Molecular Modelling and the Cray XC30 Performance Counters

Michael Bareford, ARCHER CSE Team
michael.bareford@epcc.ed.ac.uk

Reusing this material

archer

epcc

THE UNIVERSITY OF EDINBURGH

Introducing ARCHER

Advanced **R**esearch **C**omputing **H**igh **E**nd **R**esource

EPSRC

NERC SCIENCE OF THE ENVIRONMENT

CRAY
THE SUPERCOMPUTER COMPANY

archer

www.archer.ac.uk

epcc

THE UNIVERSITY OF EDINBURGH

Introducing ARCHER

Cray XC30 MPP, 4920 Compute Nodes

Dual Intel Xeon processors (Ivy Bridge), 24 cores, 64 GB

Dragonfly topology

rank 1: intra-chassis, sixteen 4-node blades (Aries interconnect)
rank 2: intra-group (two cabinets per group)
rank 3: optical, inter-group (13 groups make up ARCHER)

Tests conducted on 2-cabinet *Test Development Server*

Private to EPCC, minimises resource contention.

ARCHER supports three programming environments

Cray (v8.3.7), Intel (v14.0.4) and gnu (v4.9.2) running on CLE v5.1 OS

Cray XC30 Power Management Counters

Supported counters obtained by running `papi_native_avail` on compute node.

Running Average Power Limit Counters	Power Management Counters
PACKAGE_ENERGY (nJ)	PM_POWER:NODE (W)
DRAM_ENERGY (nJ)	PM_ENERGY:NODE (J)
PP0_ENERGY (nJ)	PM_FRESHNESS

PACKAGE = processor (two sets of RAPL counters per node)

RAPL **instantaneous**, PM energy **cumulative**

PM Library

(https://cug.org/proceedings/cug2014_proceedings/includes/files/pap136.pdf)

Hart et al. [3] have provided a library that allows one to monitor the PM counters directly

(`/sys/cray/pm_counters`)

Counter files updated every 100 ms.

Measurements cover CPU, memory and any other hardware contained on the processor daughter card.

Consumption due to the Aries network controllers and beyond is excluded however.

PM MPI Library

(https://github.com/cresta-eu/pm_mpi_lib)

Only one MPI process per node must read the PM counter file on that node.

Only one MPI process (e.g., rank 0) should collate the data, writing it to a single file.

```
CALL pm_mpi_open(out_fn)
DO i=1,nstep
  …
  CALL pm_mpi_monitor(i,1)
  …
  CALL pm_mpi_monitor(i,2)
  …
ENDDO
CALL pm_mpi_close()
```

Minimal but flexible instrumentation.

pm_mpi_open(char* out_fn)

Call MPI **get processor name** to determine unique number of the node on which calling process is running.

Do MPI **comm split** on the node number, then MPI **all reduce** to determine process that has lowest rank on each node – this is the monitoring process.

The monitoring processes open their respective PM counter files.

All monitoring processes create another sub-communicator, one that unites them all, thus rank 0 can determine the number nodes in use.

pm_mpi_monitor(int nstep, int sstep)

Monitoring processes *only* read the counter files.

Subsequent MPI **all reduce** sums energy and power counters over all nodes.

Rank 0 writes counter data to output file.

Non-monitoring processes wait at MPI barrier.

pm_mpi_close()

Monitoring processes close PM counter files.

Rank 0 also closes performance output file.

Molecular Modelling Code I

DL_POLY v4.05 (MPI)
https://www.stfc.ac.uk/SCD/44516.aspx

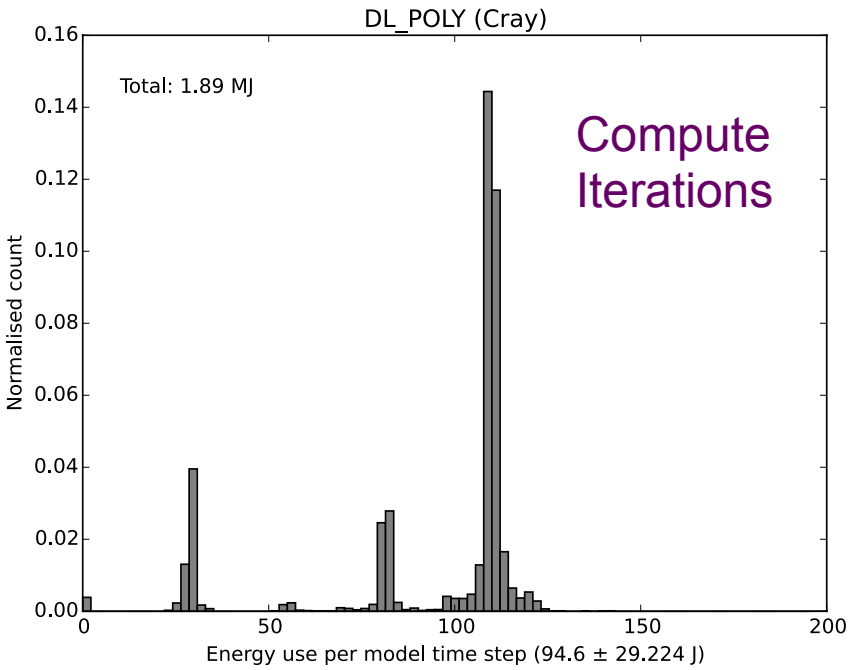
Test case 40 (ionic liquid dimethylimidazolium chloride) over four nodes (96 cores).

