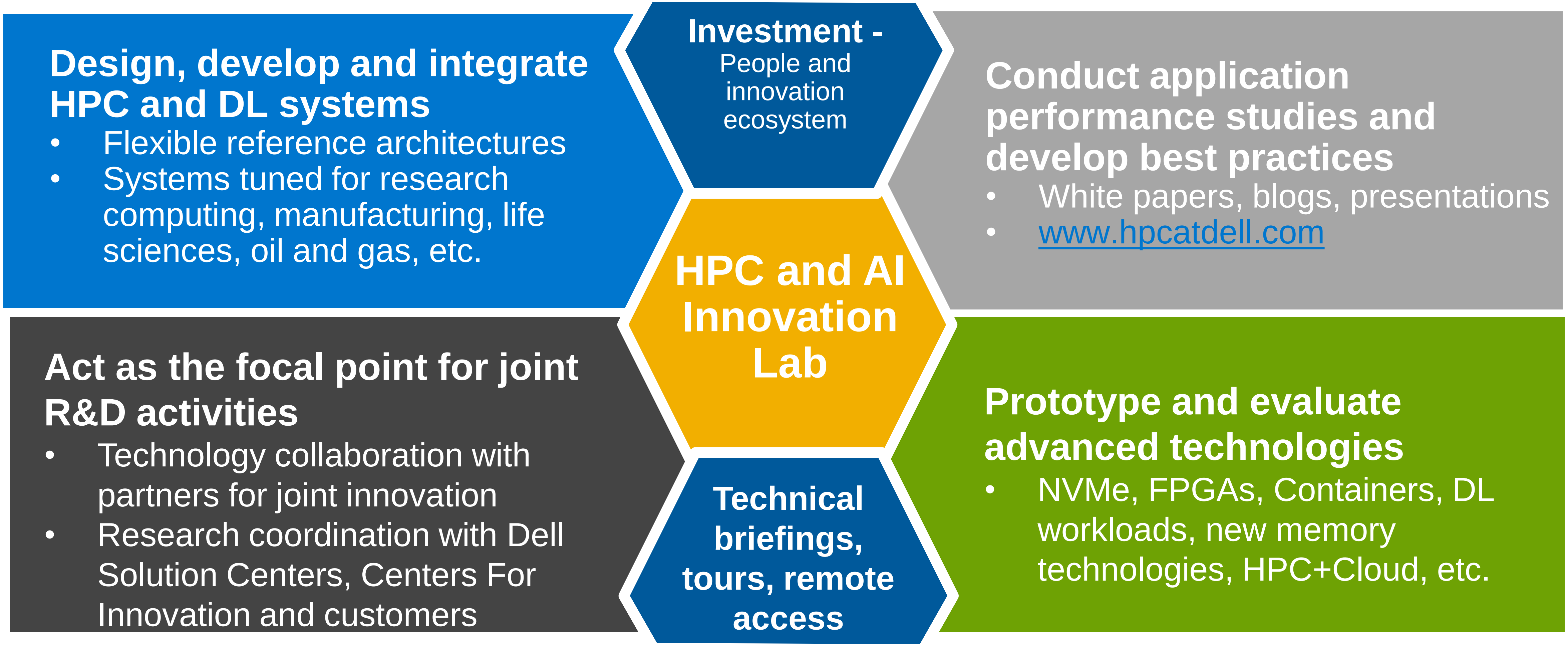
MPI Power Savings

HPC Platform Efficiency and Power Savings for Large-scale Workloads

Martin Hilgeman
Distinguished Member of Technical Staff
HPC Performance Lead
Martin.Hilgeman@dell.com

DELL Technologies

Dell Technologies HPC and AI Team Charter



HPC and AI Innovation Lab

Design, develop and integrate HPC and DL systems

- Flexible reference architectures
- Systems tuned for research computing, manufacturing, life sciences, oil and gas, etc.

Investment - People and innovation ecosystem

Conduct application performance studies and develop best practices

- White papers, blogs, presentations
- www.hpcatdell.com

Act as the focal point for joint R&D activities

- Technology collaboration with partners for joint innovation
- Research coordination with Dell Solution Centers, Centers For Innovation and customers

Technical briefings, tours, remote access

Prototype and evaluate advanced technologies

- NVMe, FPGAs, Containers, DL workloads, new memory technologies, HPC+Cloud, etc.

World-class infrastructure in the Innovation Lab

13K ft.^{2} lab, 1,300+ servers, ~10PB storage dedicated to HPC in collaboration with the community

Zenith

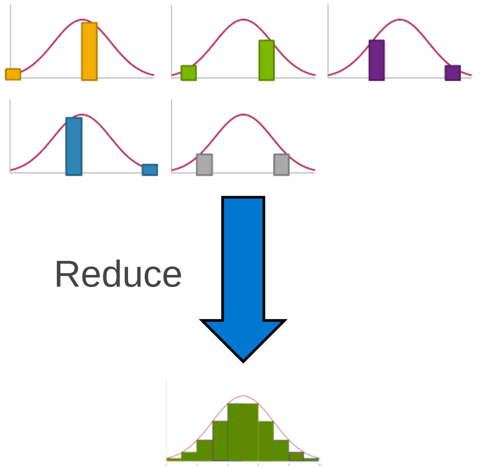
- Dell PowerEdge C6620 based on Intel Scalable processors
- Liquid cooled and air cooled

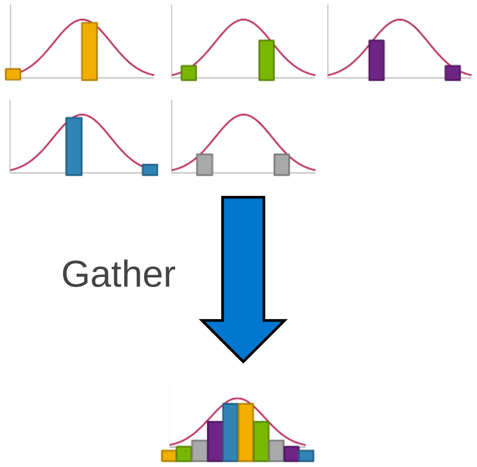
Minerva

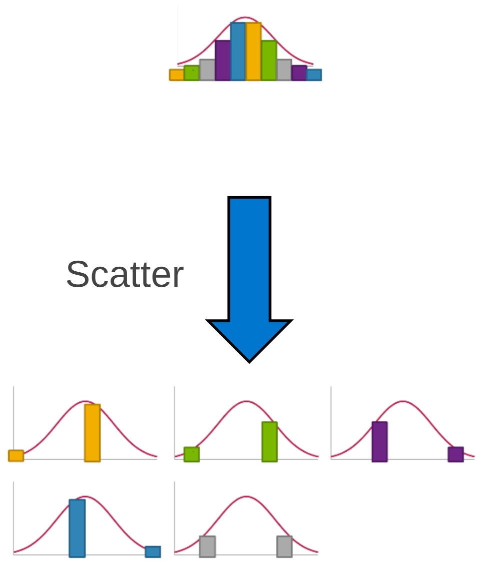
- Dell PowerEdge R6625 based on AMD EPYC processors
- Liquid cooled and air cooled

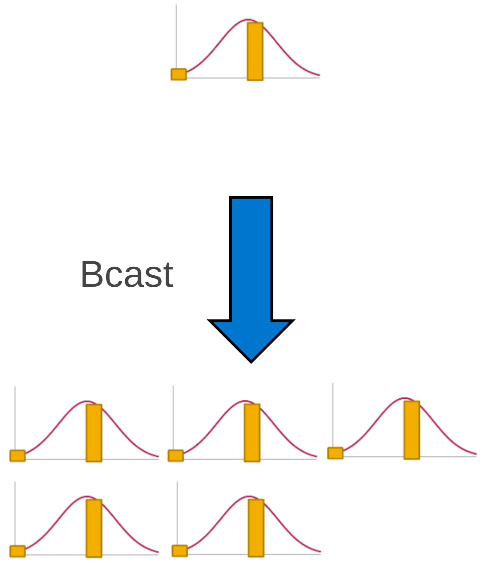
Load imbalance and MPI collectives

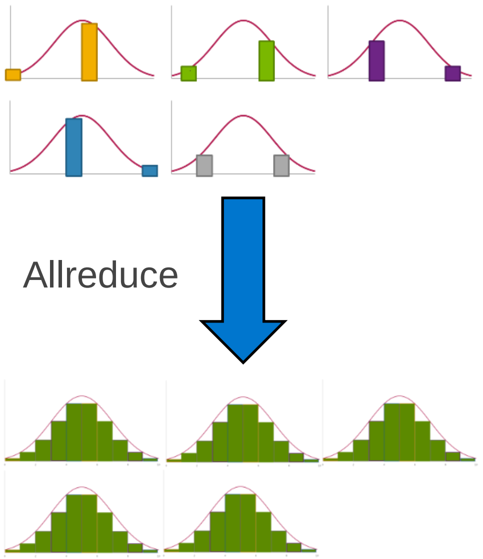
Collective MPI functions

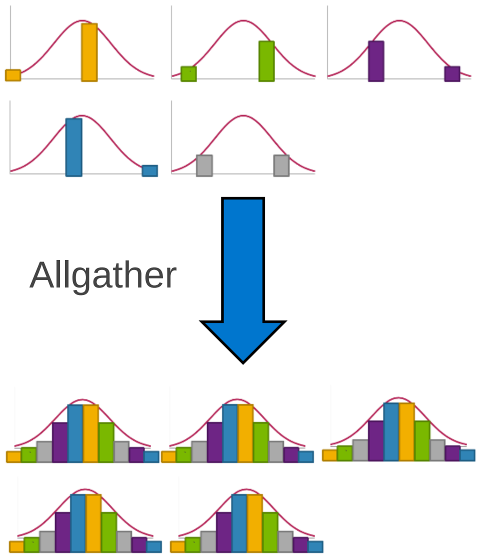


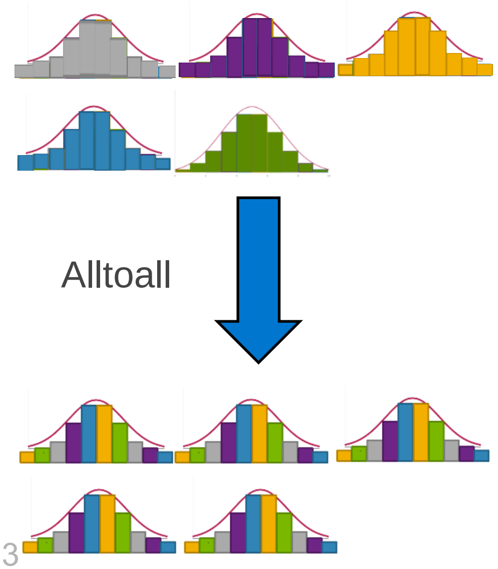




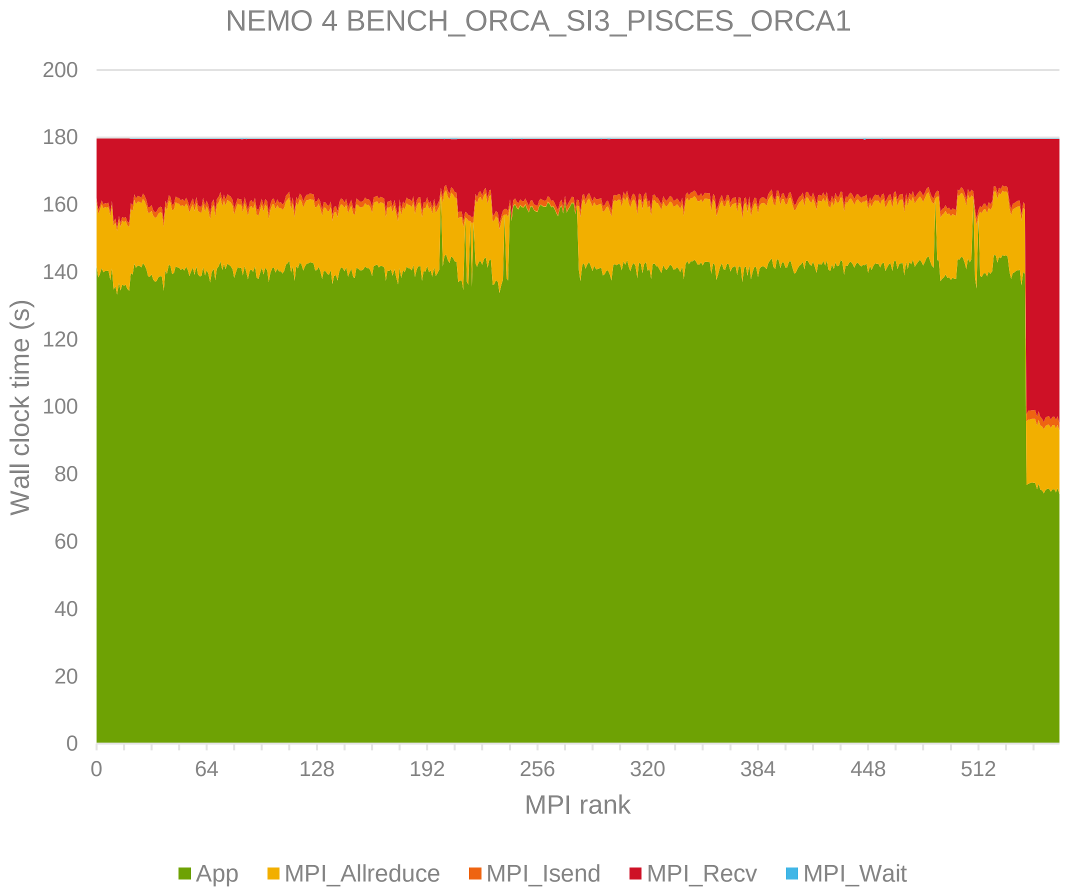








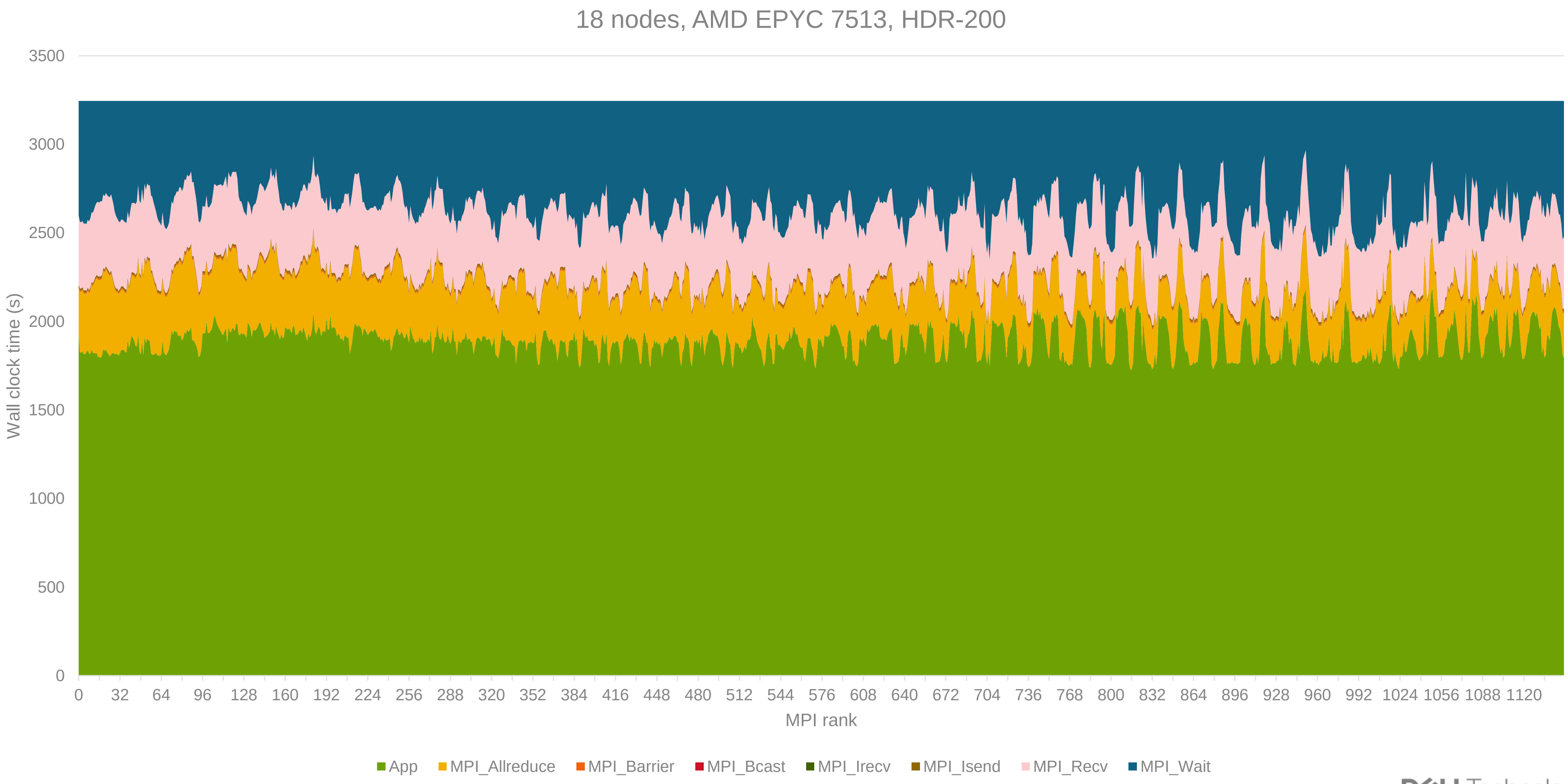
Collective MPI function rationale



- Collective functions spend a lot of time “fixing“ load imbalance

- MPI ranks spend different amounts of time in MPI_Allreduce

MOM5: load imbalance inhibits parallel scaling

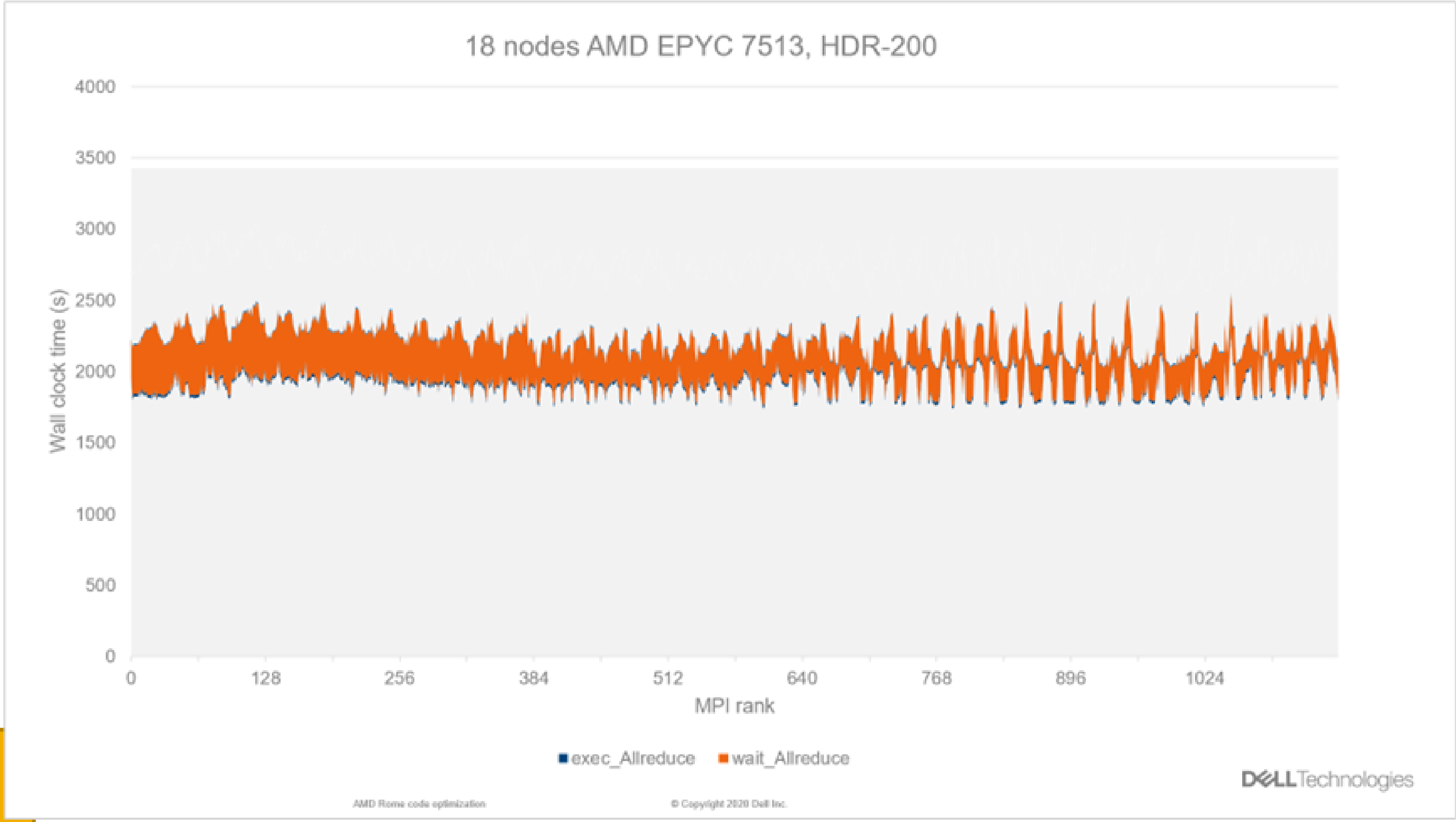


Result of perfect interconnect simulation

- Wait_*: load imbalance
- Exec_*: actual communication

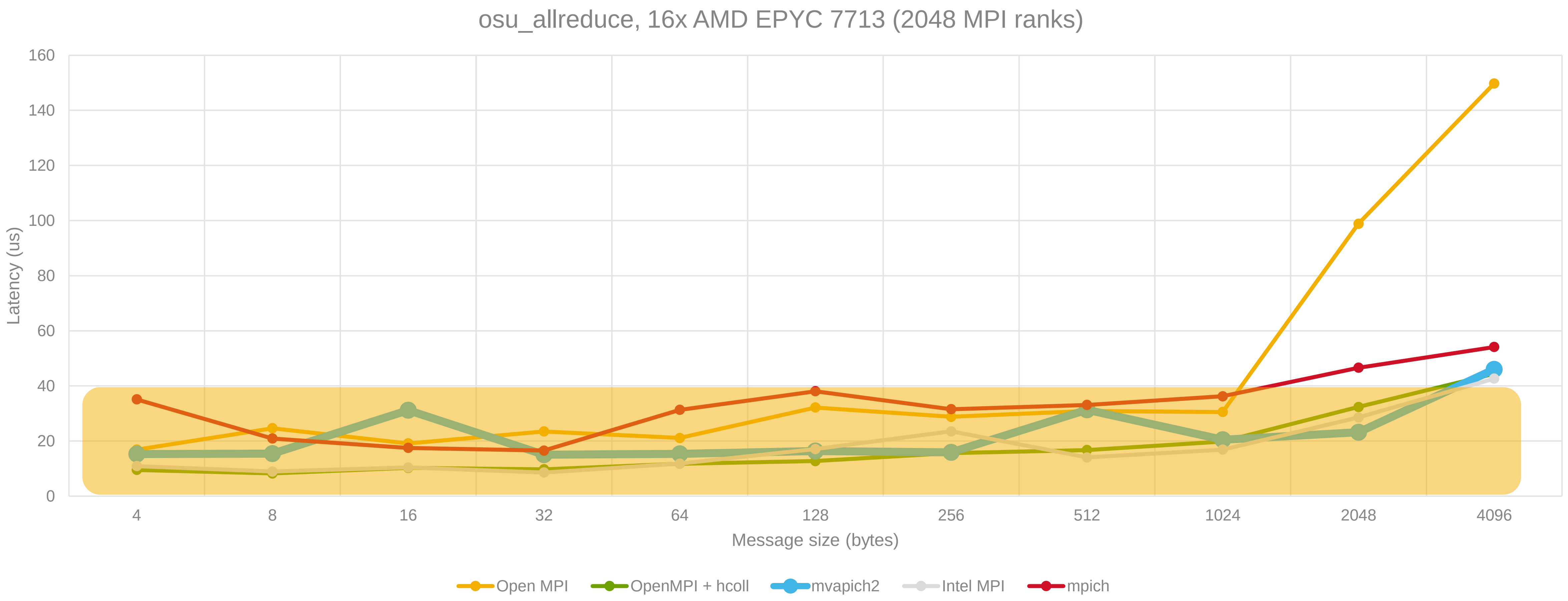
- Communication overhead is negligible

- Load imbalance is the cause of parallel overhead and potential scaling issue



123k calls, 119k < 64 bytes
Minimum time: 191 seconds
Maximum time: 463 seconds
Transfer time: 18 seconds

