PARALLEL PROGRAMMING PERFORMANCE TESTING

Matt Gordon

UWE 10026602

Introduction

This performance testing is designed to determine the effectiveness of three methods of AES brute force searching:

- Sequential
- OpenMP Parallelisation
- MPI Parallelisation

Unfortunately, due to a lack of correctly configured hardware, it is not possible to test MPI parallelisation distributed across multiple host computers.

Benchmark Setup

Tests will require each search method to start at a base key of:

0x 23 23 23 23 23 73 61 6D 00 00 00 00 23 23 23 23

Each search method will then increment this to the next valid key value (ensuring each byte contains a valid printable ASCII value between 32 and 126).

Only bytes 8 to 11 will be incremented to reduce search time. This along with the number of possible values for each byte means that the key should be found on key number 69282458.

A fixed IV will be used:

0x 01 02 03 04 05 06 07 08 09 00 0A 0B 0C 0D 0E 0F

The application is designed to measure the time taken for the search until the key is found (excluding any application set up time, but including search set up time such as key generation). The number of keys searched is also recorded to derive the number of keys the application is able to test per second.

The following tests will be conducted:

- Sequential
- OpenMP (1,2,3,4,5,6,7,8,16 threads)
- MPI (1,2,3,4,5,6,7,8,16 nodes)

Benchmarking

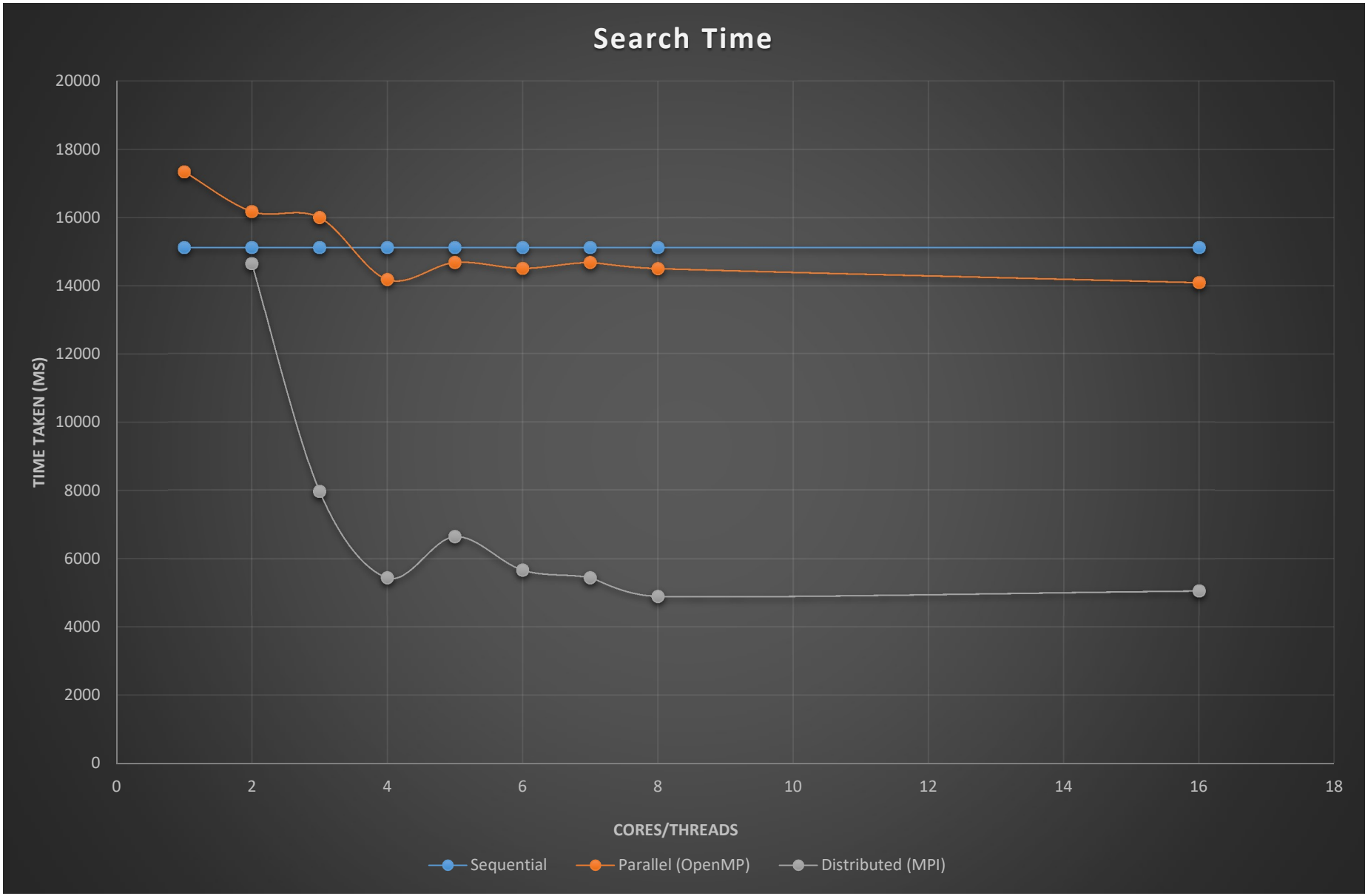
All tests were executed on the UWE hosted ‘parallel-comp-2’ which has 4 processor cores.

