BLUE WATERS

SUSTAINED PETASCALE COMPUTING

May 23, 2013

Scaling Applications on Blue Waters

New User BW Workshop May 22-23, 2013

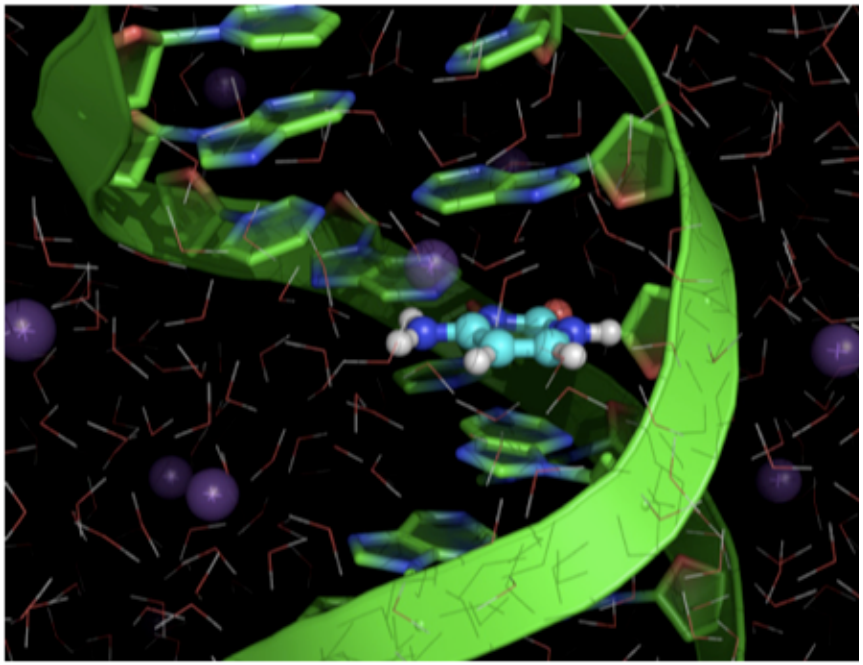
NCSA

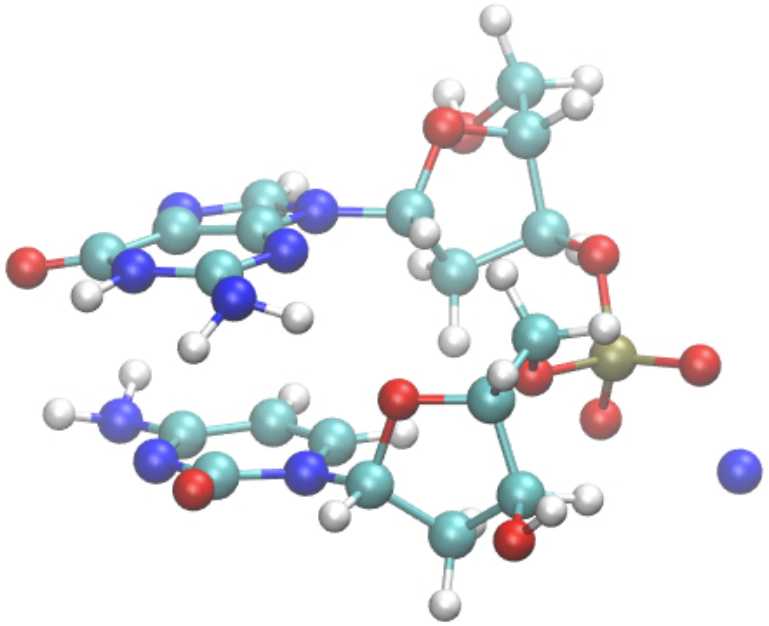
NSF

GREAT LAKES CONSORTIUM
FOR PETASCALE COMPUTATION

CRAY

NWChem





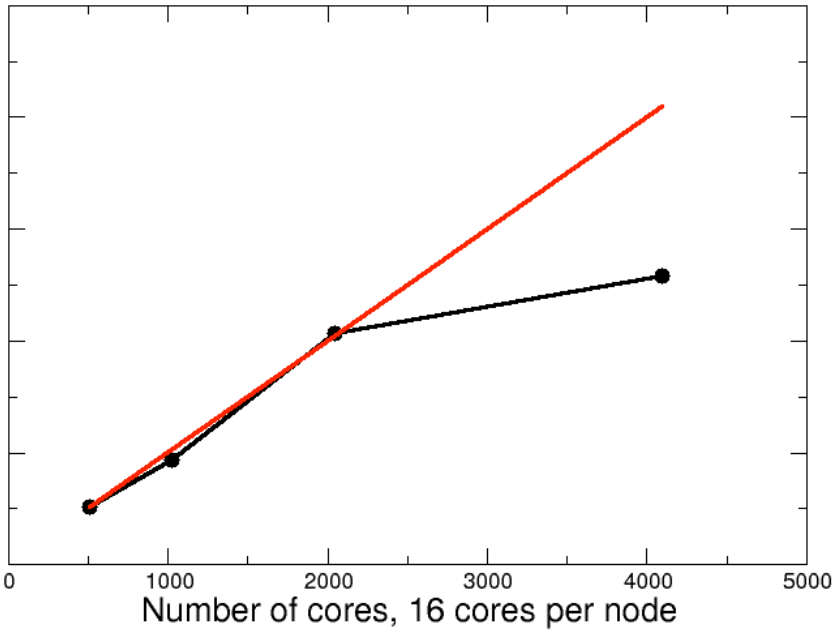
- NWChem is *ab initio* Quantum Chemistry package
- Compute electronic structure of molecular systems
- PNNL home: http://www.nwchem-sw.org

Coupled Cluster Engines in NWChem

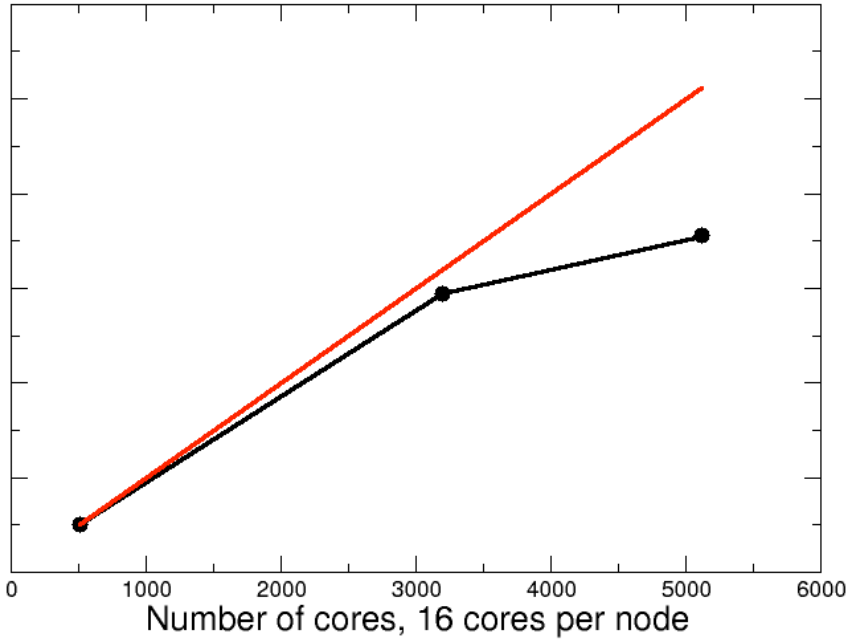
GC-dDMP cc-pVDZ basis set Nocc=150 Nvir=563	Rendell's CCSD	TCE CCSD tilesize=10
3200 cores (100 nodes)	5.4 hours	Did not fit in aggregate memory
9600 cores (300 nodes)	3.5 hours	+8 hours (did not finish)
12800 cores (400 nodes)		+8 hours (did not finish)

- Rendell's CCSD(T) code is computationally more efficient than TCE CCSD(T) for closed-shell systems
- Chemistry has lots of computational problems that need closed-shell CCSD(T)
- Interesting problems in chemistry push methods to treat larger sizes

NWChem 6.1 - Scaling on Blue Waters



CCSD 1-iter



(T)

Gua-Gua stack, 6-311++G**, 554 basis functions, Cray DMAP GA/ARMCI

Larger input sizes utilize the machine more efficiently

Perturbative triples (T) scale much better than singles and doubles CCSD

Perturbative Triples

# nodes	# cores	Time, sec	Efficiency
5,000	80,000	9117.8	
20,000	160,000	5024.4	45%

- GC-dDMP, 6-311++G**, 1042 basis functions
- Larger problem sizes could be efficiently computed