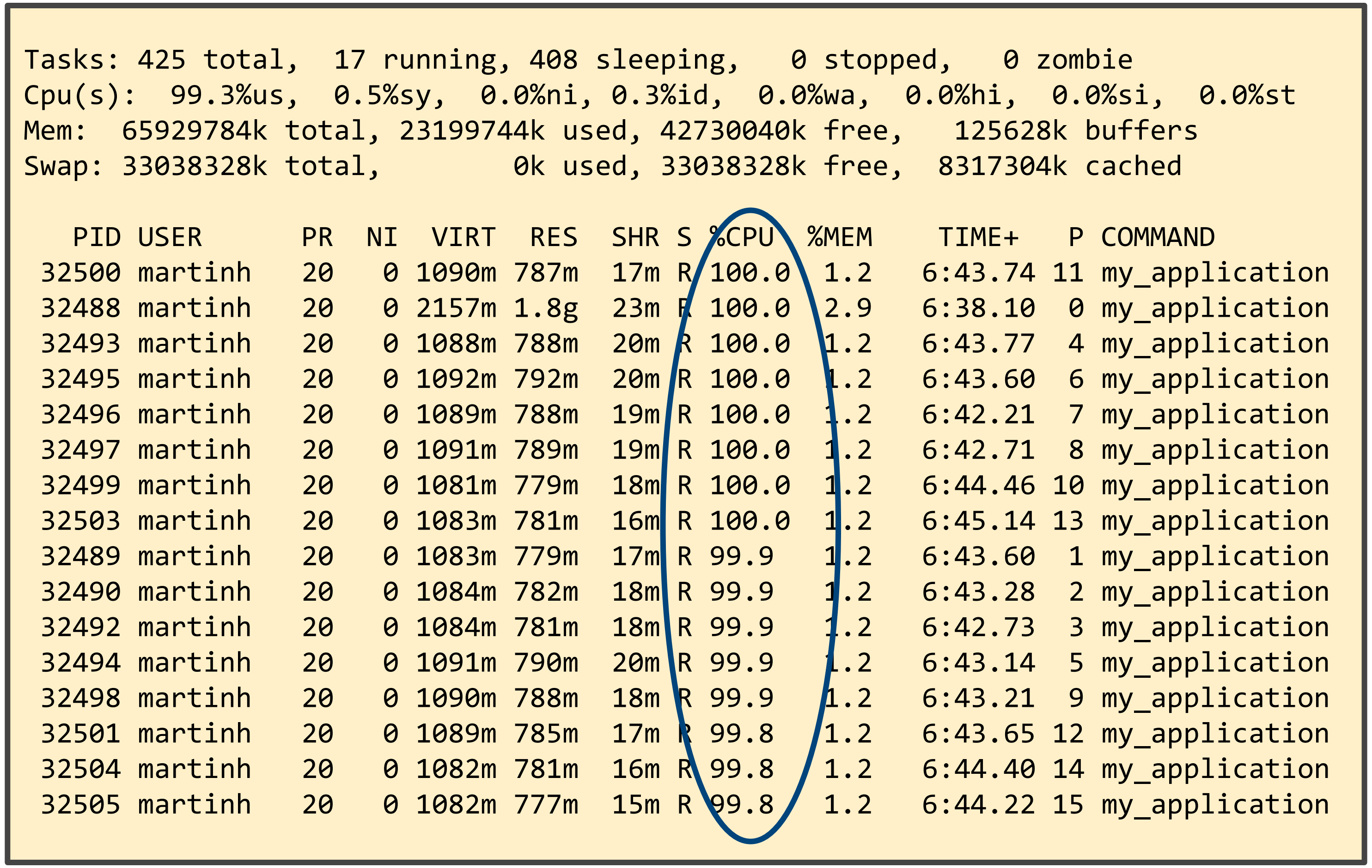
Allreduce for small message sizes



- Completion takes 20-40 us
- We have time to add a few microseconds and do something “smart”

MPI Power savings

The system load of my application



```
Tasks: 425 total,  17 running, 408 sleeping,   0 stopped,   0 zombie
Cpu(s):  99.3%us,  0.5%sy,  0.0%ni, 0.3%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  65929784k total, 23199744k used, 42730040k free,   125628k buffers
Swap: 33038328k total,        0k used, 33038328k free,  8317304k cached

   PID USER      PR  NI  VIRT  RES  SHR S %CPU  %MEM    TIME+   P COMMAND
 32500 martinh   20   0 1090m 787m  17m R 100.0  1.2   6:43.74 11 my_application
 32488 martinh   20   0 2157m 1.8g  23m R 100.0  2.9   6:38.10  0 my_application
 32493 martinh   20   0 1088m 788m  20m R 100.0  1.2   6:43.77  4 my_application
 32495 martinh   20   0 1092m 792m  20m R 100.0  1.2   6:43.60  6 my_application
 32496 martinh   20   0 1089m 788m  19m R 100.0  1.2   6:42.21  7 my_application
 32497 martinh   20   0 1091m 789m  19m R 100.0  1.2   6:42.71  8 my_application
 32499 martinh   20   0 1081m 779m  18m R 100.0  1.2   6:44.46 10 my_application
 32503 martinh   20   0 1083m 781m  16m R 100.0  1.2   6:45.14 13 my_application
 32489 martinh   20   0 1083m 779m  17m R 99.9   1.2   6:43.60  1 my_application
 32490 martinh   20   0 1084m 782m  18m R 99.9   1.2   6:43.28  2 my_application
 32492 martinh   20   0 1084m 781m  18m R 99.9   1.2   6:42.73  3 my_application
 32494 martinh   20   0 1091m 790m  20m R 99.9   1.2   6:43.14  5 my_application
 32498 martinh   20   0 1090m 788m  18m R 99.9   1.2   6:43.21  9 my_application
 32501 martinh   20   0 1089m 785m  17m R 99.8   1.2   6:43.65 12 my_application
 32504 martinh   20   0 1082m 781m  16m R 99.8   1.2   6:44.40 14 my_application
 32505 martinh   20   0 1082m 777m  15m R 99.8   1.2   6:44.22 15 my_application
```

My application is 100% busy on all cores, but is it doing any useful work?

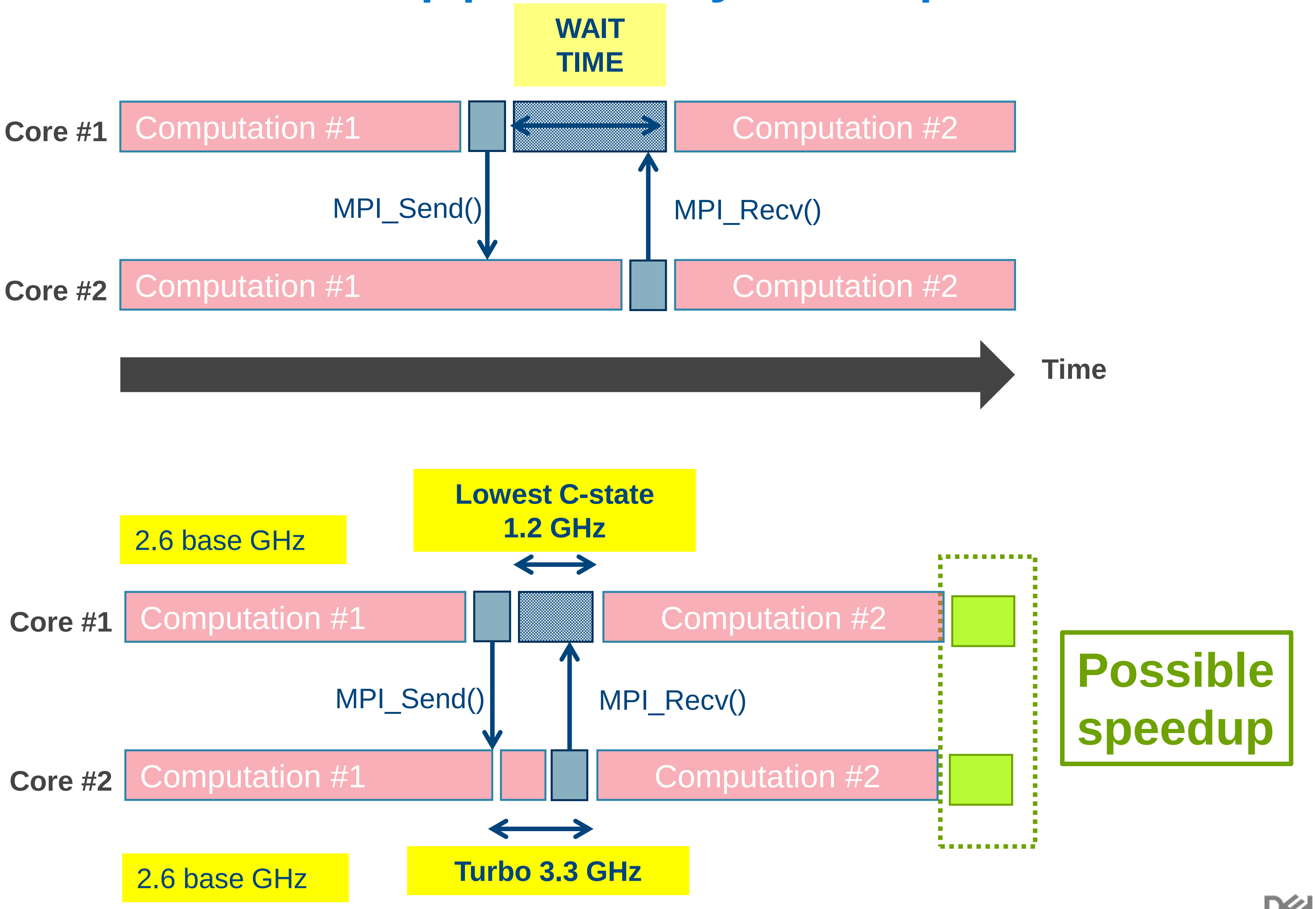
Actually…

```
 PID USER        PR  NI  VIRT  RES  SHR S %CPU  %MEM    TIME+   P COMMAND
32500 martinh    20   0 1090m 787m  17m R 100.0  1.2   6:43.74 11 my_application
```

CAN be doing (this is a debugger backtrace):

```
#4  opal_event_base_loop (base=0xe6b0ff0, flags=<value optimized out>) at event.c:855
#5  0x00007fcfd8623909 in opal_progress () at runtime/opal_progress.c:189
#6  0x00007fcfd8571f75 in opal_condition_wait (count=2, requests=0x7fff54a2ad70, statuses=0x7fff54a2ad40)
    at ../opal/threads/condition.h:99
#7  ompi_request_default_wait_all (count=2, requests=0x7fff54a2ad70, statuses=0x7fff54a2ad40) at request/req_wait.c:263
#8  0x00007fcfd45ae65e in ompi_coll_tuned_sendrecv_actual (sendbuf=0x0, scount=0, sdatatype=0x5e98fc0, dest=27, stag=-16,
    recvbuf=<value optimized out>, rcount=0, rdatatype=0x5e98fc0, source=27,
    rtag=-16, comm=0xe7945f0, status=0x0) at coll_tuned_util.c:54
#9  0x00007fcfd45b6a6e in ompi_coll_tuned_barrier_intra_recursivedoubling (comm=0xe7945f0, module=<value optimized out>) at
     coll_tuned_barrier.c:172
#10 0x00007fcfd857f282 in PMPI_Barrier (comm=0xe7945f0) at pbarrier.c:70
#11 0x00007fcfd88d6373 in mpi_barrier_f (comm=<value optimized out>, ierr=0x7fff54a2ae6c) at pbarrier_f.c:66
```

Load imbalance and opportunity for optimization



Idea – clock CPU cores down during communication

1. Intercept the MPI call – clock CPU down while waiting for the blocking Recv
2. Do the actual receive of the data
3. After the receive – reset CPU to nominal clock speed

```
int MPI_Recv(void *buf, int count, MPI_Datatype datatype,
             int source, int tag, MPI_Comm comm, MPI_Status *status)
{
    len = count * sizeof(datatype);

    if ( mpi.host_number[source] != mpi.host_number[mpi.rank] && len >= 16384 )
            ret = set_cpu_min_freq(task.cpu);

    ret = PMPI_Recv(buf, count, datatype, source, tag, comm, status);

    if ( mpi.host_number[source] != mpi.host_number[mpi.rank] && len >= 16384 )
            ret = set_cpu_max_freq(task.cpu);

    return ret;
}
```

CPU specific extensions that facilitate C-state selection

- Both AMD and Intel have released extensions to the x86_64 instruction set architecture (ISA) that can be used to force the CPU into a lower C-state when it is waiting for an event to complete
- This event can for example be a signal or a write to a memory address that is being monitored
- The use of both extensions are more or less the same, there are slightly different semantics

Intel: MONITOR/MWAIT and UMONITOR/UMWAIT

- Intel Nehalem and later contain the MONITOR and MWAIT instructions
  - MONITOR watches a store operation to a memory address to wake up the CPU
  - MWAIT puts the CPU into a lower C-state
  - MONITOR/MWAIT can only be executed in Ring 0
- Intel Alder Lake/Sapphire Rapids contain the UMONITOR/UMWAIT instructions
  - Same semantics as MONITOR/MWAIT
  - Can be called from any privilege level
  - UMWAIT instruction is controlled through the IA32_UMWAIT_CONTROL register, where bit 0 sets the C-state and bits 2:31 set the spin time in TSC cycles.

https://www.felixcloutier.com/x86/mwait

https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html

Intel: MONITOR/MWAIT and UMONITOR/UMWAIT

Bit	C-state
Bit[0] = 0	C0.2
Bit[0] = 1	C0.1
Bit[31:2]	Maximum time in P0 TSC clock ticks that processor is in C0.2 or C0.1. Maximum value is 2^{30} (10e9 cycles).

The UMONITOR/UMWAIT instructions can be enabled by calling the _umonitor and _umwait intrinsic functions

```
#include <immintrin.h>

#if defined __WAITPKG__
void _umonitor (void *a)
unsigned char _umwait (unsigned int ctrl, unsigned __int64
counter)
#endif
```

Enablement:

- GCC 12: `gcc -mwaitpkg file.c`

AMD: MONITORX/MWAITX

Similar to Intel, AMD supports similar instruction pairs: MONITORX and MWAITX

- Supported in 15h family and newer
- Semantics are slightly different
- Can be called from any privilege level

Register/bit	C-state
EAX[7:4]=1	C0
EAX[7:4]=0	C1
ECX[0]=1	Allow the CPU to wake up by an interrupt
ECX[1]=1	Maximum wait time in P0 TSC clock ticks that processor is in defined C-state. Time is set in ECX[31:2]

https://www.amd.com/en/support/tech-docs/amd64-architecture-programmers-manual-volumes-1-5 page 258

AMD: MONITORX/MWAITX

The MONITORX/MWAITX instructions can be enabled to calling the _mm_monitorx and _mm_mwaitx or __builtin_ia32_monitorx and __builtin_ia32_mwaitx intrinsic functions

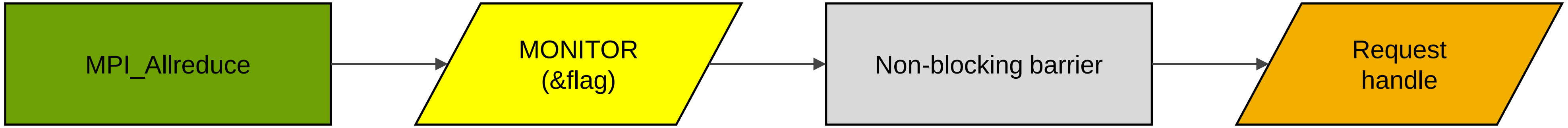
```
#include <immintrin.h>

#if defined __MWAITX__
void _mm_monitorx (void *a, unsigned int, unsigned int);
void _mm_mwaitx (unsigned int, unsigned int, unsigned int);

void __builtin_ia32_monitorx (void *a, unsigned int, unsigned int);
void __builtin_ia32_mwaitx (unsigned int, unsigned int, unsigned int);
#endif
```

Enablement:

- GCC 12: `gcc -mmwaitx file.c`

MWAIT with collective MPI functions

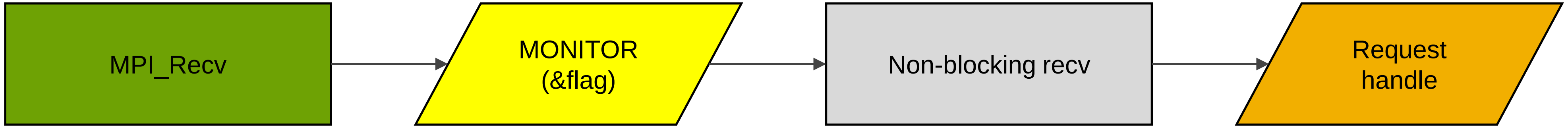
- Implementation: get a request handle and track completion



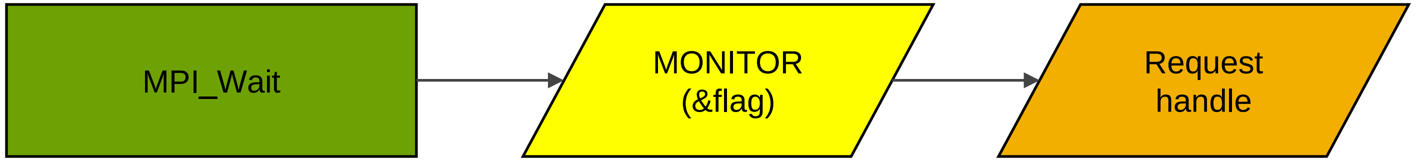
- Note: non-blocking barrier (MPI_Ibarrier) + blocking collective (MPI_Allreduce) is faster than non-blocking collective function (MPI_Iallreduce)

