Science & Technology Facilities Council

Technical Report
RAL-TR-2016-005

Energy consumption of the Jacobi Test Code on the Blue Gene/Q: does using single precision reduce energy consumption?

T Byrne, M Mawson, AD Taylor, S Thorne

April 2016

Enquiries concerning this report should be addressed to:

RAL Library
STFC Rutherford Appleton Laboratory
Harwell Oxford
Didcot
OX11 0QX

Tel: +44(0)1235 445384
Fax: +44(0)1235 446403
email: libraryral@stfc.ac.uk

Science and Technology Facilities Council reports are available online at: http://epubs.stfc.ac.uk

ISSN 1358-6254

Energy consumption of the Jacobi Test Code on the Blue Gene/Q: does using single precision reduce energy consumption?

Thomas Byrne[1], Mark Mawson[2], Andrew D. Taylor[2], Sue Thorne[13]

ABSTRACT

Power optimisation is one of the major requirements for the achievement of exascale systems. In this report, we will compare the power and energy requirements of the Jacobi Test Code when run in single and double precision on STFC Hartree Centre's IBM Blue Gene/Q, Blue Joule. We will show that switching from double to single precision can give significant energy consumption savings but only when the amount of data movement is significant. As a result, algorithm/software developers should take this into account when trying to reduce the amount of energy consumed by running their codes on computers with similar underlying architectures.

This work forms part of the EPSRC Service Level Agreement funded Software Outlook project.

Keywords: Energy Efficient Computing, Blue Gene/Q, Jacobi Test Code, Single Precision, Double Precision, CCP

March 17, 2016

[1]Scientific Computing Department, Science and Technology Facilities Council, Rutherford Appleton Laboratory, Harwell Campus, Didcot, OX11 0QX, UK.

[2]Scientific Computing Department, Science and Technology Facilities Council, Daresbury Laboratory, Sci-Tech Daresbury, Daresbury, Warrington, WA4 4AD, UK.

[3]Correspondence to: sue.thorne@stfc.ac.uk

1 Background and motivation

Power optimisation is one of the major requirements for the achievement of exascale systems. Moving from petascale to exascale computing, the number of floating point operations per second needs to increase by a factor of 1000 but the power consumption of the required supercomputer is limited to just a factor of 10 increase [3].

In this report, we will compare the power and energy requirements of the Jacobi Test Code [2] when run in single and double precision on STFC Hartree Centre's IBM Blue Gene/Q, Blue Joule [1]. If we discover significant savings by switching from double to single precision, algorithm/software developers need to take this into account to try to reduce the amount of energy consumed when running their codes on computers with similar underlying architectures. It is important to note that it is not uncommon for codes to be run with double precision accuracy when the underlying problem is of much lower accuracy and, hence, time and energy may have been wasted by computing a solution to many more significant figures than necessary.

1.1 Jacobi Test Code

The Jacobi Test Code (JTC) provides several implementations of the Jacobi algorithm for solving a fairly simple 3D partial differential equation (Poisson's equation on a cuboid) [2]: we will concentrate on the *baseline-opt* implementation and, in the rest of the paper, refer to this as `JTC_CORE`. The baseline implementation of the Jacobi algorithm consists of three nested loops: *baseline-opt* uses basic loop optimisations and, in addition, the version used for the tests in this report uses an optimised form of the inner loop to enable the compiler to vectorise the loop. The basic outline of the JTC is given in Algorithm 1.

Algorithm 1 Jacobi Test Code outline

Set-up test problem and output initial starting vector v_0
for $k = 1, \ldots, nruns$ **do**
  for $i = 1, \ldots, niter$ **do**
    $v_i =$ `JTC_CORE`(v_{i-1})
  end for
end for

Note that, within `JTC_CORE`, each entry in v_i is formed by averaging the values in v_{i-1} at neighbouring grid points. In many application codes, the current value at a particular grid point is formed by looking at values on neighbouring grid points and, hence, the JTC is a good representation of the work performed in the kernel of many codes.

1.2 Blue Gene/Q

In this report, we are only comparing the differences in power and energy consumption when running the JTC in single or double precision on Blue Joule, STFC Hartree Centre's IBM Blue Gene/Q. The Blue Gene/Q has a provided energy monitoring system called the EMON API. On each Blue Gene/Q Node-board, which is a collection of 32 Blue Gene/Q nodes, contains an FPGQ that records instantaneous power consumption with a sampling frequency of 0.25s. Each BG/Q node has 16 cores. To obtain the information from the EMON API, we use EMONSimple, a simple energy monitoring library for the Blue Gene/Q, which provides traces of power and energy consumption versus time. EMONSimple provides energy/power consumption details for seven energy/power domains:

- A2 Node (CPU execution units, L1 prefetcher, crossbar, L2 DRAM)

- Main Memory and IO drivers
- Optical Modules 1
- Optical Modules 2
- High Speed Serial cores and drivers
- SRAM Array including L1 Cache
- Link Chip

In order to understand the results produced by EMONSimple, we also use the TAU profiler [4] to provide us with further information, for example, the number of L1 and L2 cache misses. Unfortunately, the overhead from running TAU means that some of the output, especially L2 cache misses, maybe distorted but we hope that it will give us some further insights.

Note: all execution times are wall clock times.

2 Energy consumption of the JTC with OpenMP

In this section, we will concentrate our investigation on the OpenMP version of the JTC. We start by considering what happens to the execution time and energy consumption as we change the problem size and alter the number of OpenMP threads. Specifically, we will solve Poisson's equation on a cube where each side has g grid points, $g = 32, 42, 52, \ldots, 492$, and use $nthr = 1, 2, 4, 8, 16$, where $nthr$ is the number of OpenMP threads. Additionally, we use $niter = 50$ and $nruns = 10$.

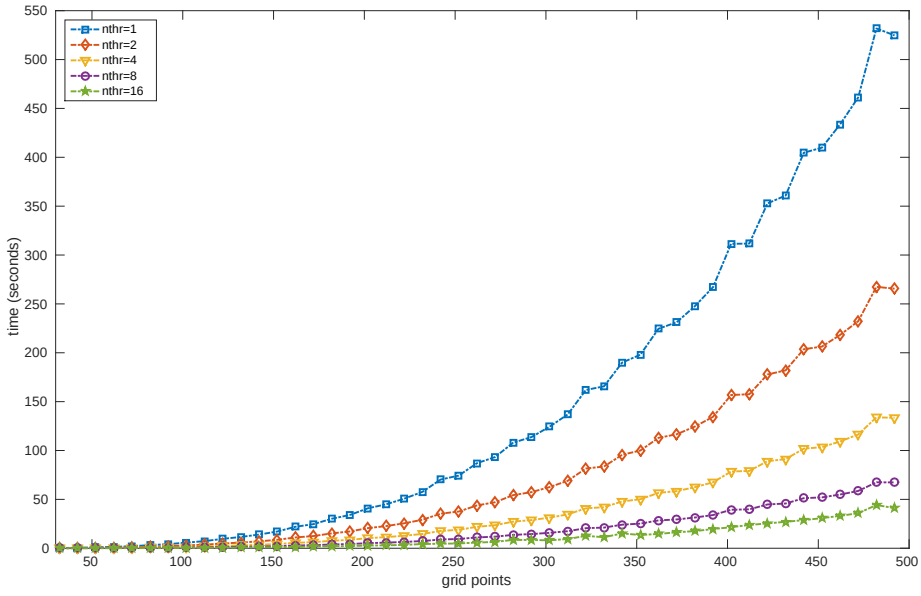

Figure 1: The execution time for running the JTC in single precision with different numbers of grid points and $nthr$.

In Figures 1 and 2, we compare the execution time and energy time, respectively, for the varying values of $nthr$ and number of grid points using the single precision version of the code. As we would expect, increasing the number of threads decreases the execution time (with it being close to linear speed-up) and,

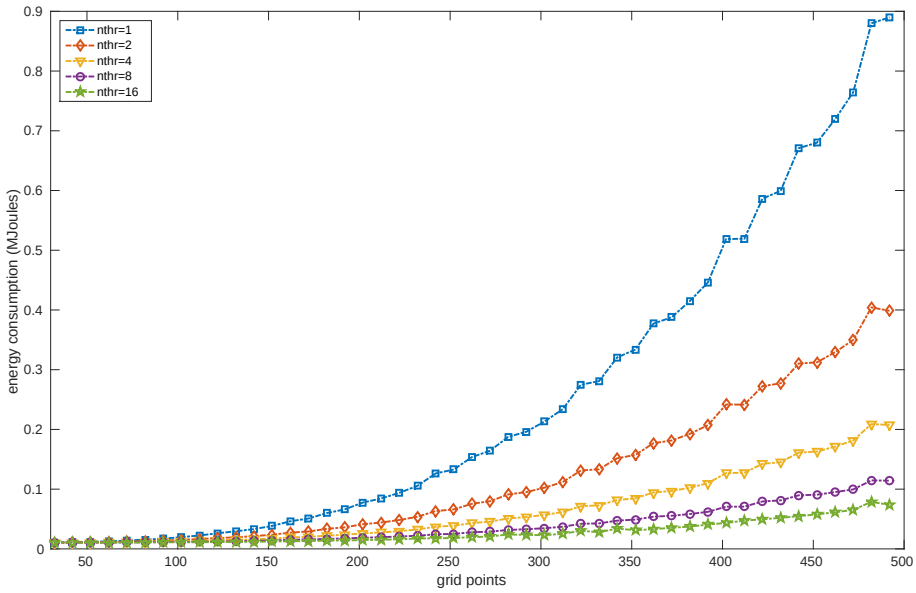

Figure 2: The total energy consumption for running the JTC in single precision with different numbers of grid points and *nthr*.

