Improved Scaling for Direct Numerical Simulations of Turbulence

Johnstone, R.
University of Southampton
Southampton, SO17 1BJ

August 2, 2012

1 Introduction

Two computer programs were addressed in the dCSE project summarised here. The first, SWT, is designed for the direct numerical simulation of turbulent flow in an infinite plane channel. It has, in the recent past, also been used to simulate the response of such a flow to irrotational strain [3], and turbulent Couette-Poiseuille flow (a channel flow driven both by a pressure gradient as well as by moving walls, [9]). Both of these investigations aimed at clarifying the effect of pressure gradients on turbulence. The second, SS3F, has so far been predominantly used to simulate the dynamics of vortices in stratified flow [2], [1], [4].

SWT is a development of the vectoral [15] channel-flow code of Kim et al [10], parallelised and translated to modern fortran [3]. SWT uses the same Runge-Kutta scheme, and the same FFT routines, as SS3F; one coordinate direction uses Chebyshev basis functions, however, which are suitable for plane channel flows with one non-periodic direction.

The SS3F code solves the incompressible Navier-Stokes equations in the Boussinesq approximation, using a Fourier representation in all three coordinate directions. The infinite boundary condition described by Corral & Jiménez [5] is implemented in one of these directions while the other two are periodic. The viscous terms are time advanced analytically, while a low-storage 3rd order Runge-Kutta scheme is used for the advective and buoyancy terms.

1.1 Numerical Methods

Both codes are pseudospectral - the nonlinear advective terms of the Navier-Stokes equations are evaluated in real space; the necessary transformation to and from wave space is carried out in slices (a one-dimensional domain decomposition), which means that a parallel transpose (all-to-all communication) is required to transform with respect to the remaining direction.

Note that the Navier-Stokes equations are quadratically nonlinear, so this procedure can generate nonzero coefficients for wavenumbers of up to $\pm 2k_{max}$, where the original wavenumber interval was $[-k_{max} : k_{max}]$. To avoid, when transforming the nonlinear terms back to wave space, the aliasing of any outside this interval to any within, we require that

$$2k_{max} - 2N \leq -k_{max} \text{ or } 2k_{max} + 2N \geq k_{max}, \tag{1}$$

and thus

$$k_{max} \leq 2N/3 \tag{2}$$

Explicit zero padding of the upper third of the spectrum is therefore sufficient to prevent this, and full dealising ($k_{max} = N/2$) is not required. Alternatives (eg. phase shift dealiasing) are less attractive as a result.

1.2 Parallelisation

Prior to the completion of this project, the SWT and SSF3D programs both used a 1-D domain decomposition. The domain is divided into 2-D slices; this permits FFTs in two dimensions but not the third, so a parallel transpose is required to transform the decomposition to another one, also of 2-D slices, but differing in one dimension.

3-D Fourier transforms are implemented here as 3 successive 1-D transforms. There is therefore no need for an individual processor to have access to more than one dimension at a time. Decomposing the domain into 'pencils' - a 2-D domain decomposition, into elements with two incomplete dimensions - would obviously increase the maximum number of processors that could be used.

However, this approach comes at a cost; an additional parallel transpose is required for every transform. The 1-D decomposition therefore remains attractive at low processor counts, though it restricts the number of processors that can be used, to, at most, the number of collocation points in one of the three spatial dimensions. As the grid is refined, however, the memory required, as well as the computational expense per time step, increases with the cube of this number. The processor count, meanwhile, can only scale linearly.

In practice the situation is even less favourable, as, in general, two different 1-D domain decompositions are required; if (as is usually the case) the numbers of collocation points in each direction are not the same, the greatest useful processor count is set by the lower of the two numbers chosen. Also, further load balancing issues arise if either one of these numbers is not exactly divisible by the processor count, and the latter is high (so that the imbalance is significant relative to the total workload).