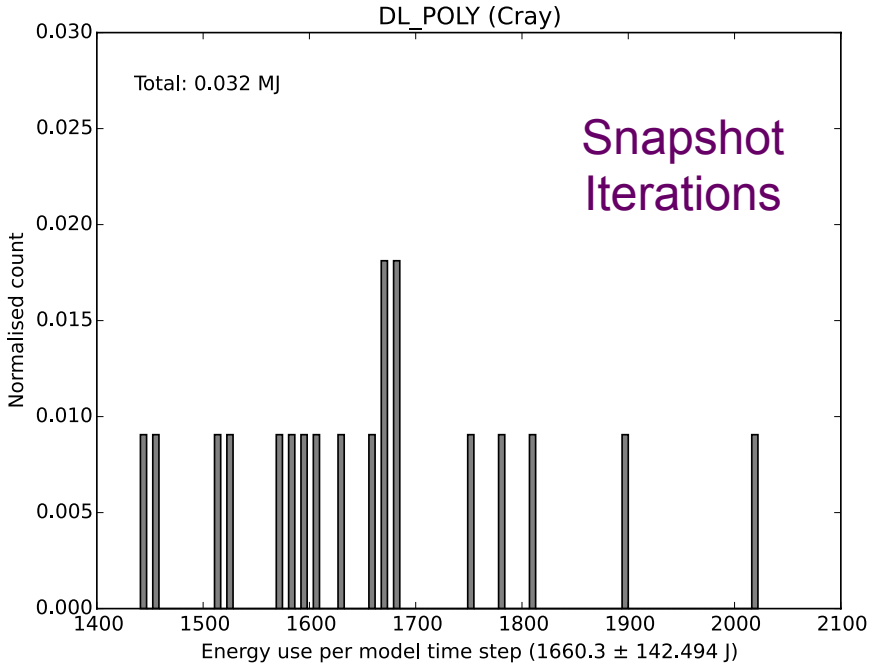
CONTROL steps = 20 000

Instrument main loop, `./VV/w_md_v.f90`

Perform six runs for each compiler environment.

Energy use per model time step (cray)





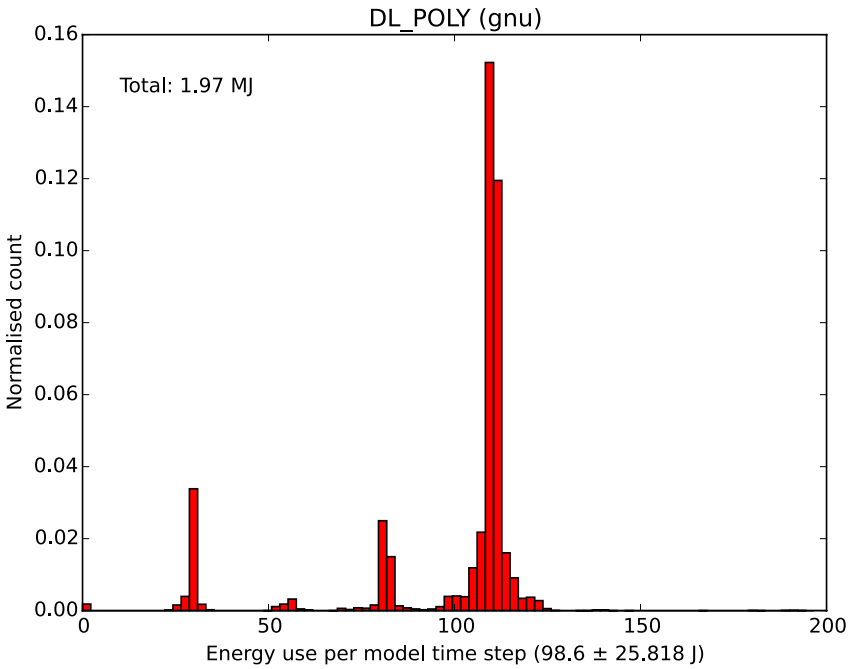
Every thousand iterations, DL_POLY restart files are written to disk – energy use increases by 16 times.

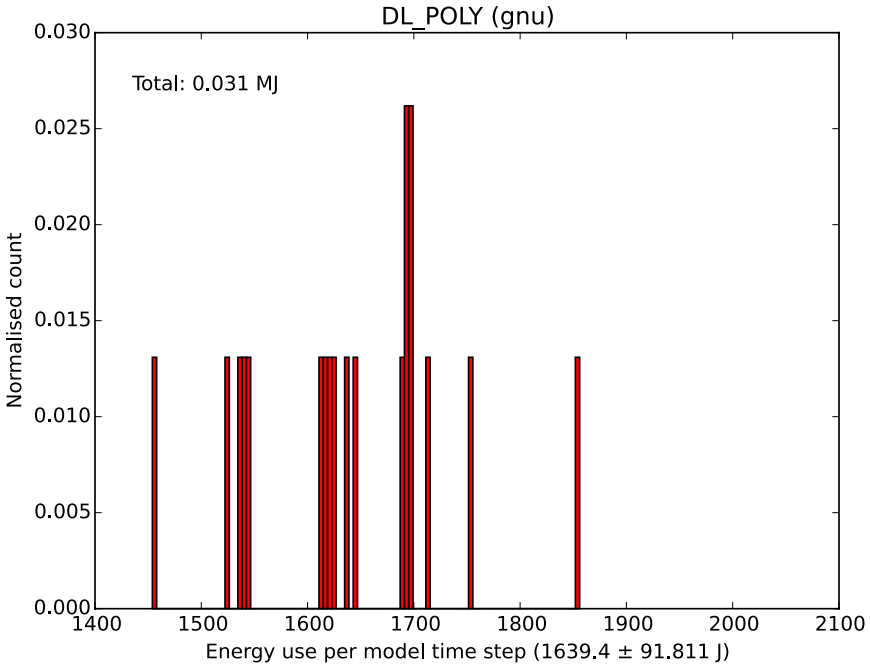
archer

epcc

THE UNIVERSITY OF EDINBURGH

Energy use per model time step (gnu)



