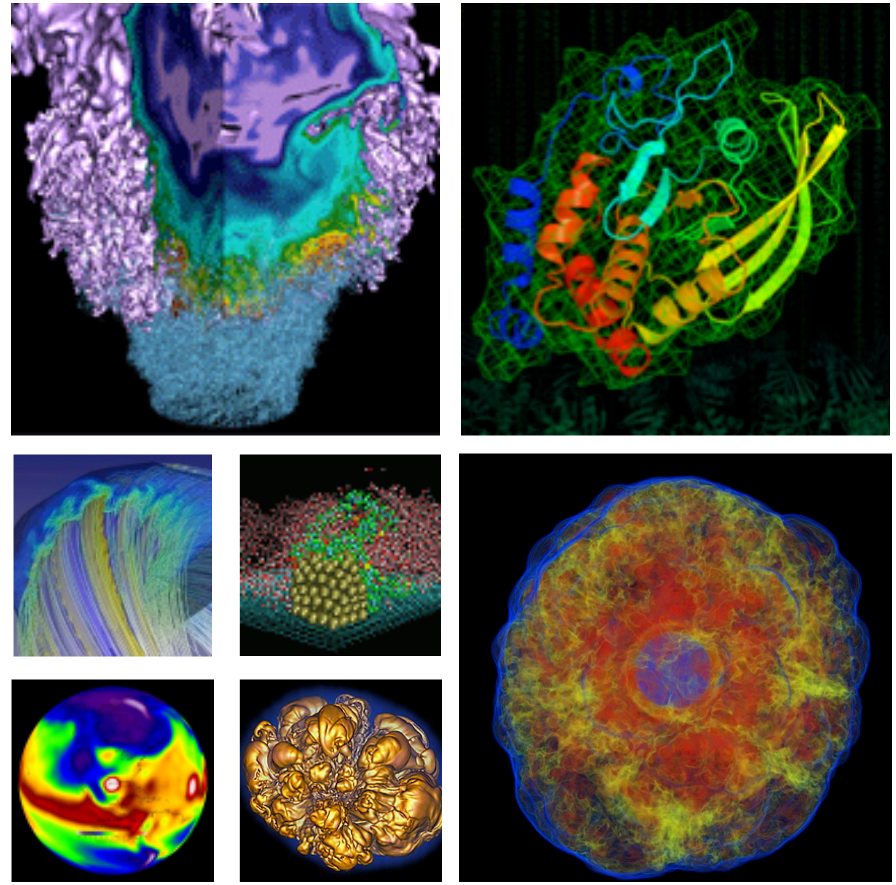
NESAP XGC1 Dungeon Update



NERSC 40 YEARS at the FOREFRONT 1974-2014

Helen He (and XGC1 team)

Cray Quarterly Meeting
July 22, 2015

U.S. DEPARTMENT OF ENERGY | Office of Science

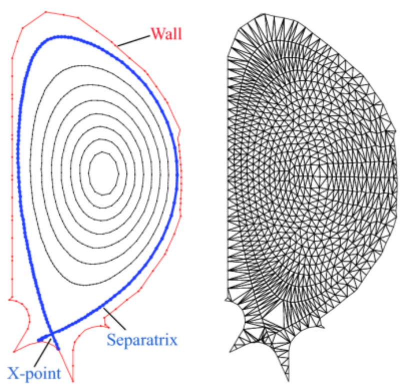
BERKELEY LAB

XGC1 NESAP Team Members

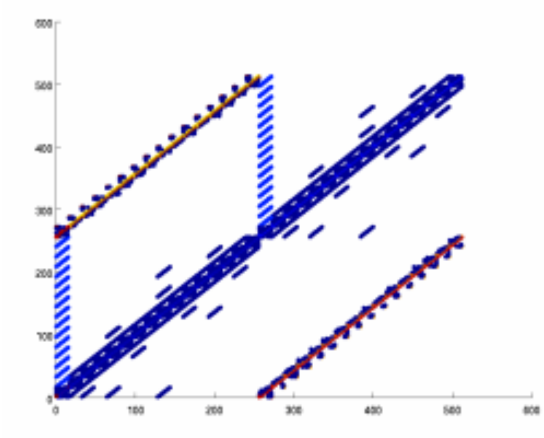
- **Name with *: Attended the Dungeon Session**
- **Code Team:**
  - PPPL: C.S. Chang (PI), Seung-Hoe Ku, Jianying Lang*, Stephan Ethier, Robert Hager
  - ORNL: Ed D'Azevedo*, Pat Worley
  - LBNL: Mark Adams
  - RPI: Eisung Yoon*
  - A good mix of physics, performance and library people
- **Cray:** Nathan Wichmann*
- **Intel:** Thanh Phung, Dmitry Nemirov*, Antonio Valles*
- **NERSC Liaison:** Helen He*