MWAIT with point-to-point or Wait/Waitall/Waitany

- MPI_Recv: get a request handle and track completion

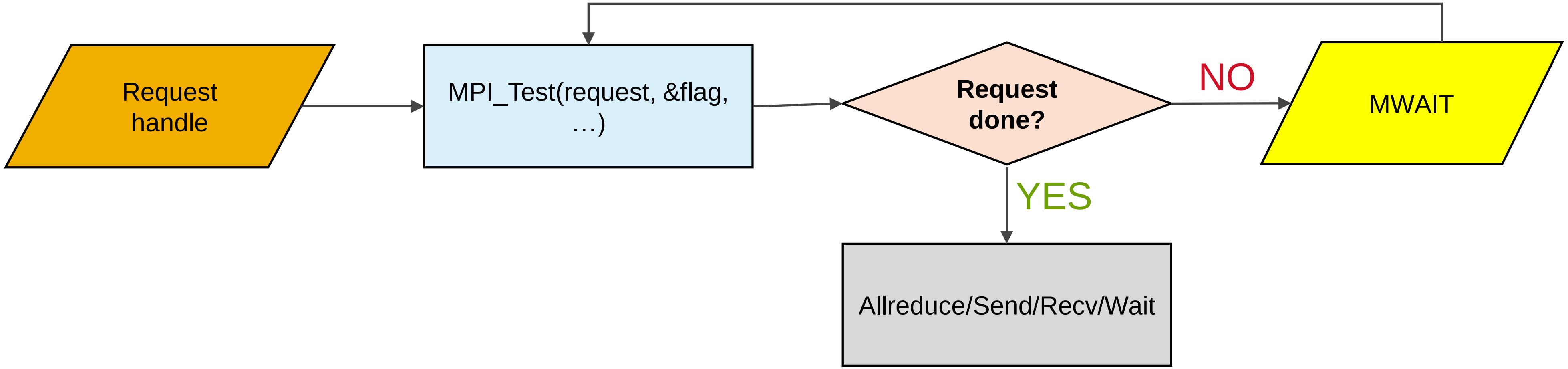


- MPI_Wait: request handle is already there, only track completion

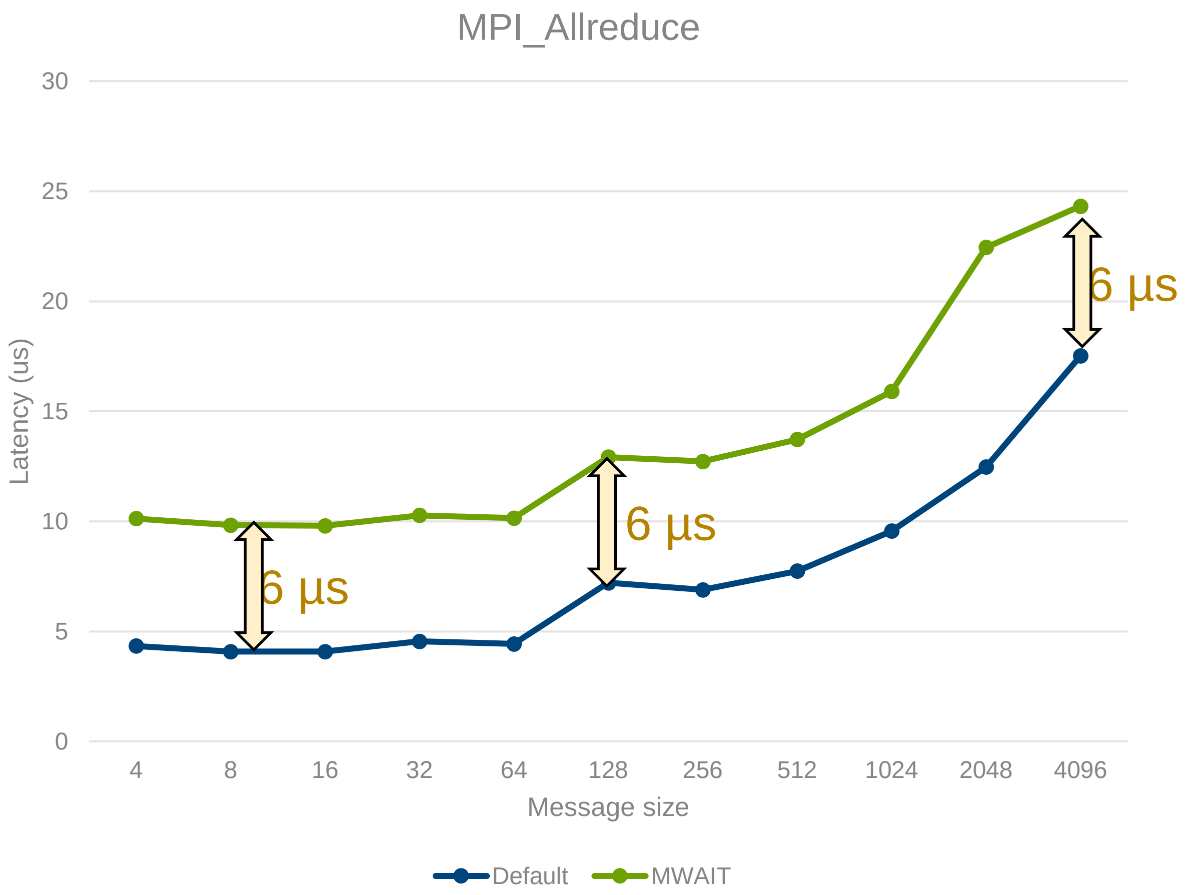


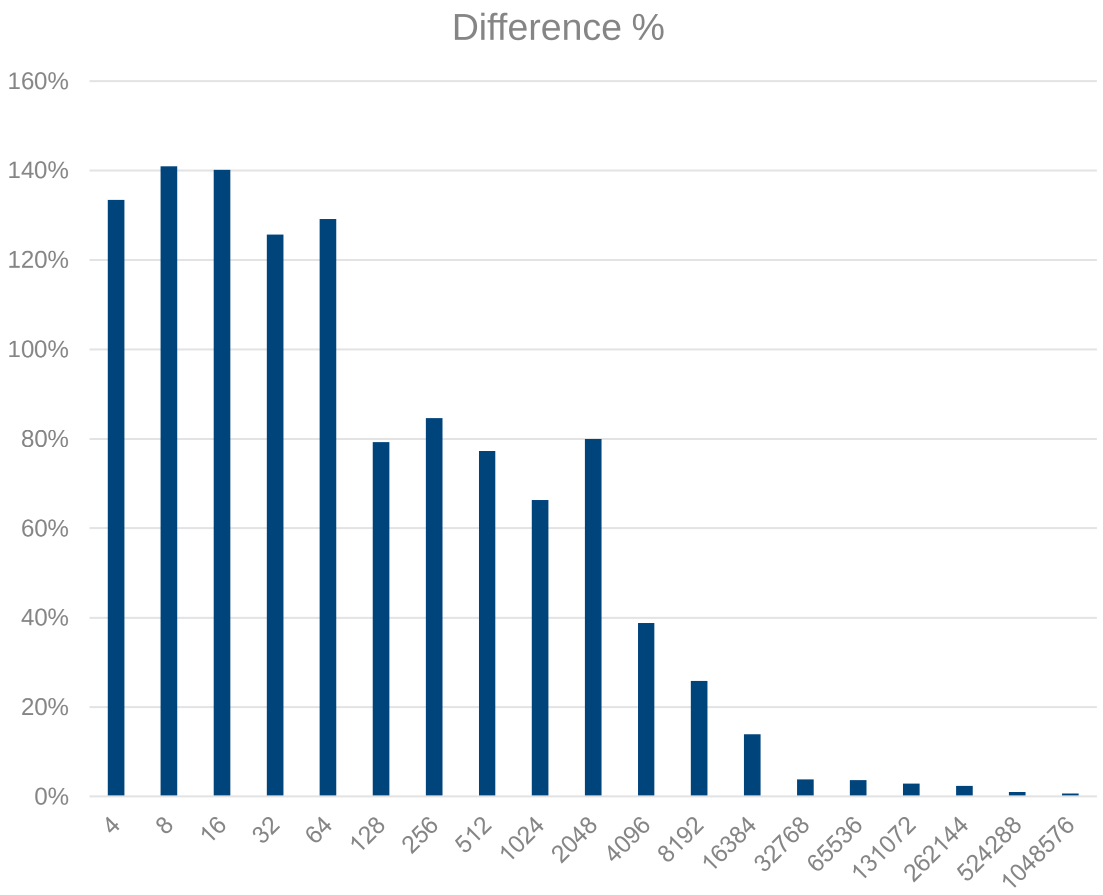
MWAIT checks completion of the request

- Implementation: use the request handle and test completion



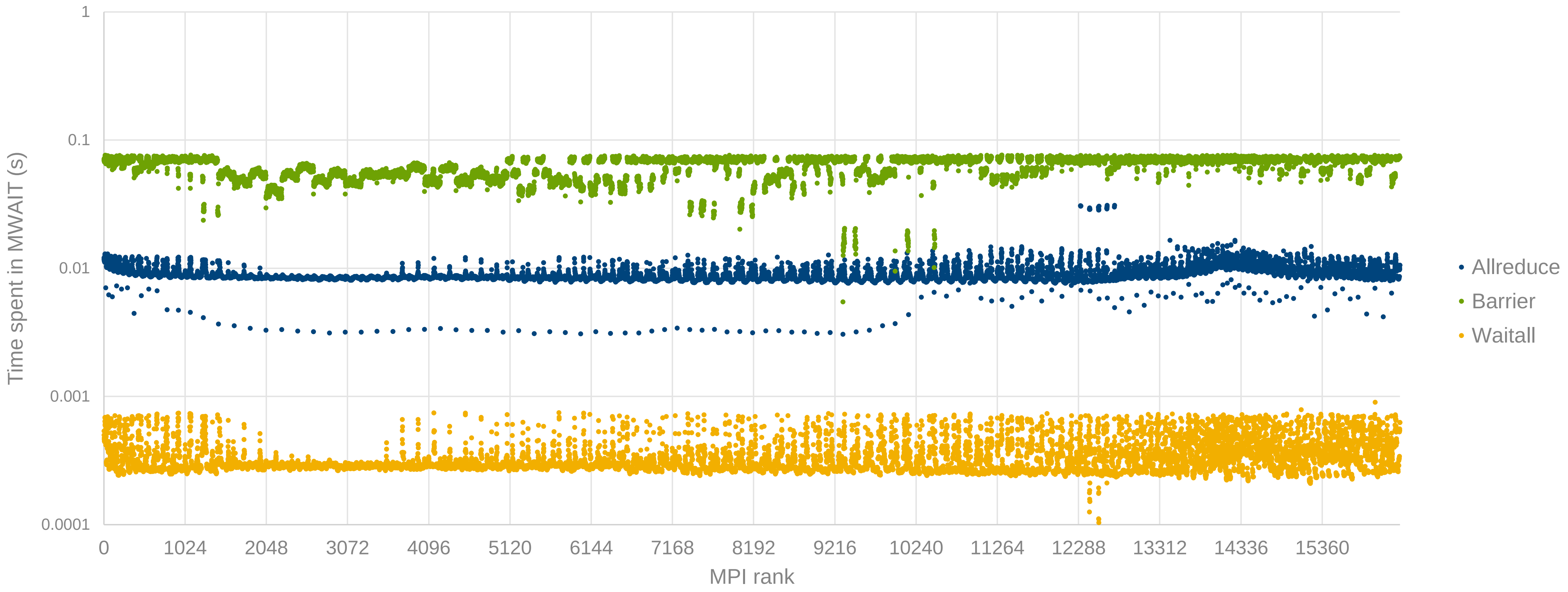
MWAIT call overhead





- Adding 6 µs to large messages is not going to hurt much
- Need to be careful with small messages

Average time spent in MWAIT



- Average time spent in MWAIT can be significant when there is load imbalance!

MPI power measurements

MPI Power measurements

Measure power for two MPI collectives

- Allreduce – 1024 kB message size
  - Commonly used
- Alltoall – 1024 kB message size
  - Most communication intensive

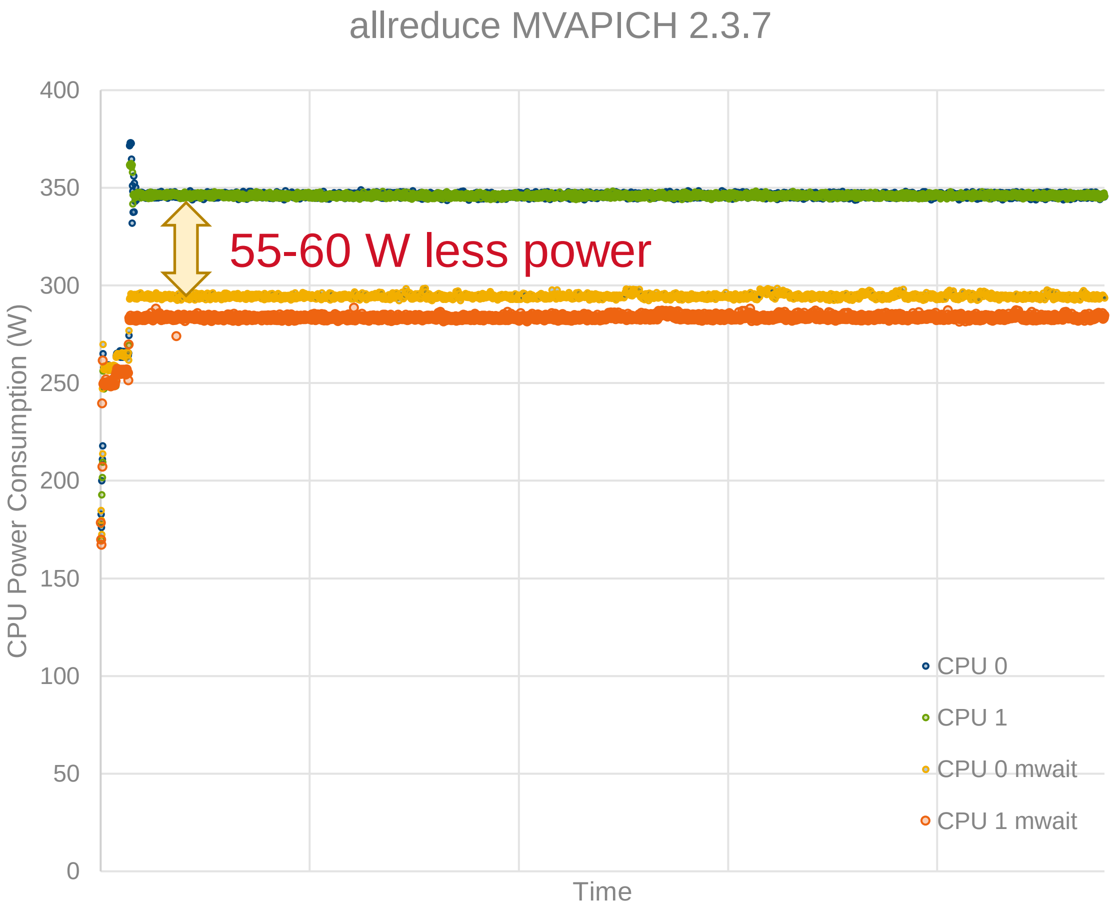
Systems tested

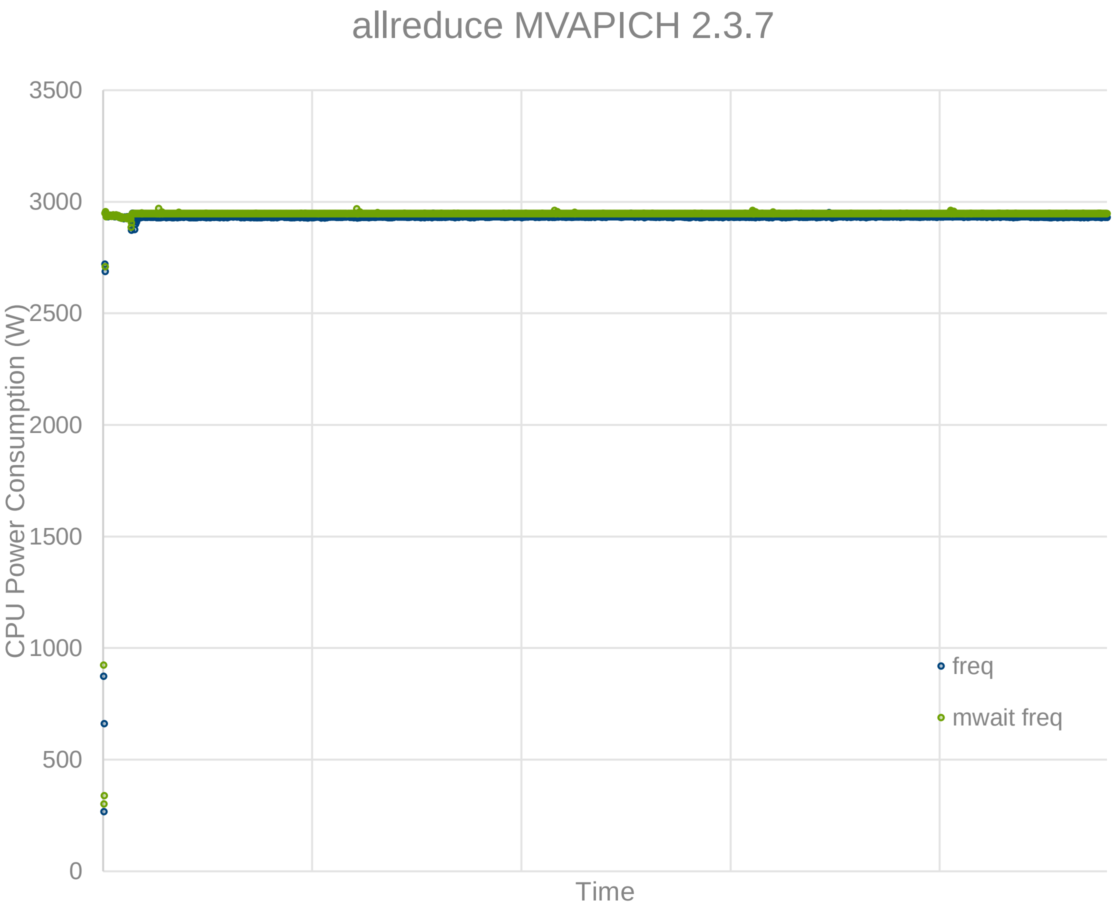
- Intel Xeon 8480 – 56C @ 2.0 GHz, 350 W TDP - NDR-400
- AMD EPYC 9654 – 96C @ 2.4 GHz, 400 W cTDP - HDR-200
- AMD EPYC 9354 – 32C @ 3.25 GHz, 280 W cTDP - HDR-200