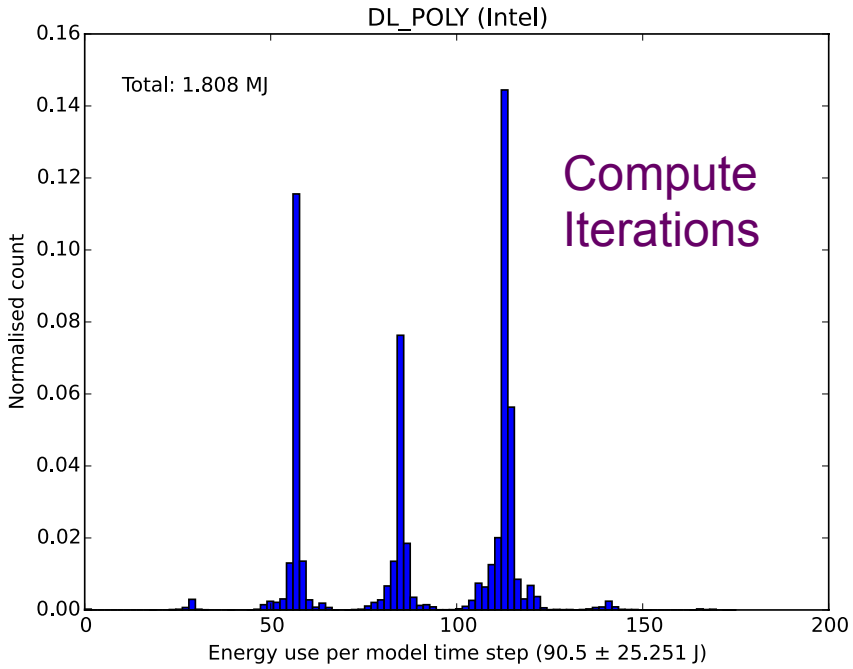


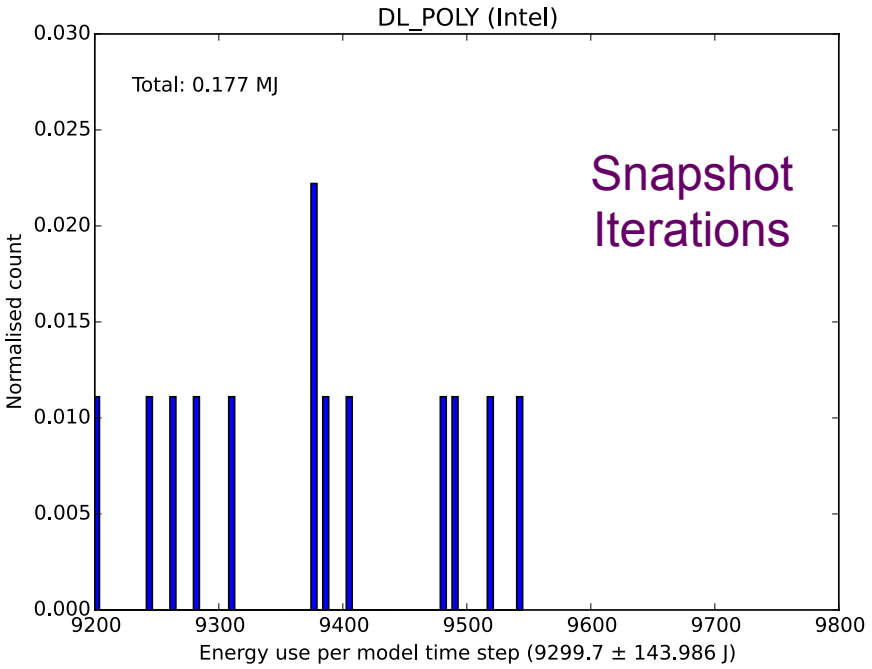
archer

epcc

THE UNIVERSITY OF EDINBURGH

Energy use per model time step (intel)





Increase now ≈ 95 times!

archer

epcc

THE UNIVERSITY OF EDINBURGH

Overall results

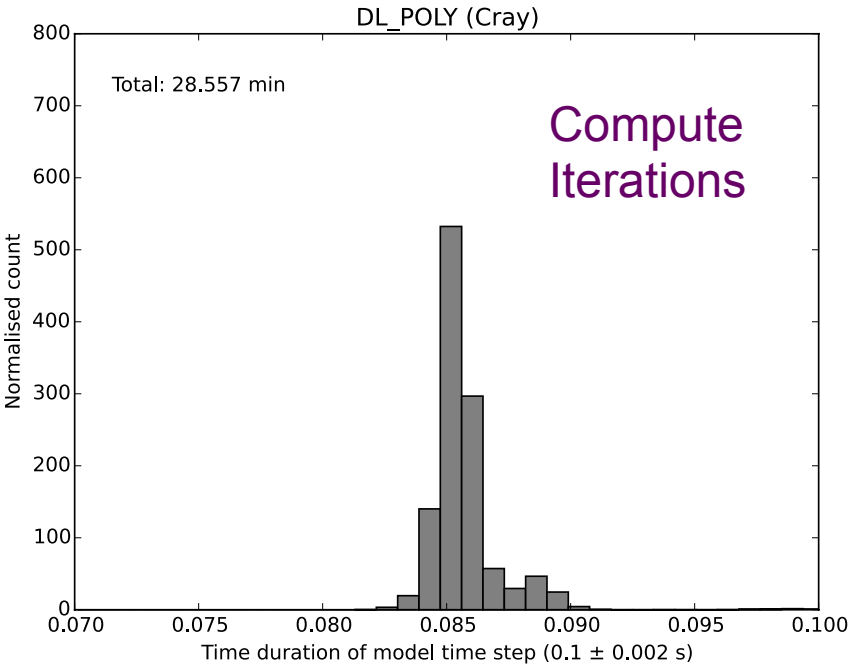
Six runs performed for each compiler environment.

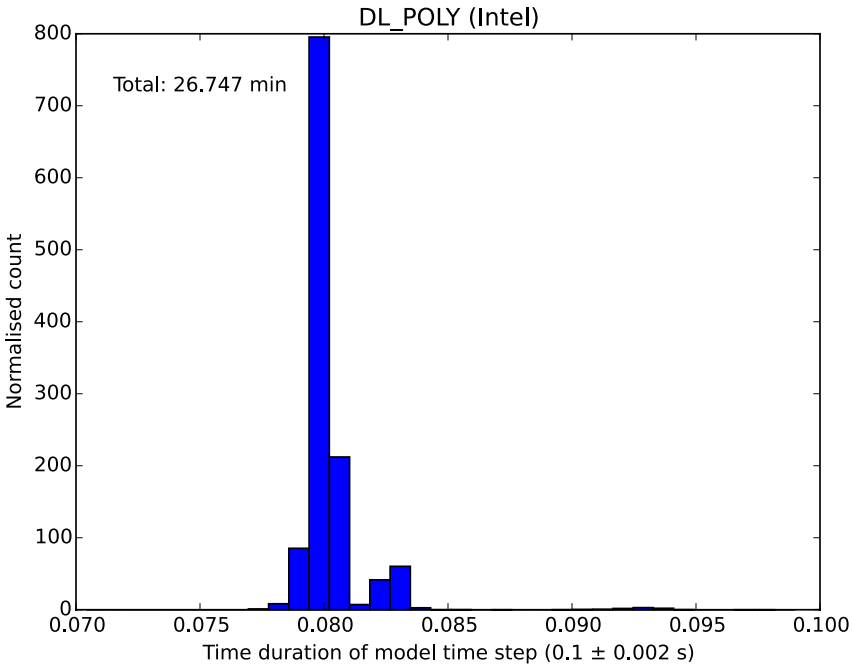
Cray: 1.92 ± 0.02 MJ over 1748 ± 2.6s

Intel: 1.97 ± 0.01MJ over 1770 ± 2.7s

gnu: 2 ± 0.02 MJ over 1823 ± 2 s

Time use per model time step (compute iterations)