These considerations limit simulations using SWT and SS3F to a few hundred processor cores at present, which results in relatively low performance in terms of wall clock time required to perform a physically realistic simulation. The most serious consequence, however, is that the scaling of memory requirements - doubling the resolution in all three directions permits at most a doubling of the core count, but guarantees the octupling of the total memory required. The expected reduction in the ratio of memory to core count arising from the increasing prevalence of multicore architectures only exacerbates the problem.

The applicability of turbulence simulations to the real world depends critically on good grid resolution (see [14] for a counterexample). In its absence, fidelity must be compromised, or the problem to be studied modified in some way - for instance, by a reduction in Reynolds number, which could adversely affect the generality of the result.

The Reynolds number is the parameter governing the range of scales present in a direct simulation of turbulence. The grid resolution required for a particular problem is expected to scale with this number. The required number of collocation points

$$N \propto \left(\frac{L}{\eta}\right)^3, \tag{3}$$

where L is the integral length scale of the largest eddies, roughly proportional to the channel height h, and η, the Kolmogorov scale, is that of the smallest, equal to

$$\eta = \left(\frac{\nu^3}{\epsilon}\right)^{1/4}, \tag{4}$$

where ν is the viscosity, and ϵ the dissipation rate. In the case of the channel flows the SWT code is designed to simulate, the following is approximately true:

$$\epsilon \propto u_\tau^3/h, \tag{5}$$

where u_τ is the friction velocity at the wall, equal to the square root of the ratio of wall shear stress and fluid density, $\sqrt{\tau/\rho}$. Substituting 5 into 4 and 3, we obtain

$$N \propto \left(\frac{u_\tau h}{\nu}\right)^{9/4} \propto \mathrm{Re}_\tau^{9/4}, \tag{6}$$

where Re is the Reynolds number. The memory requirements of a simulation therefore scale as $\mathrm{Re}^{9/4}$, however, because the time step must also be reduced to resolve the smallest eddies.

Using the same assumptions made above to obtain the lengthscale ratio, we may derive a ratio of large to small time scales T/τ, where T is the ratio of h to the mean flow velocity U (assumed above roughly $\propto u_\tau$) and the Kolmogorov time scale $\tau_\eta = \sqrt{\nu}/\eta$. This turns out to vary as

$$T/\tau \propto \mathrm{Re}^{1/2}, \tag{7}$$

a finding confirmed by the computational results of Kim et al [10].

This has implications for the scaling of computational workload with Reynolds number - i.e. that it is nearly cubic - but also for the scaling of total computational workload with grid points, namely that it is superlinear. If domain decomposition alone is relied upon for parallelisation, then - absent improvements to parallel scaling - we can probably only hope to scale the processor count with the size of the grid. A complete DNS is not, therefore, expected to scale in accordance with Gustafson's law [7], which requires a constant workload per processor - performance over a fixed number of time steps, however, may do so, if the solution algorithm scales linearly in the number of grid points.

For a DNS as a whole, Snyder's [13] Corollary of Modest Potential applies; the problem takes

$$n_t \propto N^{\frac{1}{2} \cdot \frac{4}{9}} = cN^{2/9} \tag{8}$$

time steps n_t, and the work per grid point varies at least in the same proportion (assuming, optimistically, an $O(N)$ algorithm). The total work, or wall clock time, would be

$$t = cN^{11/9}. \tag{9}$$

This is a limitation essentially imposed by the physics of turbulence; algorithms with scaling in N inferior to $O(N)$ would raise the exponent further, and time integration schemes with stability constraints might do the same.

If the application of p processors permits an increase in the problem size by a factor m, then

$$t = c(mN)^{11/9}/p, \tag{10}$$

and, if we require the wall clock time t to remain fixed,

$$m \leq p^{9/11} \tag{11}$$

Doubling p, therefore, allows at best an increase of 76% in N, which permits, as long as 6 holds, about a 29 % increase in Re. Large increases in p are therefore needed to allow significant increases in Re, and so good parallel scaling is mandatory.

