

NEKBONE PERFORMANCE PORTABILITY



SUDHEER CHUNDURI

RONALD RAHAMAN

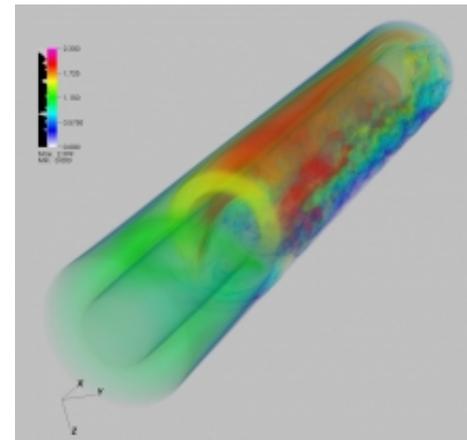
SCOTT PARKER

ARGONNE NATIONAL LABORATORY

August 23, 2017

WHAT IS NEK5000?

- Spectral element CDF Solver for:
 - Unsteady incompressible Navier-Stokes
 - Low mach number flows
 - Heat transfer and species transport
 - Incompressible magnetohydrodynamics
- Code:
 - Open source
 - <https://github.com/Nek5000/Nek5000>
 - Written in Fortran 77 and C
 - MPI for parallelization
- Features:
 - Highly scalable, scales to over a million processes
 - High order spatial discretization using spectral elements
 - High order semi-implicit time stepping



WHAT IS NEKBONE?

- Nekbone is a mini-app derived from Nek5000
 - Originally developed by Katherine Heisey, Paul Fischer
- Nek5000 is complicated to setup, run, and modify.
 - Nekbone is simpler and enables more rapid investigations
- Nekbone represents significant kernels in Nek5000
 - Large percentage of the work in Nek5000
 - Solve Poisson problem similar to pressure solve in Nek5000
 - Represents key kernels and operation mix from Nek5000
 - matrix-matrix multiplication
 - inner products
 - nearest neighbor communication
 - MPI_Allreduce
- Implemented using:
 - Fortran 77, C, MPI, OpenMP, OpenACC
- Used for:
 - DOE machine acquisitions: CORAL systems
 - Exascale co-design activities: DOE FastForward, DesignForward
 - Programming model exploration
- Available at:
 - https://cesar.mcs.anl.gov/content/software/thermal_hydraulics (Original)
 - <https://asc.llnl.gov/CORAL-benchmarks/> (CORAL)
 - <https://repocafe.cels.anl.gov/repos/nekbone/> (OpenMP)
 - <https://github.com/Nek5000/Nekbone/tree/cuda-openacc> (OpenACC)

NEKBONE IN A NUTSHELL

Solver (CG)

- solveM
- glsc3
 - AllReduce
- add2s1
- ax
 - ax_e
 - local_grad3
 - (3x) mxm
 - wr-ws-wt
 - local_grad3_t
 - (3x) mxm
 - (2x) add2
- gs_op
- add2s2
- glsc3
 - AllReduce
- add2s2
- add2s2
- glsc3
 - AllReduce

- gs_gather
- pw_exec
 - pw_exec_recvs
 - MPI_Irecv
- gs_scatter
- pw_exec_sends
 - MPI_Isend
- comm_wait
- gs_gather
- gs_scatter

Memory Bandwidth Intensive
Floating Point Intensive
Network Intensive
Memory Operation Intensive
Wrapper routines

NEKBONE IN A NUTSHELL (SINGLE NODE)

Solver (CG)

- solveM
- glsc3
 - AllReduce
- add2s1
- ax
 - ax_e
 - local_grad3
 - (3x) mxm
 - wr-ws-wt
 - local_grad3_t
 - (3x) mxm
 - (2x) add2
- gs_op
- add2s2
- glsc3
 - AllReduce
- add2s2
- add2s2
- glsc3
 - AllReduce

- gs_gather
- pw_exec
 - pw_exec_recv
 - MPI_Irecv
 - gs_scatter
 - pw_exec_send
 - MPI_Isend
 - comm_wait
 - gs_gather
 - gs_scatter

Memory Bandwidth Intensive
Floating Point Intensive
Network Intensive
Memory Operation Intensive
Wrapper routines