Known PetaFLOP reports for NWChem on (T)

- **0.487 PF**; scaling to **96,000** cores; E. Apra, et al, 2009, **SC09**: $(H_2O)_{20}$, 1020 basis functions, CCSD (T)
- Performance not reported; scaling to **210,000** cores; K. Kowalski, et al, 2011, **SC11**: Porphyrins, 780 basis functions, EOM-CCSD(T)
- **1.184 PF**; scaling to **160,000** cores; **NCSA BW** 2013, GC-dDMP, 1042 basis functions

Finally, NWChem reached 1 PF !!!

NWChem Sustained Petaflop Performance

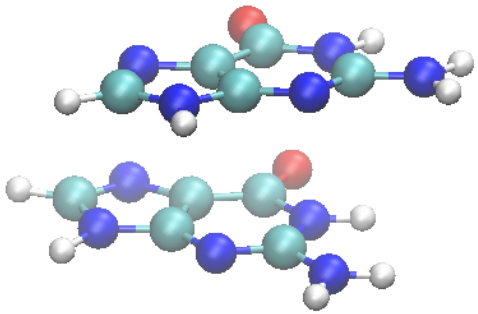
Previous results do not qualify for SPP due to benchmarking a portion of the run. On BW, SPP is higher than even best previous (T)-only results.

GC-dDMP, 6-311++G**	BW XE6 value
Wall clock time of CCSD job (1,000 nodes)	5396.4 sec
Time of single CCSD iteration	4699.0 sec
Cost of initialization, termination, and I/O	697.3 sec
Number of CCSD iterations	16
Wall clock time of (T) job (5,000 nodes)	9117.8 sec
Total time to solution, CCSD(T)	24852.2 sec (7 hours)
Performance per node on BW	47.9 GF/sec/node
Aggregate performance on 5,000 nodes (80,000 cores)	0.239 PF/sec
Aggregate performance on 20,000 nodes (160,000 cores)	**0.627 PF/sec**

Blue Waters to Kraken Comparison

Kraken	200 nodes (2400 cores)	12347.6 sec
BW	150 nodes (2400 cores)	4447.8 sec
Kraken	2.7 GF/core	32.4 GF/node
BW	7.6 GF/core	121.6 GF/node
	Time, 1 CCSD iteration	**Time, triples (T)**
Kraken	2534.7 sec	9812.9 sec
BW	1026.4 sec	3421.4 sec
BW NWChem	321.1 sec (100 nodes, 800 cores)	

Gua-Gua stack
CCSD(T)
6-311++G**
554 functions



1 BW node (16 cores) corresponds to 3.8 Kraken nodes (12 cores)
1 BW core corresponds to 2.8 Kraken cores

Profiling: CCSD kernels

Test: Gua-Gua stack 713 basis functions, 9600 cores

CCSD routines	Aggregate time, sec	Times called by each core
ga_nbget()	**4794.65**	108545.03
idx1_wrk1	13.65	108039.01
t2eri	**4887.38**	5.02
schwarz	2.84	
CCSD-iter	10031.04	

- **ga_nbget()** consumes 48% of total time; 0.044 seconds per call
- **Problem:** the parallelization model of global data distribution hits the wall (all-to-all communications do not scale with machine size)
- **Solution:** get rid of the most critical ST2 distributed global array and keep the necessary data locally in each node

NWChem CCSD Efficiency

# nodes	# cores	Time, sec	Efficiency
Official code			
5,000	10,000	6734.0	
5,000	20,000	4329.2	78%
BW code			
1,000	1,000	4699.0	
5,000	5,000	1362.2	70%

- Local ST2 array – 44GB per node (compare: BW 64GB, Titan 32GB)
- 3-fold speedup on GC-dDMP due to data localization

Sample NWChem CCSD Input

start gc-ddmp
memory stack 2000 mb heap 200 mb global 1000 mb
…
set ccsd:useinmemst2 T
set ccsd:st2parallel T
task ccsd energy

Running GC-dDMP CCSD job

aprun -n 1000 -N 1

New input options in bold

Sample NWChem PBS Script

```
#PBS -l nodes=1000:ppn=32:xe
cd $PBS_O_WORKDIR
source /opt/modules/default/init/bash
module add craype-hugepages8M

# mandatory environment variables for congestion protection
export ARMCI_DMAPP_LOCK_ON_GET=1
export ARMCI_DMAPP_LOCK_ON_PUT=1
export APRUN_BALANCED_INJECTION=63

# MP2 or (T) computation
aprun -n 1000 -N 16 -d2 /u/staff/anisimov/nw61-20130220-cray/bin/
                                        LINUX64/nwchem flops.nw > job.out
```

Things to Watch in the Job Output

Throttling is a system-wide Gemini network slow-down event to prevent data loss due to communication congestion.