Note further that for DNS of wall bounded turbulence, the ratio L/η that needs to be simulated may increase faster with Re than is indicated by the relationship 6 given above. The reason for this is that the very largest scale motions in such flows do not measurably affect flow statistics at low Re [8], so that a low Re DNS may employ a domain too small to capture them; but this ceases to be true at higher Re. The Reynolds numbers that have so far been considered using DNS are believed to fall, on the whole, into the former category.

This implies that such simulations as have already been performed are not fully independent of Re, and the results therefore lack a degree of universality that might be (and, on the basis of experimental work, is) expected at higher Re. Experimental data do not contain the same information as DNS, and so cannot always be used to address the same questions. It is therefore desirable to carry out such simulations. However, the difficulty of scaling Re implied by 6 and 11 above, added to the need to resolve still greater turbulent lengthscales, does call into question the feasibility of doing so.

2 The dCSE Project

A dCSE project to improve the performance and parallel scaling of the SWT and SS3F programs began in November 2011, approximately one month before the introduction of the HECToR phase 3 service. Final results, and comparisons with the original versions of both codes, are therefore produced on that machine, but the initial performance analysis motivating the project was carried out on phase 2b, and is summarised below.

2.1 Initial performance analysis

Both of the programs described above have properties that are bound, ultimately, to limit their parallel scalability, and these limits had, at the time of writing, been reached. In pursuit of the lines of inquiry opened in [4] and [9], simulations were planned which would require grids of approximately 3 and 1 billion collocation points, using SS3F and SWT respectively. In both of these cases, more cores would need to be allocated than can in fact be used, owing to the requirement that each core store and process an integer number of planes at a time. This follows from the use of a 1-D domain decomposition.

Initial investigations showed that, in addition, neither code performed particularly well - either in terms of per-processor performance, or of parallel scaling - on the HECToR architecture (phase 2b at the time), and that this state of affairs had worsened relative to phase 2a. Test cases smaller than the problem sizes (3 & 1 billion grid points) actually planned were used, as scaling tests would not be meaningful otherwise (certainly the core count would be completely meaningless, as many if not most cores would be left idle).

Figure 1 shows parallel scaling for SS3F on a grid (128 x 720 x 1440

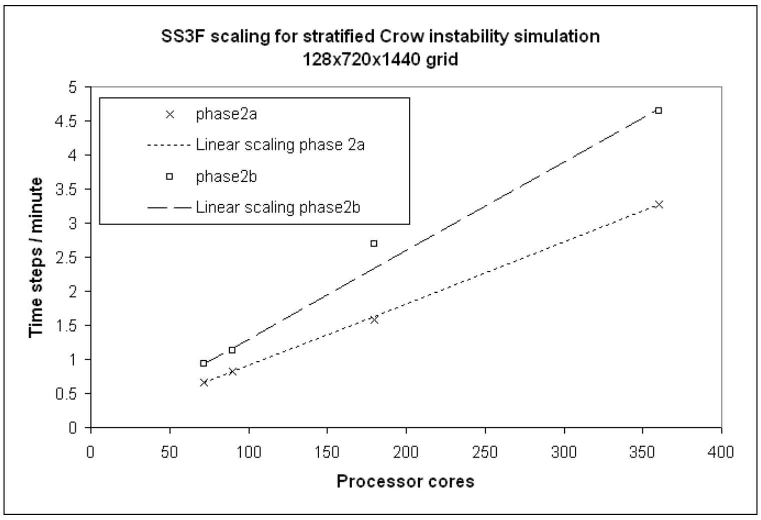

Figure 1:

modes) previously used to obtain the results published in [4]. This appears acceptable; however, it is not possible to increase the core count for this problem beyond 360 without suffering load imbalance; perfect load balance requires that the second and third dimension should both be divisible by the core count, the former without dealiasing (i.e. 1080), the latter with (1440). At the beginning of the project, this code was in fact restricted to permit only very slight load imbalance (fixed computational load was enforced for all but one process). It is reasonable to assume that merely loosening this restriction would not be a particularly good use of development effort; for the test case considered here, for instance, each process is responsible for 3 $x - z$ planes, so an imbalance of 1 would be fairly significant.