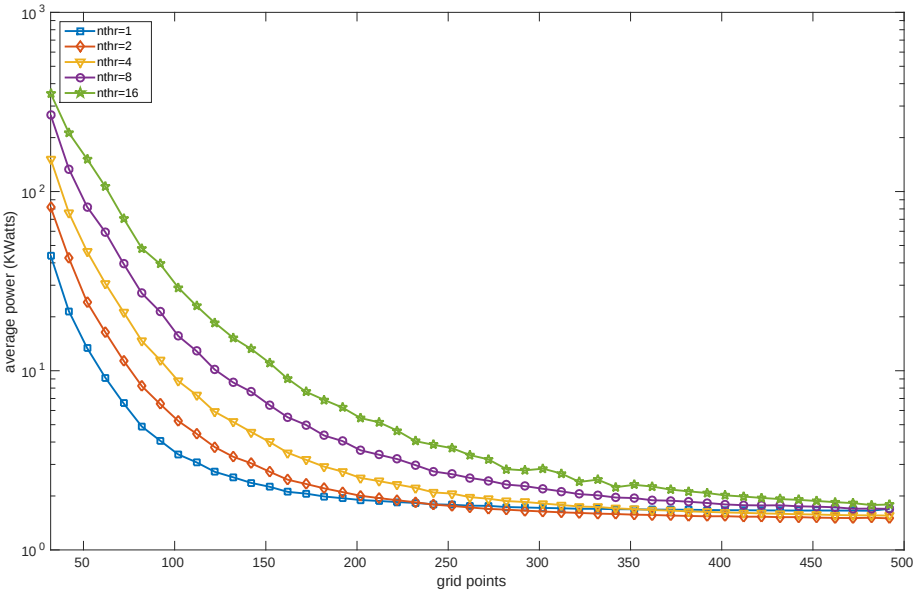

Figure 3: The average power usage for running the JTC in single precision with different numbers of grid points and *nthr*.

for large enough problem sizes, the execution time increases linearly with the problem size (note that the problem size is equal to the cube of the number of grid points). If the energy consumption of JTC runs were directly related to the execution time, then we would expect Figures 1 and 2 to look similar and

they are very similar, but there are subtle differences. We compare the average power usage in Figure 3. If the energy consumption was proportional to the execution time, then the average power usage would be identical for all problems sizes and all the different numbers of threads. However, as the problem size increases, the average power usage decreases and, for $nthr > 1$, increasing $nthr$ increases the average power consumption: this will be due to the increase in data movement. To understand this further, we look at the number of L1 and L2 cache misses when using 500 grid points, $niter = 50$ and $nruns = 10$. Within the set-up phase, the JTC initialises the vector v_0. In Table 1, we report the minimum, mean and maximum number of L1/L2 cache misses across the threads and also report the total number of cache misses for doing this initialisation. In Table 2, we report the same information but for the main component within `JTC_CORE`. We first observe that, as we would expect due to the problem size versus the size of the L1 data cache (16KB), the total number of L1 cache misses remains constant when both initialising v_0 and performing the main component within `JTC_CORE`. When initialising v_0, the total number of L2 data cache misses decreases due to data being shared out but it is important to note that the total number is not inversely proportional to $nthr$. For the main component of `JTC_CORE`, there is a significant increase in L2 data cache misses when moving from $nthr = 2$ to $nthr = 4$, which corresponds to the increase in data traffic between threads. Unfortunately, we do not have energy consumption information across the different power domains for running the JTC with 500 grid points and Blue Joule is not currently available for use. However, we do have these figures for the JTC with 492 grid points (we do not have cache data for this size of problem). In Table 3, we compare where the energy is used by the node board and, whilst the percentages do not vary much as the number of threads increases, it is possible to see that there is a slight increase in the percentage of energy being used by the main memory, which also corresponds to the increase in L2 data cache misses and, therefore, increased use of the main memory. However, the A2 node is by far the largest consumer of energy.

	L1 cache misses (10^{10})				L2 cache misses (10^7)			
nthr	min	mean	max	total	min	mean	max	total
1	1.3235	1.3235	1.3235	1.3235	9.5912	9.5912	9.5912	9.5912
2	0.5648	0.5752	0.5857	1.1504	2.7087	3.1862	3.6637	6.3724
4	0.2497	0.2744	0.3119	1.0977	1.0632	1.1640	1.3358	4.6560
8	0.1246	0.1426	0.1544	1.1406	0.4129	0.4323	0.4538	3.4580
16	0.0744	0.0811	0.0876	1.2971	0.1292	0.1353	0.1479	2.1642

Table 1: The minimum, mean and maximum number of L1/L2 cache misses across the threads and the total number of cache misses for doing the initialisation of v_0 with different values of $nthr$ using single precision and 500 grid points.

	L1 cache misses (10^{10})				L2 cache misses (10^9)			
nthr	min	mean	max	total	min	mean	max	total
1	2.0061	2.0061	2.0061	2.0061	1.5172	1.5172	1.5172	1.5172
2	1.0035	1.0035	1.0035	2.0071	0.0003	0.7551	1.5099	1.5102
4	0.5018	0.5018	0.5018	2.0071	0.0121	0.5812	1.5019	2.3250
8	0.2508	0.2509	0.2509	2.0071	0.0003	0.3663	1.4818	2.9306
16	0.1255	0.1255	0.1255	2.0074	0.0005	0.1566	0.9186	2.5057

Table 2: The minimum, mean and maximum number of L1/L2 cache misses across the threads and the total number of cache misses for main component within `JTC_CORE` with different values of $nthr$ using single precision and 500 grid points.

In Figures 4 and 5, we compare the execution time and total energy consumption for running the JTC

nthr	A2 node	main memory	HSS	SRAM
1	60.48	14.78	12.40	2.85
2	59.62	15.01	12.73	2.93
4	59.66	15.09	12.67	2.91
8	60.39	15.03	12.31	2.84
16	59.49	15.40	12.60	2.90

Table 3: Percentage of energy consumption by the A2 node, main memory and IO drivers, high speed serial cores and drivers, and the SRAM array domains when running the JTC in single precision with 492 grid points and different values of *nthr*.

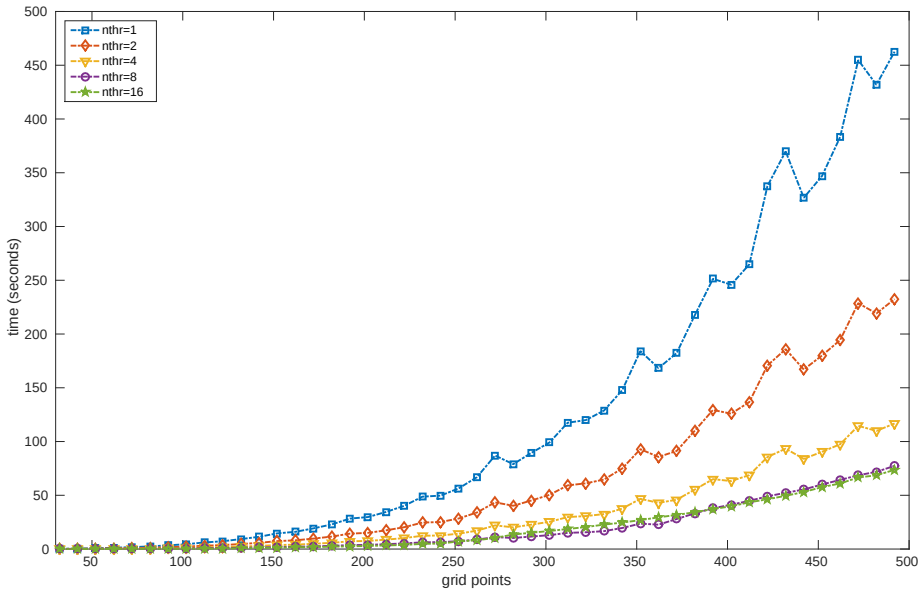

Figure 4: The execution time for running the JTC in double precision with different numbers of grid points and *nthr*.

in double precision with different numbers of threads, different numbers of grid points, $niter = 50$ and $nruns = 10$. We observe that the execution time and total energy consumption behave similarly. However, the trends in the figures are different to those when single precision is used. Most notably, when using double precision, there is little difference in execution time and energy consumption between $nthr = 8$ and $nthr = 16$. Tables 4 and 5 contain the number of L1 and L2 cache misses when initialising v_0 and performing the main component within `JTC_CORE`, respectively, using 500 grid points, $niter = 50$ and $nruns = 10$. As for the single precision case, the total number of L1 cache misses remain almost constant as we increase the number of threads and, for the initialisation of v_0, the total number of L2 data cache misses decreases as the number of threads increases. For $nthr = 1$ and $nthr = 2$, the total number of L2 data cache misses for the main component within `JTC_CORE` is nearly identical. Increasing the number of threads to four, increases the number of L2 data cache misses by a third and there is a slight drop by further increasing to $nthr = 8$. Finally, $nthr = 16$ has over double the number of L2 data cache misses compared to $nthr = 8$. This corresponds to the increasing amount of data traffic as we increase the number of threads and, as a consequence, the execution times and energy consumption values are similar for these two values of *nthr*.. In Table 6, we compare the percentage energy consumption by some of the power