CPU power is measured every 250 ms using the Dell “f1_count” internal tool

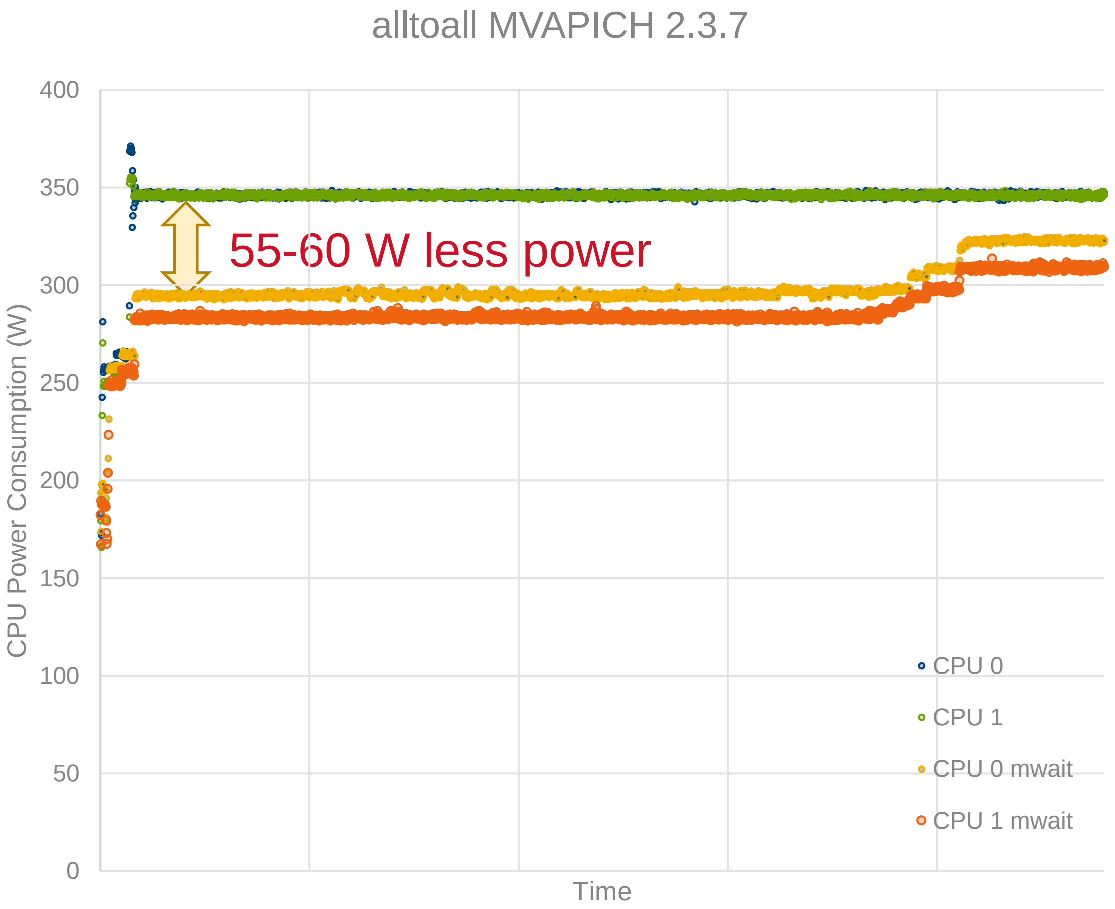
Intel

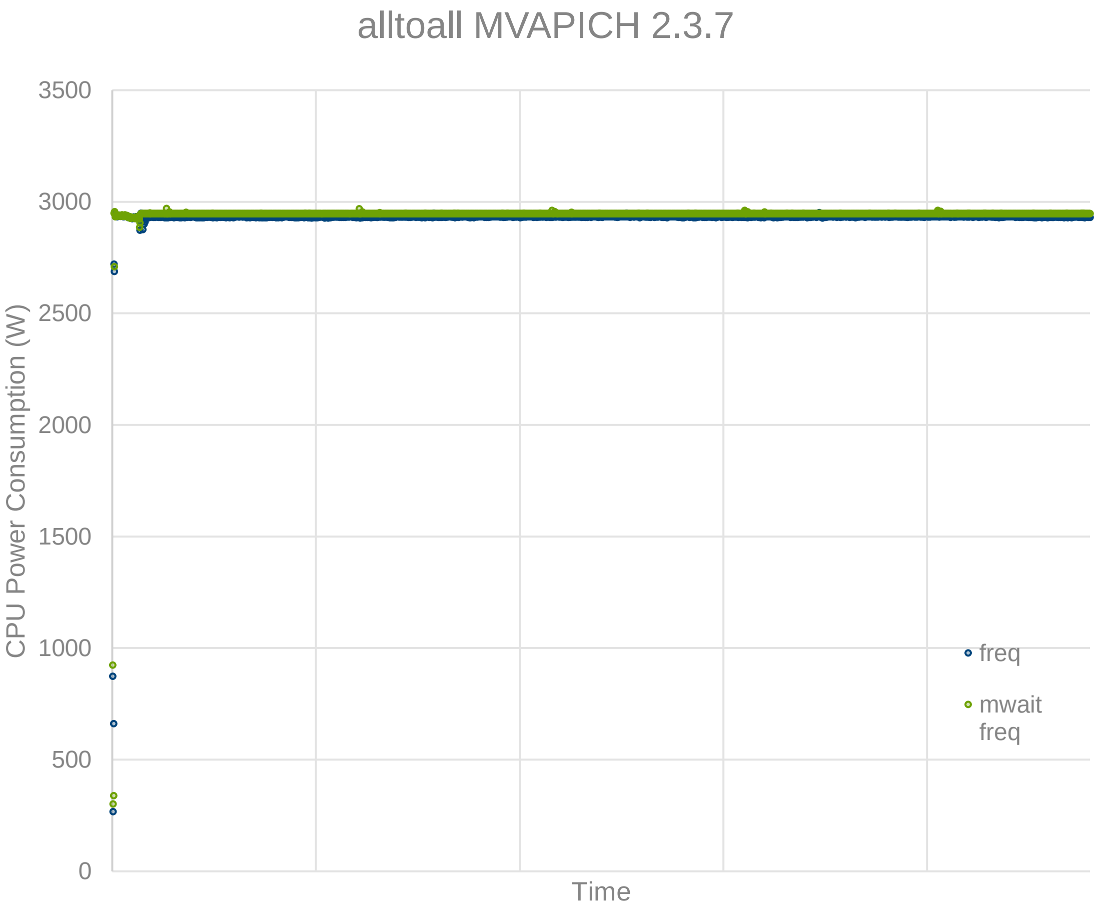
MWAIT power savings – Intel Xeon 8480



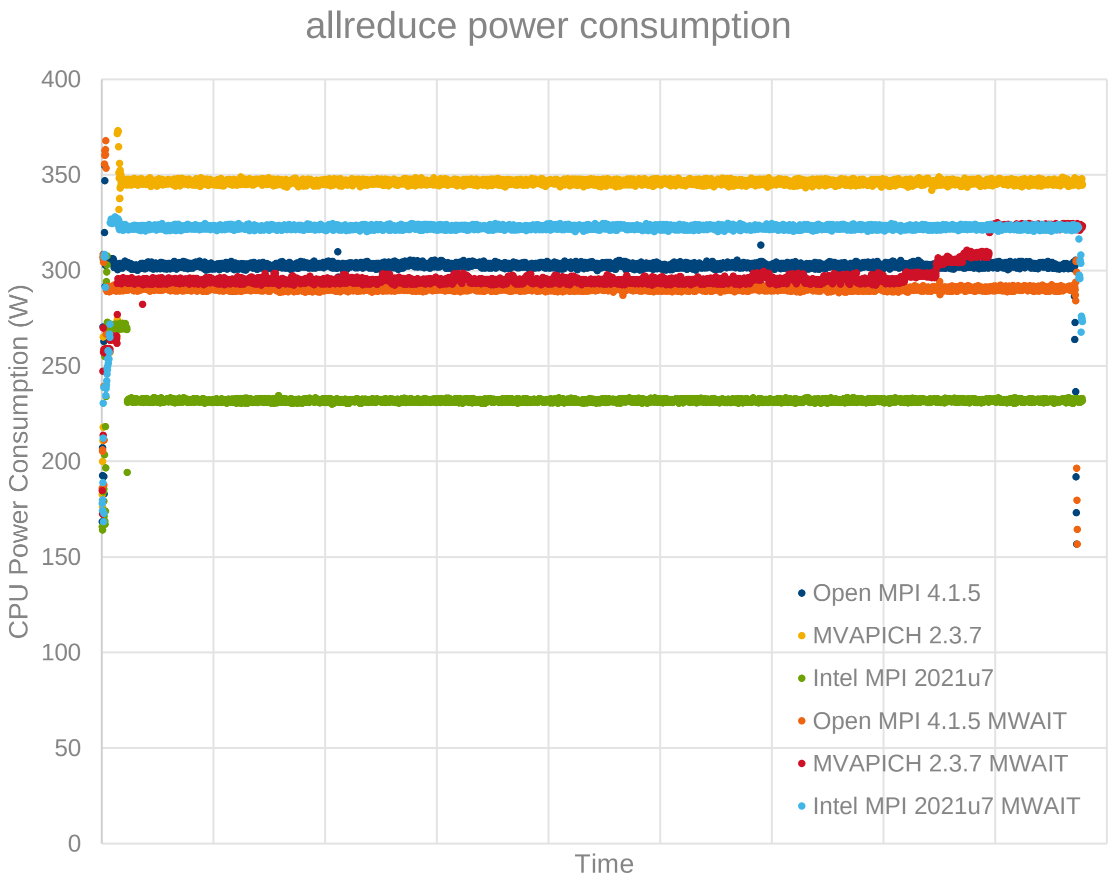


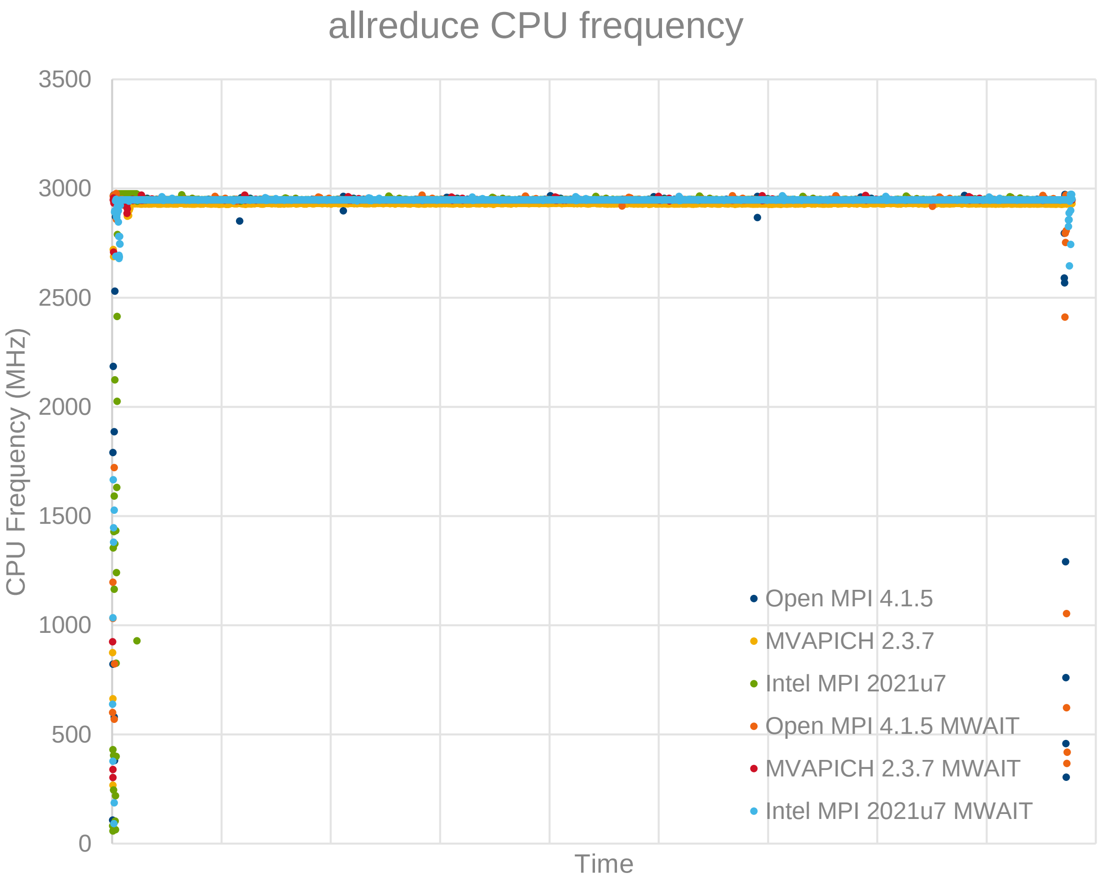
MWAIT power savings – Intel Xeon 8480



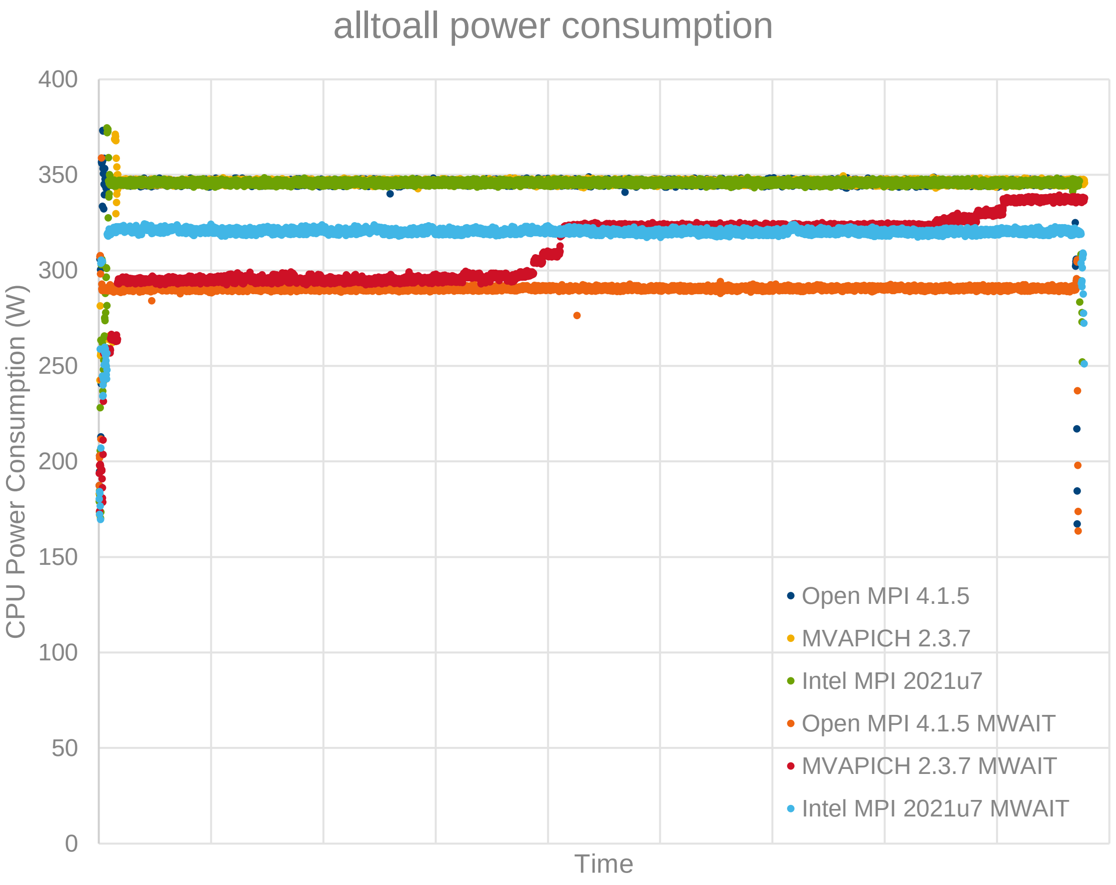


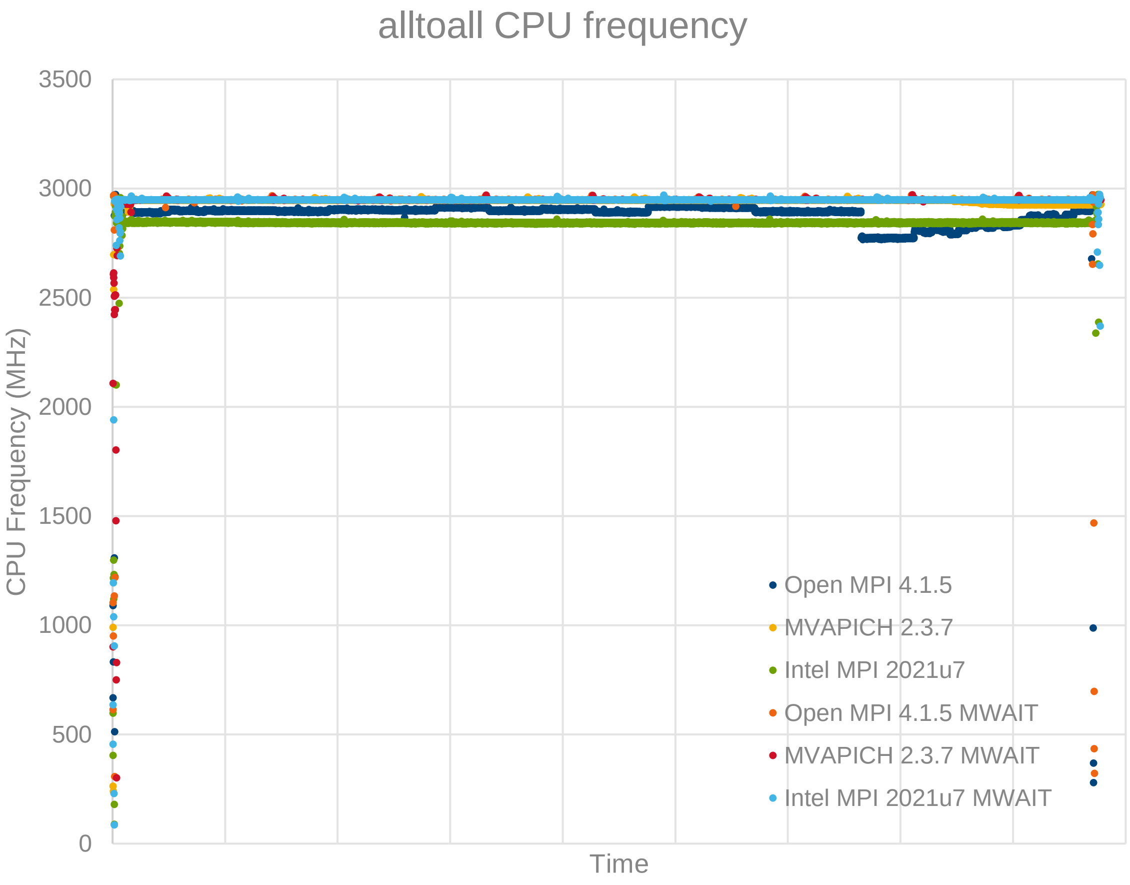
Comparison between MPI libraries – Intel Xeon 8480





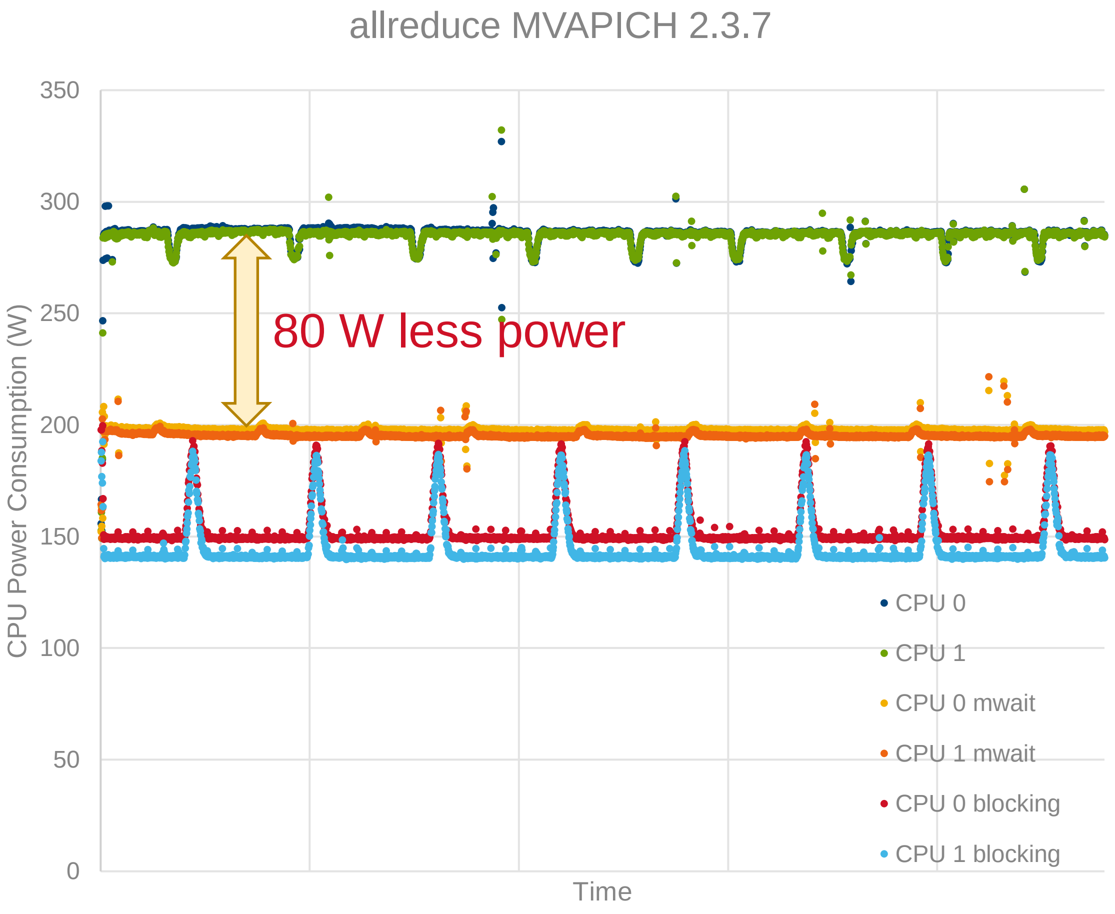
Comparison between MPI libraries – Intel Xeon 8480