NEKBONE COMPUTE PERFORMANCE MODEL

Function	Routine (major)	Data	Code	Loads	Stores	FPOps
solveM	copy	z,r	$z(i)=r(i)$	N	N	0
glsc3		r,c,z	$t=t+r(i)*c(i)*z(i)$	3N	0	3N
add2s1		p,z	$p(i)=C*p(i)*z(i)$	2N	N	2N
local_grad3	mxm (3)	p,ur,dxm1		N	0	$6N_x N$
wrswt		g,ur,us,ut	$ur(i)=g(1,i)*ur(i)+g(2,i)*us(i)+g(3,i)*ut(i)$ $us(i)=g(2,i)*ur(i)+g(4,i)*us(i)+g(5,i)*ut(i)$ $ut(i)=g(3,i)*ur(i)+g(5,i)*us(i)+g(6,i)*ut(i)$	6N	0	15N
local_grad3_t	mxm (3)	w,ur,dxtm1		0	N	$(6N_x+2)N$
gs_op	Pt-2-pt		Gather Scatter communication			
add2s2		x,p	$x(i)=x(i)+C*p(i)$	2N	N	2N

HARDWARE COMPARISON

INTEL XEON PHI KNIGHTS LANDING (7230) AND NVIDIA TESLA P100-16



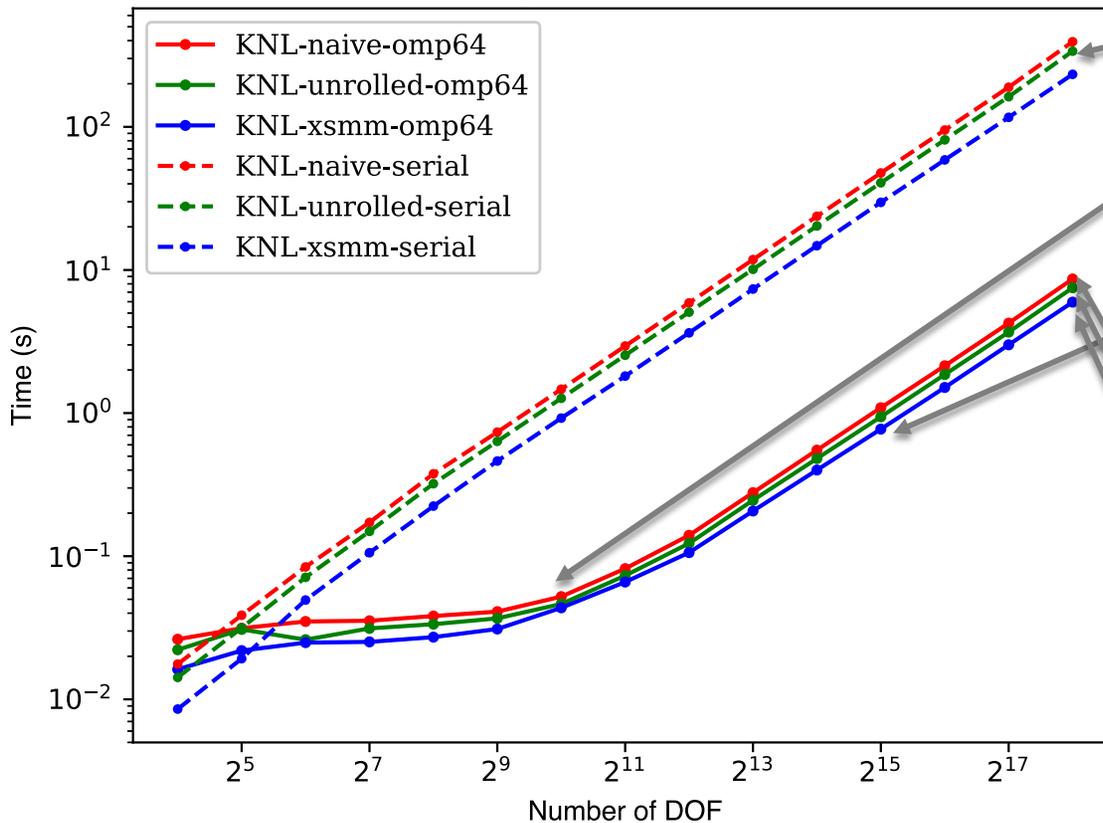
	KNL	P100 GPU
Cores/SMX	64	56
SIMD DP Width	8	32
Clock Speed [MHz]	1100	1329
Peak f32 [TFlop/s]	4.5	9.3
Peak f64 [TFlop/s]	2.25	4.7
Peak DGEMM [Tflops/s]	1.9	4.5
Memory (IPM) [GB]	16	16
IPM Bandwidth [GB/s]	~600	732
STREAM Bandwidth [GB/s]	488	574
L1 cache [kB]/ (Core/SMX)	32(D) +32(I)	64 (L1/Tex.)
L2 cache [MB]	32	4
TDP [W]	215	250