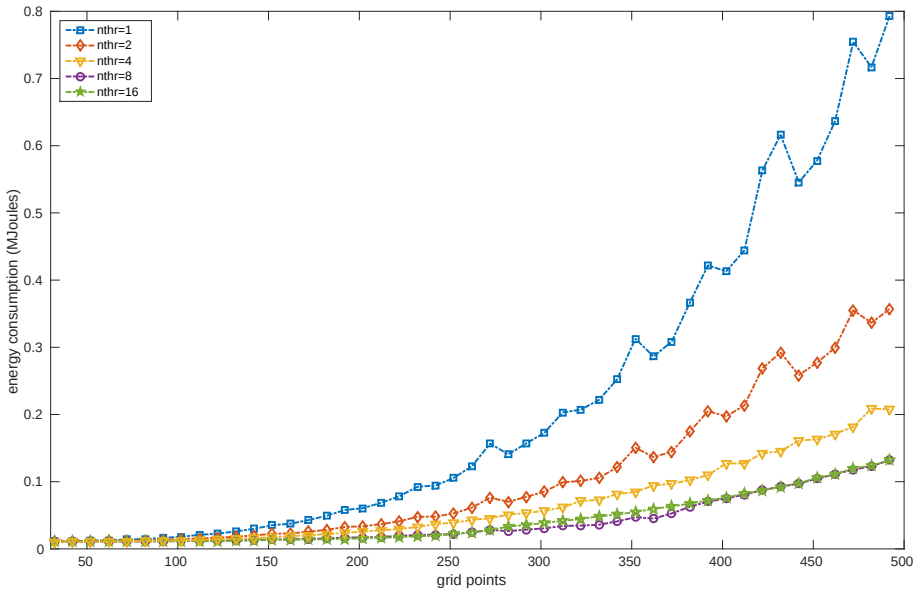

Figure 5: The total energy consumption for running the JTC in double precision with different numbers of grid points and *nthr*.

domains on the node board. We observe that, as with the single precision version, increasing the number of threads increases the proportion of the energy being consumed by the main memory. This is as we expect because of the increased data movement and L2 data cache misses.

	L1 cache misses (10^{10})				L2 cache misses (10^{8})			
nthr	min	mean	max	total	min	mean	max	total
1	1.2444	1.2444	1.2444	1.2444	1.0084	1.0084	1.0084	1.0084
2	0.6119	0.6160	0.6201	1.2321	0.2829	0.3223	0.3618	0.6447
4	0.0018	0.2191	0.3287	0.8765	0.1142	0.1162	0.1189	0.4655
8	0.1328	0.1464	0.1566	1.1711	0.0420	0.0532	0.1176	0.4258
16	0.0736	0.0792	0.0858	1.2669	0.0131	0.0138	0.0151	0.2209

Table 4: The minimum, mean and maximum number of L1/L2 cache misses across the threads and the total number of cache misses for doing the initialisation of v_0 with different values of *nthr* using double precision and 500 grid points.

In Figures 6 and 7, we provide the execution time and total energy consumption ratios for single versus double precision, respectively, varying values of *nthr* and number of grid points. Note that, for given number of grid points and *nthr*, the same number of floating point operations are performed on each thread. Start by observing that when $nthr = 1$, it is faster to run the same code in double precision instead of single precision as well as being more energy efficient, which goes against the normal assumption that single precision operations will run at twice the speed of double precision operations. In fact, the Blue Gene/Q uses Quad-Process eXtension, which means that single precision arithmetic is actually performed in double precision and the result converted back to single precision. Therefore, each single precision operation takes longer than a double precision operation and requires more energy but data movement between threads should be faster for single precision reals compared to double precision reals. Additionally,

	L1 cache misses (10^{10})				L2 cache misses (10^9)			
nthr	min	mean	max	total	min	mean	max	total
1	3.1248	3.1248	3.1248	3.1248	2.8332	2.8332	2.8332	2.8332
2	1.5711	1.5712	1.5712	3.1424	0.0139	1.4123	2.8108	2.8246
4	0.7856	0.7856	0.7856	3.1423	0.1128	1.0459	2.3328	4.1835
8	0.3907	0.3926	0.3929	3.1406	0.0013	0.4842	3.7982	3.8739
16	0.1964	0.1965	0.1965	3.1436	0.0025	0.5113	4.6825	8.1807

Table 5: The minimum, mean and maximum number of L1/L2 cache misses across the threads and the total number of cache misses for main component within `JTC_CORE` with different values of *nthr* using double precision and 500 grid points.

nthr	A2 node	main memory	HSS	SRAM
1	60.38	15.10	12.29	2.83
2	60.28	15.16	12.31	2.83
4	60.31	15.06	12.35	2.84
8	60.13	15.42	12.26	2.82
16	60.09	15.66	12.15	2.80

Table 6: Percentage of energy consumption by the node board, A2 node, main memory and IO drivers, high speed serial cores and drivers, and the SRAM array domains when running the JTC in double precision with 492 grid points and different values of *nthr*.

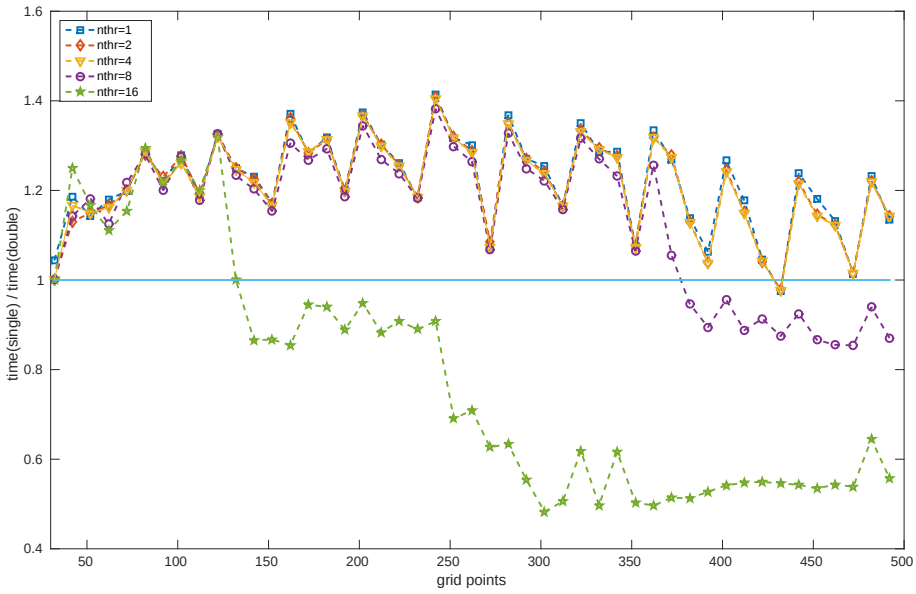

Figure 6: Ratio of execution time for running the JTC in single versus double precision with different numbers of grid points and *nthr*.

we will expect the number of cache misses to be substantially lower for the single precision variant because more values can be stored within the L1 and L2 caches.

For the case $nthr = 1$, the only data movement will be due to cache misses and, for the JTC, the time

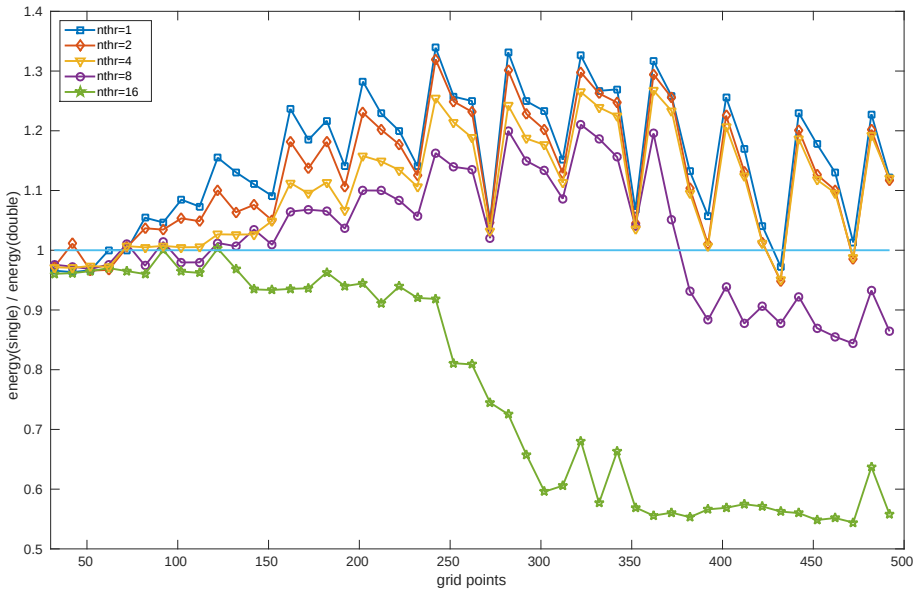

Figure 7: Ratio of total energy consumption for running the JTC in single versus double precision with different numbers of grid points and *nthr*.

spent doing floating point operations dominates the time spent moving data around. Thus, the execution time and energy consumption are, respectively, up to 40% and 35% greater for the single precision version compared to the double precision version when $nthr = 1$. For the data movement costs to start to dominate the floating point operation costs, we need to increase in the number of threads. As *nthr* is increased, we will expect the data movement due to the need for threads to communicate to increase. In addition, the floating point operations on each thread will approximately half each time the number of threads doubles.