XGC1: a PIC Fusion Code

- **Particle-in-cell code used to study turbulent transport in magnetic confinement fusion plasmas.**
- **Uses fixed unstructured grid. Hybrid MPI/OpenMP for both spatial grid and particle data. (plus PGI CUDA Fortran, OpenACC)**
- **Excellent overall MPI scalability**
- **Internal profiling timer borrowed from CESM**
- **Uses PETSc Poisson Solver** (separate NESAP effort)
- **60k+ lines of Fortran90 codes.**
- **For each time step:**
  - Deposit charges on grid
  - Solve elliptic equation to obtain electro-magnetic potential
  - Push particles to follow trajectories using forces computed from background potential (~50-70% of time)
  - Account for collision and boundary effects on velocity grid
- **Most time spent in Particle Push and Charge Deposition**



Unstructured triangular mesh grid due to complicated edge geometry

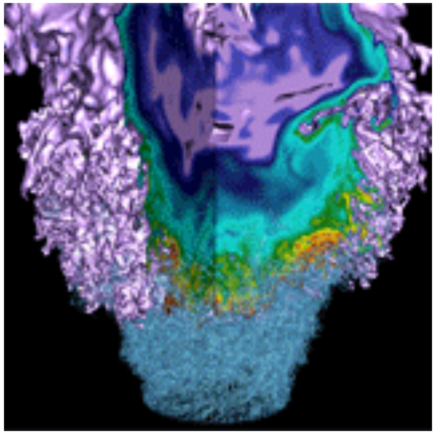


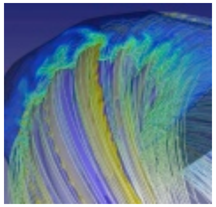
Sample Matrix of communication volume

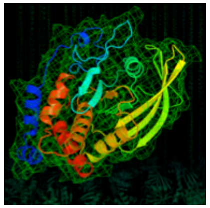
Programming Portability

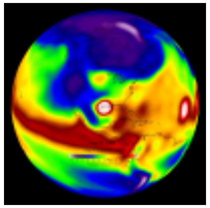
- Currently XGC1 runs on many platforms
- Part of NESAP and ORNL CAAR programs
- Applied for ANL Theta program
- Have #ifdef _OpenACC and #ifdef _OpenMP in code.
- OpenMP 4.0 target directives
- PGI CUDA Fortran
- As fewer compiler dependent directives as possible.
- Nested OpenMP is used
- Need thread safe PSPLIB and PETSc libraries.

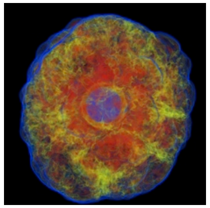
Multi Species Collision Kernel

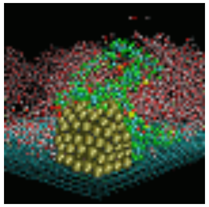












NERSC 40 YEARS at the FOREFRONT 1974-2014

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

XGC1 Multi-Species Collision Kernel

- **Had many iterations of single-species collision kernel.**
  - Good OpenMP scaling
  - Hotspot is 33% of cpu time. Multiple loops with ~80% vectorization efficiency
  - Many accesses unaligned. Non sequential access too.
- **Multi-species version ready on July 6.**
- **Nathan optimized with initialization and vectorization: 20150708 version**
- **Two main goals**
  - Vectorization
  - Evaluate for HBM analysis

"-heap-arrays 64" Compiler Flag

- Slows down both the collision and pushe kernels by >6x.
- Puts automatic arrays and arrays created for temporary computations of size (64 kbytes or larger) on the heap instead of the stack.
- Allocation and access of private copies on the heap are very expensive.
- Does no affect explicit-shape arrays.
- Removed this flag for the collision kernel, and set OMP_STACKSIZE to a large value. Now 20150708 version: Intel compiler 43 sec. (improved from 348 sec)
- Alternative: use !$OMP THREADPRIVATE. Downside: data has to be static, not allocatable.