← P100 has 2.4x DGEMM rate of KNL

← P100 has 1.2x STREAM rate of KNL

NEKBONE PERFORMANCE ON KNL

Solver time for different problem sizes
Polynomial Order 16



Using single core, time increases linearly with problem size

Using 64 cores, time is roughly flat until problem has enough parallelism to fill all cores

Using 64 cores, time increases linearly with problem size for large enough problems

Small MXM Implementation	% of peak DGEMM efficiency
Naive	18
Unrolled	26
XSMM	51

NEKBONE PERFORMANCE ANALYSIS ON KNL

Routine	Time (s)	% Solve time	GB/s	GFlop/s	% of peak
solveM	2.90E-2	3.7	462.5	0	95
glsc3	1.56E-1	20.2	351.2	43.9	72
add2s1	4.23E-2	5.4	475.6	39.6	97
local_grad3	8.31E-2	10.7		969.6	51
wrswt	1.28E-1	16.5	314.1	98.1	64
local_grad3_t	1.09E-1	14.1		754.4	40
gs_op	8.89E-2	11.5			
add2s2	1.23E-2	16.02	480.3	40.0	98

Memory bandwidth intensive kernels obtain 64-95% of STREAM

Compute intensive kernels obtains 40-51% of peak DGEMM rate

Memory operation intensive kernels are ~10% of runtime, no efficiency model yet

Network performance is not investigated

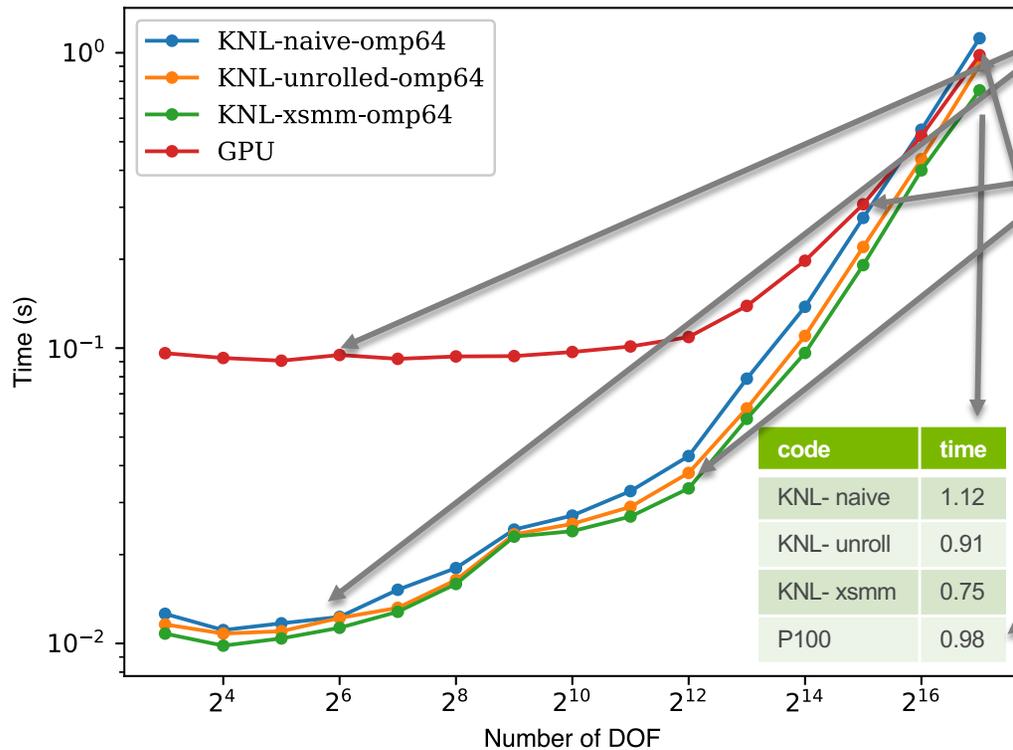
OPENACC CODE CHANGES

- Simple changes:
 - Added OpenACC Parallel Loop pragma before loop
 - Functions:
 - glsc3, add2s2, add2s1 (single loop bandwidth bound kernels)
- Moderately complex changes:
 - Manually inlined functions and merged loop nests
 - Added OpenACC Parallel Loop with Collapse to resulting outer loops
 - Functions:
 - local_grad, wrswt, local_grad_t
- Complex changes:
 - Serial data structure describing gather/scatter operations augmented to enable parallelism
 - Required for any form of shared memory parallelism including OpenMP implementation for CPU
 - Loops restructured to utilize new data structures
 - Added OpenACC loop on outer loop driving gather and scatter
 - Functions:
 - gs_gathe, gs_scatter

NEKBONE PERFORMANCE ON P100 & KNL

Solver time for different problem sizes

Polynomial Order 8



GPU performance lags CPU for small problem sizes

Sufficient DOF required to efficiently utilize cores occurs at different problem sizes for CPU and GPU

For largest DOF GPU performance exceeds naive CPU implementation

Further room to optimized GPU performance with optimized small matrix multiply, may exceed CPU performance