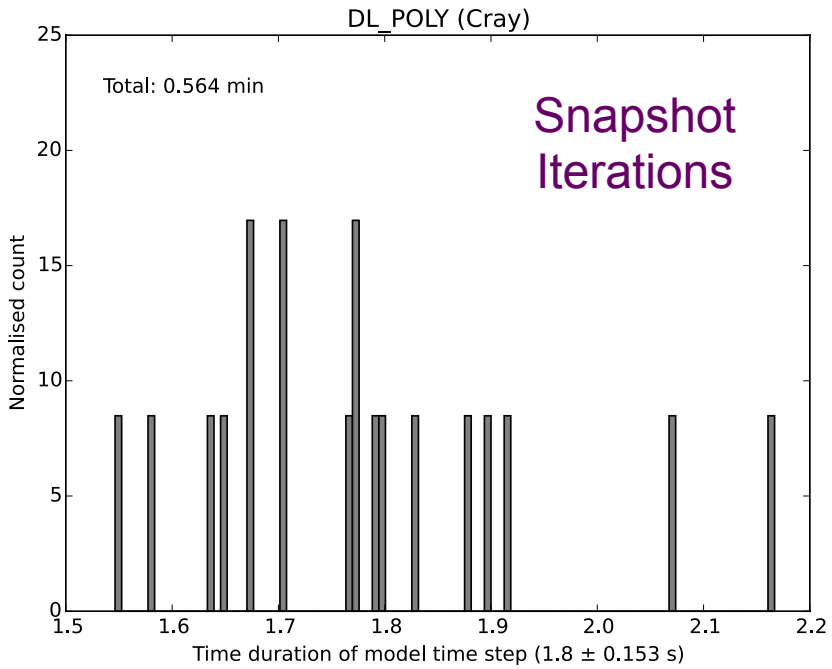
Majority of steps run faster for Intel Compared to Cray (and gnu).

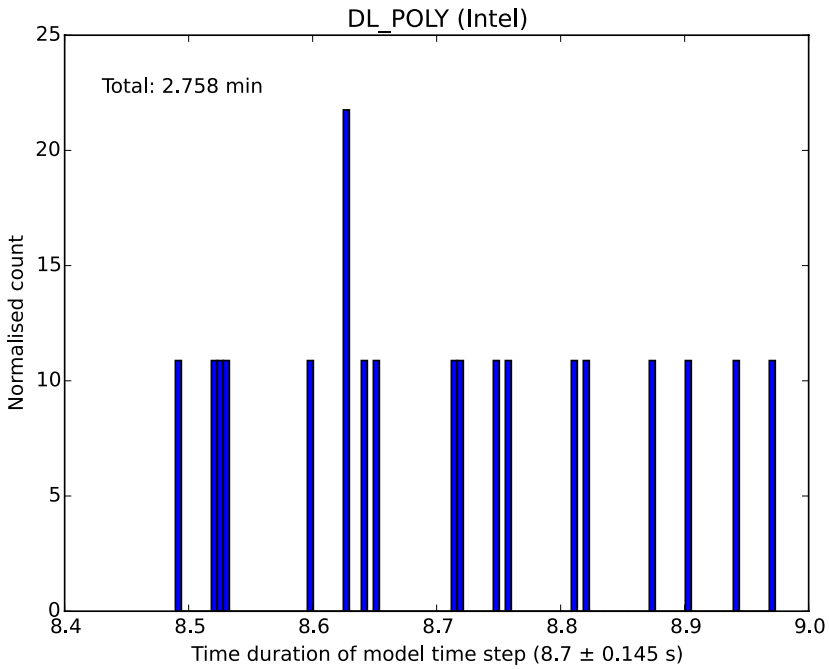
archer

epcc

THE UNIVERSITY OF EDINBURGH

Time use per model time step (snapshot iterations)





Snapshot iterations take significantly longer.

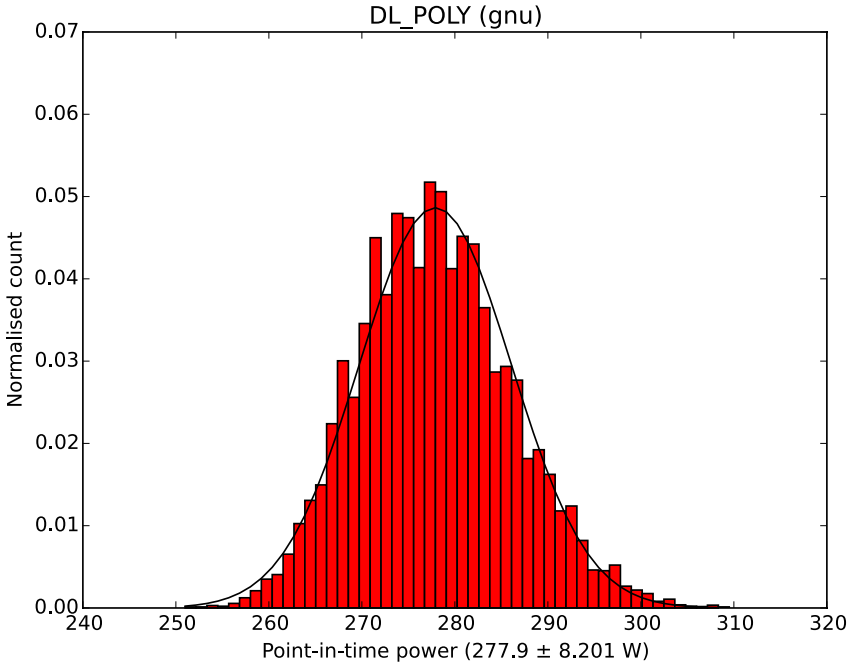
If the Intel snapshot iterations had runtimes comparable to the Cray and gnu results, the Intel compiled-code could be the most energy efficient.