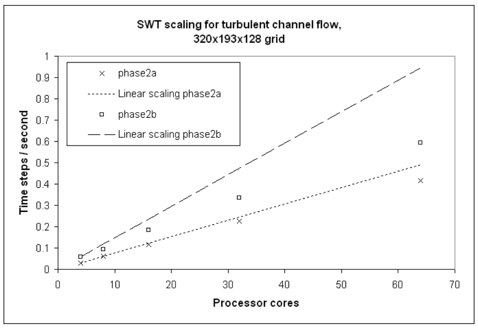

Figure 2:

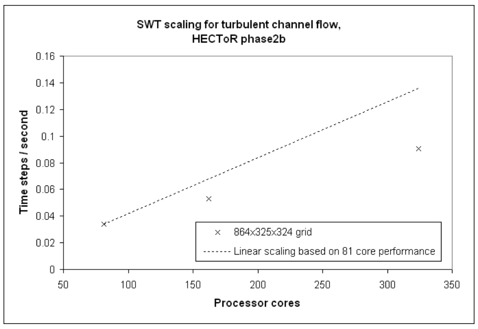

Figure 3:

The results for SWT (figures 2 and 3) appear less satisfactory, and scaling is also notably poorer on the XE6 than the XT4 for both simulation sizes (in spite of the fact that the XT4 data were collected before the introduction of the Gemini interconnect). It would appear that the SWT parallelisation may not be well suited to architectures with large numbers of cores per CPU. Performance per core, however, was superior on phase2b.

Both codes originally implemented a parallel transpose using MPI non-blocking sends and receives. On Cray machines, this approach performs less well than MPI_ALLTOALLV (used by 2DECOMP&FFT), so some parallel efficiency gains are likely to arise from this substitution.

In-depth profiling of the SWT code, using the 864 x 325 x 324 grid on 324 cores, was carried out by Dr. Ning Li of the NAG HECToR CSE team, and the results (obtained using CrayPAT) are shown below.

```
Time % | Time | Imb. Time | Imb. | Calls |Experiment=1
| | | Time % | |Group
| | | | | Function
| | | | | PE='HIDE'
100.0% | 110.186539 | -- | -- | 85892.9 |Total
|-----------------------------------------------------------------
| 63.9% | 70.354830 | -- | -- | 7983.9 |USER
||----------------------------------------------------------------
|| 19.7% | 21.745259 | 5.791653 | 21.1% | 1.0 |MAIN_
|| 13.8% | 15.219145 | 8.783854 | 36.7% | 2181.8 |radb3_
|| 9.1% | 10.063469 | 8.069853 | 44.6% | 722.2 |radf3_
|| 3.6% | 3.998016 | 2.350367 | 37.1% | 180.6 |radb4_
|| 3.3% | 3.680287 | 2.335971 | 38.9% | 180.6 |radf4_
|| 3.1% | 3.427989 | 1.016561 | 22.9% | 722.2 |passb3_
|| 2.8% | 3.075736 | 1.263592 | 29.2% | 722.2 |passf3_
|| 2.6% | 2.817066 | 1.345977 | 32.4% | 545.4 |radb4l_
|| 1.7% | 1.910091 | 1.269967 | 40.1% | 180.6 |radf4f_
|| 1.0% | 1.110035 | 0.026740 | 2.4% | 545.4 |rfftb1_
||================================================================
| 25.9% | 28.577012 | -- | -- | 77845.0 |MPI
||----------------------------------------------------------------
|| 25.4% | 27.943499 | 18.023463 | 39.3% | 38880.0 |MPI_WAIT
||================================================================
| 10.2% | 11.254697 | -- | -- | 64.0 |MPI_SYNC
||----------------------------------------------------------------
|| 9.9% | 10.945641 | 10.932999 | 93.6% | 11.0 |mpi_reduce_(sync)
|=================================================================
```

These results reveal that 25% of run time was spent in MPI_WAIT, and a further 10% in MPI_SYNC. This suggests load imbalance, which could be reduced by introducing a 2-D domain decomposition, by increasing the number of ways in which a grid could be divided for a given processor count.