XGC1 Collision Dungeon: Tools Examined

- **OMP imbalance**
  - Using 18 threads on 1 socket on Haswell EX, imbalance is 0.7%
  - Using 60 threads on the node: imbalance is 22%
  - Added KMP_BLOCKTIME=infinite variable to prevent "sleeping" of some threads
- **Vector Advisor**
  - 4 hotspots
  - Used "!DIR$ nounroll" and "!DIR$ loopcount(31)" to help vectorization
- **Vtune Memory Bandwidth analysis**
  - Reach peak bandwidth at times
- **Vtune Memory Access analysis**
  - Original collection shows the array name as unknown due to not dynamically allocated.
  - Modified array declaration, put into FASTMEM, 24% faster with 14 threads on 1 socket on Haswell EP
- **SDE for collecting instruction mixes**
  - 11% less instructions on KNL vs Haswell AVX2

Vector Advisor after Vectorization of Scalar Loop

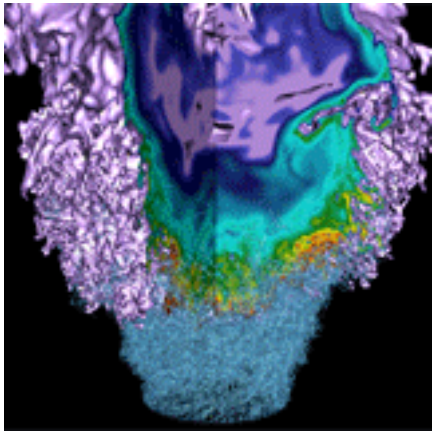
Elapsed time: 35.53s | Vectorized | Not Vectorized | FILTER: All Modules | All Sources

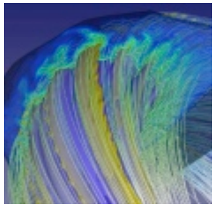
Loops		Vector Issues	Self Time	Total Time	Loop Type	Why No Vectorization?	Vectorized Loops			
							Vector...	Efficiency	Estim...	Vector Le
[loop in col_f_angle_avg_m at col_f_core_m.F90:546]		3 Ineffecti...	177.826s	177.826s	Vectorized: ...		AVX2	~87%	3.49	4
[loop in col_f_angle_avg_m at col_f_core_m.F90:590]		3 Ineffecti...	164.763s	164.763s	Vectorized: ...		AVX	~33%	1.33	4
[loop in col_f_e_and_d_m at col_f_core_m.F90:670]			54.042s	54.042s	Vectorized: ...		AVX2	~74%	2.97	4
[loop in ellip_agm_v at ellip_agm.F90:40]			42.677s	42.677s	Vectorized: ...		AVX	~100%	4.80	4
[loop in col_f_angle_avg_m at col_f_core_m.F90:526]			**40.936s**	**507.7...**	**Scalar**	**volatile a...**				
[loop in col_f_angle_avg_m at col_f_core_m.F90:530]			22.371s	22.371s	Vectorized: ...		AVX2	~88%	3.52	4
[loop in ellip_agm_v at ellip_agm.F90:31]		3 Ineffecti...	20.568s	20.568s	Vectorized: ...		AVX	~67%	2.67	4
[loop in __kmp_launch_thread at kmp_runtime.c:5600]			15.395s	597.070s	Scalar					
[loop in col_f_e_and_d_s at col_f_core_s.F90:833]			10.889s	10.889s	Vectorized: ...		AVX2	~81%	3.25	4
[loop in col_f_e_and_d_s at col_f_core_s.F90:866]		2 Ineffecti...	8.810s	8.810s	Vectorized: ...		AVX2	~83%	3.31	4
[loop in col_f_e_and_d_s at col_f_core_s.F90:811]			6.449s	26.148s	Scalar	inner loop w...				
[loop in col_f_angle_avg_s at col_f_core_s.F90:586]			6.416s	13.221s	Scalar	outer loop w...				
[loop in ellip_agm_v at ellip_agm.F90:62]		3 Ineffecti...	5.180s	5.180s	Vectorized: ...		AVX	~57%	2.27	4
[loop in ellip_agm_v at ellip_agm.F90:54]		2 Ineffecti...	3.980s	3.980s	Vectorized: ...		AVX2	~56%	4.45	8
[loop in __intel_avx_rep_memset]			3.549s	3.549s	Scalar					
[loop in col_f_e_and_d_m at col_f_core_m.F90:669]			2.190s	56.232s	Scalar					
[loop in col_f_angle_avg_m at col_f_core_m.F90:522]			0.792s	508.520s	Scalar	inner loop th...				
[loop in col_f_lu_matrix at col_f_core_s.F90:1003]			0.700s	1.859s	Scalar	inner loop w...				
[loop in col_f_e_and_d_m at col_f_core_m.F90:659]			0.220s	56.452s	Scalar	inner loop w...				
[loop in col_f_e_and_d_s at col_f_core_s.F90:910]			0.190s	0.190s	Scalar	inner loop w...				
[loop in col_f_picard_step at col_f_core_s.F90:1098]		3 Gather/S...	0.140s	0.140s	Vectorized: ...		AVX2	~13%	1.03	8
[loop in col_f_core_m at col_f_core_m.F90:248]		1 System f..	0.090s	0.200s	Vectorized: ...	1 inner loop ...				
[loop in col_f_core_m at col_f_core_m.F90:245]		1 System f..	0.080s	0.170s	Vectorized: ...	1 inner loop ...				
[loop in bsolver at bsolver.F90:93]		1 Assumed..	0.080s	0.150s	Threaded (O...	vector depe...				
[loop in bsolver at bsolver.F90:96]		1 Assumed..	0.070s	0.070s	Scalar	vector depe...				
[loop in col_f_core_m at col_f_core_m.F90:259]			0.070s	0.070s	Vectorized: ...		AVX	~100%	4.16	4
[loop in col_f_f_df at col_f_core_s.F90:716]			0.060s	0.060s	Vectorized: ...		AVX	~78%	3.11	4
[loop in col_f_core_m at col_f_core_m.F90:261]			0.060s	0.060s	Vectorized: ...		AVX2	~100%	3.99	4
[loop in col_f_e_and_d_s at col_f_core_s.F90:802]			0.060s	26.208s	Scalar	inner loop w...				
[loop in col_f_core_m at col_f_core_m.F90:130]			0.050s	0.050s	Scalar	inner loop w...				