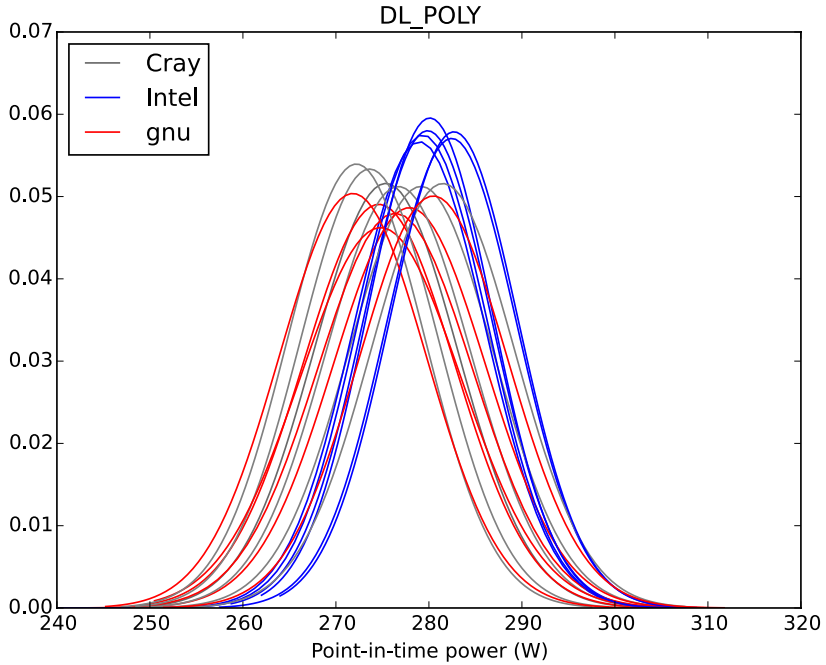
archer

epcc

THE UNIVERSITY OF EDINBURGH

Point-in-time Power Distributions





Distributions from all 18 simulations.

Intel runs draw slightly (~2%) more power.

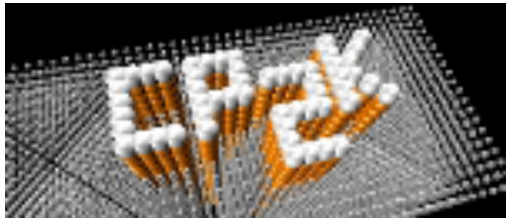
archer

epcc

THE UNIVERSITY OF EDINBURGH

Molecular Modelling Code II

CP2K v2.6.14482 (MPI/OpenMP)
http://www.cp2k.org



GNU programming environment only
./tests/QS/benchmark/H2O-1024.inp over eight nodes (192 cores)