All tests ran successfully, and a sanity check was carried out where the tests were carried out a second time to ensure results were consistent within a reasonable degree of error.

Results

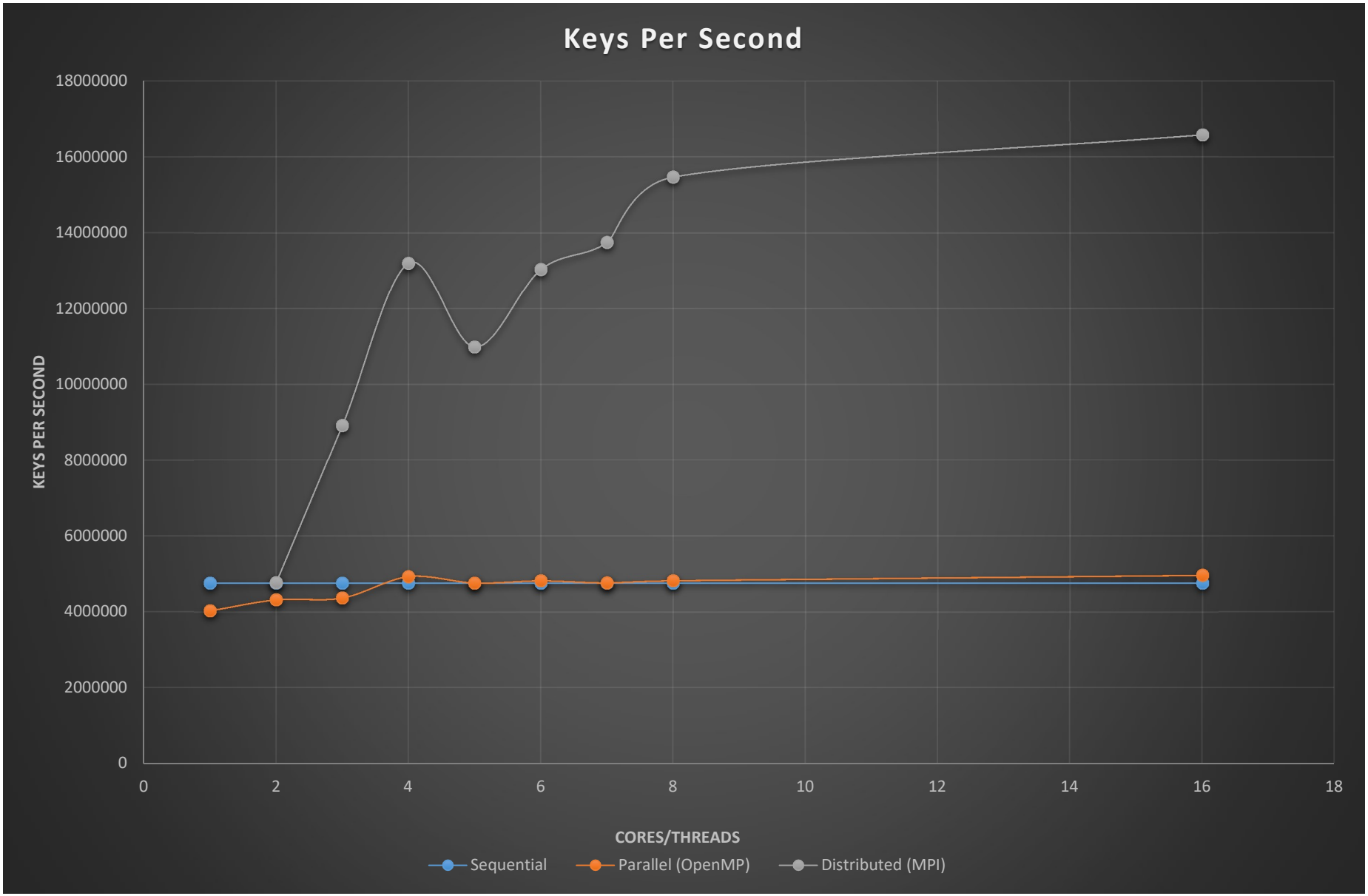
Overall Search Time

Search Time (ms)									
	Threads/Cores								
Search Method (69282458 keys)	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**16**
Sequential	15125	15125	15125	15125	15125	15125	15125	15125	15125
Parallel (OpenMP)	17343	16179	16003	14185	14685	14513	14683	14507	14093
Distributed (MPI)	N/A	14648	7960	5452	6646	5674	5453	4914	5068


Search Time
TIME TAKEN (MS)
20000
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
0
2
4
6
8
10
12
14
16
18
CORES/THREADS
Sequential
Parallel (OpenMP)
Distributed (MPI)

