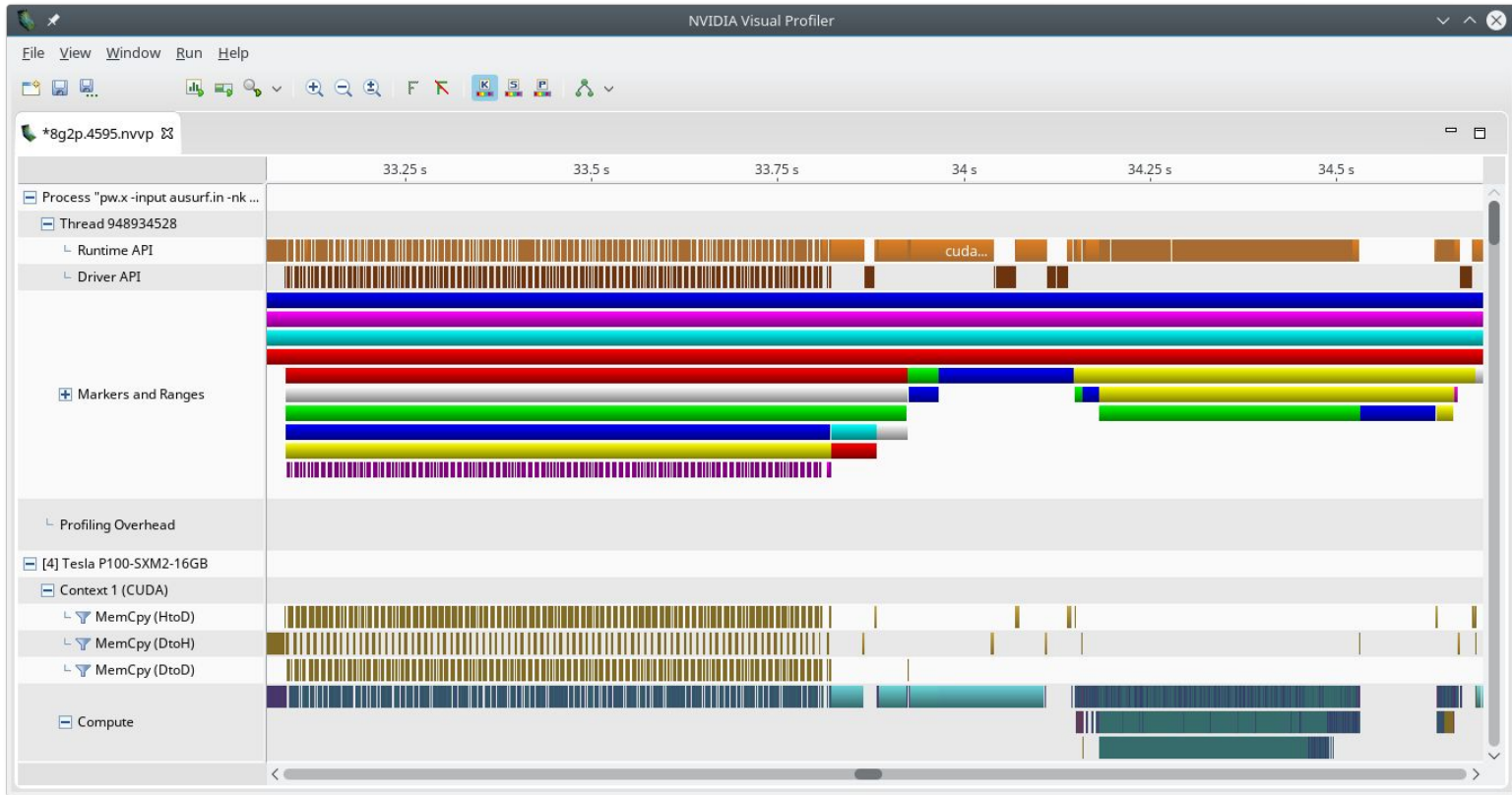
Sample NVVP segment from AUSURF112 on NVIDIA DGX-1 System

GPU Porting of Key Computational Routines

- The iterative diagonalization and computation of charge density are dominated by three basic operation types:
    - Level-3 BLAS routines, predominantly Z/DGEMM
    - 3D Fast Fourier Transforms (FFT), typically distributed
    - dense-matrix diagonalization via LAPACK or ScaLAPACK

- BLAS routines easily ported using available routines in CUBLAS library

- 3D FFT and dense-matrix diagonalization more involved

- Remaining routines ported to GPU as necessary for performance or to remove redundant host/device data movement