MOTION.MD.STEPS = 100

`./src/motion/md_run.F, qs_mol_dyn_low()`
Real MD Loop

archer

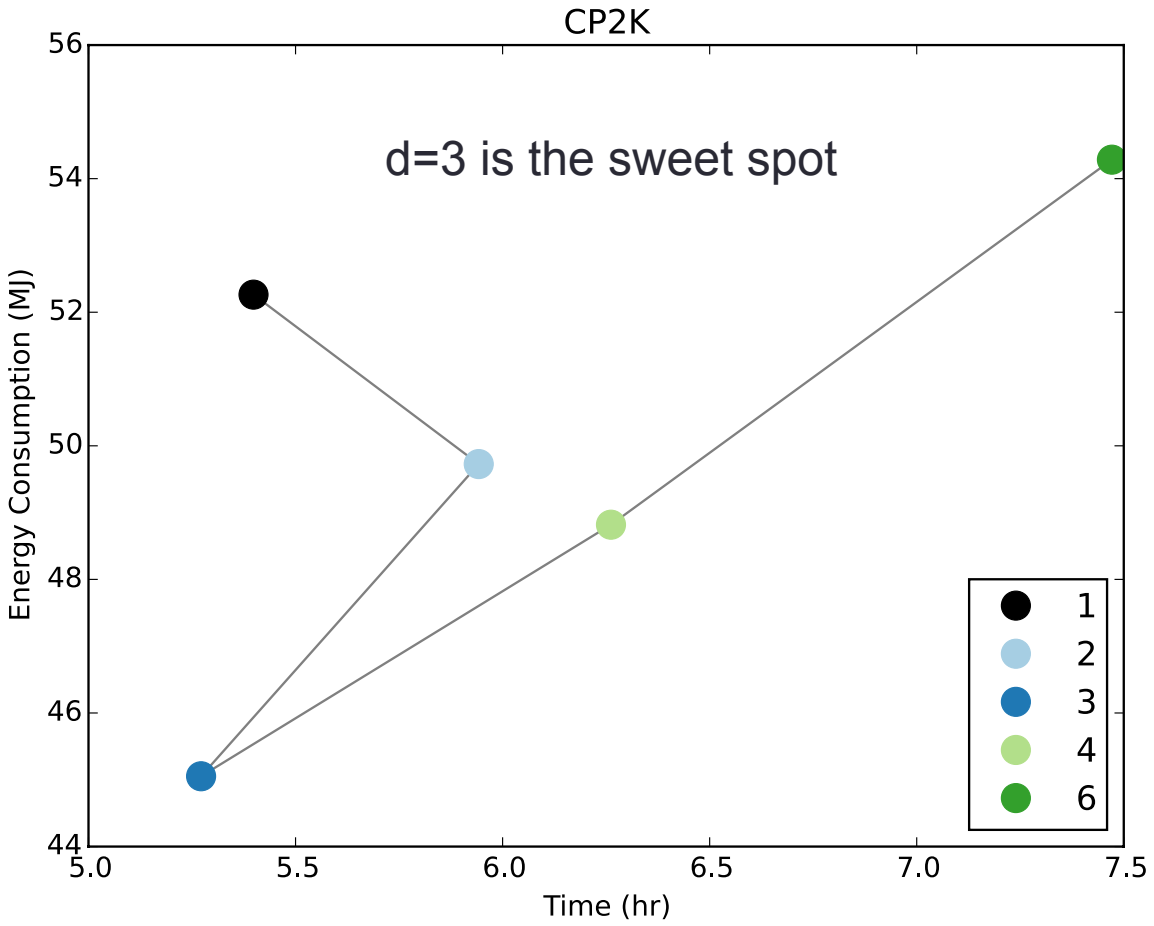
epcc

THE UNIVERSITY OF EDINBURGH

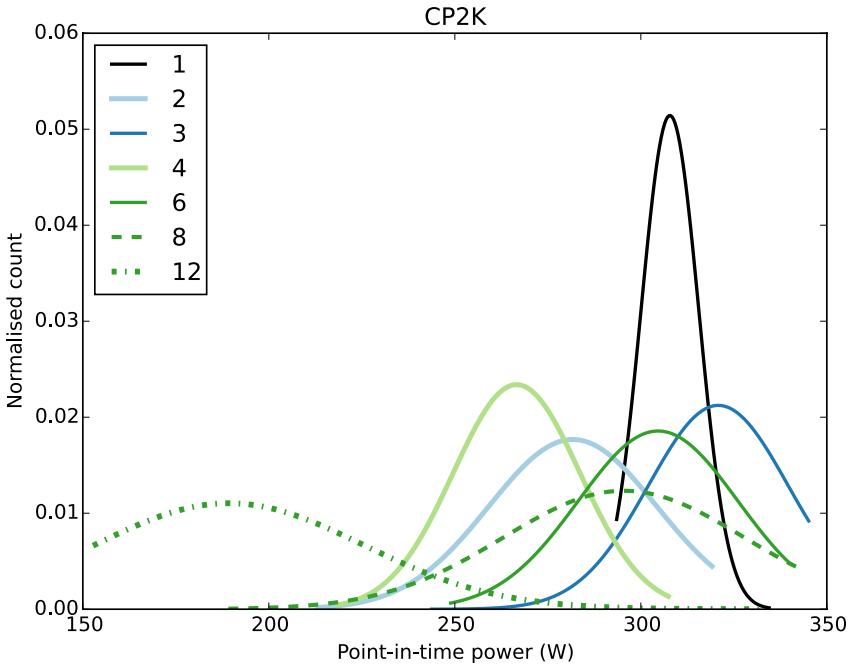
Cumulative energies and run times for different OpenMP threading options

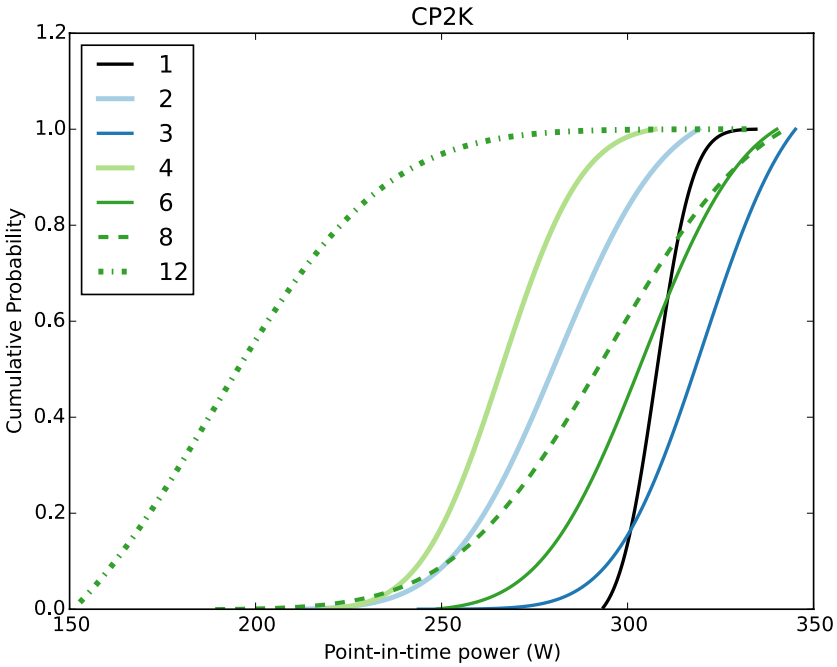
`aprun` options	Energy (MJ)	Run Time (hr)
-n 192 -N 24 -S 12 -d **1**	52.263	5.4
-n 96 -N 12 -S 6 -d **2**	49.727	5.94
-n 64 -N 8 -S 4 -d **3**	45.052	5.27
-n 48 -N 6 -S 3 -d **4**	48.819	6.26
-n 32 -N 4 -S 2 -d **6**	54.284	7.47
-n 24 -N 3 -d **8**	71.54	11.57
-n16 -N 2 -S 1 -d **12**	91.342	16.72

Energy usage against run time for different OpenMP threading options



Normal distributions and CDFs inferred from point-in-time power histograms





In general, power deviation increases with thread count.

archer

epcc

THE UNIVERSITY OF EDINBURGH

CrayPat Alternative (perftools module)

Instead accessing PM counter files directly it is possible to use CrayPat API calls.

```
CALL PAT_region_begin(id, label, istat)
IF (monitoring process) THEN
  CALL PAT_record(PAT_STATE_ON)
ELSE
  CALL PAT_record(PAT_STATE_OFF)
ENDIF
DO i=1,nstep
  …
  IF (monitoring process) THEN
    CALL PAT_counters(PAT_CTRS_PM, names, values)
  ENDIF
  …
ENDDO
CALL PAT_region_end(id)
```

Must load **`perftools`** module before compilation, then instrument exe with **`pat_build -w`** command.

Need to set **`PAT_RT_PERFCTR`** environment variable in job submission script.

Also tied to a particular counter category.

PAT MPI Library

(https://github.com/cresta-eu/pat_mpi_lib, coming soon)

pat_mpi_open(*char* out_fn*)

Monitoring processes turn PAT recording on.

And call `PAT_counters(cat[i],0,0,&nc)` for each counter category specified by MY_RT_CTRCAT environment variable.

Allocate memory required to hold counters.

pat_mpi_monitor(*int nstep, int sstep*)

Call `PAT_counters(cat[i],&name[j],&val[j],&nc)` for each counter category specified by MY_RT_CTRCAT, where the actual counter names are given by PAT_RT_PERFCTR.

Example Job Script

```
...
module load perftools
...
export PAT_RT_SUMMARY = 1
export MY_RT_CTRCAT = PAT_CTRS_RAPL, PAT_CTRS_PM
export PAT_RT_PERFCTR = PACKAGE_ENERGY, PP0_ENERGY, DRAM_ENERGY,
PM_POWER:NODE, PM_ENERGY:NODE
...
aprun –n 96 ./DL_POLY.z+pat >& stdouterr
```