NEKBONE PERFORMANCE ANALYSIS ON P100

Routine	Time (s)	% Solve time	Model GB/s	Profiled GB/s	Model GFlops	Profiled GFlops	% Peak
glsc3	9.23E-2	7.9	656.54	545.31	109.4	109.42	95%
add2s1	3.61E-2	3.1	558.20	512.81	46.5	46.52	89%
local_grad3 + wrswt	3.35E-1	28.7	160.41	187.01	278.2	278.22	6%
local_grad3_t	2.96E-1	25.4	136.00	85.04	272.00	272.00	6%
gs_op	5.56E-2	4.8					
add2s2	1.09E-1	9.3	554.37	512.36	46.5	46.20	89%

- Memory bandwidth intensive kernels obtain 89-95% of STREAM, better than KNL
- Compute intensive kernels obtains 6% of peak DGEMM rate, worse than KNL

- Model GB/s are within 8-10 % of profiles (except glsc3, which is 20% greater)
- Model GFlops/s are within 0.03% of profiles

CONCLUSIONS AND NEXT STEPS

Conclusions:

- Nekbone performance on KNL and P100:
 - Kernels efficiencies are comparable on both for un-optimized kernels
 - Higher small matrix multiply efficiencies require architecture specific routines
 - Lowest time to solution was on KNL with optimized matrix multiply, performance is 1.3x P100
 - GPU optimized small matrix multiply could yield ~2.5x improvement in solve time on P100
 - P100 could show 1.9x better performance than KNL
- Nekbone portability:
 - “Single source” portability can be achieved with low performance on both architectures
 - OpenACC can be used performance portably for the memory bandwidth intensive routines
 - Architecture specific small matrix multiply are required for optimal performance on either architecture

Next Steps:

- Optimize small matrix multiply operations for GPU
- Merge OpenMP and OpenACC branches
- Convert OpenACC directives to OpenMP 4.5
- Look at other portability frameworks RAJA, Kokkos

QUESTIONS?

www.anl.gov

OPENACC CODE CHANGES

- Glsc3 – added OpenACC Parallel Loop pragma before loop
- Add2s2 - added OpenACC Parallel Loop pragma before loop
- Add2s1 - added OpenACC Parallel Loop pragma before loop
- Local_grad, wrswt, local_grad_t
 - Inlined mxm calls
 - Merged 3 mxm loops
 - Inlined local_grad and local_grad_t
 - Merged local_grad and wrswt
 - Outer loop uses ACC Parallel Loop with Collapse
- gs_gather & gs_scatter
 - Serial data structure modified to express parallelism for gather/scatters
 - OpenACC loop on over outer loop driving gather and scatter