Evidence of Application throttling the High Speed Network fabric can be found at the end of your job output

Application 122336 **network throttled**: 5000 nodes throttled, 58:05:33 node-seconds

Application 122336 balanced injection 100, after throttle 63

Application 122336 resources: utime ~736228827s, stime ~1730307s

To prevent throttling make sure your PBS script includes

export ARMCI_DMAPP_LOCK_ON_GET=1

export ARMCI_DMAPP_LOCK_ON_PUT=1

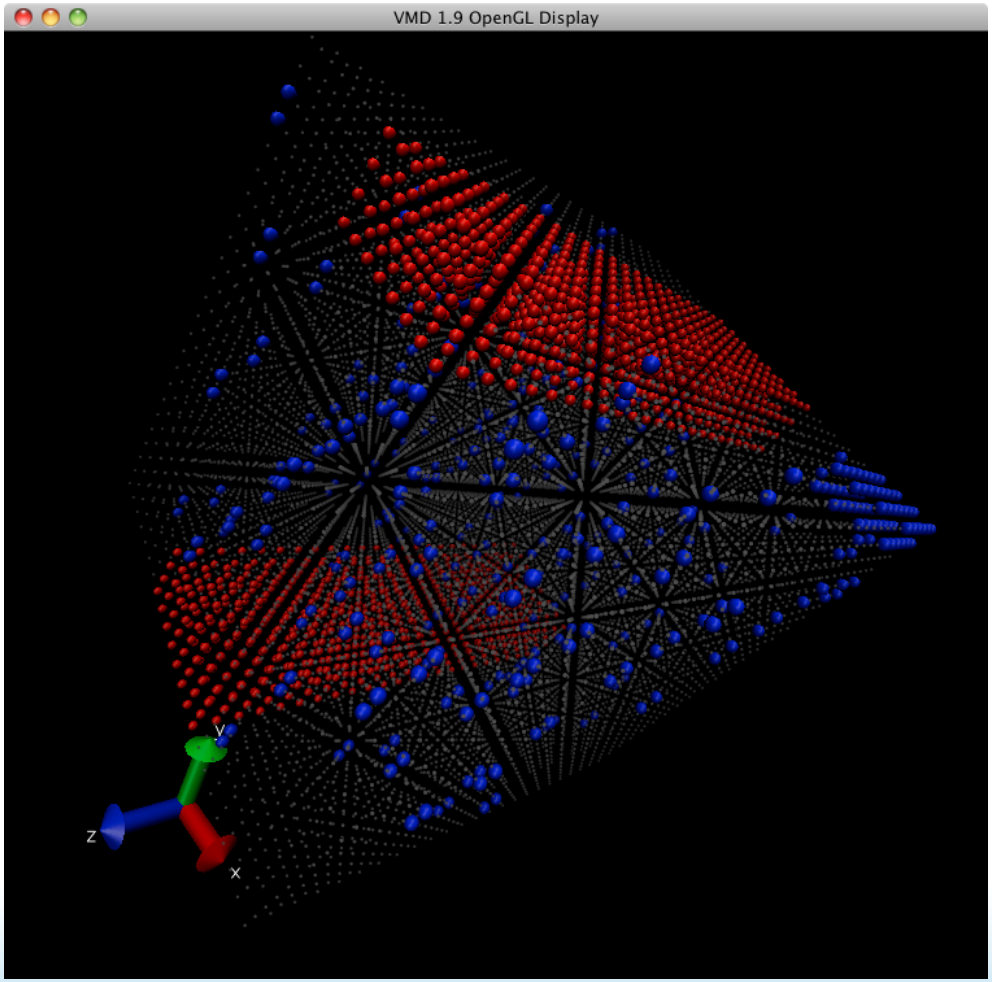
export APRUN_BALANCED_INJECTION=63 set to smaller value, if necessary