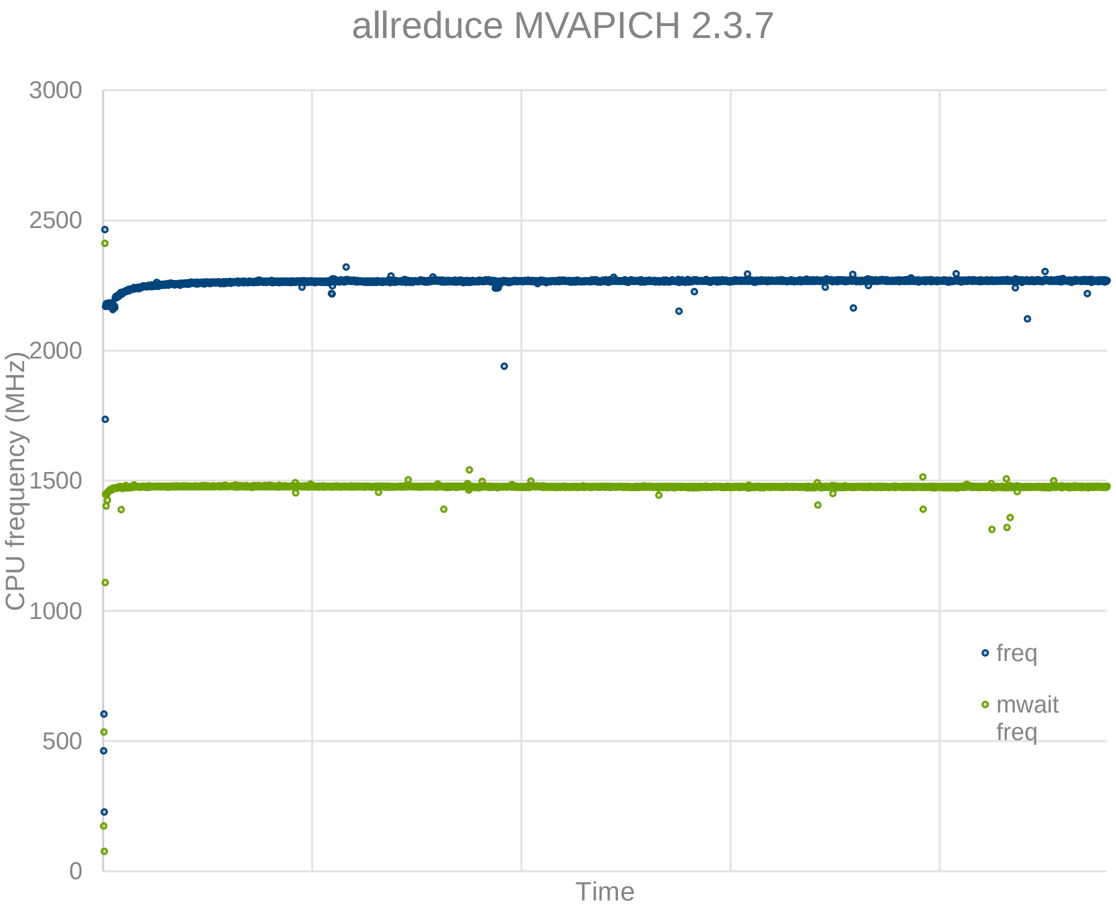




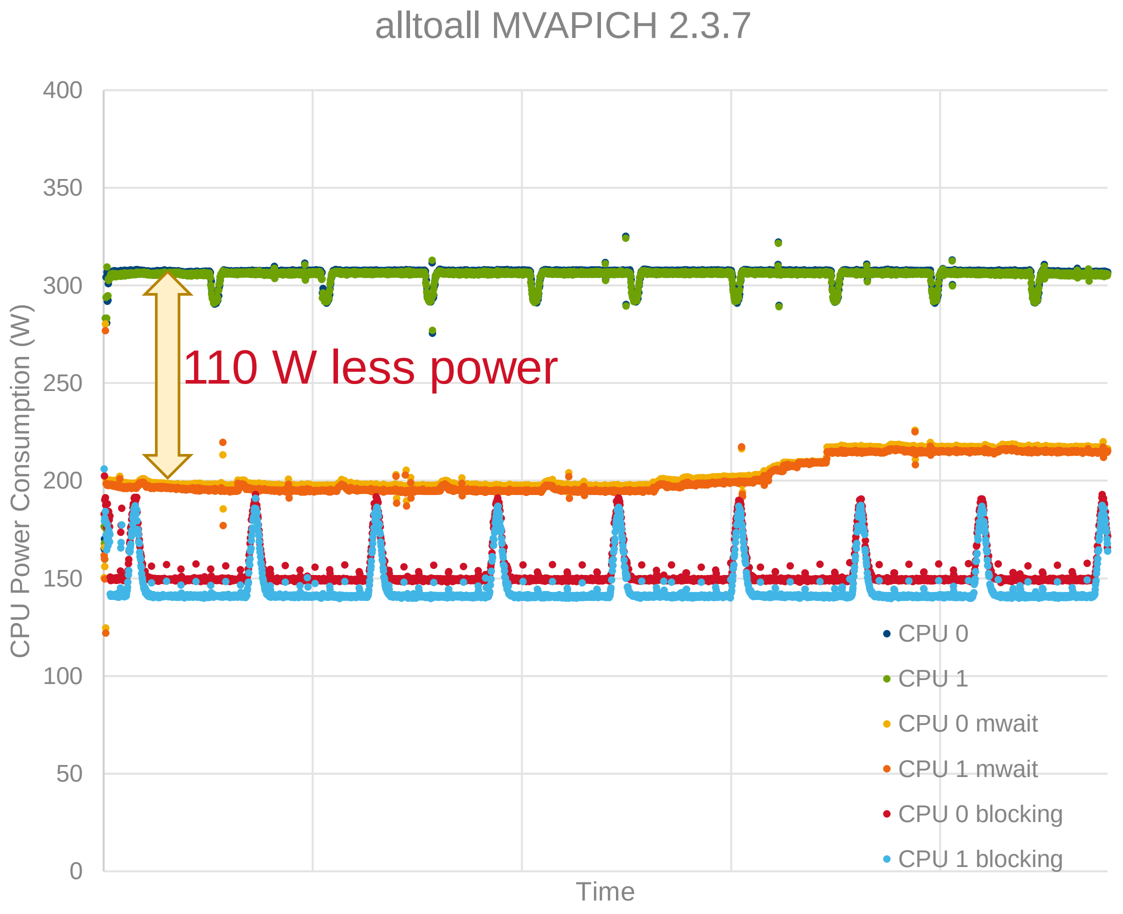
AMD

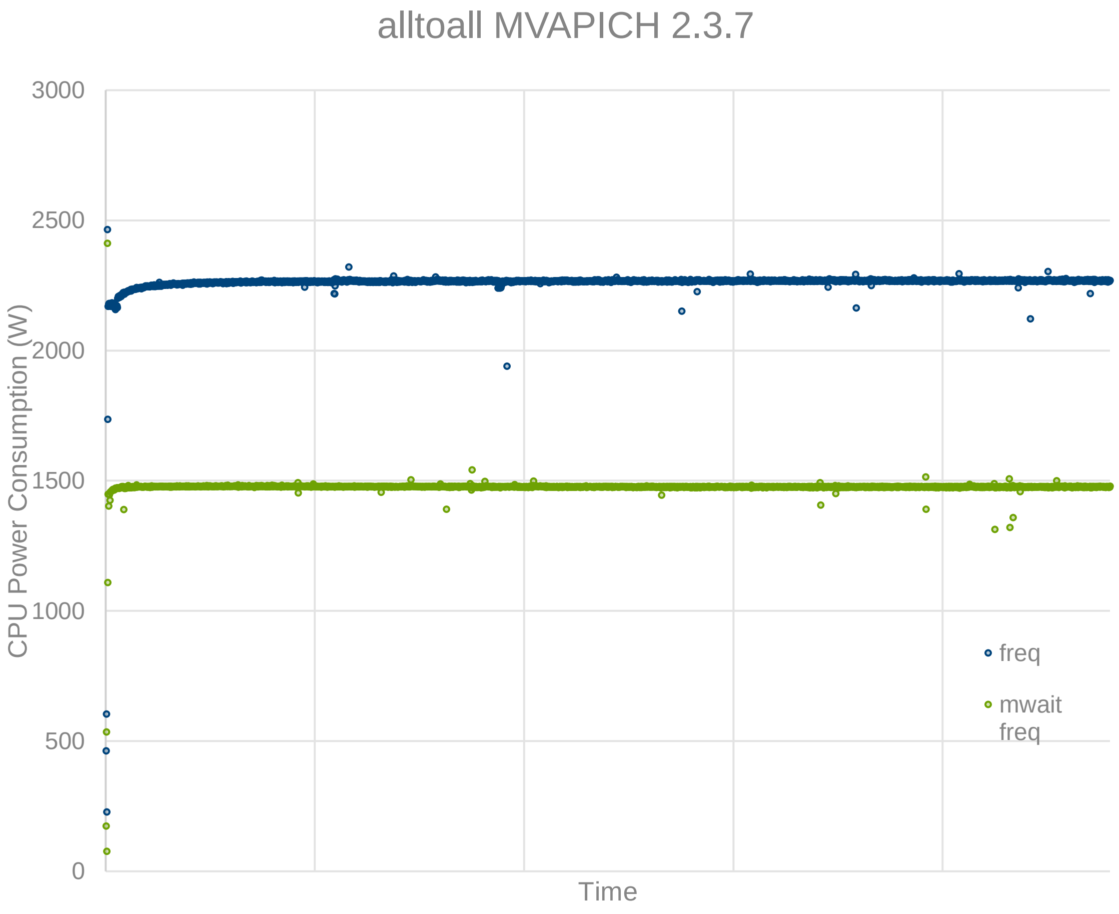
MWAIT power savings – AMD EPYC 9654



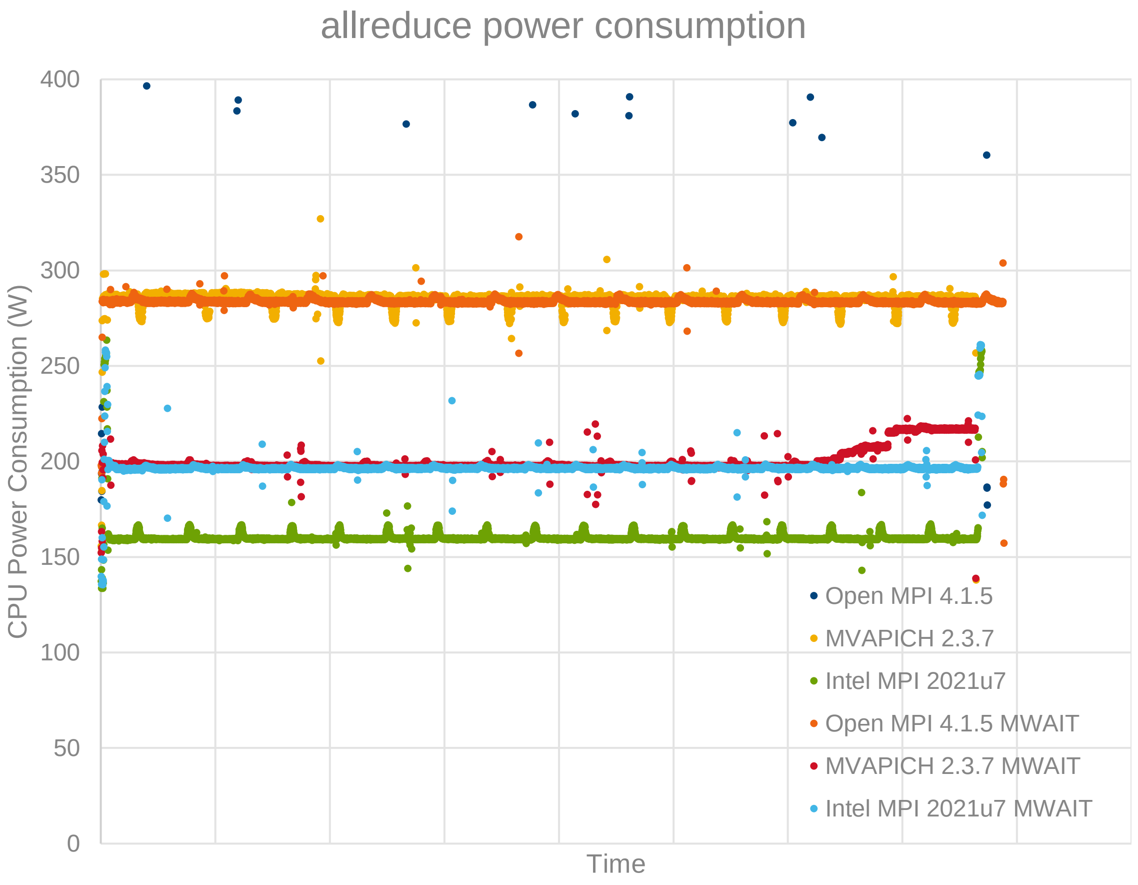


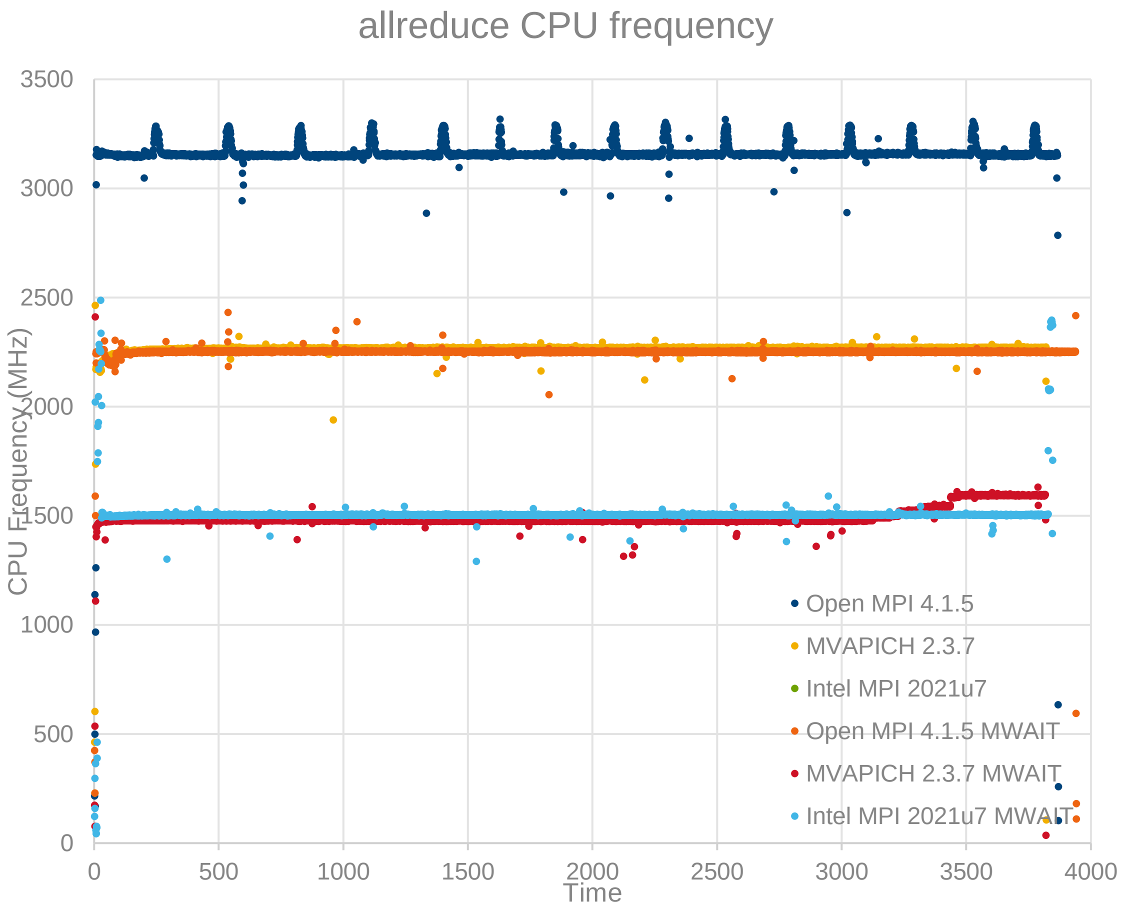
MWAIT power savings – AMD EPYC 9654



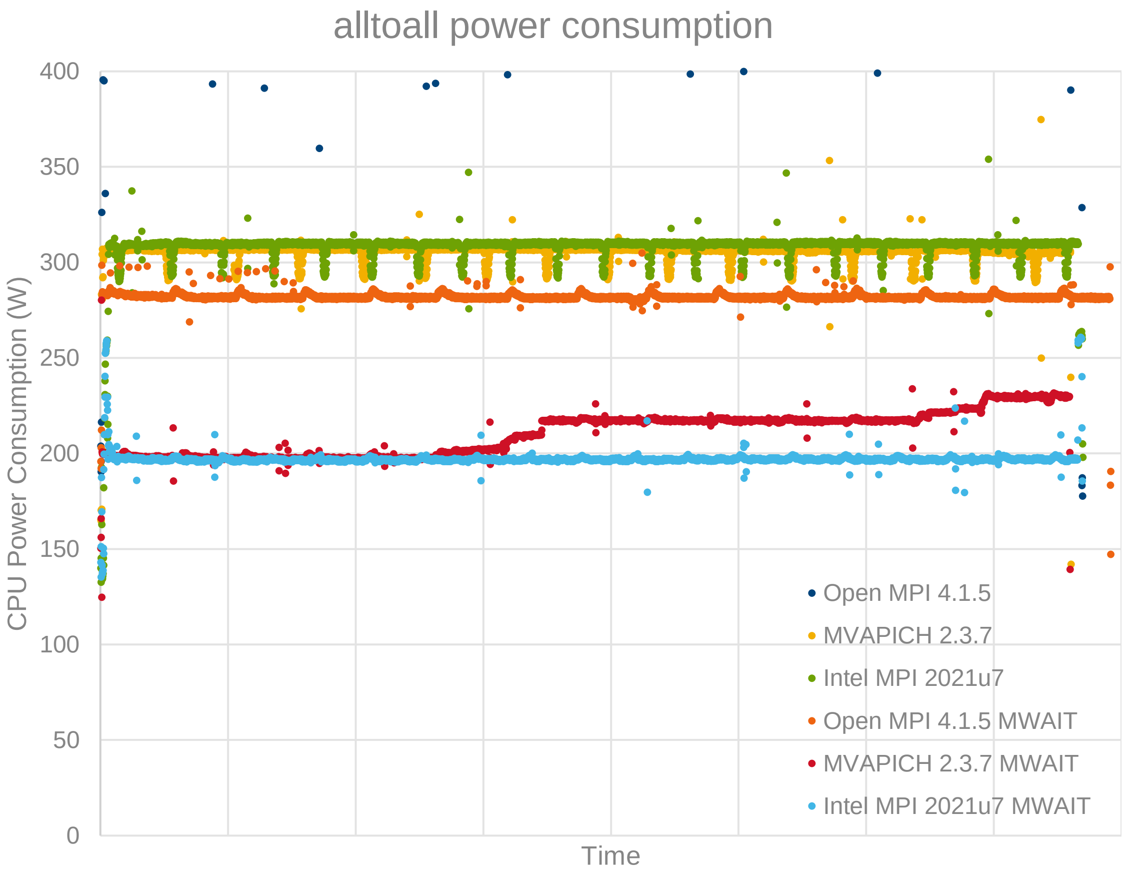


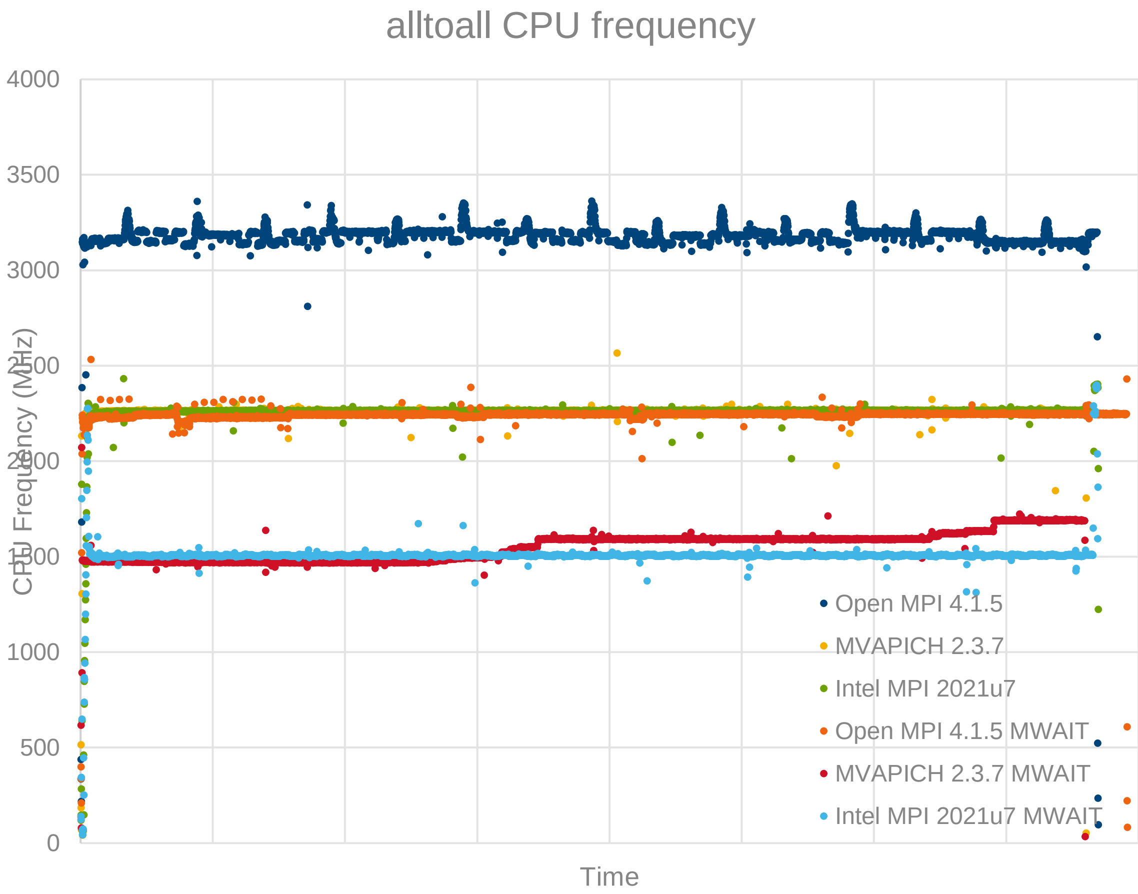
Comparison between MPI libraries – AMD EPYC 9654



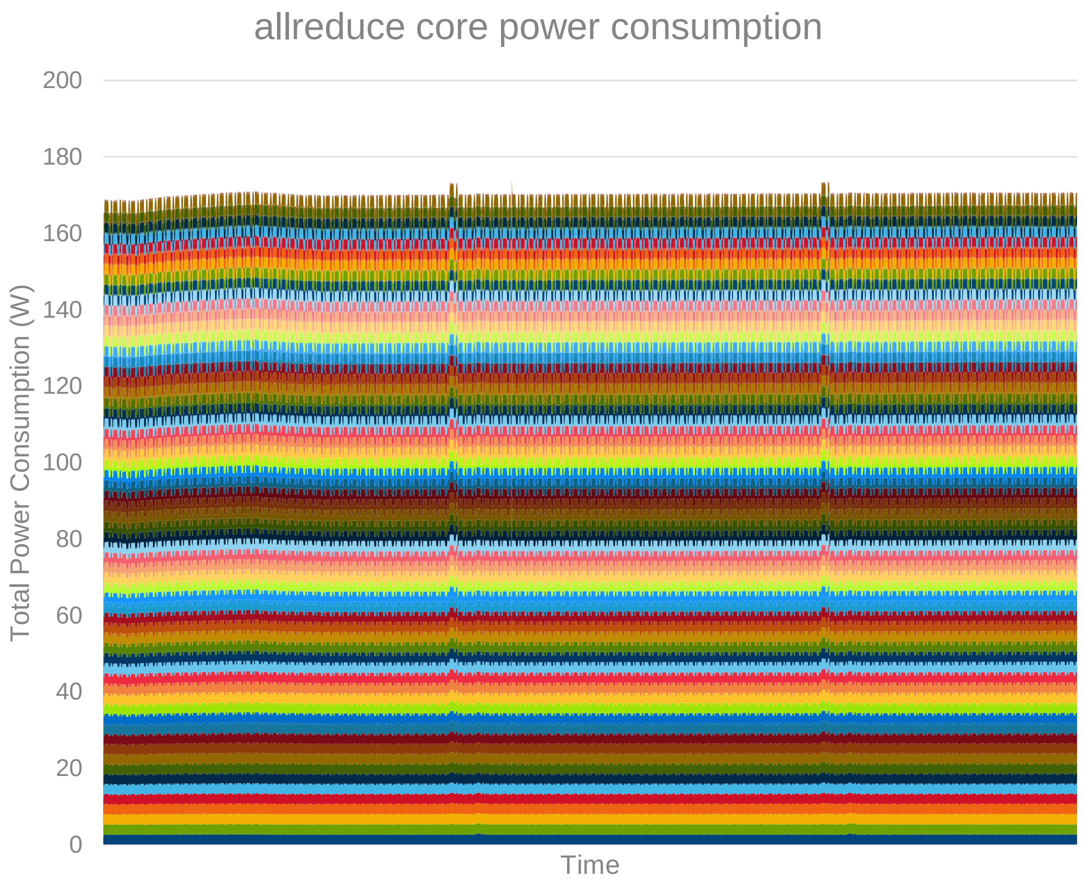


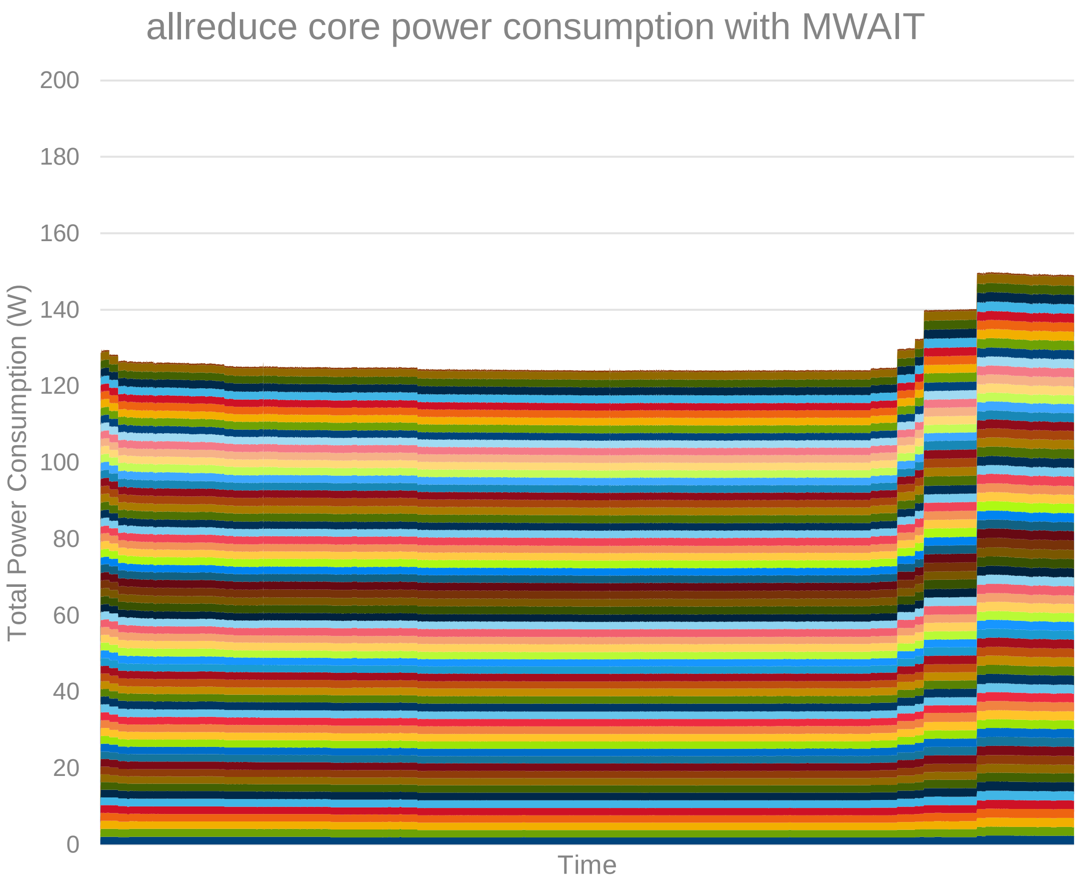
Comparison between MPI libraries – AMD EPYC 9654