For $nthr = 2$ and 4, the ratios for execution time are very similar to those of $nthr = 1$. However, increasing the number of threads reduces the ratio in terms of energy consumption. If we compare the number of L1 cache misses for the core component of `JTC_CORE` with 500 grid points, we observe that there are approximately 50% more misses for the double precision version; for the L2 data cache misses, there are 80-90% more misses for $nthr = 1, 2$ and 4. In addition, increasing the number of threads results in the data movement becoming more dominant when running the JTC and, hence, we obtain the decrease in ratios.

For $nthr = 8$ and less than 380 grid points, the double precision version is preferable in terms of execution time and energy consumption. Increase the number of threads beyond 380 and the single precision version becomes preferable. We expect that this is due to the problem size resulting in a lot more cache misses. Unfortunately, we had not collected the cache data required to support this hypothesis before Blue Joule became unavailable.

Setting $nthr = 16$ and using less than 132 grid points, the execution time is generally between 15 and 35% larger when using single precision instead of double precision but the energy consumption is decreased by 4%. However, the interesting cases are for the larger numbers of grid points. Here, the single precision version provides about a 45% reduction in execution time and energy consumption. For 500 grid points, we note that there is a 226% increase in L2 data cache misses in the main component of `JTC_CORE` when using double precision instead of single precision and it is these large differences in cache misses that make the single precision variants preferable.

In the above tests, we set $nruns = 10$ and altered the number of grid points. Suppose that we fix the

number of grid points to 452 and alter $nruns$. We can express the wall clock time, t_{tot}, and total energy consumption, e_{tot}, as

$$t_{tot}(nruns) \approx t_{setup} + nruns \times t_{Jacobi}$$

and

$$e_{tot}(nruns) \approx e_{setup} + nruns \times e_{Jacobi},$$

where t_{setup} and e_{setup} are the time and energy consumed setting-up the test problem, respectively, and t_{Jacobi} and e_{Jacobi} are the time and energy consumed when performing the inner loop of Algorithm 1 with $niter = 50$, respectively, Using $nruns = 5, 10, \ldots, 50$, we can compute t_{setup}, t_{Jacobi}, e_{setup} and e_{Jacobi} by minimizing

$$\sum_{i=1}^{10} \left(t_{tot}(5i) - t_{setup} - nruns \times t_{Jacobi}\right)^2 \tag{1}$$

and

$$\sum_{i=1}^{10} \left(e_{tot}(5i) - e_{setup} - nruns \times e_{Jacobi}\right)^2 . \tag{2}$$

In Tables 7 and 8, we provide the computed values of t_{setup} and t_{Jacobi}, and e_{setup} and e_{Jacobi}, respectively, for both single and double precision. We observe that it takes slightly less time to set-up the problem in double precision compared to single precision. For $nthr = 1$, the Jacobi iterations take 20% longer to execute in single precision compared to double. As the value of $nthr$ increases, the time difference between the single and double precision versions for performing the Jacobi iterations reduces and, for $nthr = 8$, the single precision version is 15% faster than the double precision version. For $nthr = 16$, the time to perform the Jacobi iterations in single precision is half the time for the double precision version. The ratios differ when we consider the energy consumption, which, again, shows that the energy consumption is not proportional to the execution time. For $nthr = 1$, 22% more energy is required to set up the problem in single precision compared to double precision: this is due to the overhead in performing single precision operations compared to double precision ones. The energy consumed when performing 50 Jacobi iterations is 16% higher in the single precision case. For $nthr = 2$ and 4, the energy consumed by running 50 Jacobi iterations is 15% higher for the single precision version. However, the ratio decreases as we move to 8 threads, with the set-up phase using 17% less energy when using single precision instead of double precision. For $nthr = 16$, the energy consumed by the 50 Jacobi iterations in the single precision version is just 54% of that for the double precision. It is important to note that these ratios can vary a lot for other numbers of grid points because of cache behaviour. Note that 452 grid points was chosen because it exhibited some of the worst cases with respect to single precision's energy consumption over the larger problems.

	single		double		ratio (single/double)	
$nthr$	t_{setup}	t_{Jacobi}	t_{setup}	t_{Jacobi}	t_{setup}	t_{Jacobi}
1	42.17	36.77	41.71	30.51	1.011	1.205
2	22.12	18.43	21.78	15.82	1.016	1.165
4	11.21	9.22	11.01	7.94	1.018	1.161
8	5.83	4.62	5.67	5.44	1.029	0.850
16	3.11	2.79	2.92	5.47	1.067	0.510

Table 7: Computed values of t_{setup} and t_{Jacobi} for single and double precision version of JTC with 452 grid points and different values of $nthr$, and the ratio of single/double.

We can conclude that, using OpenMP on a Blue Gene/Q, we will only start to get benefits from using single precision over double precision, in execution times and energy consumption, if the costs of data movement outweigh the additional costs of performing the floating point operations. However, we also

	single		double		ratio (single/double)	
$nthr$	e_{setup}	e_{Jacobi}	e_{setup}	e_{Jacobi}	e_{setup}	e_{Jacobi}
1	0.1253	0.0598	0.1025	0.0517	1.222	1.157
2	0.0133	0.0300	0.0103	0.0259	1.291	1.158
4	0.0106	0.0152	0.0112	0.0131	0.946	1.160
8	0.0099	0.0080	0.0119	0.0090	0.832	0.889
16	0.0109	0.0049	0.0116	0.0091	0.940	0.538

Table 8: Computed values of e_{setup} and e_{Jacobi} for single and double precision version of JTC with 452 grid points and different values of $nthr$.

observe that, whilst the execution time is a clear factor in determining the energy consumption when data movement is at a minimum, the cost of data movement, often associated with cache misses, is a significant factor when the amount of data movement becomes large.

3 Energy consumption of the JTC with MPI

As well as there being an OpenMP version of the JTC, there is also a MPI version. Within the MPI version, the cuboid domain is divided up into equal-sized cuboids: the user decides how this division is done by specifying p_x, p_y and p_z, which specify how many regions to split each of the x, y and z axis, respectively, into. Note that the total number of MPI processes, p, satisfies $p = p_x p_y p_z$.

In this section, we will investigate the energy consumption of the JTC when running with different numbers of MPI processes and different configurations of p_x, p_y and p_z.

We start by considering the single precision version of the JTC and let $p = 2^i$, $i = 0, \ldots, 5$. For each value of p, we choose p_x, p_y and p_z to be powers of 2 such that $p_x \geq p_y \geq p_z$ and $p_x \leq 2p_z$. In Figures 8 and 9, we note that, as expected, increasing the number of MPI processes gives a drop in execution times and there is also a drop in energy consumption. Interestingly, the switch between $p = 8$ and $p = 16$ gives a much smaller drop in energy consumption than any of the other times that p is doubled.

In Figures 10 and 11, we consider the double precision version of the JTC. As with the single precision case, we observe significant drops in the execution time and energy consumption when moving between $p = 1$, 2, 4 and 8, but the execution time and energy consumption for $p = 8$ and $p = 16$ are almost identical. There is then a drop in energy consumption when moving from $p = 16$ to $p = 32$. Increasing the value of p, increases the amount of communication required between processes and, hence, the amount of data movement increases. From our results, we can conclude that the energy running the CPU execution units dominates the energy requirements for data movement when $p < 16$. Thus, doubling the number of processes approximately halves the number of floating point operations being performed on a single CPU, hence, the time spent operating the CPU approximately halves. At the same time, the amount of data movement will increase, meaning that the total energy consumption is not inversely proportional to the number of processes.

We compare the use of single and double precision in terms of execution time and energy consumption in Figures 12 and 13, respectively. As with the OpenMP version of the JTC, for large problems, the double precision version is preferable when the the level of data movement is low compared to the floating point operations ($p \leq 8$). For $p = 16$, the execution time and energy consumption are roughly the same but, by setting $p = 32$, the single precision version gives a 17-18% reduction in execution time and a 15-17% reduction in energy consumption.

As in Section 2, we can fix the number of grid points and alter $nruns$ to determine the execution time and energy consumption of the set-up phase and 50 Jacobi iterations. Again, we will use 452 grid points and data from $nruns = 5, 10, \ldots, 50$ to compute t_{setup}, t_{Jacobi}, e_{setup} and e_{Jacobi} by solving Equations 1 and 2. In Tables 9 and 10, we give the computed values of t_{setup}, t_{Jacobi}, e_{setup} and e_{Jacobi} for different

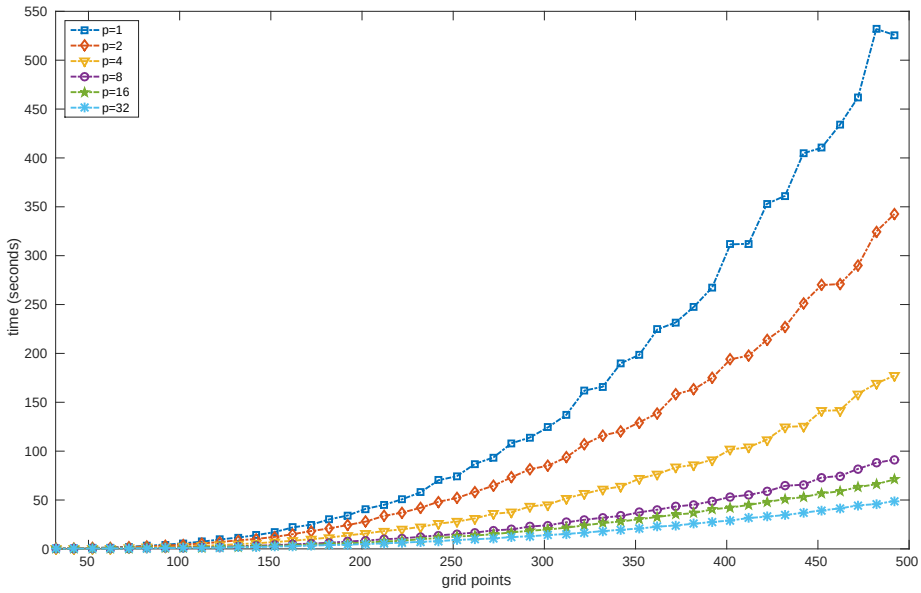

Figure 8: The execution time for running the JTC in single precision with different numbers of grid points and p.