Throttling Emergency Manual

- Stop running your application until the cause of throttling is identified and the fix is applied
- Contact you Point-of-Contact or send request to help+bw@ncsa.illinois.edu asking for assistance
- Consider adding BI API calls (C-library) in your code
- Set a smaller value APRUN_BALANCED_INJECTION=33
- Use fewer nodes
- Choose the job placement on 3D torus

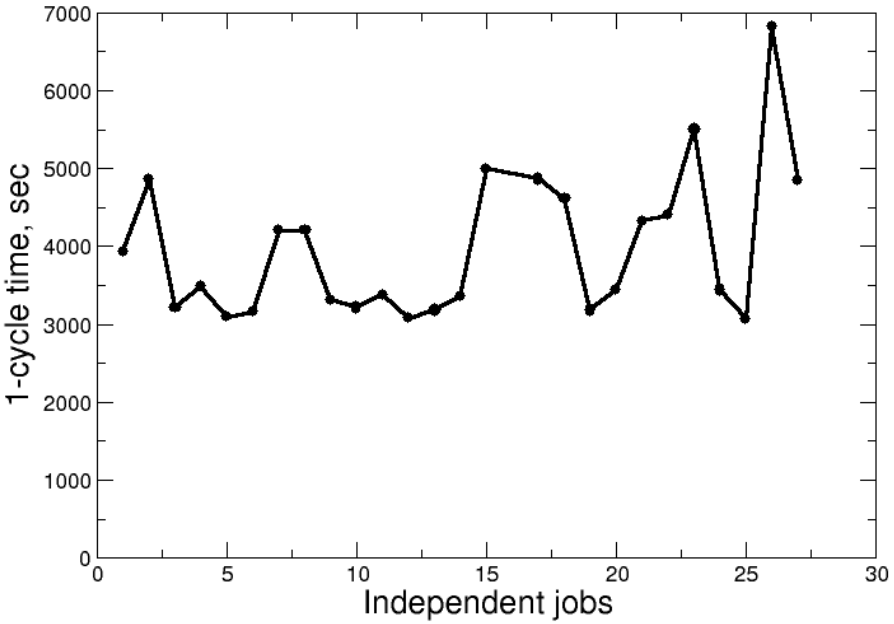
Cray 3D Torus Topology

- Periodic box in 3D
- XK nodes – red
- Service nodes – blue
- Compute nodes – gray
- Slower – Y, X, Z – Faster
- Routing X, then Y, then Z
- Routing path depends on application placement on 3D torus