OUTLINE

1. Nek5000 description (Scott)
2. Nekbone description (Scott)
3. Nekbone code overview (Scott)
4. Nekbone performance model (Scott)
5. KNL and P100 hardware comparison (?)
6. KNL (table or plot) for opt vs. unoptimized (Sudheer)
7. KNL comparison vs perf model and rate limit identification (Sudheer)
8. code mods for OpenACC (Scott)
9. P100 perf comparison with KNL (Ron/Sudheer)
10. P100 comparison vs perf model and rate limit identification (Ron/Sudheer)
11. Strong scaling limit discussion[Sudheer]
12. Summary of how portable[Scott]

NEKBONE IN A NUTSHELL (OPENMP)

```
cg() [loop 1 -> numCGIterations]
• solveM() [z(i) = r(i)]
• glsc3() [inner product]
  • AllReduce()
• add2s1() [a(i) = c*a(i)+b(i)]
• ax()
  • ax_e()
    • local_grad3() [gradient]
      • (3x) mxm()
    • wr-ws-wt [wx(i) = f(g,ur,us,ut)]
    • local_grad3_t() [gradient]
      • (3x) mxm()
      • (2x) add2()
    • add2s2() [a(i) = a(i) + c*b(i)]
• glsc3() [inner product]
  • AllReduce
• add2s2() [a(i) = a(i) + c*b(i)]
• add2s2() [a(i) = a(i) + c*b(i)]
• glsc3() [inner product]
  • AllReduce
```

- **Bandwidth Bound**
- **Compute Bound**

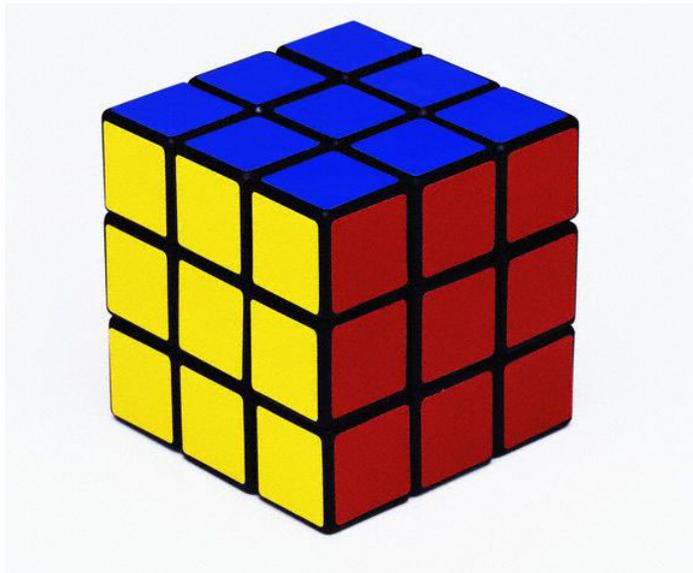
NEKBONE COMMUNICATION

- Point to Point Communication

- 26 send/receives per rank
 - 8 vertex values sent/received (8 Bytes per message, for 512x16 case)
 - 12 edges sent/received (128 Bytes per message, for 512x16 case)
 - 6 faces sent/received (16,384 Bytes per message, for 512x16 case)