3D Fast Fourier Transforms

- Required in iterative diagonalization and charge computation

- Component 1D FFT computations computed using CUFFT

- Generally distributed among the processes in each k-point pool:
  - requires transposition and data communication across processes using `MPI_Alltoall` or similar communication pattern

  - Many 3D FFT computations for each k-point, one for each band index

3D Fast Fourier Transforms

- Existing CPU implementation not amenable to a performant GPU port:
  - Individual FFTs for each band too small to saturate GPU resources

  - No attempt to overlap FFT computation with MPI communication:
    - problematic on GPU systems in cases where communication buffers must be staged through the host

- To address these issues, implemented a batched FFT strategy where multiple band FFTs computed together
  - More available concurrent work for better GPU utilization
  - Provides straightforward mechanism for pipelining data movement and computation
  - Requires more memory, but this was not an issue in tested cases

3D Fast Fourier Transforms

- As a further optimization, implemented all-to-all communication using non-blocking `MPI_Isend/MPI_Irecv`
  - Important on systems which are capable of multiple concurrent peer-to-peer (P2P) transfers between GPUs

- A number of MPI distributions we tried showed suboptimal utilization of available P2P bandwidth on systems with multiple P2P connections
  - For all-to-all, implemented explicit handling of P2P communication using CUDA interprocess communication (IPC), with non-peer transfers handled by linked MPI library

Diagonalization

- The dense-matrix diagonalization, used for the block Davidson method, is another computationally expensive routine.

- Consists of computing eigenvalues and eigenvectors of a modest size system ($N \times N$ with $N \sim O(10^3)$) using a dense eigensolver

- On CPU, this operation is typically distributed over N_D processes and computed using ScaLAPACK, or similar library

Diagonalization

- Current GPU port targets serial path ($N_D = 1$) using a custom developed GPU eigensolver
  - one GPU per k-point pool performs the dense-matrix diagonalization

- Custom solver used in lieu of several existing options for GPU, like MAGMA:
  - Written to reduce dependencies on CPU resources for computation, only reduced tridiagonal solve completed on CPU using LAPACK
  - Beneficial on "fat" nodes, with high GPU to CPU socket ratios, where bottlenecks due to limited CPU resources can arise

- See GTC talk for more info on the solver development

Benchmarking and Results

Testing Details

- Performance results for two benchmark cases were obtained on several GPU systems and a reference CPU system.

- On reference CPU system:
  - Distributed ELPA solver used for diagonalization ($N_D > 1$)
  - MKL for other BLAS/LAPACK routines
  - OpenMP enabled, tried many configurations with best cases reported

- On GPU systems:
  - Custom serial eigensolver used for diagonalization ($N_D = 1$)
  - CUBLAS for BLAS routines on GPU, MKL/ESSL for any BLAS/LAPACK CPU routines
  - GDR features tested on systems with P2P connectivity (CUDA-aware MPI + custom IPC)
  - OpenMP enabled on Intel-based systems
  - OpenMP disabled on IBM system in favor of using multithreaded ESSL

Wilkes-2 (Cambridge)
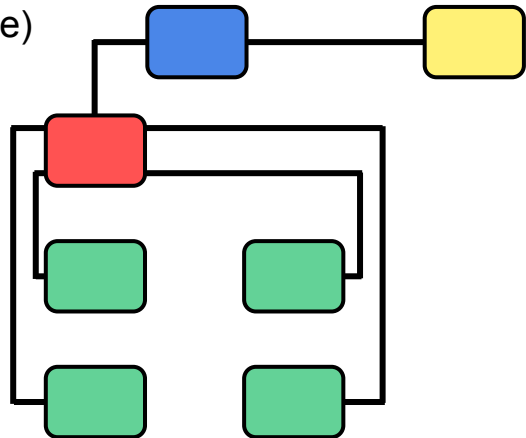
NVIDIA DGX-1
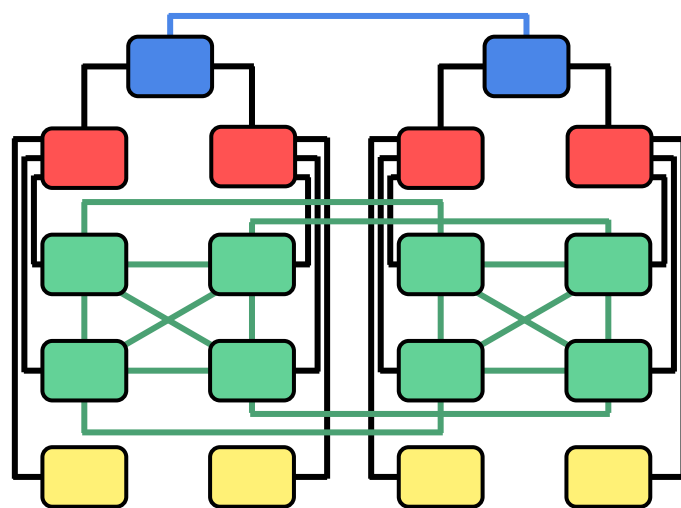
Piz Daint (CSCS)