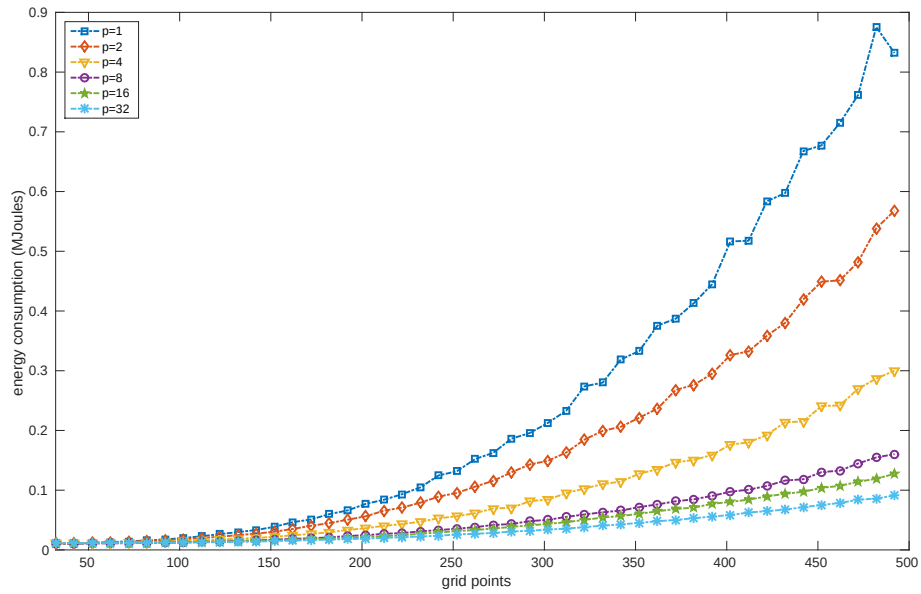

Figure 9: The total energy consumption for running the JTC in single precision with different numbers of grid points and p.

values of p. We start by observing that there is little difference in the set-up time for a given value of p when comparing single and double precision. However, for $p > 1$, the amount of energy consumed by the set-up stage is much lower for the single precision case: just 10% of the double precision's energy

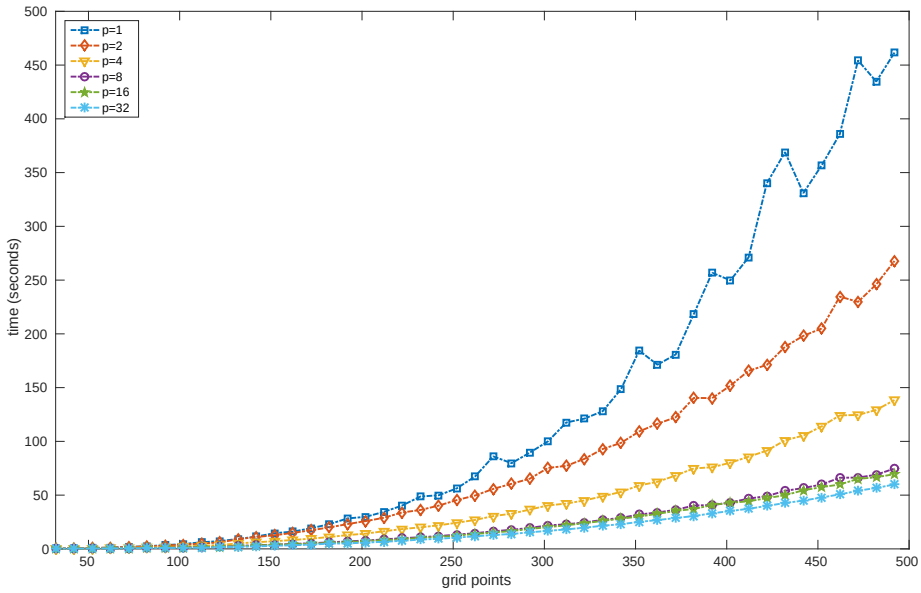

Figure 10: The execution time for running the JTC in double precision with different numbers of grid points and p.

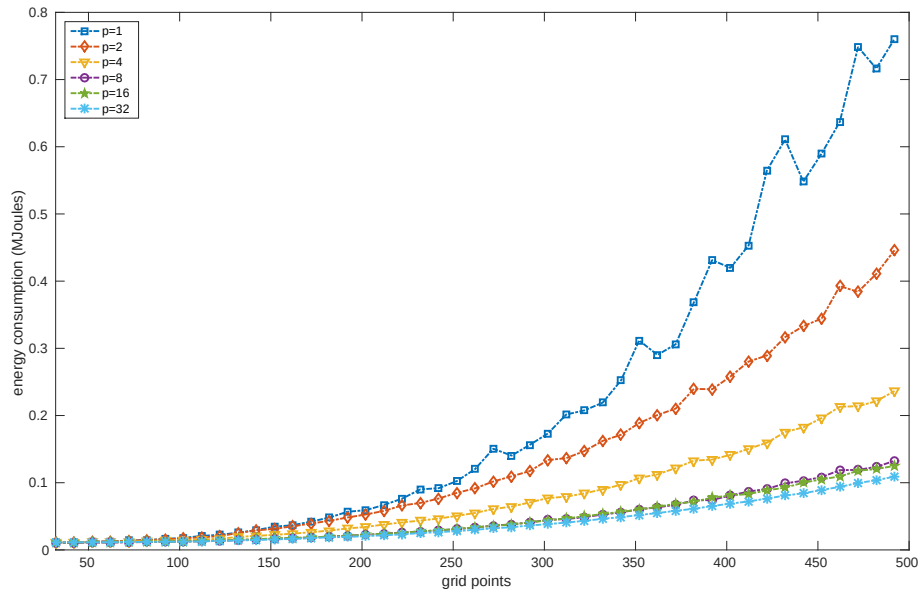

Figure 11: The total energy consumption for running the JTC in double precision with different numbers of grid points and p.

consumption for $p = 8$ and $p = 16$. For the Jacobi iteration phase, when $p = 1, 2, 4, 8$, the execution time of the single precision variant is between 17 and 35% larger than the double precision variant whilst the energy consumption is increased by between 15 and 36%. For $p = 16$, the increased data movement results

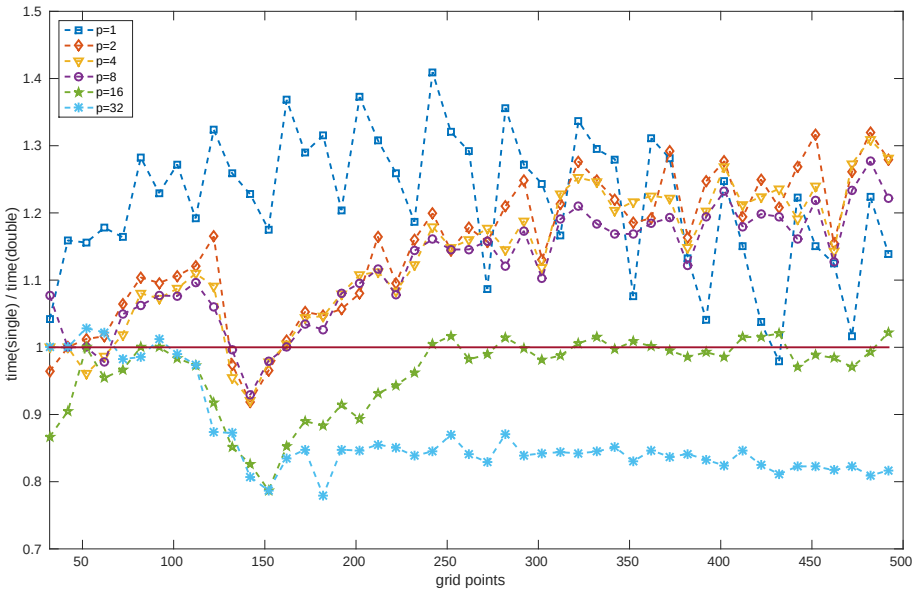

Figure 12: Ratio of execution time for running the JTC in single versus double precision with different numbers of grid points and p.