MWAIT per-core power savings – AMD EPYC 9354

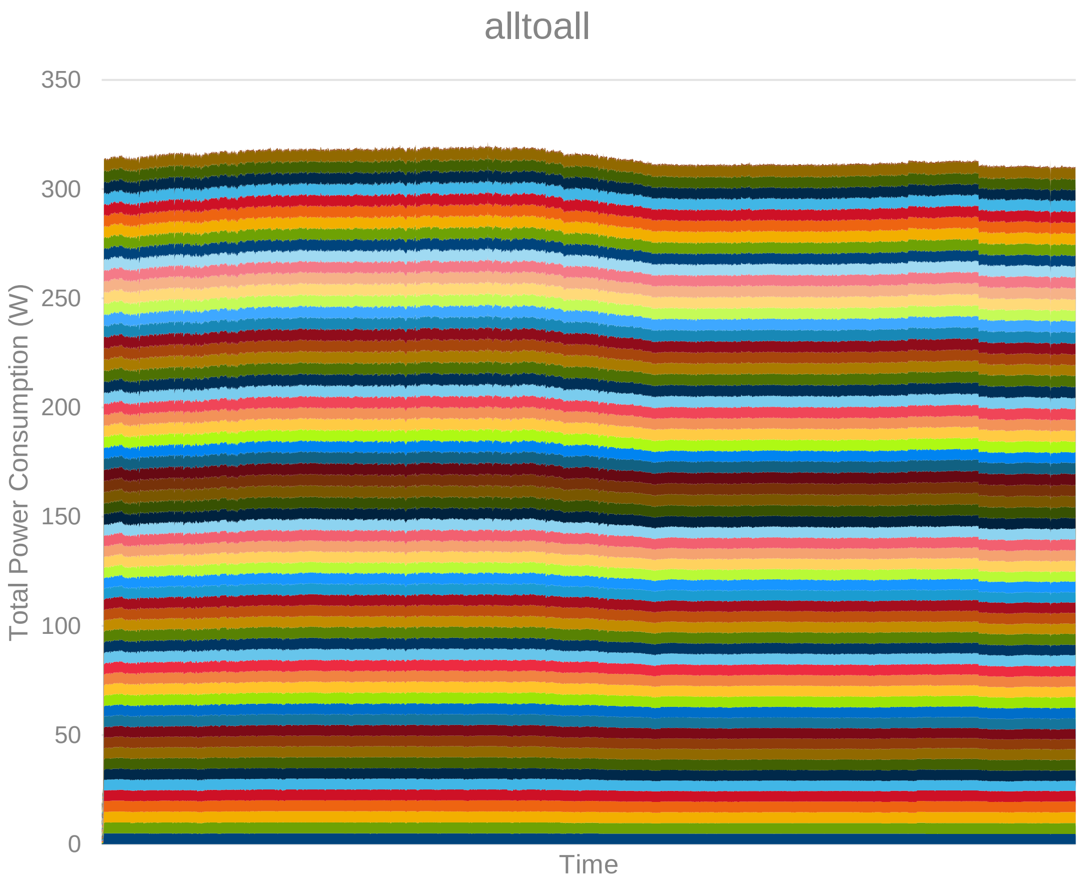




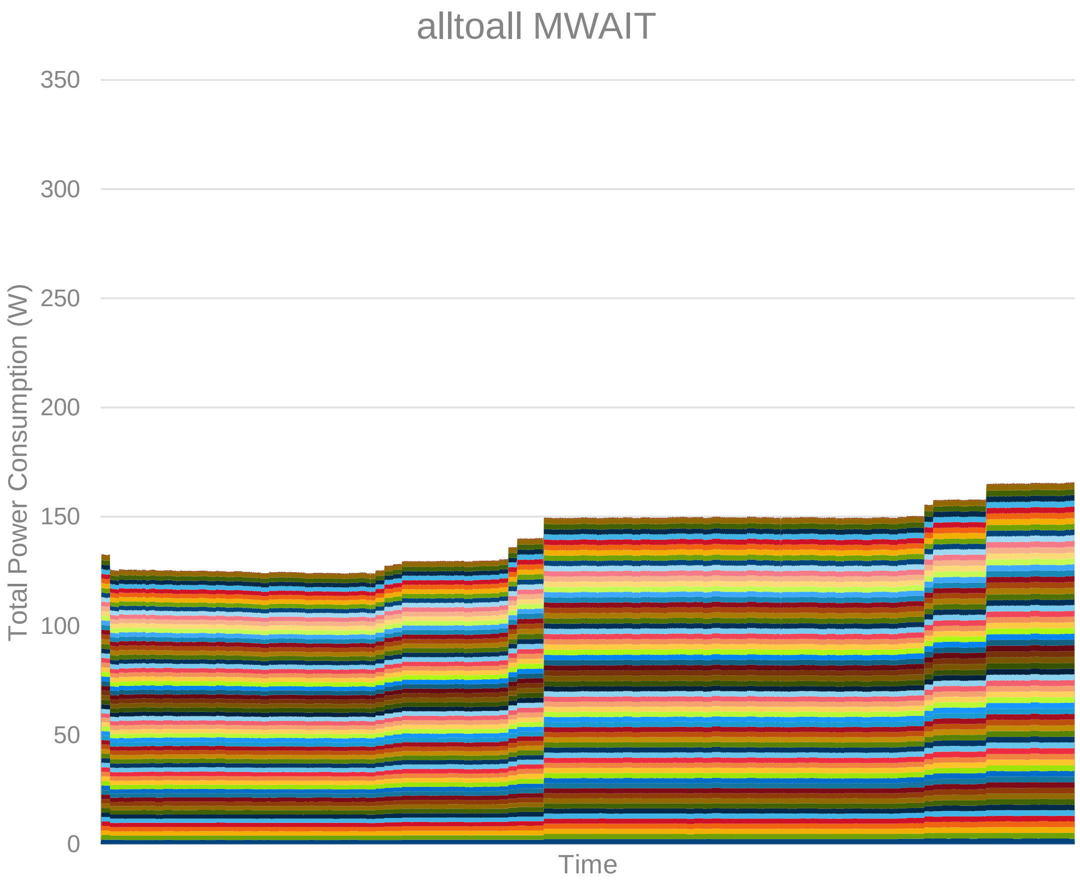
2.6-2.7 W per core

2.0-2.4 W per core

MWAIT per-core power savings – AMD EPYC 9354



4.9-5.0 W per core



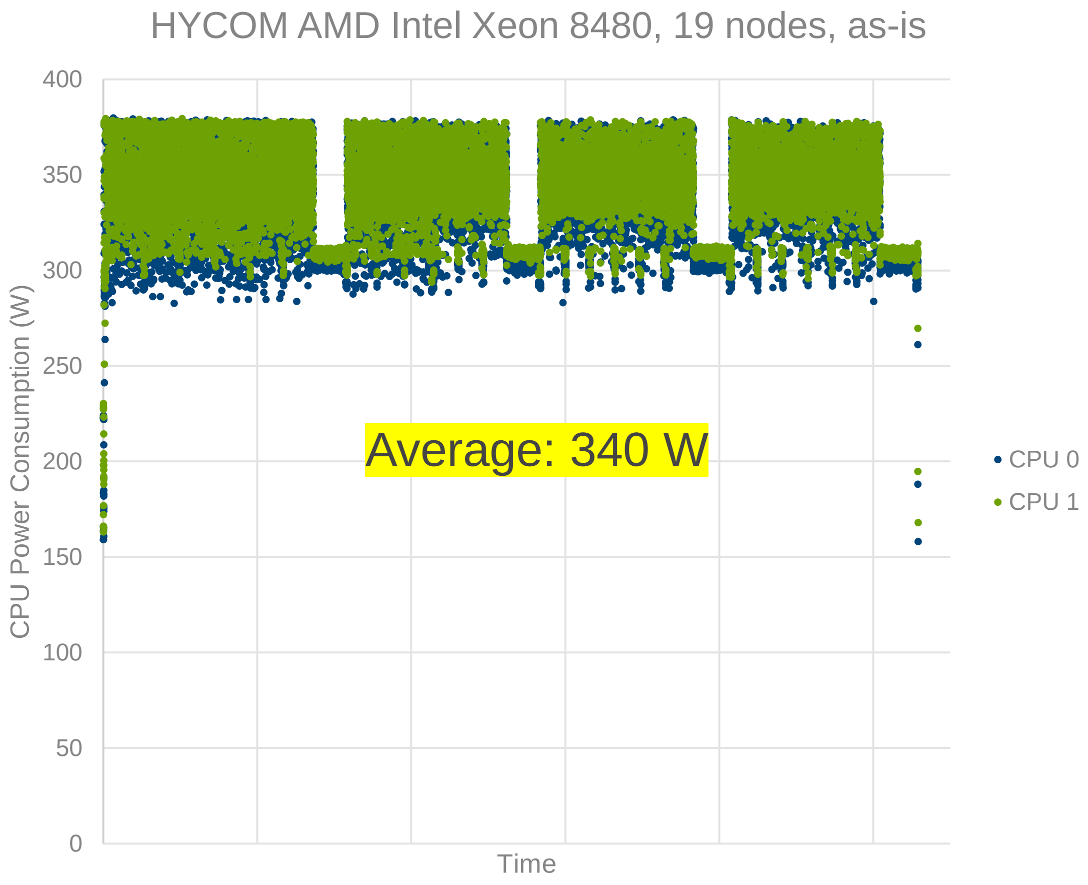
2.0-2.5 W per core

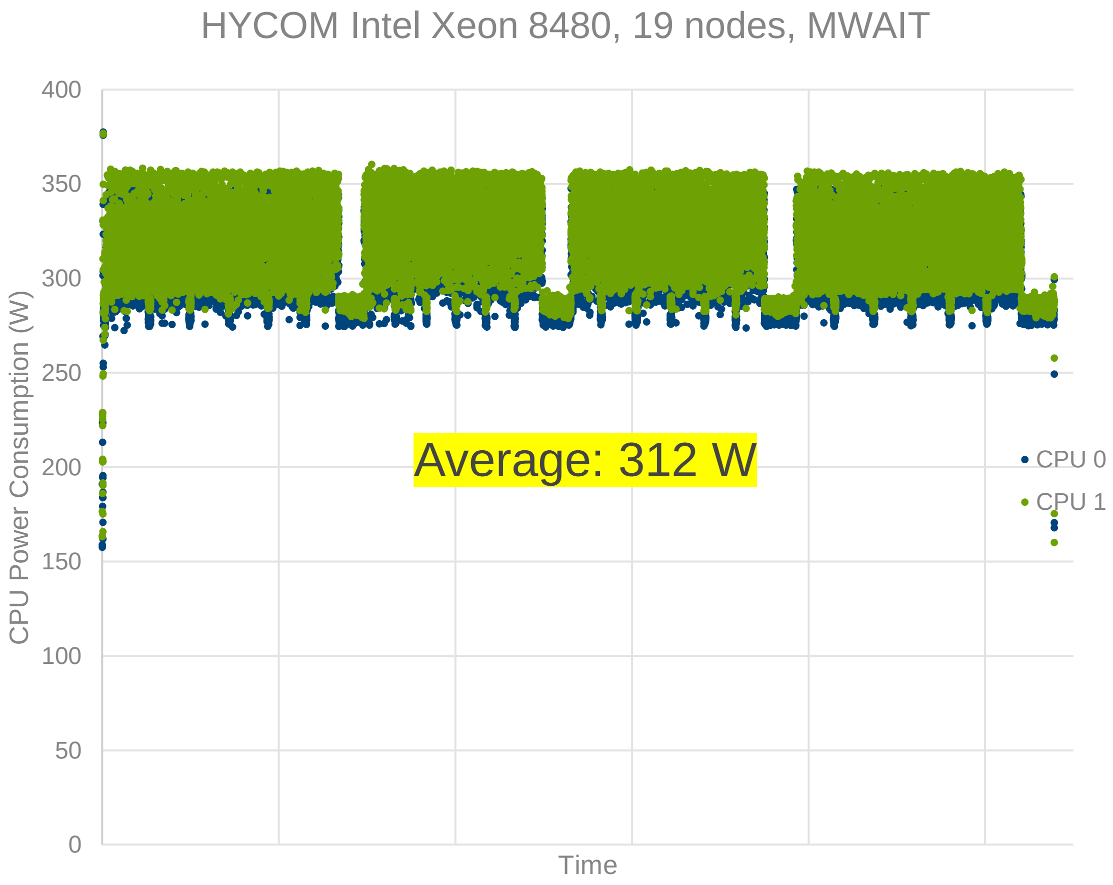
Application power measurements

Applications tested

Application name	Segment	Communication pattern	MPI footprint
HYCOM	Weather/Climate	Nearest neighbor	Large Allreduce
MOM5	Weather/Climate	Nearest neighbor	Large Allreduce
OpenFOAM	Computational Fluid Dynamics	Point-to-point	Small Allreduce
Quantum Espresso	Materials Science	Global Transposition	Alltoall
CP2k	Chemistry	Global Transposition	Allreduce/Alltoall

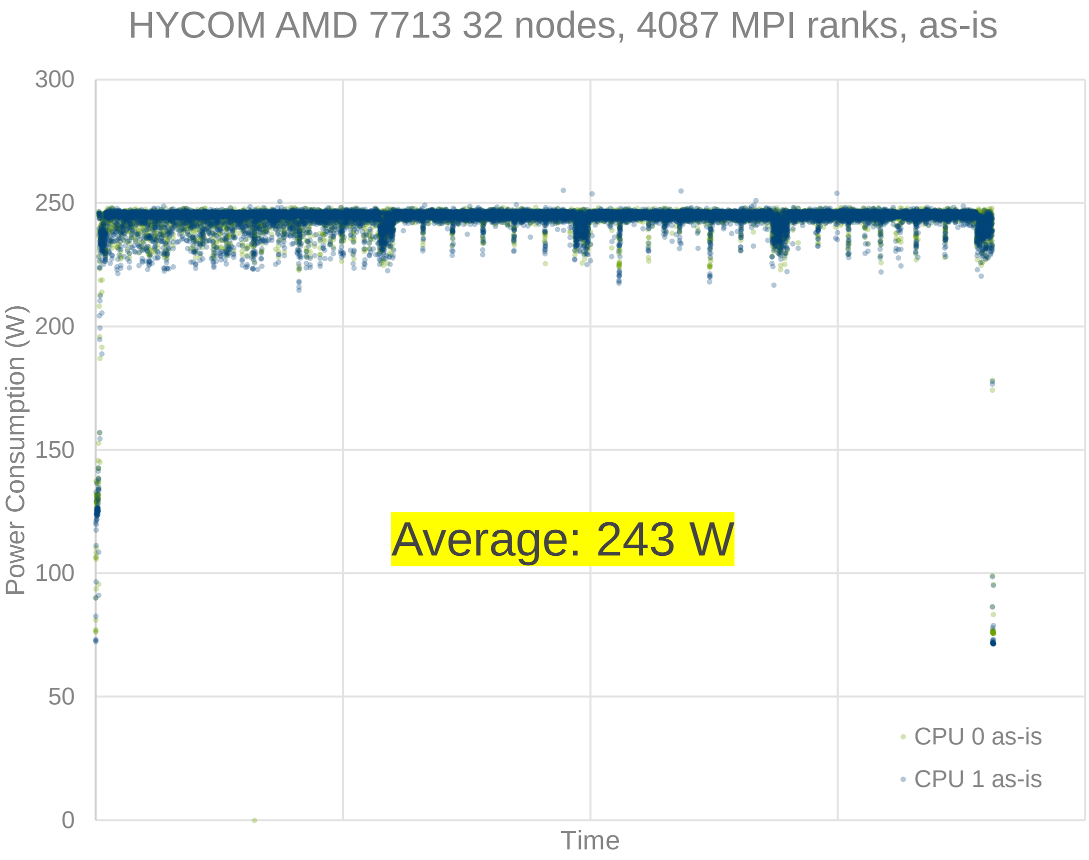
HYCOM on Intel Xeon 8480 (350 W TDP)

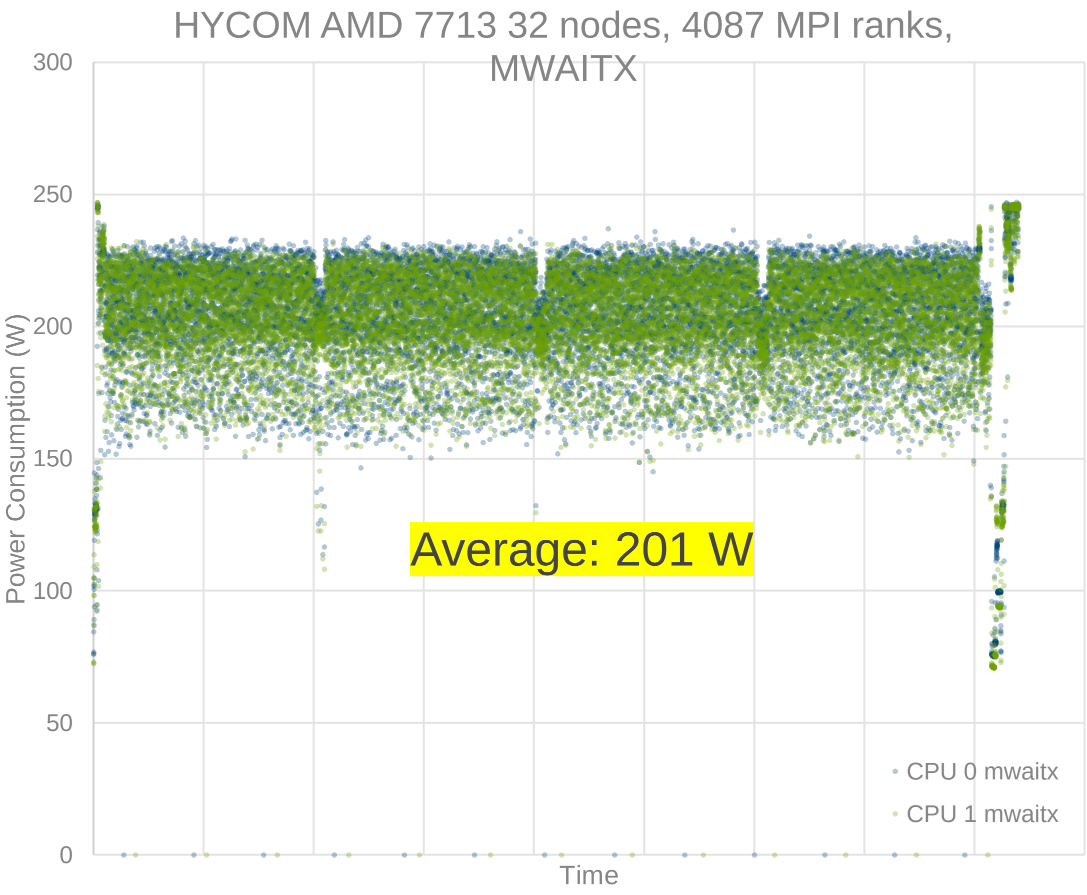




- 12% power savings *per* CPU
- 1.8% performance loss

HYCOM on AMD 7713 (240W cTDP)

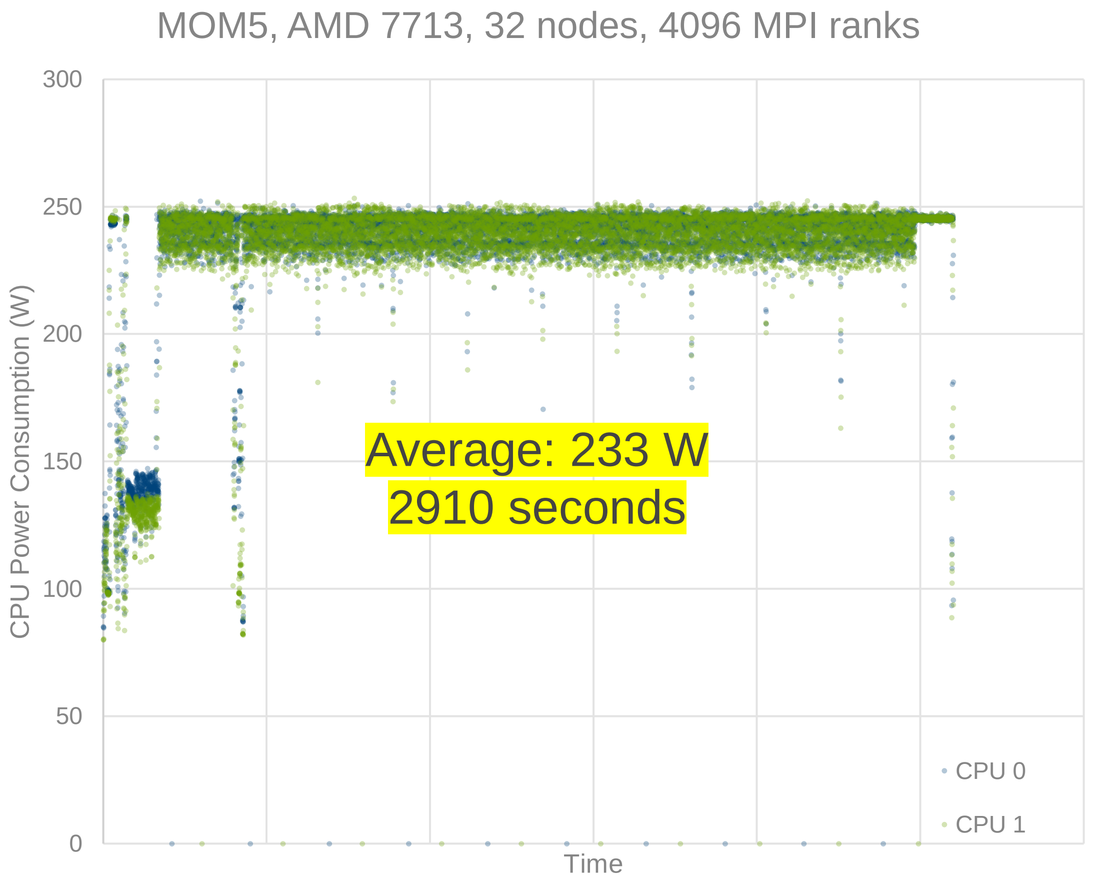


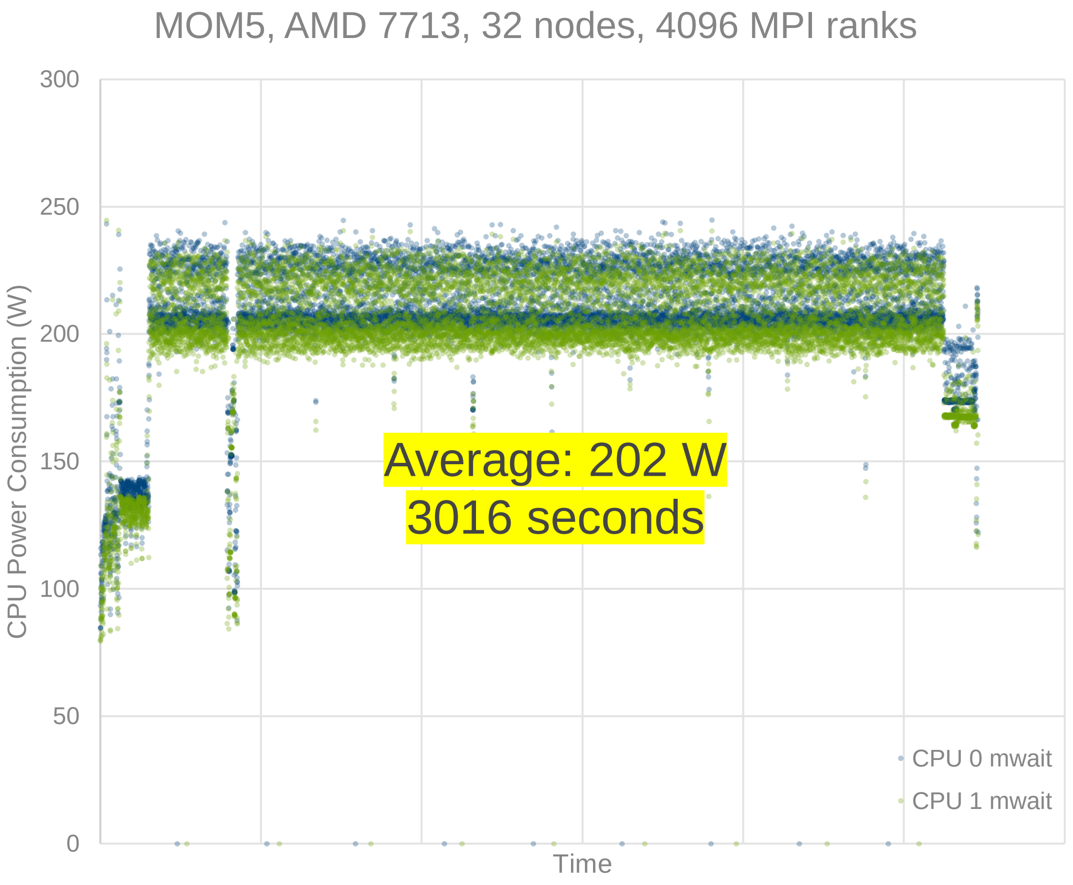


- 17% power savings *per* CPU
- 10% *faster*

DELLTechnologies

MOM5 on AMD 7713 (240W cTDP)

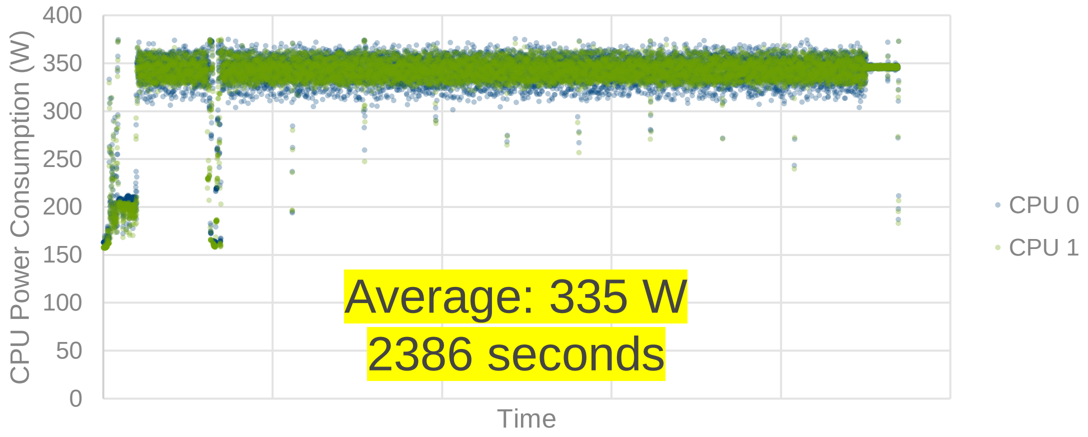




- 13.3% power savings *per* CPU, 3.6% performance loss

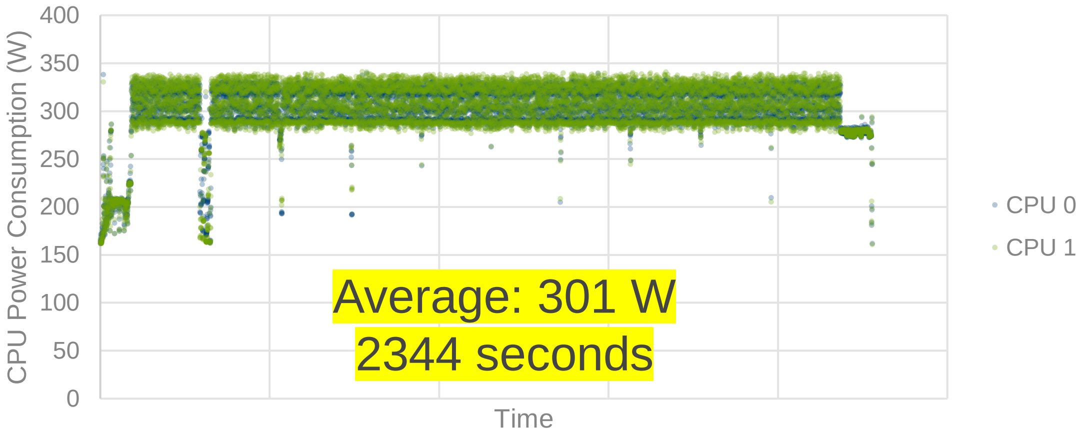
MOM5 on Intel 8480+ (350W TDP)