Summit Dev (ORNL)

CPU	PLX	NIC
GPU	—— PCIe	—— NVLink
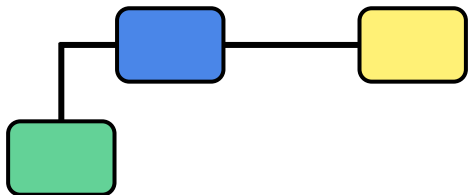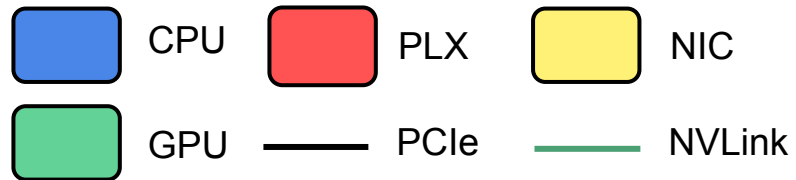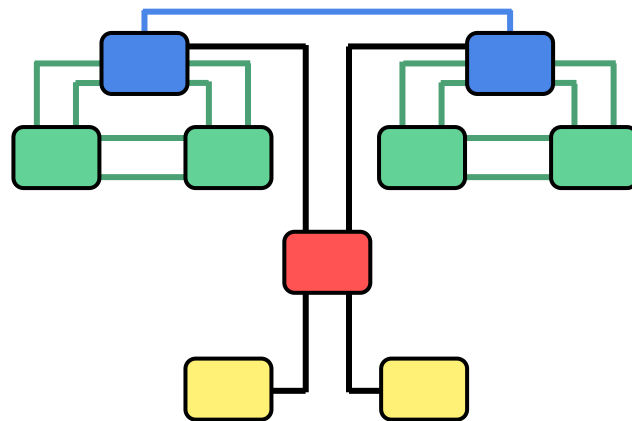
Benchmark Cases

- AUSURF112:
  - Gold surface with 112 atoms and 2 *k*-points
  - Smaller case suitable for workstations and small distributed systems

- Ta2O5:
  - Tantalum pentoxide with 96 atoms and 26 *k*-points.
  - Larger case suitable for scaling from small to large distributed systems

Parameter	Benchmark case	
	AUSURF112	Ta2O5
Number of atomic species	1	2
Number of atoms	112	96
Number of electrons	$1,232$	544
Number of Kohn-Sham states	739	326
Number of k-points	2	26
Number of plane waves	$100,747$	$477,247$
Kinetic energy cutoff	25 Ry	130 Ry
Charge density cutoff	200 Ry	520 Ry
Dimension of dense FFT grid	$\{180, 90, 288\}$	$\{198, 168, 220\}$

AUSURF112: PWscf Time

- Factor of 2-3 speedup using GPU relative to CPU system

- Fixing number of resources per pool gives nearly linear scaling with increased resources

- Increasing number of resources per pool less efficient

System	N_K	Number of CPUs or GPUs used				
		2	4	8	16	32
Broadwell (CPU)	1	1142.24	642.03	369.66	272.00	266.20
	2	1190.13	586.84	335.00	196.54	**144.07**
Piz Daint	1	286.24	219.91	171.80	—	—
	2	—	149.21	**115.87**	—	—
DGX-1	1	347.82	271.37	210.67	—	—
	2	—	184.10	142.15	—	—
DGX-1, GDR	1	270.21	190.12	174.75	—	—
	2	—	142.43	**100.54**	—	—
Summit Dev	1	321.69	234.32	187.69	—	—
	2	—	176.50	128.85	—	—
Summit Dev, GDR	1	308.52	227.74	188.39	—	—
	2	—	169.60	**124.22**	—	—
Wilkes-2	1	395.26	326.71	227.61	—	—
	2	—	226.89	167.80	—	—
Wilkes-2, GDR	1	300.03	226.13	203.59	—	—
	2	—	164.63	**116.50**	—	—
Workstation	1	334.23	—	—	—	—
Workstation, GDR	1	**279.54**	—	—	—	—