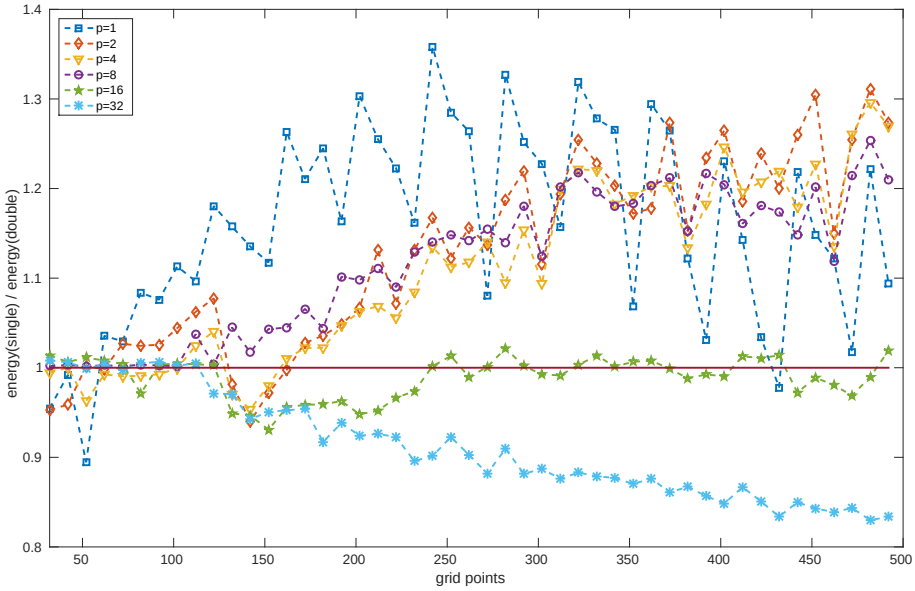

Figure 13: Ratio of total energy consumption for running the JTC in single versus double precision with different numbers of grid points and p.

in the execution time and energy consumption being nearly identical for the Jacobi iteration within the single and double precision versions. Finally, when $p = 32$, the data movement costs are now high enough

that we see the advantage of using single precision over double precision. Here, for the Jacobi iteration phase, the execution time is reduced by 18% and the energy consumption by 12%.

	single		double		ratio (single/double)	
p	t_{setup}	t_{Jacobi}	t_{setup}	t_{Jacobi}	t_{setup}	t_{Jacobi}
1	42.23	36.77	41.82	31.48	1.010	1.168
2	22.30	24.79	21.73	18.32	1.026	1.353
4	11.42	12.98	11.30	10.26	1.011	1.265
8	5.94	6.70	5.89	5.40	1.009	1.242
16	3.04	5.38	3.01	5.44	1.010	0.988
32	1.68	3.77	1.62	4.62	1.041	0.816

Table 9: Computed values of t_{setup} and t_{Jacobi} for single and double precision version of JTC with 452 grid points and different values of p, and the ratio of single/double.

	single		double		ratio (single/double)	
p	e_{setup}	e_{Jacobi}	e_{setup}	e_{Jacobi}	e_{setup}	e_{Jacobi}
1	0.0787	0.0600	0.0672	0.0523	1.171	1.147
2	0.0975	0.0409	0.0580	0.0301	1.680	1.359
4	0.0282	0.0212	0.0353	0.0169	0.794	1.254
8	0.0028	0.0120	0.0257	0.0089	0.109	1.356
16	0.0016	0.0097	0.0155	0.0090	0.103	1.074
32	0.0031	0.0068	0.0131	0.0076	0.242	0.883

Table 10: Computed values of e_{setup} and e_{Jacobi} for single and double precision version of JTC with 452 grid points and different values of p, and the ratio of single/double.

We conclude that, as with the OpenMP version, the advantages of using single precision over double precision only becomes apparent when the data movement costs start to be significant. The advantage of using single precision within the JTC is less pronounced than the OpenMP case.

4 Energy consumption of the JTC with OpenMP and MPI

We have seen that the OpenMP and MPI versions of the JTC give a reduction in energy consumption as we increase the number of threads/processes. In this section, we consider the mixed OpenMP-MPI version of the JTC.

We start by considering what happens when the number of MPI processes is fixed ($p = 4$) and the number of OpenMP threads is altered ($nthr = 1, 2, 4, 8, 16$). Each node has 16 cores and, hence, for $nthr > 4$ we will be running the JTC across different nodes. In Figures 14 and 15, we compare the execution time and energy time, respectively, for the varying values of $nthr$ and grid points using the single precision version of the code. For $nthr = 1, 2$ and 4, we observe that doubling the number of threads does not halve the execution time but there is a substantial decrease. In comparison, in the pure OpenMP version, the execution time does halve. As we move to using two and then four nodes ($nthr = 8$ and 16), there is little to be gained in execution time over using the combination $p = 4$ and $nthr = 4$. For the pure OpenMP version, doubling the number of threads roughly halved the total energy consumption. Here, moving from $nthr = 2$ to $nthr = 4$ decreases the total energy consumption by roughly a third. There is no gain to be had in terms of total energy consumption by switching from $nthr = 4$ to $nthr = 8$, and, contrary to the execution time, switching to $nthr = 16$ increases the total energy consumption: for large

numbers of grid points, the energy consumption is similar to that with $nthr = 2$. this is due to the extra energy required to move data between nodes.

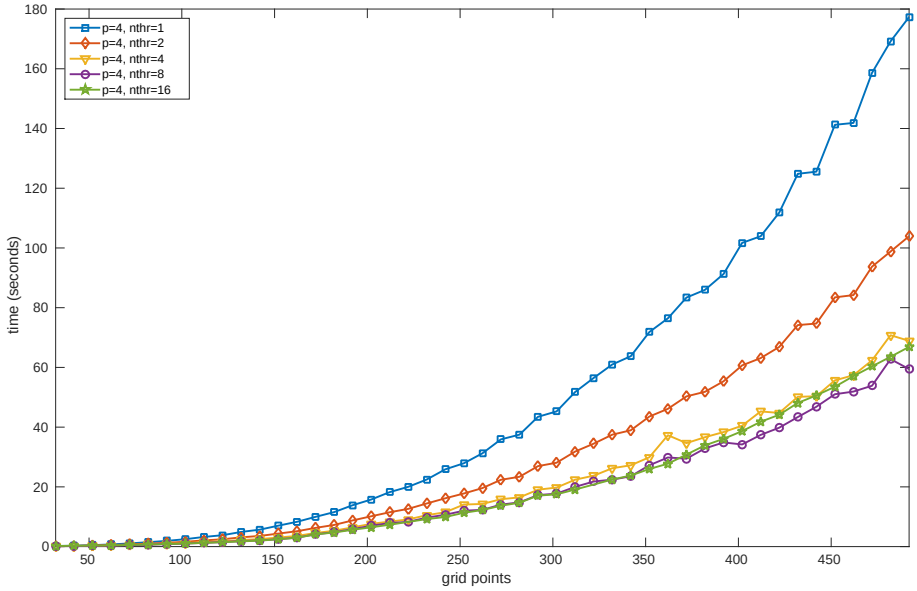

Figure 14: The execution time for running the JTC in single precision with $p = 4$ and different numbers of grid points and $nthr$.

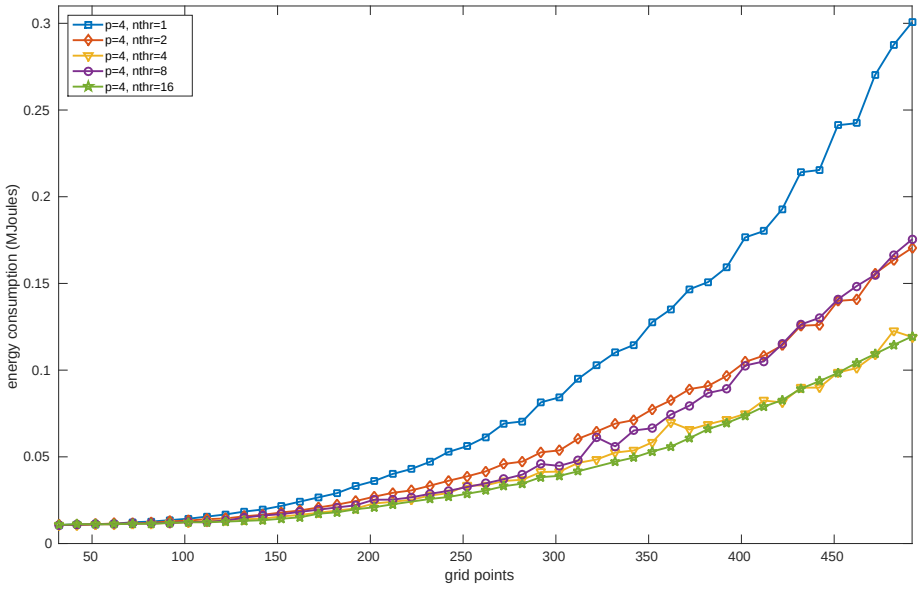

Figure 15: The total energy consumption for running the JTC in single precision with $p = 4$ and different numbers of grid points and $nthr$.

For the double precision version with $p = 4$, see Figures 16 and 17, we move even further away from the behaviour of the pure OpenMP version as we alter $nthr$. In terms of both the execution time and total energy consumption, $nthr = 4$ is favourable: the extra cost of moving double precision data between nodes completely outweighs any gains from parallelism of the floating point operations.