MOM5, 8480+ 10 nodes, 1120 cores



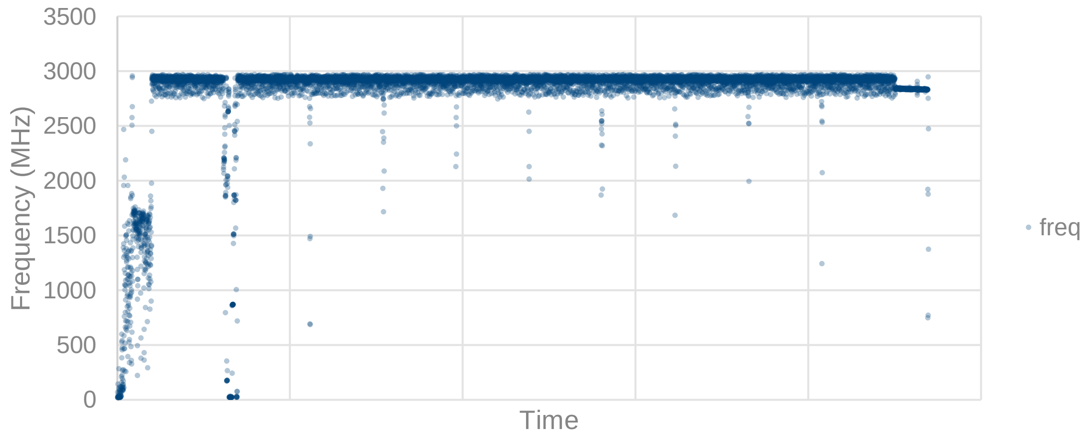
Average: 335 W
2386 seconds

MOM5, 8480+ 10 nodes, 1120 cores, umwait

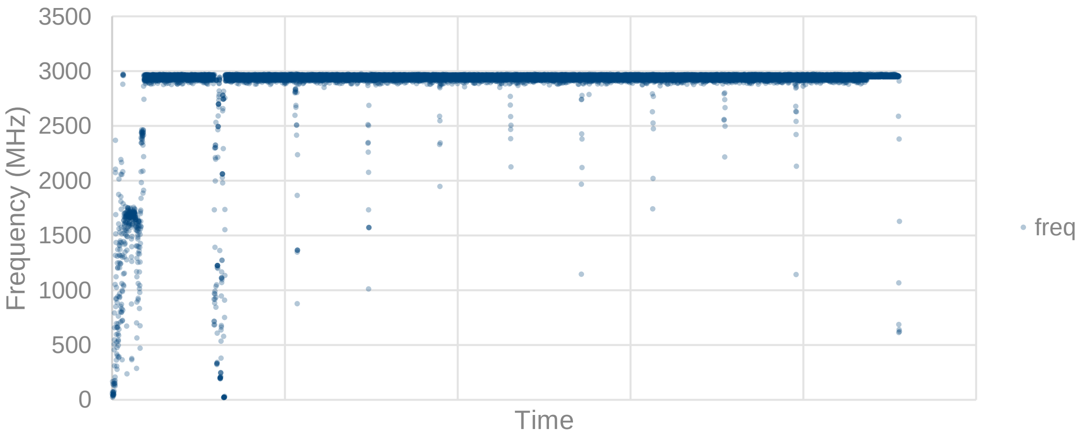


Average: 301 W
2344 seconds

MOM5, 8480+ 10 nodes, 1120 cores

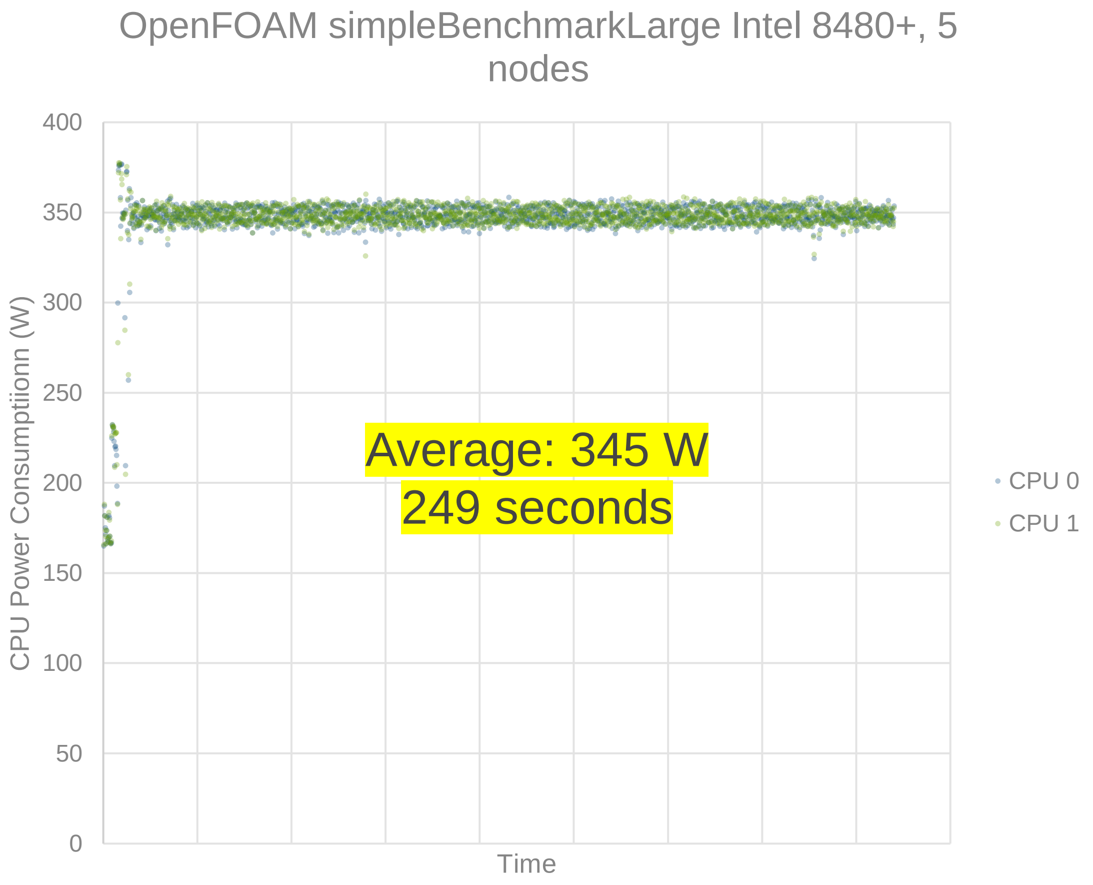


MOM5, 8480+ 10 nodes, 1120 cores, umwait

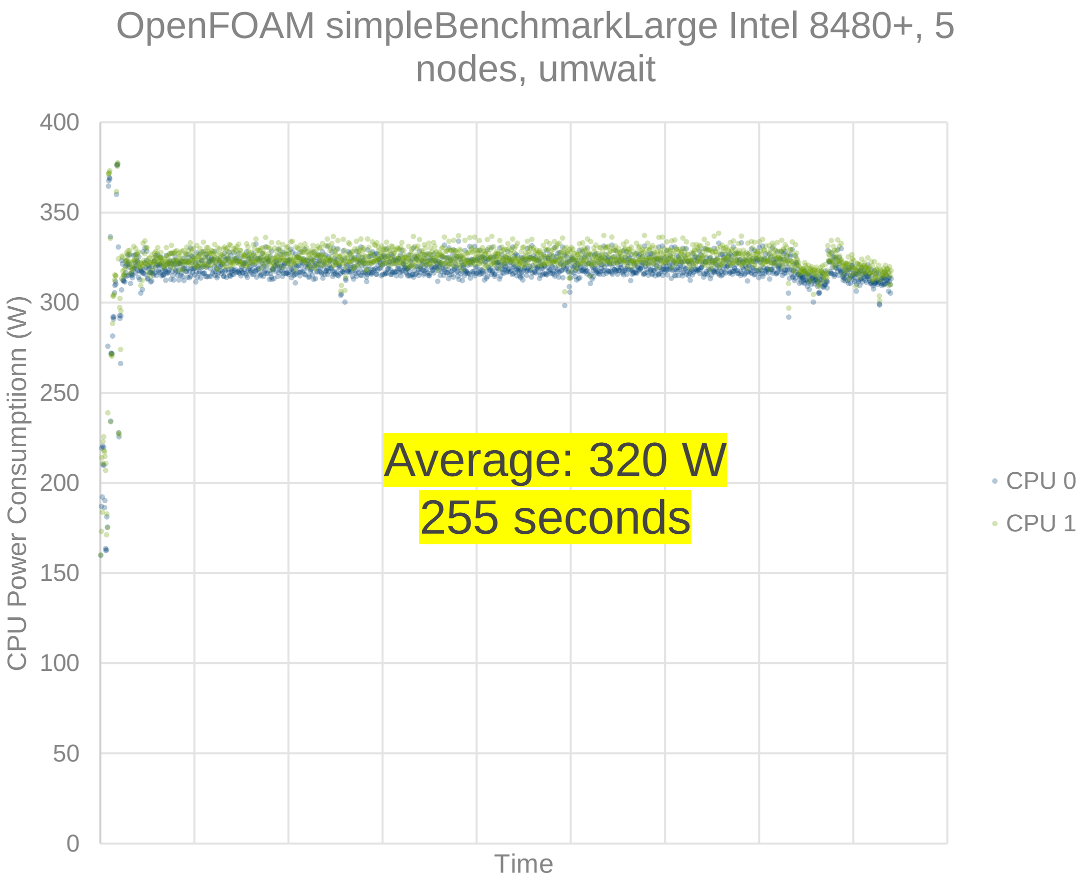


- 30 Watts power savings *per* CPU

OpenFOAM on Intel 8480+



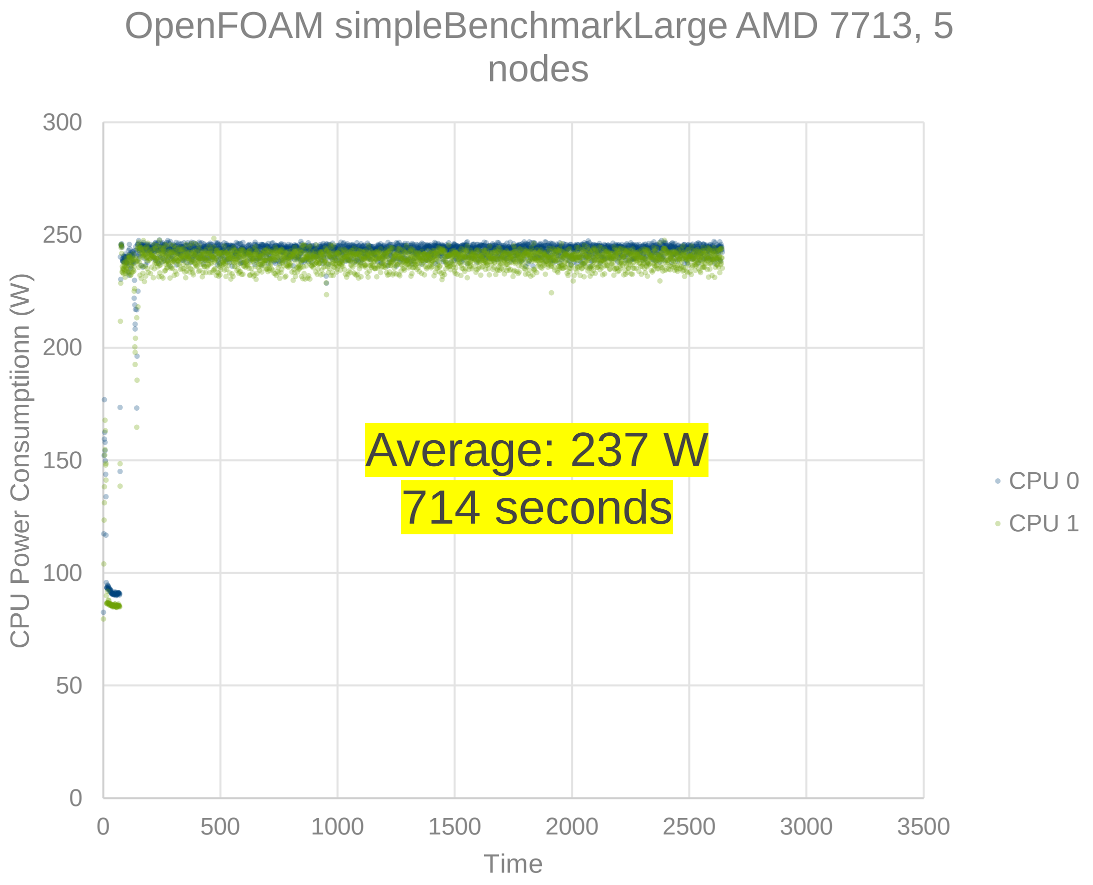
Average: 345 W
249 seconds



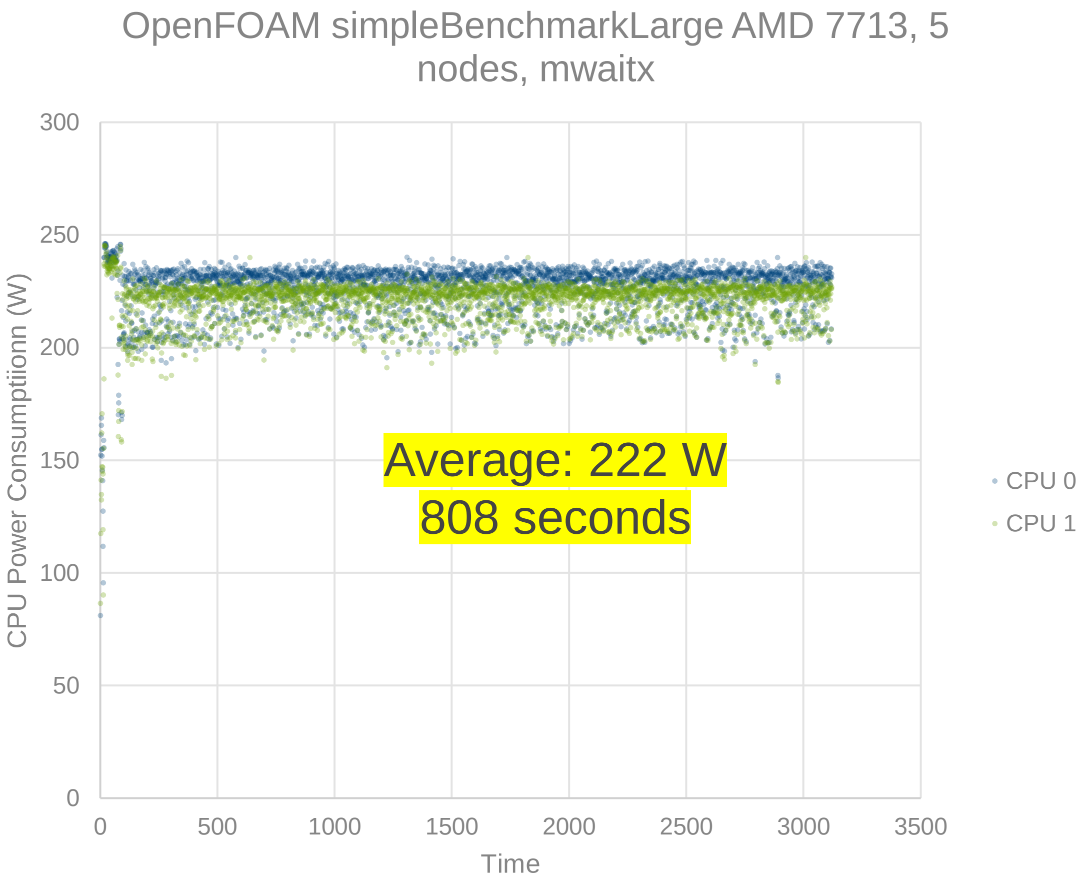
Average: 320 W
255 seconds

- 25 Watts power savings *per* CPU

OpenFOAM on AMD 7713



Average: 237 W
714 seconds

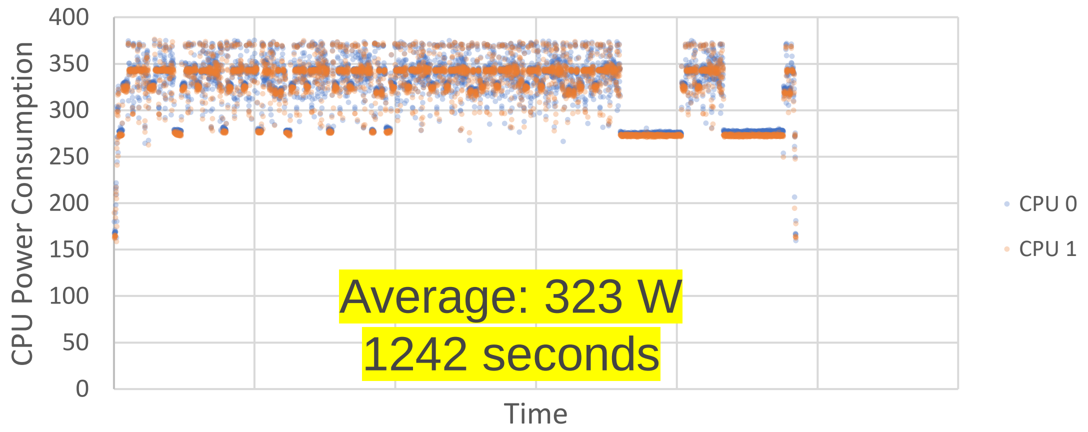


Average: 222 W
808 seconds

- 25 Watts power savings *per* CPU

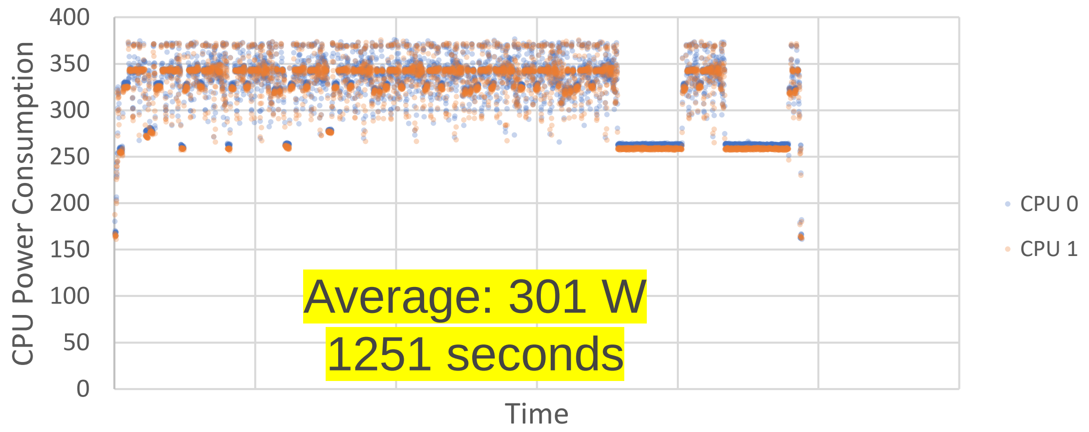
QE on Intel 8480+ (350W TDP)

QE grir686, 10 nodes Intel 8480+

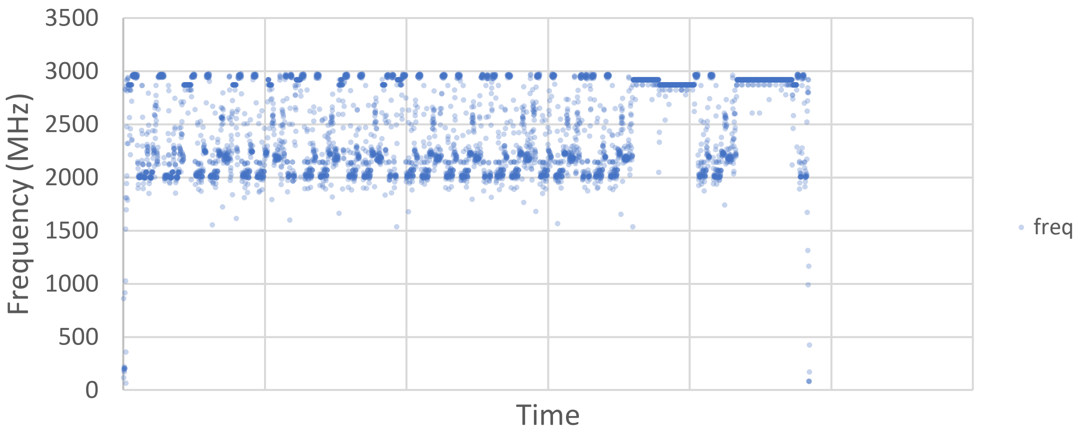


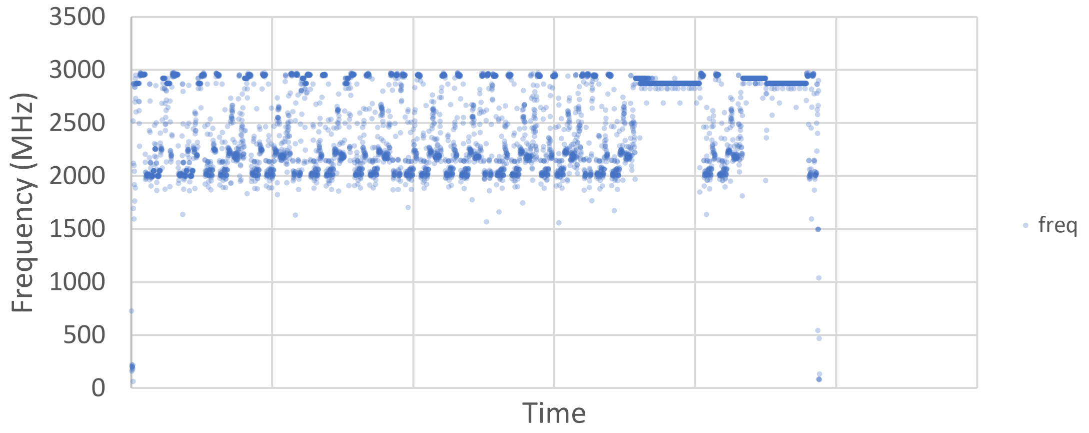
Average: 323 W
1242 seconds

QE grir686, 10 nodes Intel 8480+, umwait



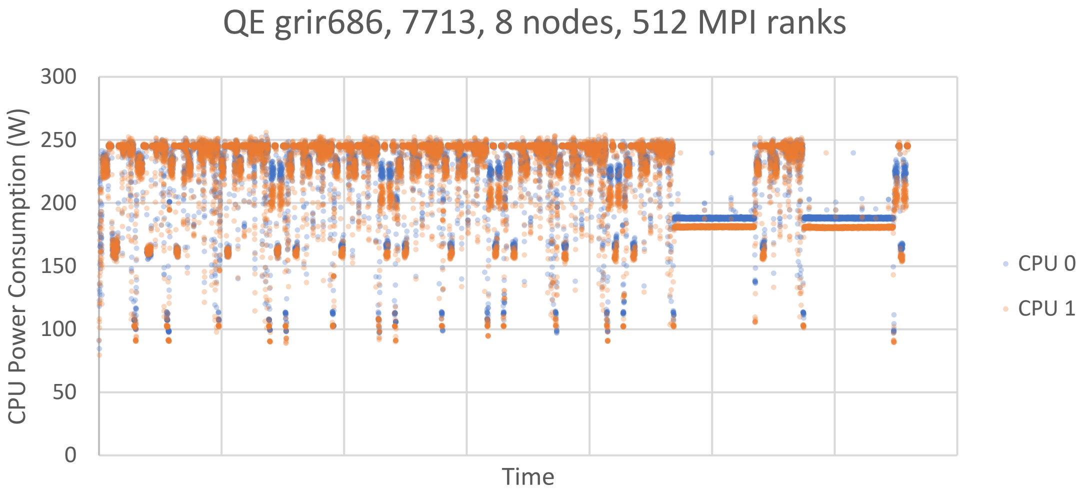
Average: 301 W
1251 seconds



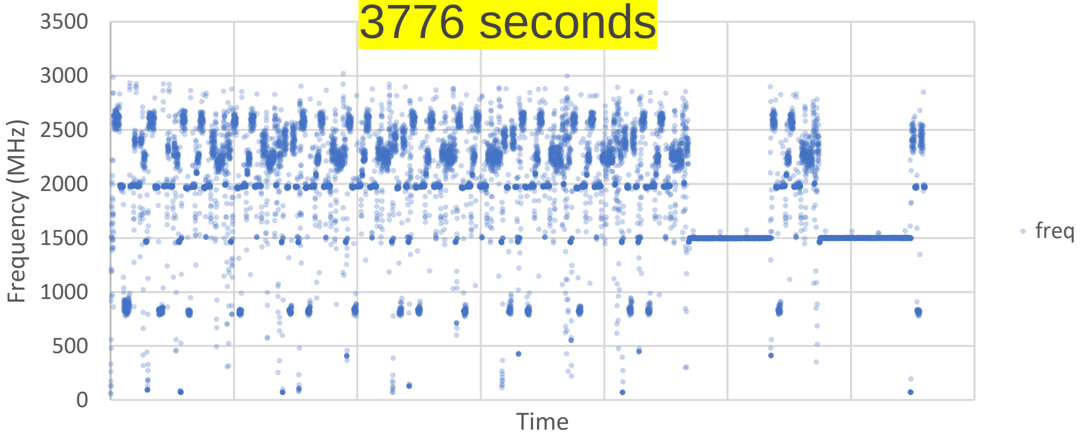


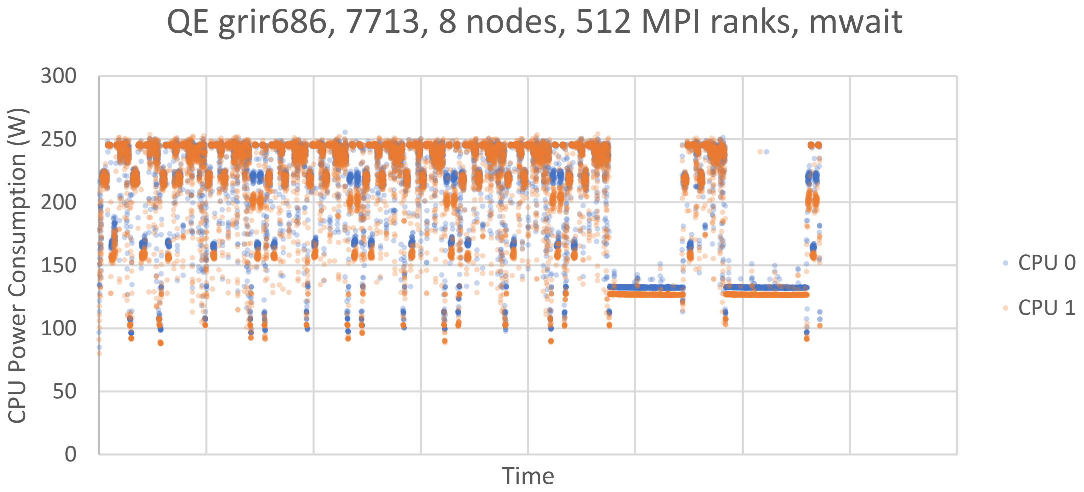
- 22 Watts power savings *per* CPU

QE on AMD 7713 (240 W TDP)