Of the routines in the USER section, all but the main program (much of the runtime of which is associated with one-time initialisations, and not significant in a production run) are part of the existing FFT package (vecfft). It is important, therefore, that these should be efficient.

However, profiling of the most heavily used FFT routine (rad3b) using hardware performance counters showed that:

- ⇒ D1+D2 cache utilisation is 90.1% (which is quite poor).
- ⇒ There is no SSE vectorisation at all.

The first result may explain the good per-core performance on phase2b relative to phase2a; poor caching means the performance of the FFT routines is likely bounded by memory access speed, and the memory bandwidth per core of phase2b is superior. Note that the vecfft routines were designed to perform optimally on traditional vector processors (SWT and SS3F have previously been used on the X2 component of the HECToR service). The second result shows that the vecfft optimisations are unsuitable for the short SSE vector instructions.

A good case can therefore be made that the serial performance of both codes is poor on HECToR. However, it also appears that this is largely attributable to the poor performance of the FFT routines used by them, which dominate the run time.

2.2 Plan of Work

On the basis of the measurements summarised in section 2.1 above, it was considered that both SWT and SS3F would benefit from a modernisation of the FFT routines used, and conversion to a 2-D domain decomposition, to improve per-core performance and parallel scaling respectively.

The 2DECOMP&FFT library [12], [11] enables applications using a three-dimensional structured mesh to use a 2-D (pencil) domain decomposition, providing the parallel transpose operations required by some numerical methods (such as FFTs) to use such a decomposition. In addition, a higher-level interface (a 'black-box' 3-D FFT) is provided, as are routines for parallel I/O and halo support (relevant to hybrid spectral-finite difference methods). These were not to be used in this project; both SWT and SS3F already use MPI-I/O to read and write restart files, and neither uses finite differences.

Although 2DECOMP&FFT supports several FFT routines, a decision was made to begin by converting the existing FFT routines to instead use the FFTW library [6]. This course of action was adopted in part because SWT requires Chebyshev transforms; although these can be implemented using FFT routines, 2DECOMP&FFT does not yet support these directly. They could, however, be implemented by using SWT's Chebyshev transforms and replacing the FFTs with those of an FFT library supported by 2DECOMP&FFT (one such is FFTW). In addition, it was reasoned that early conversion to FFTW would aid development work, by separating any possible slight numerical differences arising from the use of a different FFT library from those arising from parallelisation. Results from the existing codes (modified to use FFTW) could be compared with those obtained using the new versions in the knowledge that the underlying FFT library would be the same (provided, of course, that 2DECOMP&FFT was compiled to use FFTW).

The following work plan was therefore adopted:

- ⇒ The vecfft routines would be replaced by FFTW equivalents, in both codes, for both Fourier and Chebyshev transforms.
- ⇒ A 2-D domain decomposition would be provided for SS3F by rewriting it to use the 'black-box' 3-D FFTs provided by the 2DECOMP&FFT library.
- ⇒ A package of Chebyshev transforms would be produced for the 2DECOMP&FFT library.
- ⇒ These transforms would then be used to convert SWT in the same way as SS3F.

2.3 Implementation

2.3.1 FFT conversion

The replacement of the FFT routines was reasonably straightforward; a naive approach using 1-D individual transforms was chosen. Multiple cosine

Domain size	Wall clock time per time step		% reduction
	FFTW3	vecfft	
128 x 720 x 1440	6.66 s	13.5 s	51%
256 x 1440 x 2880	32.3 s	72.3 s	55%

Table 1:

transforms were tested for the Chebyshev transform routines described in section 3 below, but there was no significant performance benefit in that case (testing was carried out using the SWT code and a 3072 x 325 x 1024 mode problem on 6144 cores).