AUSURF112: PWscf Time

- Factor of 2-3 speedup using GPU relative to CPU system

- Fixing number of resources per pool gives nearly linear scaling with increased resources

- Increasing number of resources per pool less efficient

System	N_K	2	4	8	16	32
		Number of CPUs or GPUs used			
Broadwell (CPU)	1	1142.24	642.03	369.66	272.00	266.20
	2	1190.13	586.84	335.00	196.54	**144.07**
Piz Daint	1	286.24	219.91	171.80	—	—
	2	—	149.21	**115.87**	—	—
DGX-1	1	347.82	271.37	210.67	—	—
	2	—	184.10	142.15	—	—
DGX-1, GDR	1	270.21	190.12	174.75	—	—
	2	—	142.43	**100.54**	—	—
Summit Dev	1	321.69	234.32	187.69	—	—
	2	—	176.50	128.85	—	—
Summit Dev, GDR	1	308.52	227.74	188.39	—	—
	2	—	169.60	**124.22**	—	—
Wilkes-2	1	395.26	326.71	227.61	—	—
	2	—	226.89	167.80	—	—
Wilkes-2, GDR	1	300.03	226.13	203.59	—	—
	2	—	164.63	**116.50**	—	—
Workstation	1	334.23	—	—	—	—
Workstation, GDR	1	**279.54**	—	—	—	—

AUSURF112: PWscf Time

- Factor of 2-3 speedup using GPU relative to CPU system

- Fixing number of resources per pool gives nearly linear scaling with increased resources

- Increasing number of resources per pool less efficient

System	N_K	Number of CPUs or GPUs used				
		2	4	8	16	32
Broadwell (CPU)	1	1142.24	642.03	369.66	272.00	266.20
	2	1190.13	586.84	335.00	196.54 →	**144.07**
Piz Daint	1	286.24	219.91	171.80	—	—
	2	—	149.21 →	**115.87**	—	—
DGX-1	1	347.82	271.37	210.67	—	—
	2	—	184.10	142.15	—	—
DGX-1, GDR	1	270.21	190.12	174.75	—	—
	2	—	142.43 →	**100.54**	—	—
Summit Dev	1	321.69	234.32	187.69	—	—
	2	—	176.50	128.85	—	—
Summit Dev, GDR	1	308.52	227.74	188.39	—	—
	2	—	169.60 →	**124.22**	—	—
Wilkes-2	1	395.26	326.71	227.61	—	—
	2	—	226.89	167.80	—	—
Wilkes-2, GDR	1	300.03	226.13	203.59	—	—
	2	—	164.63 →	**116.50**	—	—
Workstation	1	334.23	—	—	—	—
Workstation, GDR	1	**279.54**	—	—	—	—

AUSURF112: 8 GPU/CPU

- GPU vs. CPU systems:
  - Faster performance on GPU systems
  - GPU eigensolver outperforming ELPA
- GPU systems:
  - FFT performance improvement with GDR
  - Eigensolver on Summit Dev slower than on Intel systems, more consistent across Intel systems

Legend: FFT, Eigensolver, Other

System	N_K	Wall Time [s]
Broadwell	1	369.66 s
	2	335.00 s
DGX-1	1	210.67 s
	2	142.15 s
DGX-1, GDR	1	174.75 s
	2	100.54 s
Piz Daint	1	171.80 s
	2	115.87 s
Summit Dev	1	187.69 s
	2	128.85 s
Summit Dev, GDR	1	188.39 s
	2	124.22 s
Wilkes-2	1	227.61 s
	2	167.80 s
Wilkes-2, GDR	1	203.59 s
	2	116.50 s

Wall Time [s]

Results from paper

AUSURF112: 8 GPU/CPU

- GPU vs. CPU systems:
  - Faster performance on GPU systems
  - GPU eigensolver outperforming ELPA
- GPU systems:
  - FFT performance improvement with GDR
  - Eigensolver on Summit Dev slower than on Intel systems, more consistent across Intel systems