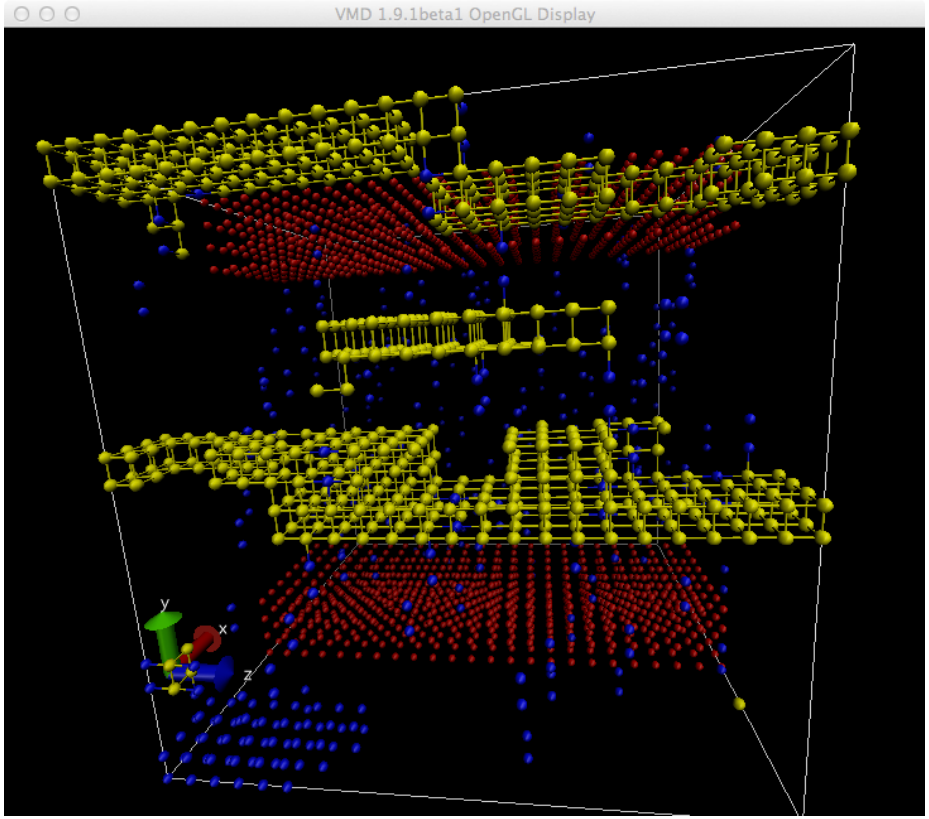


Why My Jobs Run Slow - Performance Variation

NWChem: Geometry Optimization: MP2/6-311++G**, 2098 basis functions



- Discuss your aprun options with support staff
- BW scale is unprecedented – Use advanced job placement