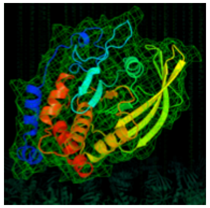
HBM Simulation on Haswell

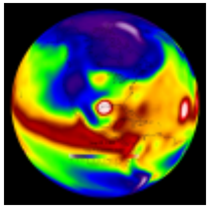
- **To simulate using HBM directive following options should be added :**
  - !DIR$ ATTRIBUTES FASTMEM :: <data name> should be add immediately after 'allocatable' in source code
  - '-ljemalloc -lmemkind -lpthread -lnuma -L/<PATH_to_jemaloc> -L/<PATH_to_memkind>' to compile options
  - set MEMKIND_HBW_NODES=0
  - Run the application using 'numactl --membind=1 --cpunodebind=0 <binary>'

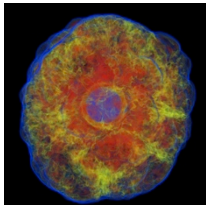
PushE Kernel

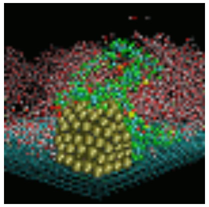












XGC1 PushE Kernel

- **Lots of iterations to reach barebone:**
  - No PETSc or ADIOS, no ChargeE
  - Only initialization and PSPLINE are kept
- **Does not vectorize, need major code work**
  - Many loop counts smaller than vector length
- **Initially only pushes 1 particle at a time, appears good cache hit rate**
- **Not memory bandwidth bound**
- **Two main goals:**
  - Vectorization
  - Evaluate for HBM analysis

Particle Push Kernel

- Code modified to push groups of particles (group size is sml_veclen in input) to encourage vectorization
- Removed “-heap-arrays” option and adjusted OMP_STACKSIZE
- Code modified to avoid vector notation with modulo(). Bug filed for Intel compiler.