FFTW planning flags were also investigated; FFTW_ESTIMATE was found to perform just as well as FFTW_PATIENT. The relevant test was carried out with the SS3F code on the test case described in section 2.1 above, using 360 cores over 12 nodes.

In spite of the simplicity of this implementation, substantial performance gains were realised for both codes. Test results for SS3F are summarised in table 1.

A similar test was carried out for SWT, using a domain size of 3072 x 325 x 1024; a percentage reduction of 53 % was observed in that case (see section 4.2 below).

2.3.2 2-D domain decomposition using the 2DECOMP&FFT library

The original work plan called for use of 2DECOMP&FFT's FFT API - that is, its 'black-box' 3-D FFT routines. On investigation, it turned out that a lower-level approach was required. It is in fact more convenient to do so, because the original codes have been programmed in the expectation that optimisations involving intermediate arrays (that would be hidden by a 3-D FFT) are possible; removing these would mean writing new code that would probably be less efficient. The need to implement dealiasing is also an impediment.

Dealiasing is most efficiently performed by pruning the array at each parallel transposition. The order of operations eventually adopted is as follows (the initial letter of each item denotes the direction in which the domain is not decomposed, e.g. y implies y-pencils containing all points in y but only a subset in x and z):

$\Rightarrow$ y: In wave space - global domain size N_x x $3N_y/2$ x N_z.

$\Rightarrow$ y: SS3F - Fourier transform in y, SWT - Chebyshev transform in y.

$\Rightarrow$ Transpose y to z.

$\Rightarrow$ z: Expand domain size to N_x x $3N_y/2$ x $3N_z/2$.

$\Rightarrow$ z: Fourier transform in z.

$\Rightarrow$ Transpose z to x.

$\Rightarrow$ x: Expand domain size to $3N_x/2$ x $3N_y/2$ x $3N_z/2$.

$\Rightarrow$ x: Fourier transform in x.

$\Rightarrow$ x: In real space - calculate non-linear terms of Navier-Stokes equations.

$\Rightarrow$ x: Fourier transform in x.

$\Rightarrow$ x: Prune domain to N_x x $3N_y/2$ x $3N_z/2$.

$\Rightarrow$ Transpose x to z.

$\Rightarrow$ z: Fourier transform in z.

$\Rightarrow$ z: Prune domain to N_x x N_y x $3N_z/2$.

$\Rightarrow$ Transpose z to y.

$\Rightarrow$ y: SS3F - Fourier transform in y; SWT - Chebyshev transform in y.

$\Rightarrow$ y: If dealiasing in y - zero high wavenumbers.

This significantly reduces the total data volume that requires transposition, relative to operating using a $3N_x/2$ x $3N_y/2$ x $3N_z/2$ global domain size. For backward and forward transformation of a single variable, the cost is $3/2+9/4+9/4+3/2 = 15/2$, rather than $27/8+27/8+27/8+27/8 = 27/2$, a saving of 44 %. A general 3-D FFT supporting this approach would require information relating to the nature of the dealiasing to be performed in each direction (for instance, as noted in section 1 above, the 3/2 rule is specific to equations with quadratic nonlinearity).

Note that x, y and z above correspond to the notation used in SWT and SS3F, but *not* that of 2DECOMP&FFT. To translate, exchange y and z.

It proved reasonably straightforward to implement the above approach using the domain decomposition API of 2DECOMP&FFT, and the results of doing so are discussed in section 4 below.

3 Chebyshev Transforms

A necessary side product of the conversion of the SWT code to use FFTW was a routine to carry out Chebyshev transforms. Had the FFT API of 2DECOMP&FFT been used, it would have been necessary to incorporate this into that library; as it stands, the domain decomposition API was chosen instead, so this is not necessary. However, it was considered appropriate to adapt and generalise this routine so as to make it suitable for use in or with 2DECOMP&FFT. This would broaden the capabilities of this library by reusing a side-product of this project.