1-cycle time: 4033.6 sec

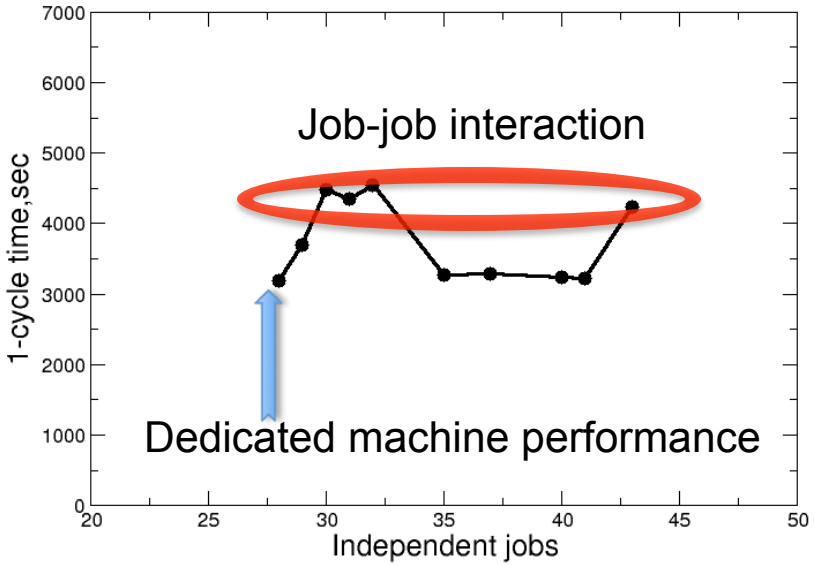
Exact Hostlist Node-Allocation

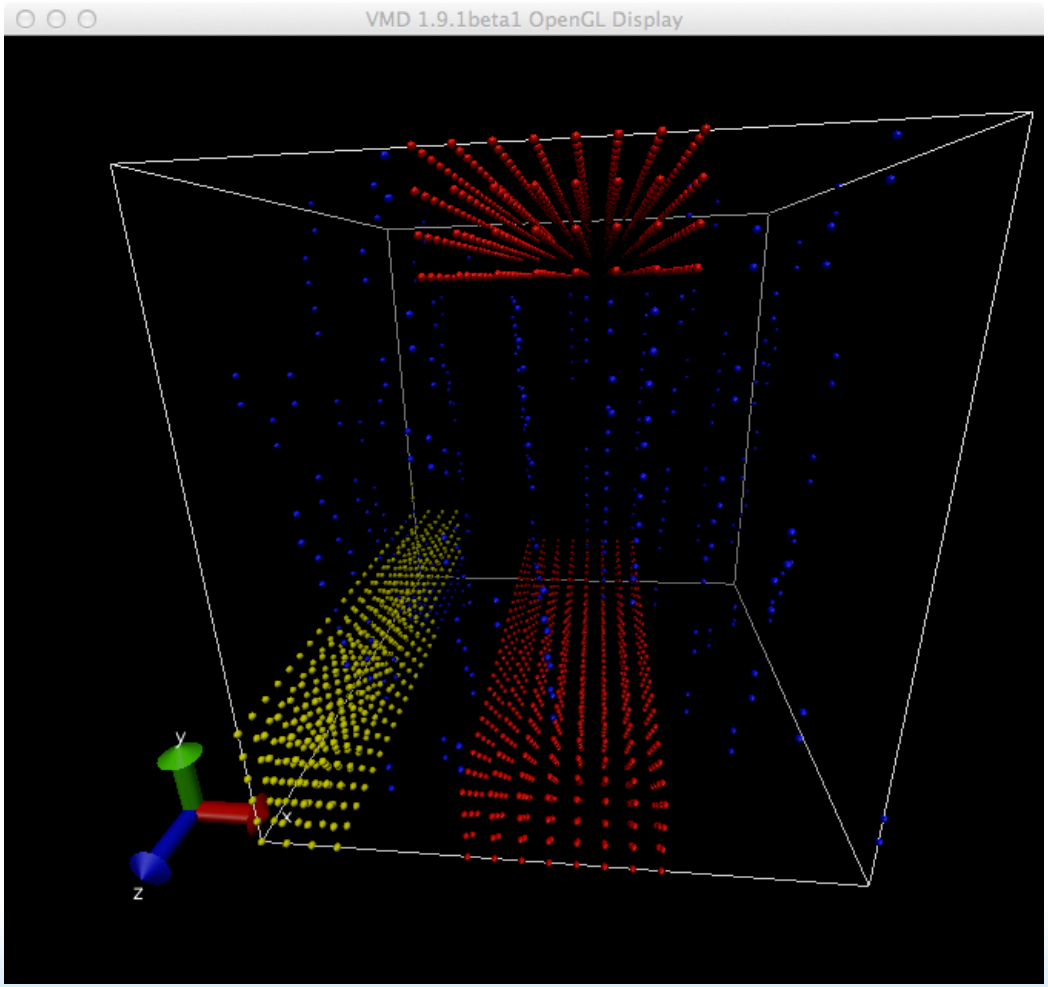
```
#!/bin/bash
#PBS -j oe
#PBS -l nodes=1000:ppn=32
#PBS -l walltime=14:00:00
#PBS -l hostlist=1824+... 5079+5082+5083+5084+5085+5086+5087^
cd $PBS_O_WORKDIR
source /opt/modules/default/init/bash
checkjob ${PBS_JOBID} > nodelist.${PBS_JOBID}
aprun -n 16000 -N 16 -d 2 ./a.out > job.out
```

Adaptive documentation:

http://docs.adaptivecomputing.com/mwm/Content/topics/resourceManagers/rmextensions.html#hostlist