Legend: FFT, Eigensolver, Other

	Wall Time [s]	
Broadwell: $N_K = 1$	369.66 s	
$N_K = 2$	335.00 s	
DGX-1: $N_K = 1$	201.13 s	-5%
$N_K = 2$	133.03 s	-6%
DGX-1, GDR: $N_K = 1$	155.58 s	-11%
$N_K = 2$	87.93 s	-13%
Piz Daint: $N_K = 1$	150.13 s	-13%
$N_K = 2$	103.98 s	-10%
Summit Dev: $N_K = 1$	173.49 s	-8%
$N_K = 2$	112.20 s	-13%
Summit Dev, GDR: $N_K = 1$	171.14 s	-9%
$N_K = 2$	108.96 s	-12%
Wilkes-2: $N_K = 1$	208.86 s	-8%
$N_K = 2$	151.22 s	-10%
Wilkes-2, GDR: $N_K = 1$	182.42 s	-10%
$N_K = 2$	99.64 s	-14%

Updated results using V1.0

Ta2O5: PWscf Time

- Similar performance trends to AUSURF112 case

- Larger number of available *k*-points allows for scaling out further

System	N_K	8	26	52	104	208
				Number of CPUs or GPUs used		
Broadwell (CPU)	13	—	—	1374.26	809.36	540.64
	26	—	3055.46	1566.95	682.05	**378.73**
Piz Daint	1	5273.93	—	—	—	—
	2	3602.07	—	—	—	—
	13	—	—	617.58	419.39	330.85
	26	—	—	—	315.60	**217.29**
DGX-1	1	7253.06	—	—	—	—
	2	5008.94	—	—	—	—
DGX-1, GDR	1	4139.18	—	—	—	—
	2	2701.00	—	—	—	—
Summit Dev	1	4122.03	—	—	—	—
	2	3236.12	—	—	—	—
	13	—	—	581.15	394.62	289.30
	26	—	—	—	305.66	216.95
Summit Dev, GDR	1	3994.21	—	—	—	—
	2	2959.70	—	—	—	—
	13	—	—	544.83	398.91	292.87
	26	—	—	—	284.90	**207.37**
Wilkes-2	1	7394.40	—	—	—	—
	2	6103.55	—	—	—	—
	13	—	—	1035.20	656.85	—
	26	—	—	—	515.78	—
Wilkes-2, GDR	1	5032.00	—	—	—	—
	2	3264.26	—	—	—	—
	13	—	—	572.43	460.16	—
	26	—	—	—	**273.86**	—

Ta2O5: PWscf Time

- Similar performance trends to AUSURF112 case

- Larger number of available *k*-points allows for scaling out further

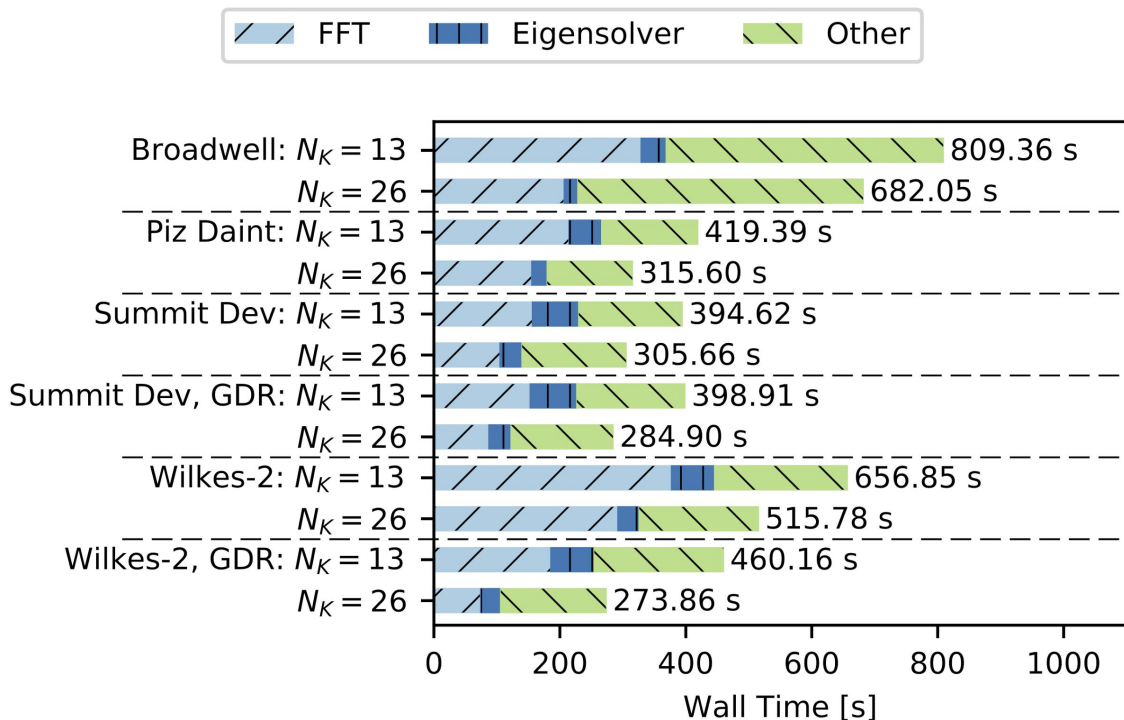| System	N_K	8	26	52	104	208
				Number of CPUs or GPUs used		
Broadwell (CPU)	13	—	—	1374.26	809.36	540.64
	26	—	3055.46	1566.95	682.05	**378.73**
Piz Daint	1	5273.93	—	—	—	—
	2	3602.07	—	—	—	—
	13	—	—	617.58	419.39	330.85
	26	—	—	—	315.60	**217.29**
DGX-1	1	7253.06	—	—	—	—
	2	5008.94	—	—	—	—
DGX-1, GDR	1	4139.18	—	—	—	—
	2	2701.00	—	—	—	—
Summit Dev	1	4122.03	—	—	—	—
	2	3236.12	—	—	—	—
	13	—	—	581.15	394.62	289.30
	26	—	—	—	305.66	216.95
Summit Dev, GDR	1	3994.21	—	—	—	—
	2	2959.70	—	—	—	—
	13	—	—	544.83	398.91	292.87
	26	—	—	—	284.90	**207.37**
Wilkes-2	1	7394.40	—	—	—	—
	2	6103.55	—	—	—	—
	13	—	—	1035.20	656.85	—
	26	—	—	—	515.78	—
Wilkes-2, GDR	1	5032.00	—	—	—	—
	2	3264.26	—	—	—	—
	13	—	—	572.43	460.16	—
	26	—	—	**273.86**	—	—