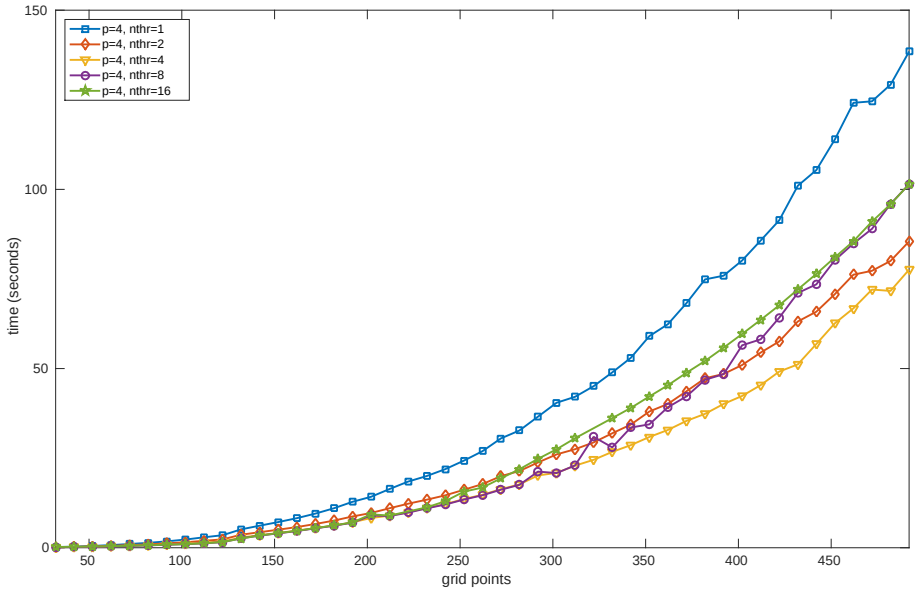

Figure 16: The execution time for running the JTC in double precision with $p = 4$ and different numbers of grid points and $nthr$.

In Figures 18, 19 and 20, we compare the execution times, energy consumption and average power usage for the single and double precision versions of the JTC with four MPI processes, varying the number of grid points and the values of $nthr$. As expected, for very small problems (less than 75 grid points), the single precision version reduces the execution time by up to 8% but we see more modest reductions in energy consumption. Note that the average power usage is greater for the single precision version but the drop in execution time is enough to give a small drop in energy consumption. For grid points between 80 and 130, the average power usage is generally lower for the single precision version than the double precision version. The biggest drop in power usage is for $nthr = 1$ where there is up to an 8% decrease. Increasing the number of threads brings the average power usage closer for the two different precisions. For this range of grid points, the single precision execution times are larger than those of the double precision version and, as a consequence, the total energy consumption is comparable. As the number of grid points increases, there are suddenly huge gains to be had in execution time when using the single precision version instead of double precision: this is due to the cache behaviour and the double precision version having to move a lot of data between caches and main memory but the single precision version is still mainly able to store its data in the caches. For $nthr = 16$ and approximately 150 grid points, the execution time of the single precision version is 40% lower than that of the double precision version. This difference in execution time decreases as the number of threads increases. The increase in data movement in the double precision version slows down the rate at which floating point operations are executed and, hence, the overall power usage is much lower for the double precision version. However, the overall energy consumption is lower for the single precision version.

If we continue to increase the number of grid points, the amount of data traffic for the single precision version ramps up as the caches fill-up. For greater than 200 grid points, the size of problems are such

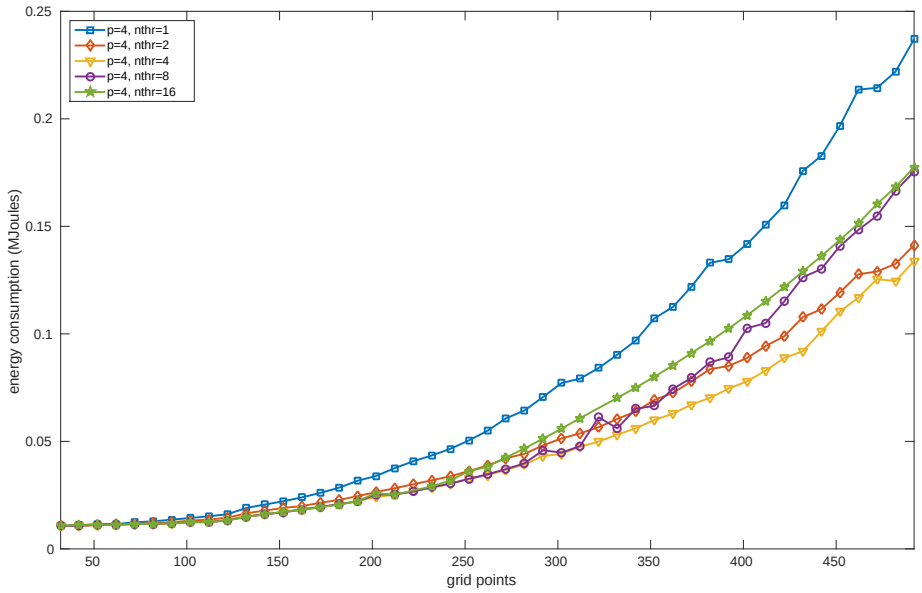

Figure 17: The total energy consumption for running the JTC in double precision with $p = 4$ and different numbers of grid points and $nthr$.

that the costs of data movement start to dominate for both the single and double precision variants. Where the data movement levels are still modest compared to floating point operations, as expected, the double precision version has lower execution times and, although the average power usage is slightly lower for the single precision version, the double precision variant uses the smaller amount of energy. In particular, for $nthr = 1$, the single precision version consumes approximately 20% more energy; for $nthr = 2$, approximately 15% more energy is consumed by the single precision version. For $nthr = 4$, the increase in data traffic over $nthr = 1$ or $nthr = 2$ results in the average power usage being similar but the execution time for for single precision version is between 3 and 10% lower than that of the double precision version. Hence, we see similar gains in energy consumption. As expected, as the data movement increases further with more data having to flow between nodes, the advantages of using single precision increase even more. In particular, for $nthr = 8$ and greater than 380 grid points, the single precision version reduces the execution time by 33% and energy consumption by 31%. For $nthr = 16$ and greater than 400 grid points, the execution time and total energy consumption are reduced by at least 35% and at least 33%, respectively.

Whilst fixing the value of p and varying $nthr$ is of interest (or fixing $nthr$ and varying p), we normally have a maximum number of cores available for us to use but we can use different combinations of $nthr$ and p such that we use the maximum number of cores. Thus, we wish to know which combination is best for a specific precision and which combination gives the best gain when comparing the single and double precision version. In the following set of results, we set $nthr$ and p such that $nthr \times p = 64$, that is, we assume that we have four nodes available. In Figures 21 and 22, we compare the execution time and total energy consumption for running the JTC in single and double precision with different numbers of grid points, $niter = 50$ and $nruns = 10$.. We start by noting that the level of data traffic is such that all of the single precision variants are faster than the double precision variants with $p \leq 16$ and use less energy. For both the double and single precision versions, $p = 32$ and $nthr = 2$ gives the lowest execution times and energy consumption. Additionally, $p = 4$ and $nthr = 16$ is always the worse in terms of both execution time and total energy consumption. It is not true that doubling p and halving $nthr$

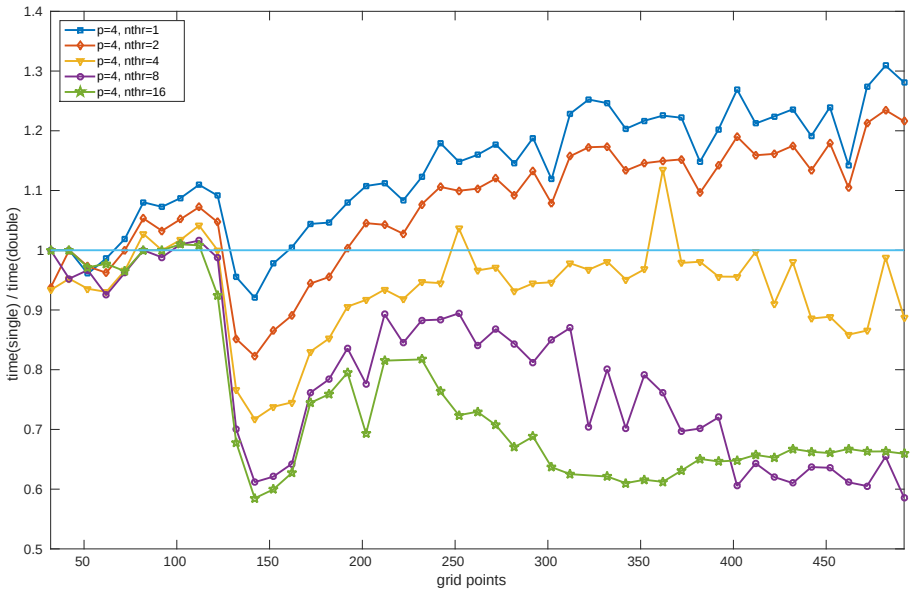

Figure 18: Ratio of execution time for running the JTC in single versus double precision with $p = 4$ and different numbers of grid points and $nthr$.