- Collective Communication

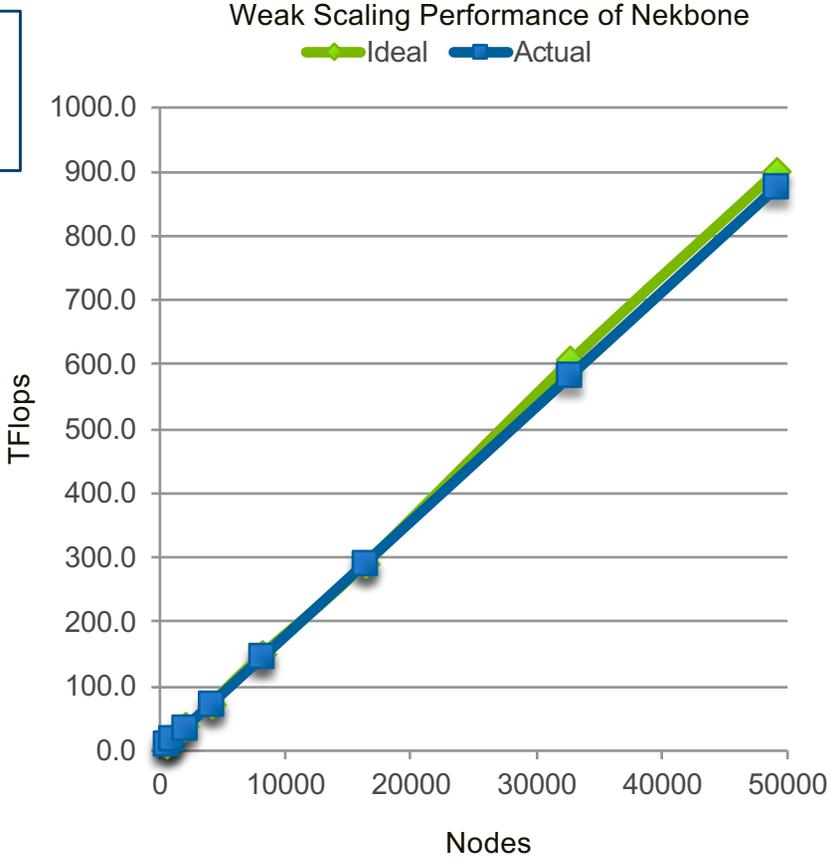
- Calls MPI_Allreduce 3 times per CG iteration
- 8 Byte (1 double) reduction per call
- 24 bytes per iteration



NEKBONE SCALING ON MIRA

Grid Points per thread: ~10k
 FLOP Rate: 9% of peak
 Parallel Efficiency: 99%

Rank S	Thread S	TFlops
512	64	9.5
1024	64	18.9
2048	64	36.9
4096	64	73.9
8192	64	150.5
16384	64	291.1
32768	64	606.9
49152	64	900.8



TYPICAL RATIOS ON REPRESENTATIVE BG/Q RUNS

Ratio	Value
FLOPS/(Bytes Loaded & Stored)	0.94
Loaded Bytes/ Stored Bytes	4
FLOPS/AllReduce	158,000,000
FLOPS/Pt2Pt Byte	4,744
FLOPS/MPI-Message	9,111,545

Routine Type	Percentage
Memory Bound	45%
Compute Bound	35%
Point to Point Comm.	18%
Collective Comm.	2%

ADDING OPENMP TO NEKBONE

- Adding OpenMP:
 - Relatively straightforward: 90% trivial, 10% required detailed understanding
 - Basic approach: partition element across threads
 - Easy:
 - Add a single OMP parallel region at top of cg() routine
 - Modify routines (add2s2, glsc3, axi, etc) to take a range of elements as an arg
 - Modify routines to use locally declared work arrays (ax_e)
 - A bit more complex:
 - Restructure gather/scatter maps for parallel execution
 - Add synchronization and barriers around communication operations (gs_aux, pw_exec)
- Impact:
 - Little impact on compute performance
 - Little impact on memory usage
 - Some impact on communication performance, most noticeable at large scale
 - Eliminates some data copies to/from MPI buffers
 - Fewer messages sent
 - Provides opportunity to overlap communication and computation

NEKBONE ON KNL

- Nekbone is up and running on KNL
 - Simulations and estimates of performance based on KNL specs
 - Run on pre-release KNL hardware
 - Performance as expected based on compute performance model
 - Tuning use of AVX-512 instructions
 - Utilizing LIBXSMM for matrix multiplication

WHY NEKBONE?

- System designers need representative applications to study
 - HPC has unique characteristics
- In comparison to Nek5000 Nekbone is:
 - More easily configurable
 - Number of spectral elements per rank
 - Polynomial order of element
 - Quicker to run
 - Run time is adjustable over a wide range
 - Typical run time is a few seconds
 - Allows multiple cases in one run
 - A range of elements can be specified
 - A range of polynomial orders can be specified
 - More easily instrumented
 - More easily modified
 - Has ~8K lines of code vs 60k lines of code for Nek5000
 - Re-implemented using other programming models: OpenMP, OpenACC, CUDA