Ta2O5: PWscf Time

- Similar performance trends to AUSURF112 case

- Larger number of available *k*-points allows for scaling out further

System	N_K	8	26	52	104	208
				Number of CPUs or GPUs used		
Broadwell (CPU)	13	—	—	1374.26	809.36	540.64
	26	—	3055.46	1566.95	682.05 →	**378.73**
Piz Daint	1	5273.93	—	—	—	—
	2	3602.07	—	—	—	—
	13	—	—	617.58	419.39	330.85
	26	—	—	—	315.60 →	**217.29**
DGX-1	1	7253.06	—	—	—	—
	2	5008.94	—	—	—	—
DGX-1, GDR	1	4139.18	—	—	—	—
	2	2701.00	—	—	—	—
Summit Dev	1	4122.03	—	—	—	—
	2	3236.12	—	—	—	—
	13	—	—	581.15	394.62	289.30
	26	—	—	—	305.66	216.95
Summit Dev, GDR	1	3994.21	—	—	—	—
	2	2959.70	—	—	—	—
	13	—	—	544.83	398.91	292.87
	26	—	—	—	284.90 →	**207.37**
Wilkes-2	1	7394.40	—	—	—	—
	2	6103.55	—	—	—	—
	13	—	—	1035.20	656.85	—
	26	—	—	—	515.78	—
Wilkes-2, GDR	1	5032.00	—	—	—	—
	2	3264.26	—	—	—	—
	13	—	—	572.43	460.16	—
	26	—	—	—	**273.86**	—

Ta2O5: 104 GPU/CPU

- GPU vs. CPU systems:
  - ELPA faster in this case, but GPU eigensolver remains competitive

- GPU systems:
  - On fat systems, GDR required for high FFT performance
  - Summit Dev has high FFT performance without GDR due to host-device NVLink