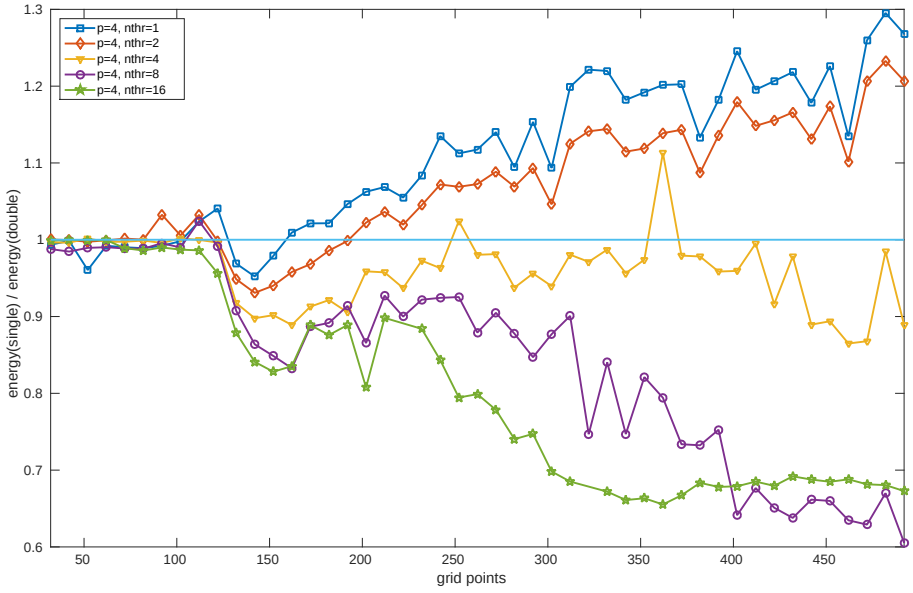

Figure 19: Ratio of total energy consumption for running the JTC in single versus double precision with $p = 4$ and different numbers of grid points and $nthr$.

always decreases execution time and total energy consumption: in fact, for large number of grid points, $p = 16$ and $nthr = 4$ outperforms $p = 8$ and $nthr = 8$. We therefore conclude that different combinations of OpenMP threads and MPI processes can have profound effects on the total energy consumption and,

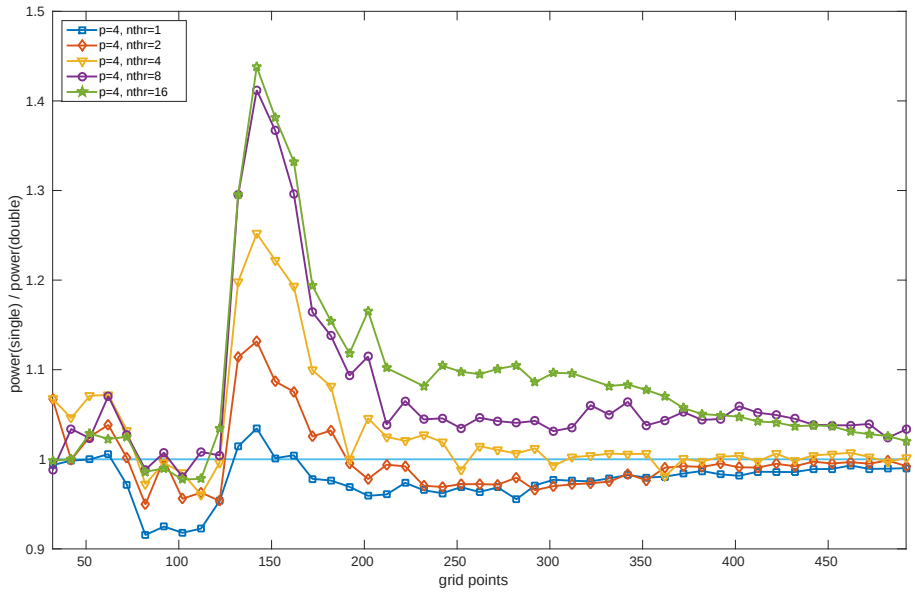

Figure 20: Ratio of total energy consumption for running the JTC in single versus double precision with $p = 4$ and different numbers of grid points and $nthr$.

whilst the results from Sections 2 and 3 might lead us to assume that it is better to focus on more threads, this assumption is not true.

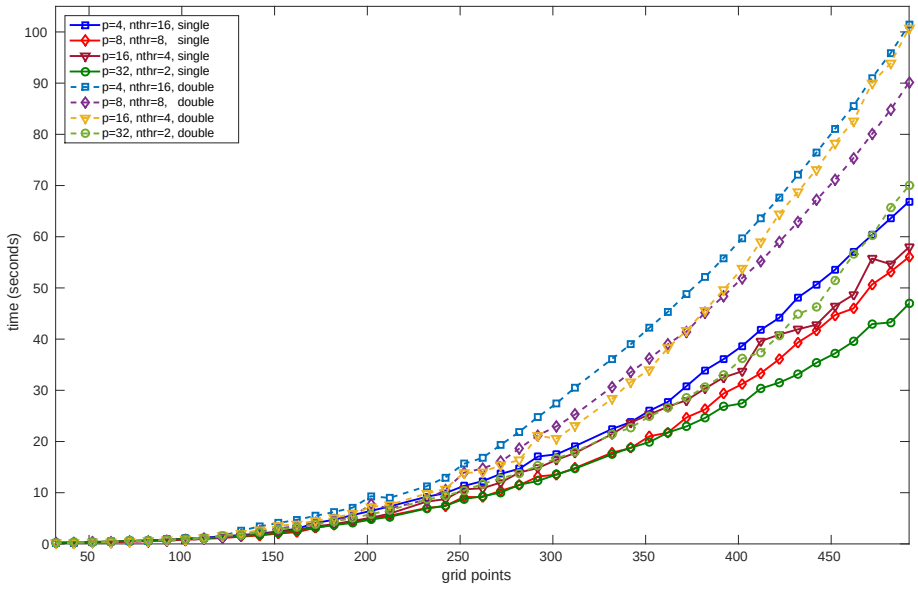

Figure 21: The execution time for running the JTC in single and double precision with $p \times nthr = 64$ and different values of $size$.

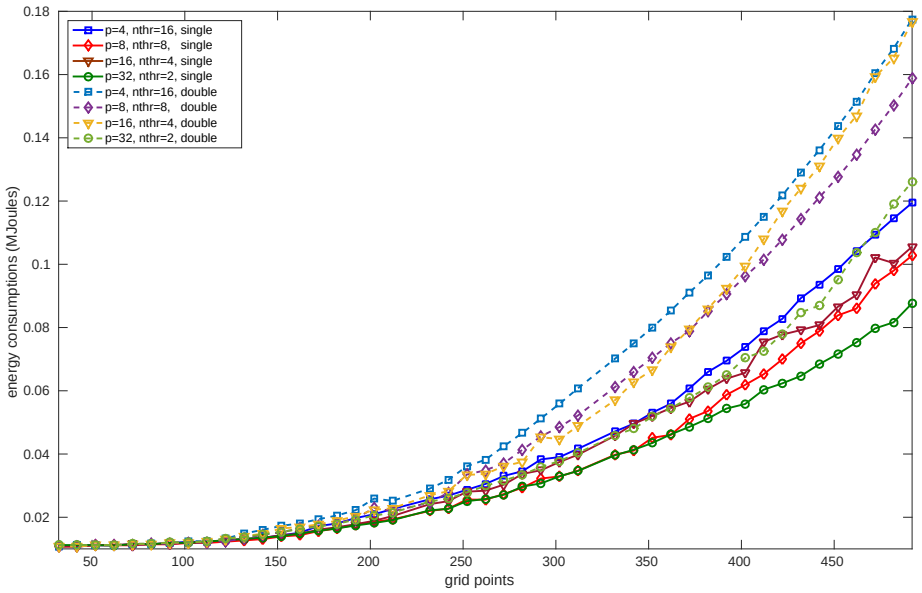

Figure 22: The total energy consumption for running the JTC in single and double precision with $p \times nthr = 64$ and different values of $size$.

5 Conclusions

We conclude that the advantages of using single precision instead of double precision within the JTC only become apparent on the BG/Q when the costs of data movement becomes significant. This can be achieved by using large enough problem sizes with a large enough number of OpenMP threads and/or MPI processes. Increasing the number of threads in the pure OpenMP version gave more significant reductions in energy consumption than increasing the number of processes in the pure MPI version. If we wish to try to reduce energy consumption by altering the number of threads and processes in the mixed version without altering the precision, then we found that it was better to use a smaller number of threads with a larger number of processes.

The JTC benchmark does not allow the user to specify the level of accuracy that the Poisson problem should be solved to. If we were able to do this, then we would expect the double precision version to reach a given level of accuracy in fewer iterations than the single precision version. Hence, we will not expect the practical gains in using single precision to be so impressive for the larger number of threads/processes.

Future work includes the running of similar JTC experiments on other architectures so that we can compare their individual energy consumption characteristics.

References

[1] See http://community.hartree.stfc.ac.uk/wiki/site/admin/blue%20gene%20q%20further%20info.html.

[2] L. Anton, M. Mawson, A. D. Taylor, and V. Szeremi, *Performance profiles with Jacobi Test code suite*, tech. rep., 2015. http://purl.org/net/epubs/work/12145766.

[3] S. Ashby, P. Beckman, J. Chen, P. Colella, B. Collins, D. Crawford, J. Dongarra, D. Kothe, R. Lusk, P. Messina, and Others, *The opportunities and challenges of exascale*

computing, summary report of the advanced scientific computing advisory committee (ASCAC) subcommittee at the US Department of Energy Office of Science, (2010).

[4] S. S. Shende and A. D. Malony, *The TAU Parallel Performance System*, International Journal of High Performance Computing Applications, 20 (2006), pp. 287–311.