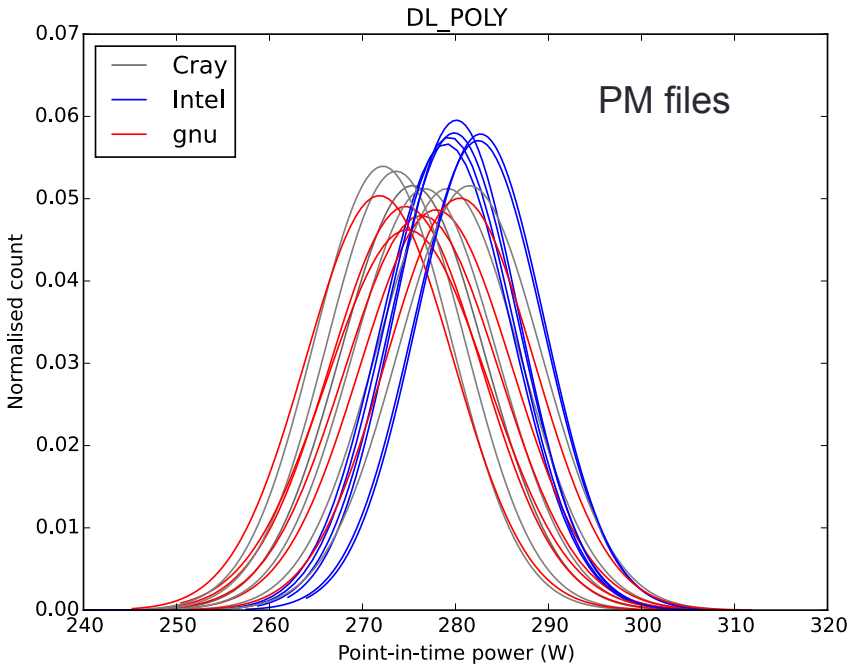
PM files vs PAT API

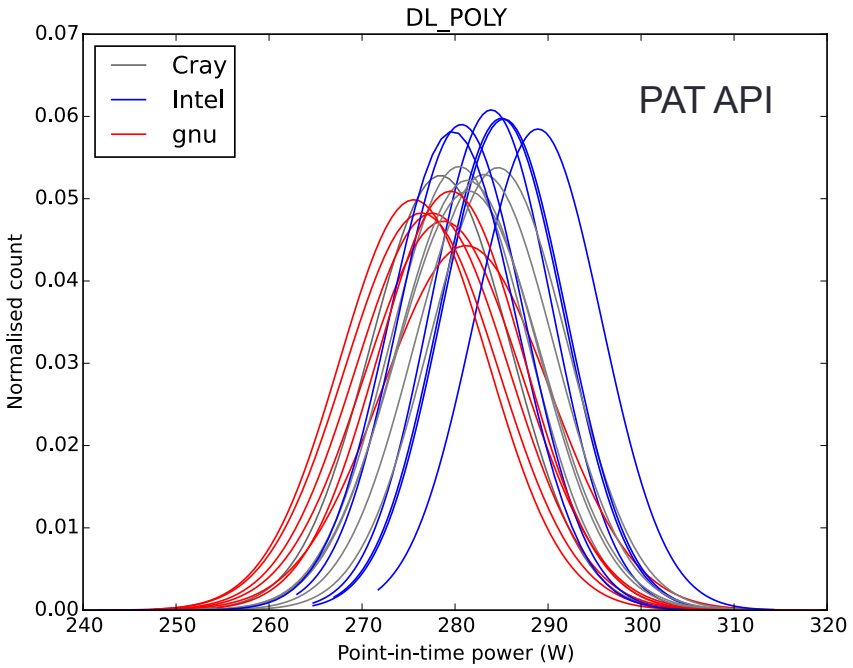
Average DL_POLY power consumption and runtimes for six runs per compiler environment.

	PAT API	PM Files
Cray	1.96 MJ (1747 s)	1.92 MJ (1748 s)
Intel	1.99 MJ (1762 s)	1.97 MJ (1770 s)
gnu	2 MJ (1819 s)	2 MJ (1823 s)

Higher Cray energy due to different node assignment.