Keys Per Second

Keys Per Second									
	Threads/Cores								
Search Method (69282458 keys)	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**16**
Sequential	4761000	4761000	4761000	4761000	4761000	4761000	4761000	4761000	4761000
Parallel (OpenMP)	4036000	4326000	4374000	4934000	4766000	4823000	4767000	4825000	4967000
Distributed (MPI)	N/A	4778000	8919000	13206000	10984000	13041000	13753000	15466000	16574000


Keys Per Second
KEYS PER SECOND
18000000
16000000
14000000
12000000
10000000
8000000
6000000
4000000
2000000
0
0
2
4
6
8
10
12
14
16
18
CORES/THREADS
Sequential
Parallel (OpenMP)
Distributed (MPI)

Analysis

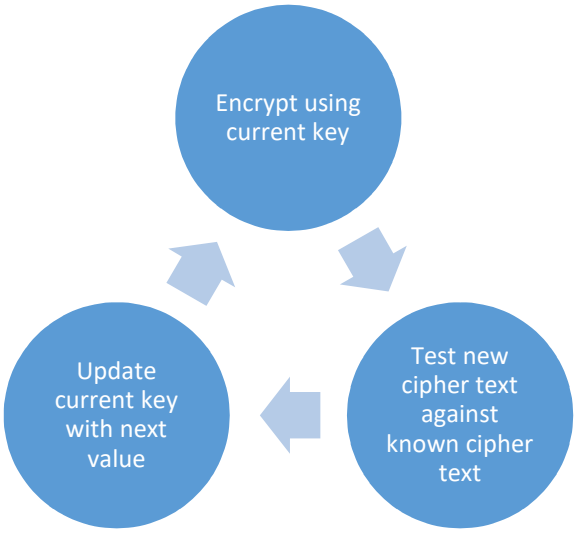
Sequential

The sequential search result shows that the 69M key combinations were able to be tested in a very efficient way, taking only 15.1 seconds (4.76M keys per second). This shows the sequential algorithm developed will serve as a realistic and consistent baseline for comparison of the parallel search results.

Parallel (OpenMP)

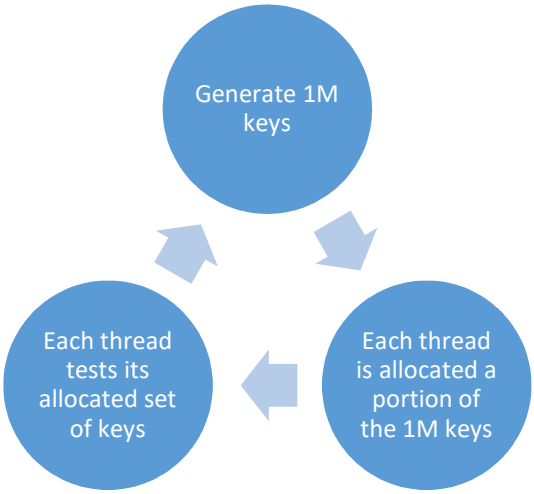
The observation regarding the OpenMP results is that when using 1 thread the time taken is increased significantly against the sequential method. This is likely due to the necessary difference in execution of these two methods.

The sequential algorithm searches data in the following way:



This was not possible for OpenMP parallelisation as multiple threads would need to access and increment the 'current key' at the same time leading to race conditions, corruption or data and missed keys.

To solve this, the OpenMP method searches in the following way:



Whilst necessary, the additional overhead and memory management (as well as standard OpenMP overhead including thread management) required to create and store keys in batches adds to the execution time significantly.

As the number of threads used increases, only minimal performance gains are seen. This is likely due to:

- Additional overhead due to thread management
- Batch key generation must be done sequentially and cannot be parallelised due to shared memory issues

When observing system monitor applications (e.g. TOP) in Linux, it was observed that CPU utilisation for the search process never exceeded 250% suggesting there are additional performance bottlenecks at the system level (possibly relating to thread management).

Parallel (MPI)

The MPI search method shows much greater performance gains than OpenMP, with an almost linear performance gain from 2 to 4 nodes.

At 2 nodes, the performance is near identical to that of the sequential method, this is due to only 1 of these nodes testing the keys, whilst the other is generating new 'start keys' ready for when other nodes have finished. Unlike the OpenMP batch key generation, the MPI function is able to use the same key generation technique that is used in the sequential function due to the way MPI operates using multiple processes, isolating memory and therefore providing additional performance gains over OpenMP.

Results after 4 nodes show non-linear decreases and increases which have shown to be repeatable in subsequent sanity tests. It is likely the reason for increased performance after 4 nodes is due to hyper threading functionality of the processor but will eventually reach an upper limit.

Conclusion

The controlled testing carried out has shown that using the AES-NI hardware accelerated extended x86 instruction set, exceptional performance can be achieved even through sequential search operation.

OpenMP parallelisation was disappointing, showing little performance gains and also requiring significant modification of the search algorithm so that it could be implemented.

MPI shows major performance gains, with linear improvement up to the number of cores present in the host and some additional minor performance gains after this due to hyper threading.