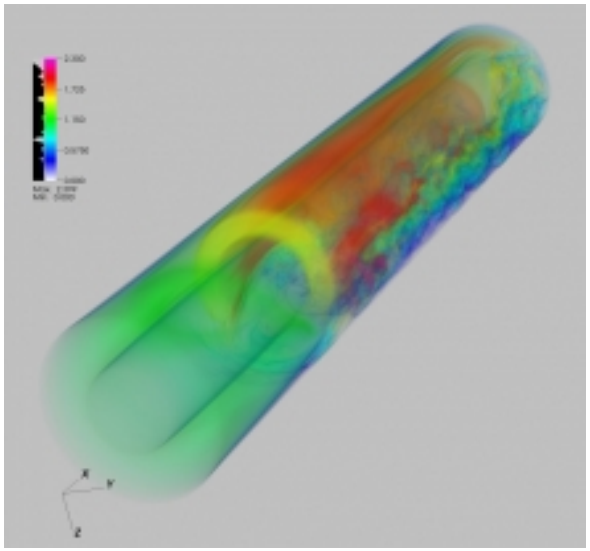
Argonne NATIONAL LABORATORY

Nekbone

Scott Parker
Argonne
Leadership
Computing
Facility

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
  - 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 benchmark derived from Nek5000**
  - Developed by Katherine Heisey, Paul Fischer, and Scott Parker
- **Solves 3D Poisson problem in rectangular geometry**
  - Spectral Element method with Conjugate Gradient linear solver
  - Large percentage of the work 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, and OpenMP
- **Used for**
  - Exascale co-design activities: DOE FastForward, DesignForward
  - DOE machine acquisitions: CORAL systems
- **Available at**
  - https://cesar.mcs.anl.gov/content/software/thermal_hydraulics
  - https://asc.llnl.gov/CORAL-benchmarks/

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

Nekbone in a nutshell

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()**
  - **gs_op()** *[ptp communication]*
  - **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**
- **Network Bound**

gs_op()

- **gs_gather()** *[while, out[j] = out[j] + in[i]]*
- **pw_exec()**
  - **pw_exec_recvs()** *[MPI_Irecv]*
  - **gs_scatter()** *[while, out[j] = in[i]]*
  - **pw_exec_sends()** *[MPI_Isend]*
  - **comm_wait()** *[MPI_Waitall]*
  - **gs_gather()** *[while, out[j] = out[j] + in[i]]*
- **gs_scatter()** *[while, out[j] = in[i]]*

Nekbone Compute Performance Model

Tabulated Operation Counter Per Routine

Routine	Routine	Routine	Routine	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
	gop	mpi_allreduce						
add2s1				p,z	p(i)=C*p(i)*z(i)	2N	N	2N
axi								
	ax_e							
		local_grad3						
			mxm	p,ur,dxm1		N	0	$2N_xN$
			mxm	p,us,dxtm1		0	0	$2N_xN$
			mxm	p,ut,dxtm1		0	0	$2N_xN$
		wrwswt		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	w,ur,dxtm1		0	0	$2N_x*N$
			mxm	t^*,us,dxm1		0	0	$2N_x*N$
			add2	w,t^*	w(i)=w(i)+t(i)	0	0	N
			mxm	t^*,ut,dxm1		0	0	$2N_x*N$
			add2	w,t*	w(i)=w(i)+t(i)	0	N	N
	gs_op							
	add2s2			w,p	w(i)=w(i)+c*p(i)	2N	N	2N
	mask							
glsc3				w,c,p	t=t+w(i)*c(i)*p(i)	3N	0	3N
	gop	mpi_allreduce						
add2s2				x,p	x(i)=x(i)+C*p(i)	2N	N	2N
add2s2				r,w	r(i)=r(i)+C*w(i)	2N	N	2N
glsc3				r,c,r	t=t+r(i)*c(i)*r(i)	2N	0	3N

Nekbone Compute Performance Model

Measured time and performance for each kernel invocation

Routine	Av Time	% Time	Bytes Loaded/it	Bytes Stored/it	FP Operations/it	GB/s	Gflop/s	Est time	Perf Ratio
Solver Time	1.77E+00								
rzero	5.06E-04	0.03%	0	113,246,208	0	224.03	0.00		
copy	6.11E-04	0.03%	113,246,208	113,246,208	0	370.66	0.00		
glsc3a	7.36E-04	0.04%	226,492,416	0	42,467,328	307.64	57.68		
gopa	1.56E-05	0.00%	0	0	0	0.00	0.00		
solveM	6.56E-02	3.71%	113,246,208	113,246,208	0	345.43	0.00	0.053292333	0.81
glsc3b	1.09E-01	6.16%	339,738,624	0	42,467,328	312.20	39.03	0.0799385	0.73
gopb	1.50E-03	0.08%	0	0	0	0.00	0.00		
add2s1	8.29E-02	4.69%	226,492,416	113,246,208	28,311,552	410.01	34.17	0.0799385	0.96
localgrad3	2.57E-01	14.56%	113,246,208	0	1,358,954,496	44.00	528.04	0.060322909	0.23
wrwswt	2.34E-01	13.23%	679,477,248	0	212,336,640	290.54	90.79	0.159877	0.68
localgradt	3.31E-01	18.75%	0	113,246,208	1,387,266,048	34.17	418.62	0.061579636	0.19
gsop	2.26E-01	12.77%	0	0	0	0.00	0.00		
add2s2a	8.76E-02	4.95%	226,492,416	113,246,208	28,311,552	388.03	32.34	0.0799385	0.91
glsc3c	1.12E-01	6.33%	339,738,624	0	42,467,328	303.72	37.96	0.0799385	0.71
gopc	1.25E-03	0.07%	0	0	0	0.00	0.00		
add2s2b	8.33E-02	4.71%	226,492,416	113,246,208	28,311,552	407.96	34.00	0.0799385	0.96
add2s2c	8.53E-02	4.83%	226,492,416	113,246,208	28,311,552	398.07	33.17	0.0799385	0.94
glsc3d	7.82E-02	4.42%	226,492,416	0	42,467,328	289.62	54.30	0.053292333	0.68
gopd	1.28E-03	0.07%	0	0	0	0.00	0.00		

Summary of Current Nekbone Performance on KNL

- **Nekbone solver time**
  - KNL Bin3, 16GB MCDRAM, 64 Core, 1.1-1.3 GHz : **1.77 seconds**
  - KNL A0, 16 GB, 72 Core : **2.87 seconds**
  - Haswell E5-2699 v3, dual socket, 36 cores, 2.30GHz**: 3.6 seconds**
- **KNL kernel mix:**
  - Streaming kernels – 53% of time
  - Matrix multiply – 33% of time
  - Communication – 13% of time
- **KNL kernel performance:**
  - Streaming kernels are achieving 70-95% of Stream bandwidth
  - Small matrix multiply kernel is achieving 20-25% of peak floating point performance
    - Working on integrating improved matrix multiply routines