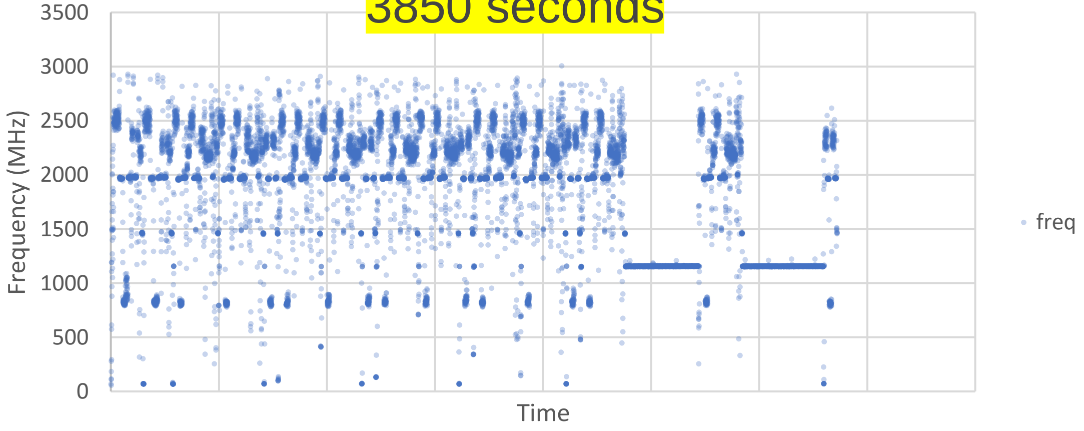


Average: 215 W
3776 seconds



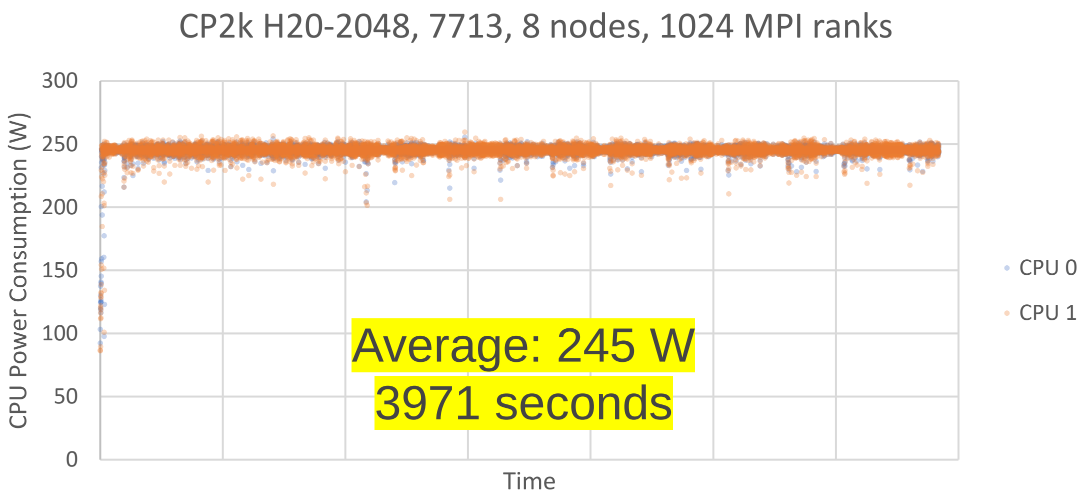


Average: 197 W
3850 seconds



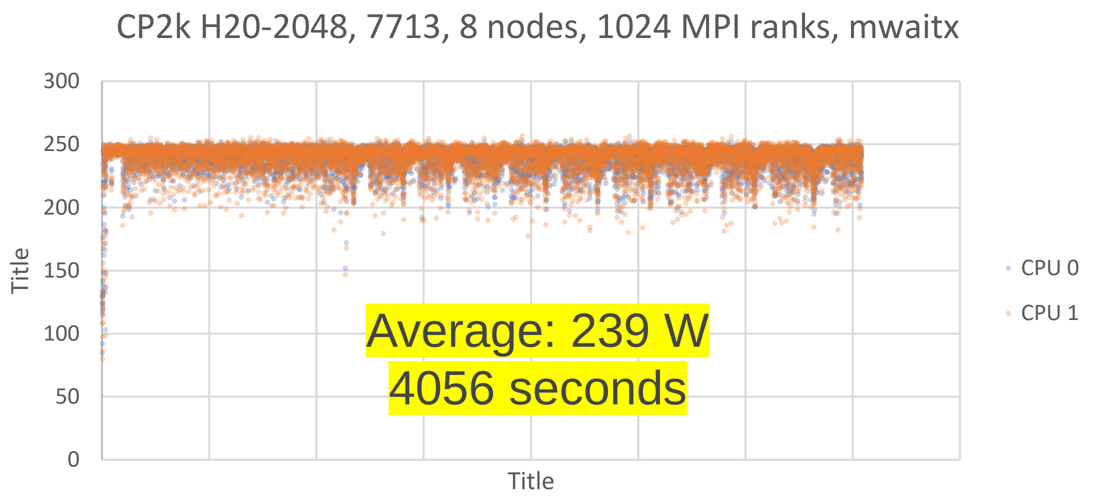
- 8% power savings *per* CPU

CP2k on AMD 7713 (240 W TDP)



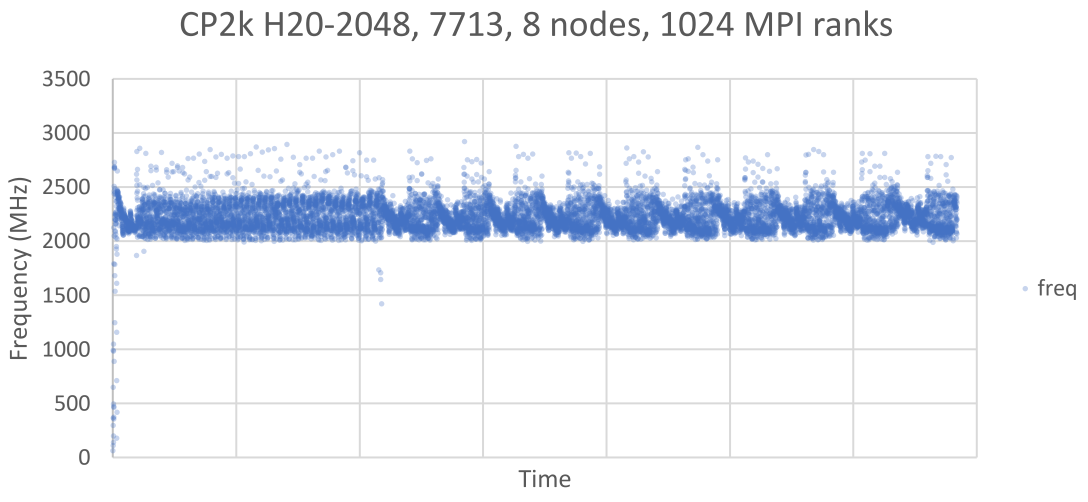
CP2k H20-2048, 7713, 8 nodes, 1024 MPI ranks

Average: 245 W
3971 seconds

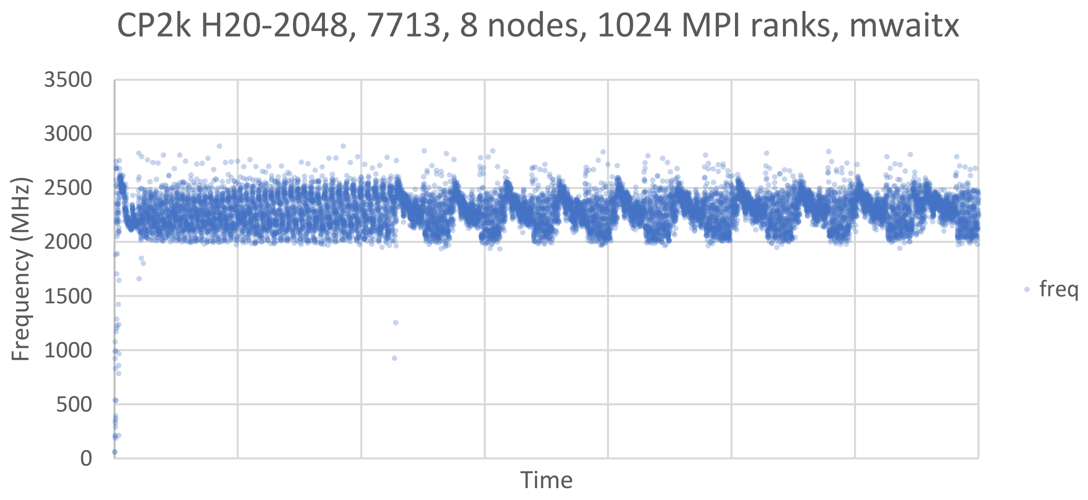


CP2k H20-2048, 7713, 8 nodes, 1024 MPI ranks, mwaitx

Average: 239 W
4056 seconds



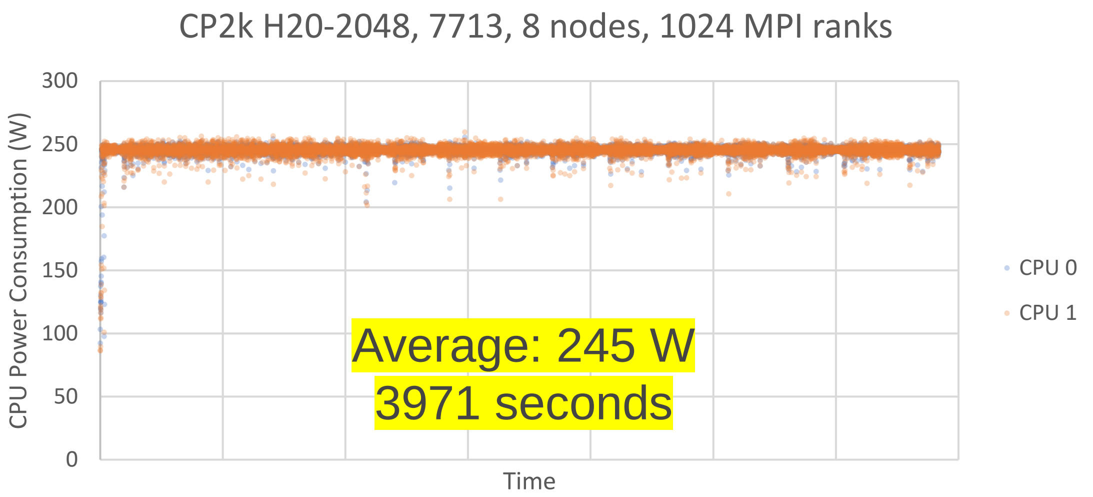
CP2k H20-2048, 7713, 8 nodes, 1024 MPI ranks



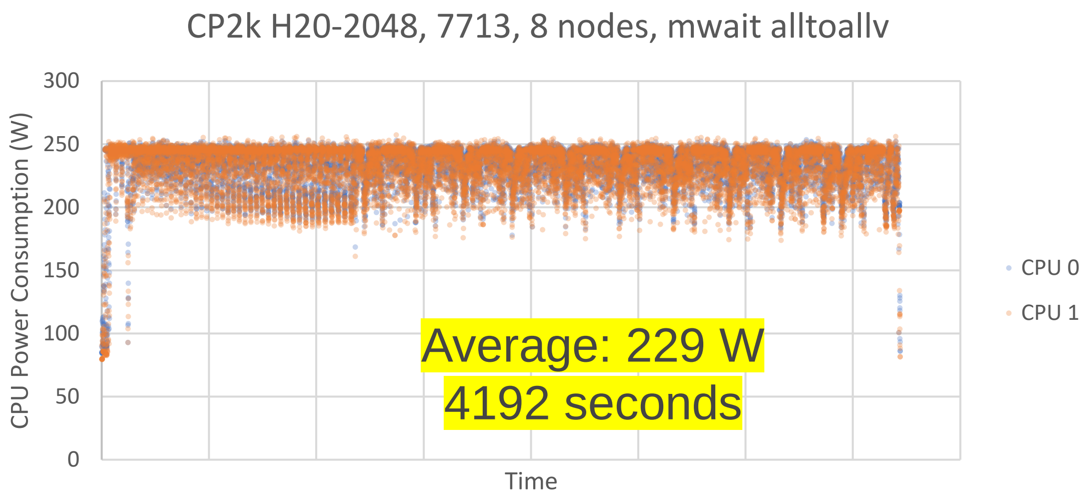
CP2k H20-2048, 7713, 8 nodes, 1024 MPI ranks, mwaitx

- 3% power savings *per* CPU

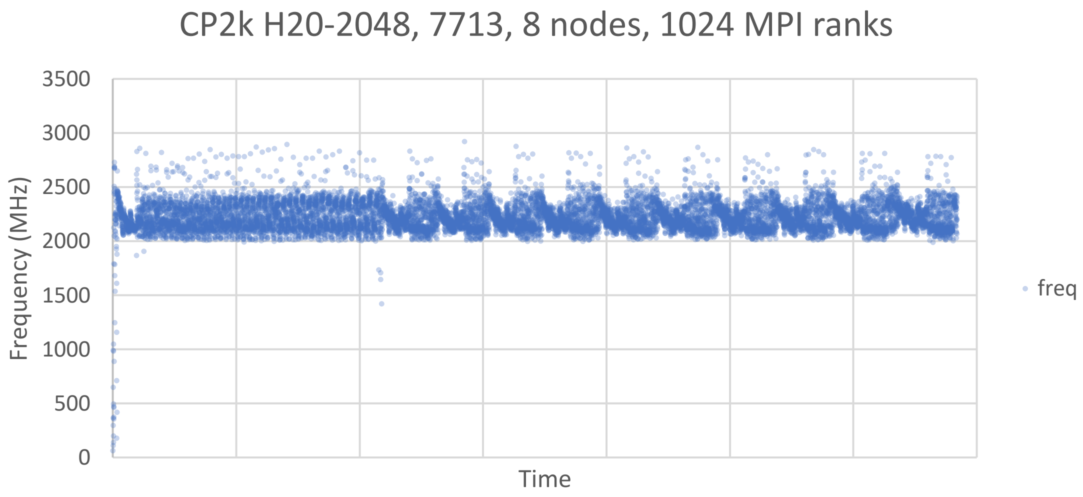
CP2k on AMD 7713 (240 W TDP) – MPI_Ialltoall

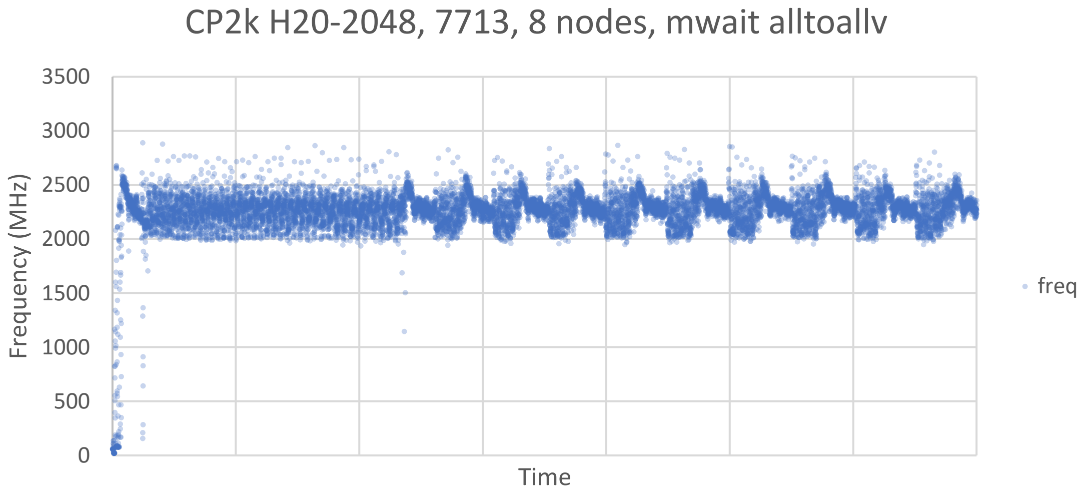

CP2k H20-2048, 7713, 8 nodes, 1024 MPI ranks

Average: 245 W
3971 seconds


CP2k H20-2048, 7713, 8 nodes, mwait alltoallv

Average: 229 W
4192 seconds


CP2k H20-2048, 7713, 8 nodes, 1024 MPI ranks

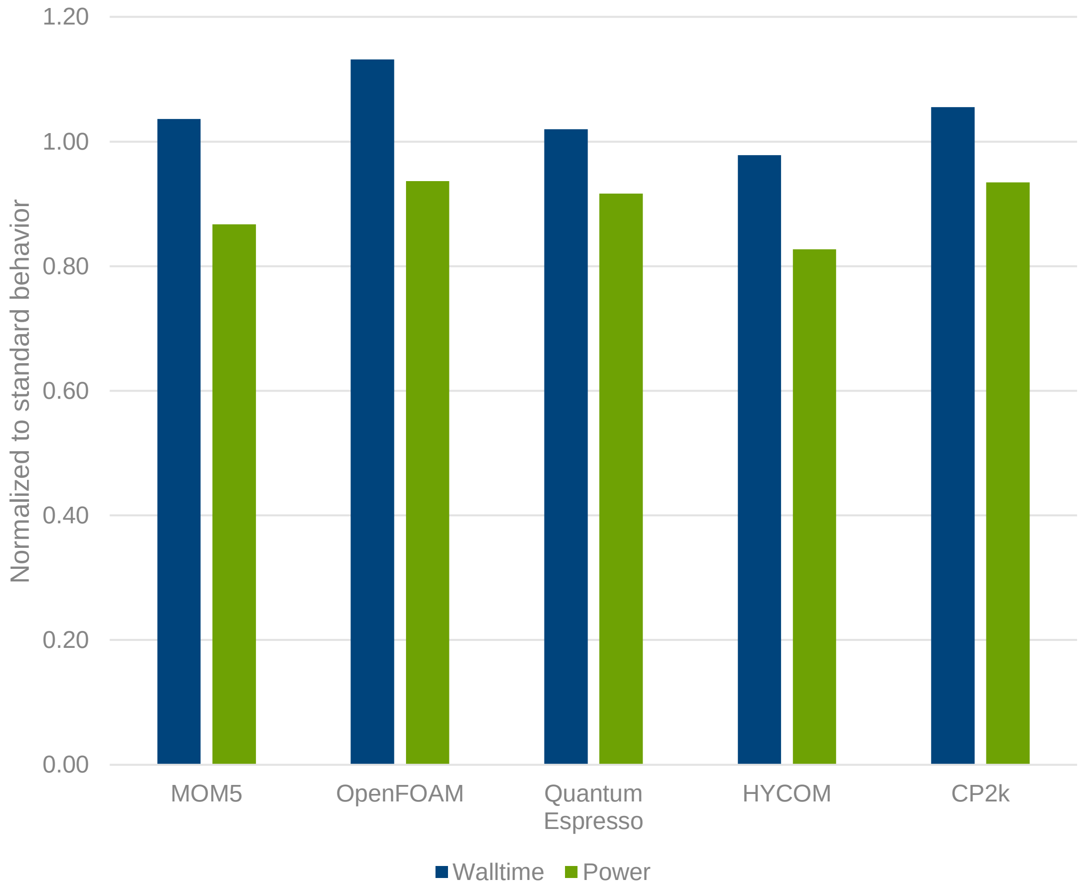

CP2k H20-2048, 7713, 8 nodes, mwait alltoallv

- 7% power savings *per* CPU

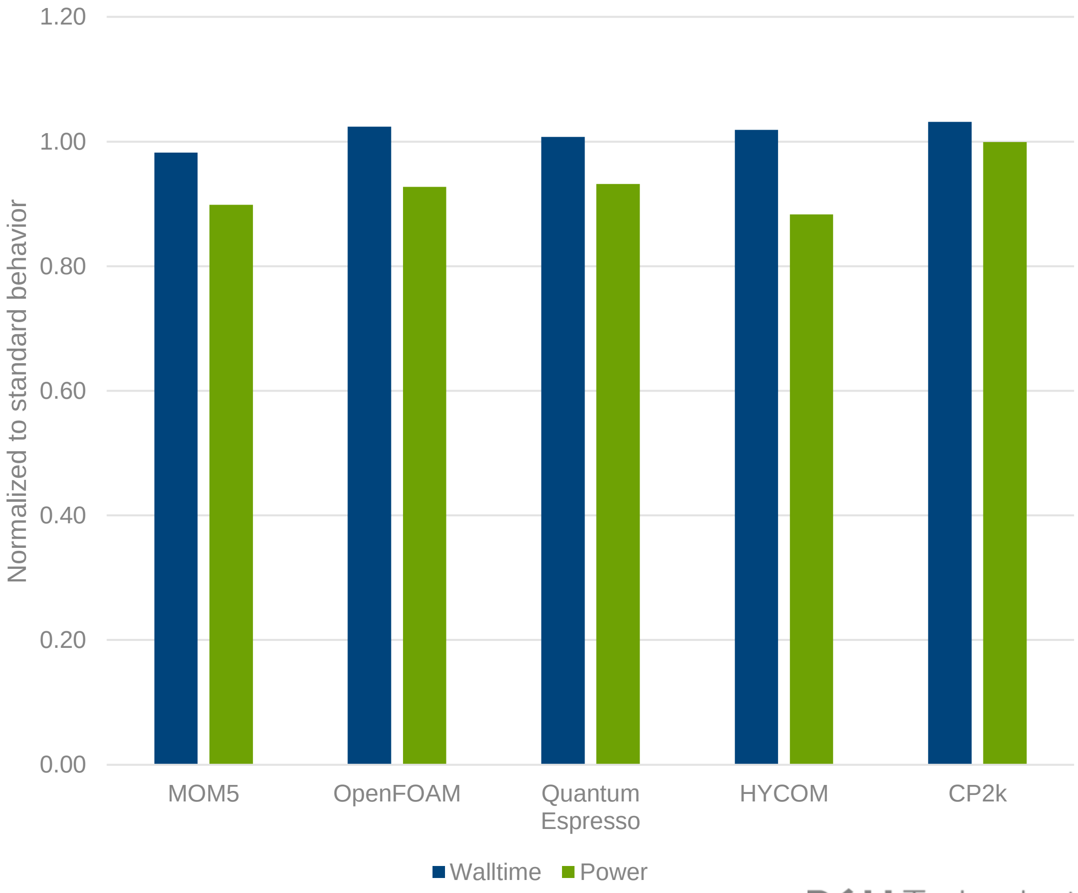
Summary

Usage of x86_64 MWAIT instruction can give power savings up to 17% for HPC apps that have load imbalance

AMD EPYC 7713 mwaitx



Intel Xeon 8480 umwait



DELL Technologies