Top Hotspots

- Top hot spots related to particle search and evaluation of bicubic spline interpolation
- Performance limited by data movement

Grouping: Source Function / Function / Call Stack

Source Function / Function / Call Stack	Hardware Event Count by H		
	INST_RETIR...	CPU_CLK_U...	MEM_LOA...
search_tr2	35,544,400,000	33,794,200,000	124,250,000
i_interpol_wo_pspline	40,543,600,000	25,217,400,000	310,000
eval_bicub_1	55,242,200,000	23,616,400,000	22,760,000
__kmp_wait_template<kmp_flag_64>	20,727,400,000	19,204,800,000	10,000
field_following_pos2	10,608,000,000	15,491,400,000	660,000
efield_gk_elec2	25,787,200,000	14,144,800,000	47,580,000
derivs_elec_vec	22,361,400,000	12,652,400,000	17,970,000
eval_bicub_2	33,096,400,000	10,645,000,000	16,420,000
bicub_interpol1	11,648,600,000	8,936,400,000	1,980,000
bvec_interpol	4,794,200,000	8,673,600,000	3,340,000
derivs_single_with_e_elec_vec	13,198,600,000	7,091,400,000	13,160,000
Selected 1 row(s):	35,544,400,000	33,794,200,000	124,250,000

Search and Spline Evaluation

- **Search operation: Find which triangle contains the particle to perform interpolation. Current grid has O(10^5) triangles. Future grid has O(10^6) triangles**
  - Geometric hashing of particle coordinates to 2D uniform rectangular grid
  - Search short list of triangles that overlap with that grid cell (fewer than 10 triangles)
- **Spline evaluation:**
  - Geometric hashing to 2D rectangular grid
  - Evaluate bicubic polynomial using coefficients from table in that grid cell

Original Search Routine

- Initial version has short vectors of length 2
- Indirect addressing in "itrig"
- Early exit when the triangle is found

```
jlo = lbound( grid%guess_table, 2 )
jhi = ubound( grid%guess_table, 2 )

ij = (xy - grid%guess_min)*grid%inv_guess_d + 1
i = max(ilo, min(ihi, ij(1)) )
j = max(jlo, min(jhi, ij(2)) )

istart = grid%guess_xtable(i,j)
iend = istart + grid%guess_count(i,j) - 1
itr = -1
do k=istart,iend
   itrig = grid%guess_list(k)
  dx(1:2) = xy(1:2) - grid%mapping(1:2,3,itrig)
  p(1:2)= grid%mapping(1:2,1,itrig)*dx(1) +
          grid%mapping(1:2,2,itrig)*dx(2)
  p(3)=1.0d0 - p(1) - p(2)
   if (minval(p) .ge. -eps) then
      itr = itrig; exit
   endif
enddo
```

Strip-mine (Loop blocking / Loop tiling)
Search Routine

Assembly | Assembly grouping: Address

Source	CPU Time: Effective Time	Spi...	Ov..	Ins... Re...
start = grid%guess_xtable(i,j)	0.231s	0s	0s	1,2..
end = istart + grid%guess_count(i,j) - 1	0.646s	0s	0s	1,6..
tr = -1	1.331s	0s	0s	3,6..
se_vector = (iend-istart+1 .ge. veclen)	0.222s	0s	0s	199..
f (use_vector) then	0.055s	0s	0s	45,..
do kstart=istart,iend,veclen	0.055s	0s	0s	201..
kend = min(iend, kstart+veclen-1)	0.038s	0s	0s	111..
klen = kend - kstart + 1	0.191s	0s	0s	388..
grid_mapping(1:2,1:3,1:klen) = grid%mapping(1:2,1:3,grid%guess_list(kstart:kend))	2.734s	0s	0s	7,3..
ir unroll 4				
do kk=1,min(klen,veclen)	0.260s	0s	0s	1,2..
k = (kstart-1) + kk				
! itrig = grid%guess_list(k)				
dx_vec1(kk) = xy(1) - grid_mapping(1,3,kk)	0.108s	0s	0s	487..
dx_vec2(kk) = xy(2) - grid_mapping(2,3,kk)	0.153s	0s	0s	838..
p_vec1(kk)= grid_mapping(1,1,kk)*dx_vec1(kk) + &	0.279s	0s	0s	1,8..
grid_mapping(1,2,kk)*dx_vec2(kk)	0.035s	0s	0s	203..
p_vec2(kk)= grid_mapping(2,1,kk)*dx_vec1(kk) + &	0.122s	0s	0s	916..
grid_mapping(2,2,kk)*dx_vec2(kk)	0.020s	0s	0s	61,..

Future Work

- **Collision kernel**
  - Explore nested OpenMP
- **Pushe kernel**
  - Explore particle sorting for each grid cell and global data rearrangement before push. Needs major code modification.
  - Design prototype test to examine effectiveness of sorting.
  - Explore replicating triangle data structure to avoid indirect addressing, may use 5X more memory (~200 MB per MPI task)
  - Reorganize OpenMP outer loop over grid cells and pushe particles in cell

Thank you.