Legend: FFT, Eigensolver, Other

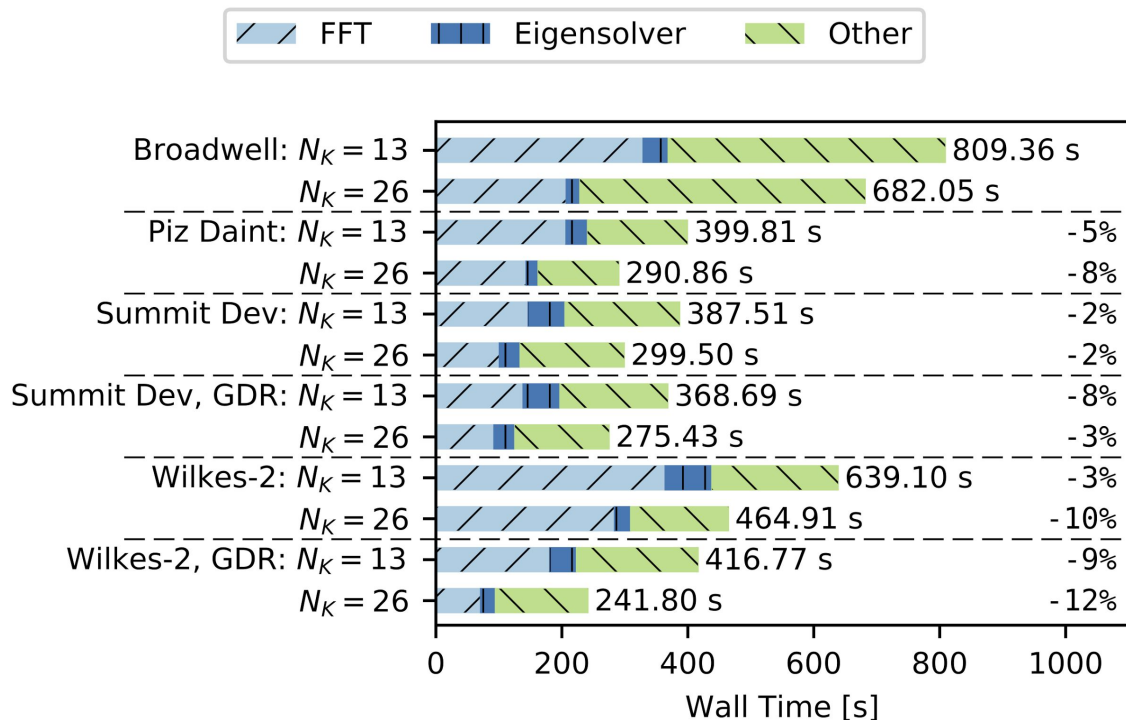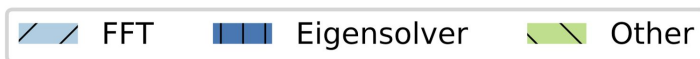| System	N_K	Wall Time [s]
Broadwell	13	809.36 s
	26	682.05 s
Piz Daint	13	419.39 s
	26	315.60 s
Summit Dev	13	394.62 s
	26	305.66 s
Summit Dev, GDR	13	398.91 s
	26	284.90 s
Wilkes-2	13	656.85 s
	26	515.78 s
Wilkes-2, GDR	13	460.16 s
	26	273.86 s

Wall Time [s]

Results from paper

Ta2O5: 104 GPU/CPU

- GPU vs. CPU systems:
  - ELPA faster in this case, but GPU eigensolver remains competitive

- GPU systems:
  - On fat systems, GDR required for high FFT performance
  - Summit Dev has high FFT performance without GDR due to host-device NVLink

Legend: FFT | Eigensolver | Other

System	Wall Time	%
Broadwell: $N_K = 13$	809.36 s	
$N_K = 26$	682.05 s	
Piz Daint: $N_K = 13$	399.81 s	-5%
$N_K = 26$	290.86 s	-8%
Summit Dev: $N_K = 13$	387.51 s	-2%
$N_K = 26$	299.50 s	-2%
Summit Dev, GDR: $N_K = 13$	368.69 s	-8%
$N_K = 26$	275.43 s	-3%
Wilkes-2: $N_K = 13$	639.10 s	-3%
$N_K = 26$	464.91 s	-10%
Wilkes-2, GDR: $N_K = 13$	416.77 s	-9%
$N_K = 26$	241.80 s	-12%

Wall Time [s]