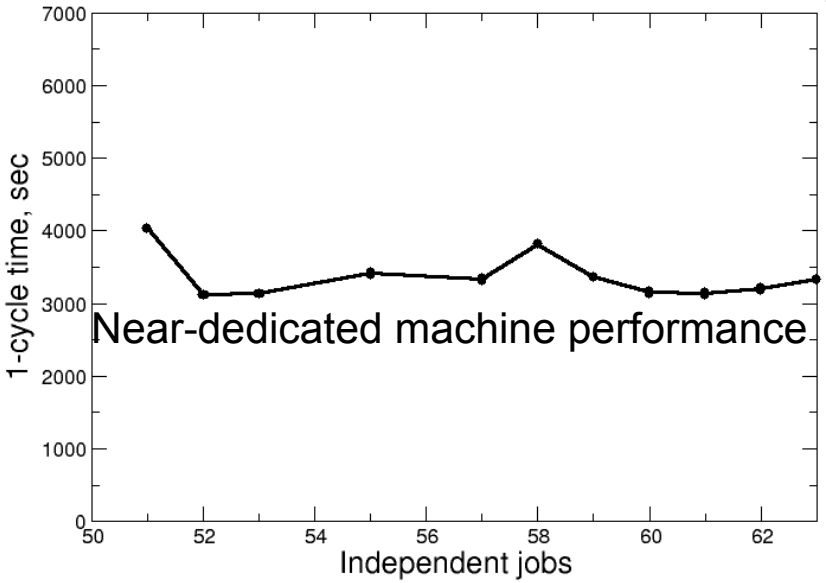
Performance Optimization on 3D Torus

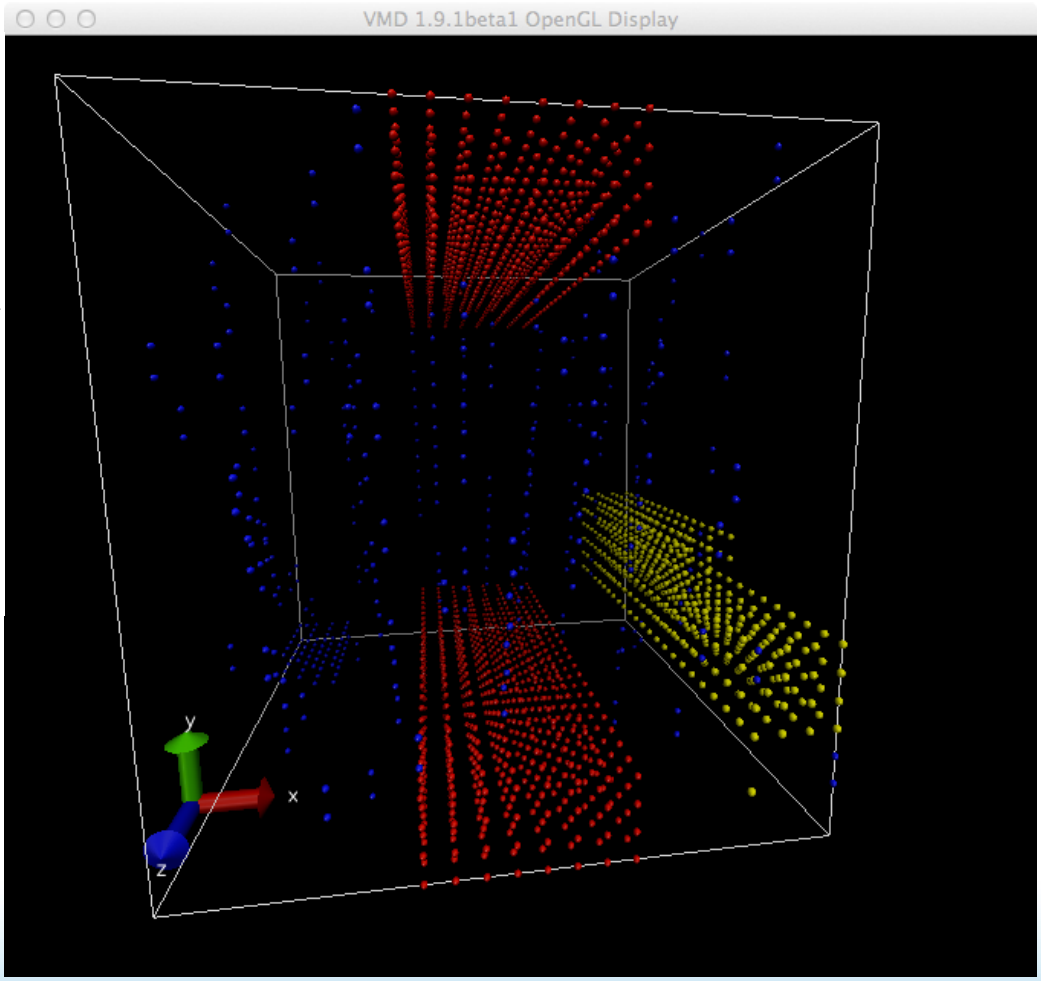




- 1000-node job – yellow
- Nice performance improvement
- Throttling still happens because of interaction with other jobs

Throttling-Free Placement on 3D Torus





- No throttling is observed
- Best performance near to that on dedicated system
- Variations still exist but the magnitude is small

Further Reading on Cray 3D Torus Topology

- Bob Fiedler, Cray Inc.: https://bluewaters.ncsa.illinois.edu/documents/10157/12008/AdvancedFeatures_PRAC_WS_2013-02-27.pdf

When to Approach for Help

- **Help Lines:**
  - Phone (217) 244-6689
  - Online chat via BW Portal bluewaters.ncsa.illinois.edu
  - Email help+bw@ncsa.illinois.edu
  - Online access to Jira ticket system via BW Portal
- Issues with authentication
- Jobs not running
- How to submit jobs on Blue Waters (XE, XK, bulldozer cores)
- Application fail to compile
- How to use the available resources efficiently
- Need help to improve application performance