PAT API Comparison



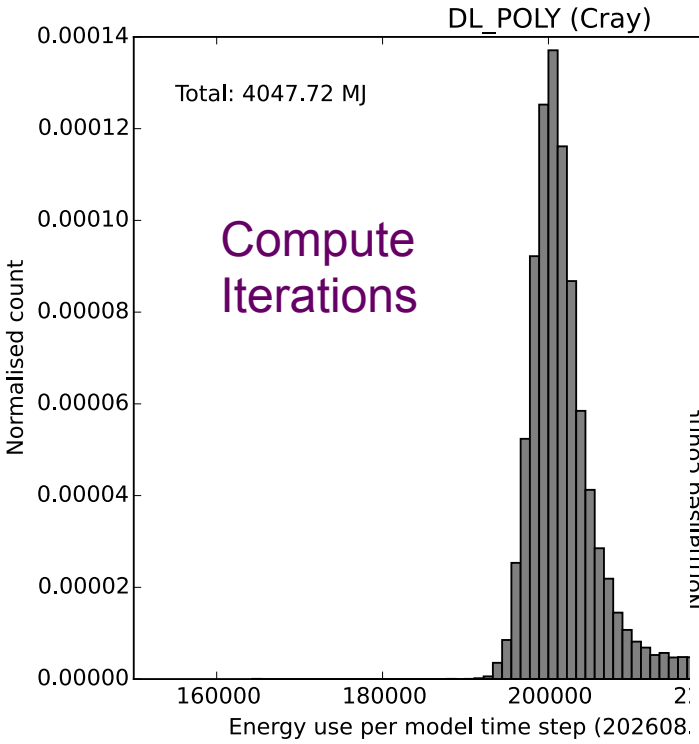


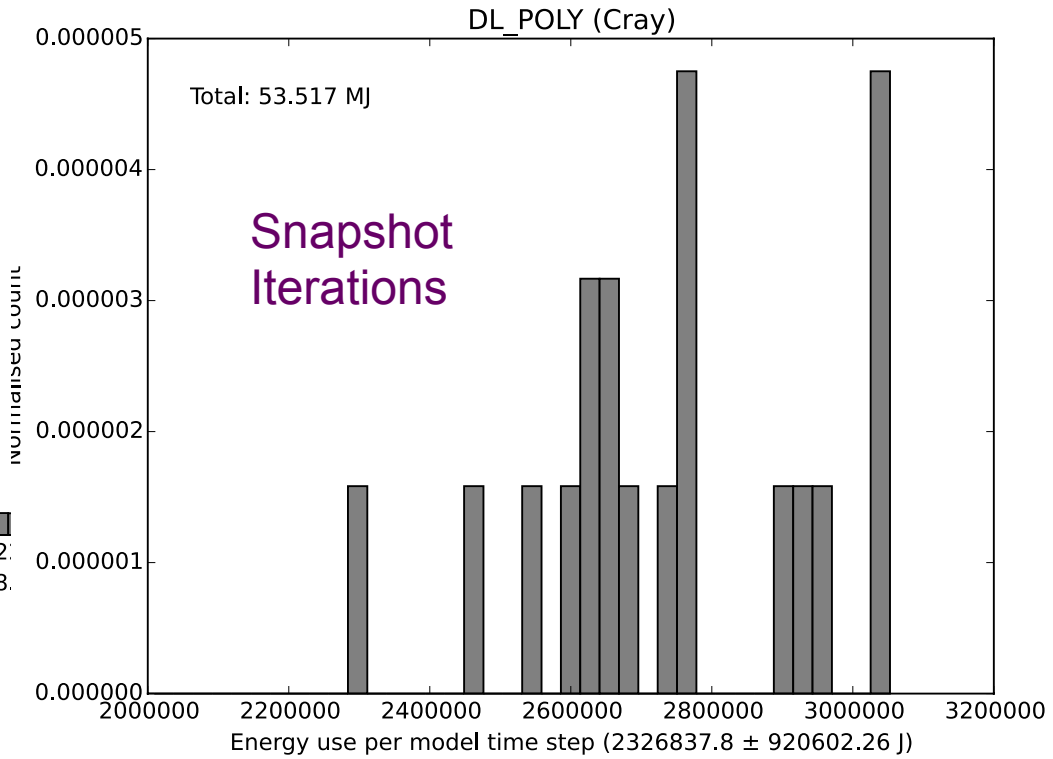
archer

epcc

THE UNIVERSITY OF EDINBURGH

PAT API DRAM Energy (Cray 1/6)



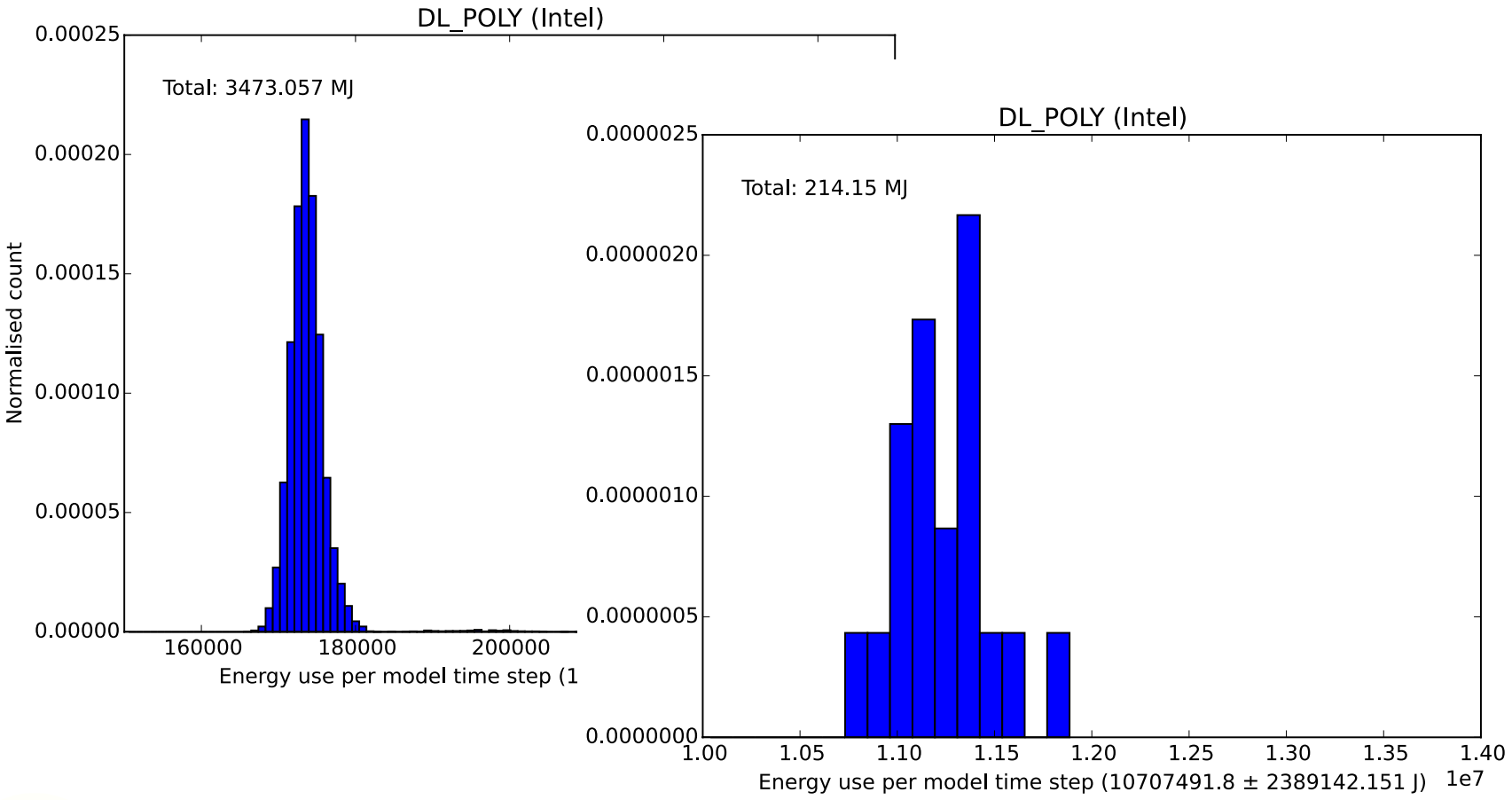


archer

epcc

THE UNIVERSITY OF EDINBURGH

PAT API DRAM Energy (Intel 4/6)



archer

epcc

THE UNIVERSITY OF EDINBURGH

Conclusions I

DL POLY results show the expected correlation between energy use and runtime.

Cray-compiled code uses the least energy, followed by Intel then gnu. Although differences are slight.

Closer examination of the data, reveals that the Intel runs *might* use the least energy, if the compiler options could be set such that the Intel snapshot iterations had runtimes comparable with the Cray and gnu results.

Conclusions II

Energy use will depend on the number of threads per MPI process: using multiple threads can reduce runtimes and energy usage but not beyond a certain thread count.

Three threads is the **optimum thread count** for CP2K running over eight nodes with the H2O-1024.inp data set.

Further work could investigate the importance of node assignment within the ARCHER dragonfly topology as regards energy consumption.

Further work (CP2K)

Running with three threads per MPI process, one could compare energy usages for the following scenarios.

1) All eight nodes from the **same chassis**.
2) Four nodes from one chassis and four nodes from a **different chassis**.
3) Same as scenario two but involving a chassis from a **different group**.

The usefulness of this work would be in understanding the energy cost of communicating via the rank 2 and/or rank 3 networks.