Updated results using V1.0
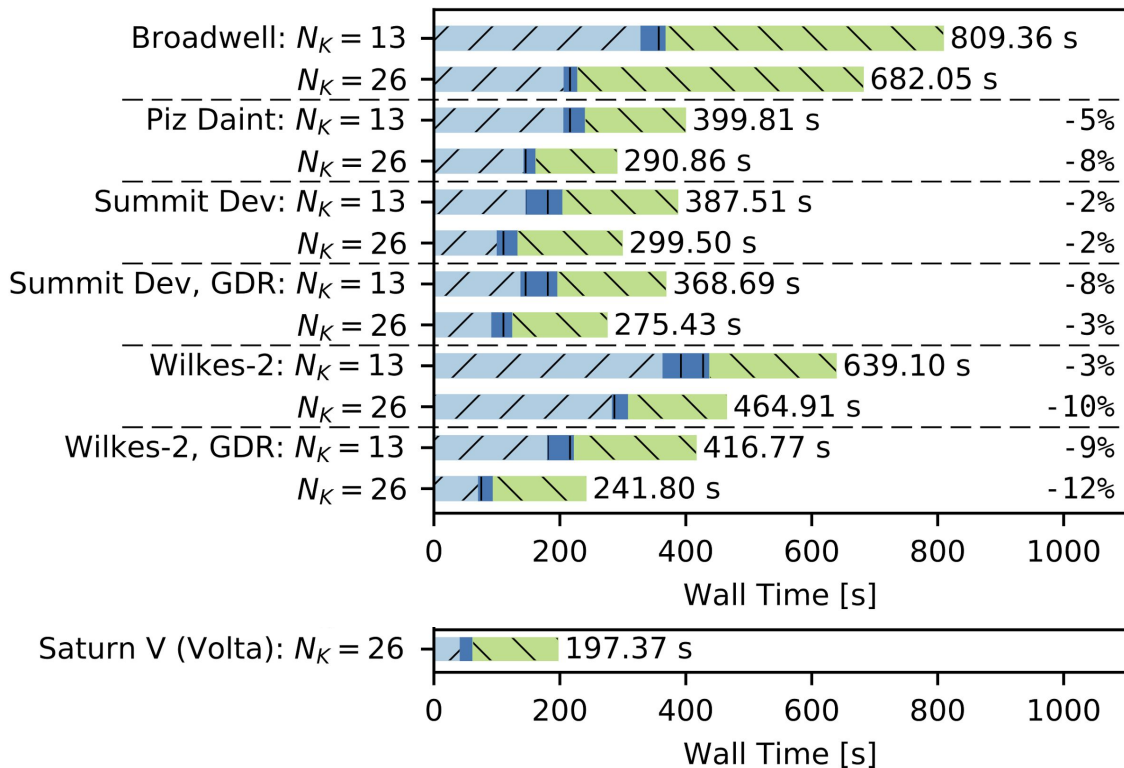
Ta2O5: 104 GPU/CPU

- GPU vs. CPU systems:
  - ELPA faster in this case, but GPU eigensolver remains competitive

- GPU systems:
  - On fat systems, GDR required for high FFT performance
  - Summit Dev has high FFT performance without GDR due to host-device NVLink

Updated results using V1.0

Si63Ge (vc-relax)

	QE-GPU CSCS		QE CSCS	QE Cineca	
	1 P100	10 P100	20 BW (360c)	1 KNL (60c)	10 KNL (640c)
npool	1	10	10	5	10
init_run	15.92s	7.50s	4.45s	21.61s	10.33s
electrons	668.06s	108.78s	235.58s	1542.72s	292.86s
update_pot	1.37s	1.04s	10.42s	31.95s	7.94s
forces	12.06s	3.03s	13.20s	60.91s	11.93s
stress	74.28s	15.82s	75.69s	260.82s	38.55s
cdiaghg	71.38s	6.89s	15.51s	147.97s	76.15s
PWSCF	**774.49s**	**138.70s**	**342.26s**	**1934.28s**	**400.29s**

Fermi energy	6.5908 ev	6.5908 ev	6.5908 ev	6.5908 ev	6.5908 ev
Total energy	-813.93522903 Ry	-813.93522903 Ry	-813.93522904 Ry	-813.93522904 Ry	-813.93522903 Ry
Total force	0.002992	0.002992	0.002992	0.002992	0.002992
Total stress	0.00000062	0.00000062	0.00000062	0.00000062	0.00000062
Pressure	0.09	0.09	0.09	0.09	0.09

BW/KNL results from *https://github.com/electronic-structure/benchmarks*

Conclusions

Conclusions

- New GPU implementation can reduce time to solution by a factor of 2 - 3 relative to the reference CPU system

- Custom serial GPU eigensolver provides competitive performance relative to ScaLAPACK and ELPA with limited sensitivity to host resources. Available on Github at: *https://github.com/NVIDIA/Eigensolver_gpu*

- Full utilization of P2P resources essential for high performance, especially on systems with large GPU to CPU socket ratios

- CUDA-accelerated version of QE is open-source and available on Github at: *https://github.com/fspiga/qe-gpu*