Given a function $f(x)$ on the interval $-1 \leq x \leq 1$, and making a change of variable

$$x = \cos(\theta), \tag{12}$$

where $0 \leq \theta \leq \pi$, we define $\phi(\theta) = f(\cos(\theta))$. ϕ can be expanded in a cosine series

$$\phi(\theta) = \frac{1}{2}a_0 + \sum_{k=1}^{\infty} a_k \cos(k\theta) \tag{13}$$

The Chebyshev polynomials T_k of the first kind are defined

$$T_k(x) = T_k(\cos(k\theta)) = T_k(k \arccos(x)) \tag{14}$$

so

$$\phi(\theta) = \frac{1}{2}a_0 + \sum_{k=1}^{\infty} a_k T_k(x) \tag{15}$$

Note that the coefficients a_k in 13 and 15 are the same. It is, therefore, possible to obtain them using a cosine transform, computable with the FFT algorithm. It is also possible, with some manipulation, to use a standard FFT. The SWT code originally used a complex-to-real FFT for this purpose, and Chebyshev transform routines in x, y and z (or rather the array indeces usually corresponding to those dimensions) were developed from this. However, as FFTW3 supports cosine transforms, an alternative set of routines was created taking advantage of these. Both sets have been made available to Dr. Ning Li, the developer of 2DECOMP&FFT.

Collocation methods based on Chebyshev polynomials usually use the Chebyshev-Gauss points located at the roots of T_k, and given by

$$x_k = \cos\left(\frac{2k+1}{2N}\pi\right), k = 0, 1.., N \tag{16}$$

or the Gauss-Lobatto points, located at the extrema of T_k (in other words, at the zero points of T'_k rather than T_k). These are given by

$$x_k = \cos\left(\frac{k}{N}\pi\right), k = 0, 1.., N \tag{17}$$

For applications such as SWT - which is designed to simulate flow through a channel of fixed depth - the Gauss-Lobatto points are preferred; the zeroth and Nth points are at 1 and -1, and do not depend, as the Chebyshev-Gauss points do, on N. Both grids are supported; at present, the default is to use the Gauss-Lobatto points, and an optional argument is used (if present) to switch to Chebyshev-Gauss.

4 Results

4.1 SS3F

Figure 4 shows parallel scaling for a planned simulation using SS3F. The original code is included for reference (open square), although it was necessary to reduce the cores used per node from 32 to 6 in order to run this case. Node count is therefore chosen in preference to core count for the x-axis; this reflects the actual resources occupied in running this simulation in a way that core count does not.

The improvement in efficiency - ie. performance at the minimum (192) node count relative to the original code - appears at first glance to be entirely due to the use of all 32 cores per node (it is a factor of approximately 5, not far off 32/6). If true, the contribution expected from the replacement of the original FFT routines - and confirmed for smaller test cases - is absent. However, the 32 AMD Interlagos processors on each node share many resources, notably L3 cache and interconnect bandwidth, so this may not be a fair comparison.

Scaling to over 12000 cores is efficient, but in this case the same good efficiency does not extend to >18000 cores.

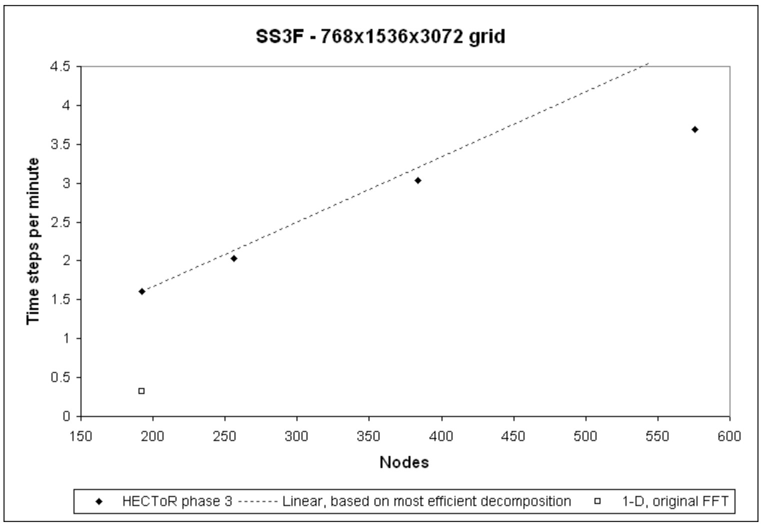

Figure 4:

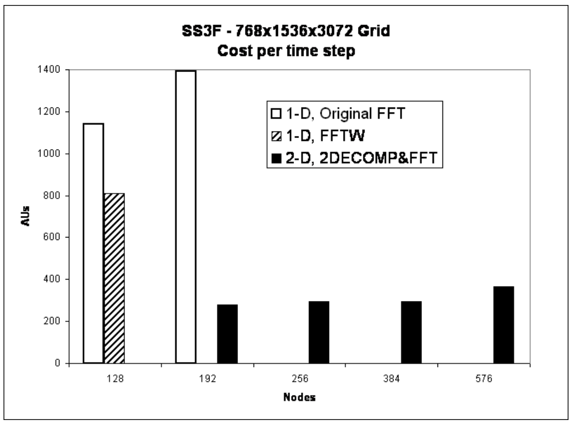

Figure 5:

4.2 SWT

Figure 6 shows absolute performance plotted against node count, for a test case (3072 x 321 x 1024 modes) chosen to be representative of the simulations planned using SWT in the near future. Node count is again chosen in preference to core count, because the original code was not quite able to run this case using all 32 cores per node, owing to its 1-D domain decomposition. The original code is represented in the figure with an empty square symbol. Replacing the original FFT routines with FFTW3 equivalents resulted in a significant performance improvement - using the same number of cores, wall clock time per time step fell by about 53%.

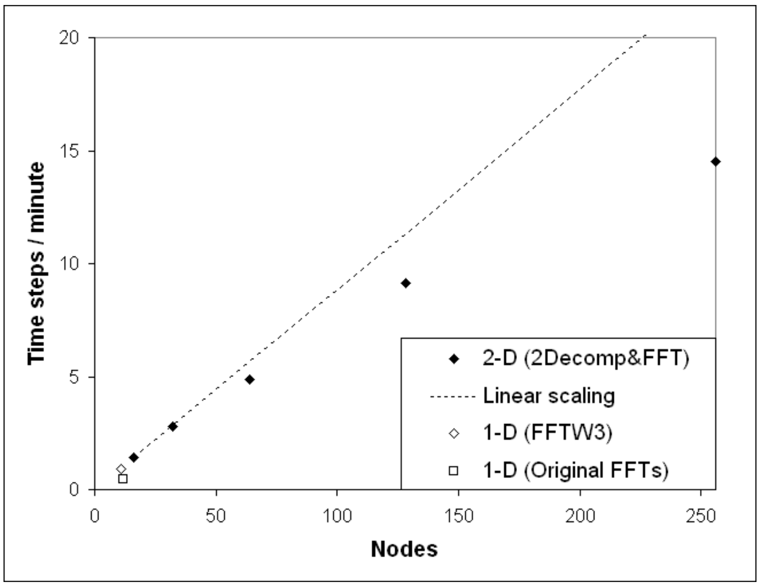

Figure 6: SWT, parallel scaling for 3072 x 321 x 1024 modes.

In addition, memory requirements were slightly reduced. This made it possible to reduce the required node count by 1 to 11, with all nodes but one fully populated (the last could almost certainly also have been fully utilised but for the fact that one grid dimension was not a multiple of 32). This (11 node) run is shown in the figure as an empty diamond; using 12 nodes instead improved performance by about 5% (this is not shown in figure 6).

Scaling to a higher number of processors was not possible in either of these cases, owing to the 1-D domain decomposition (nor would it have been possible to use fewer nodes; reducing the core count would only have been possible by reducing the number of cores used per node). This case therefore represents the limit of what could realistically be achieved using the 1-D decomposition. Note that - perhaps owing to the dimensions of the large static arrays used - both 1-D code versions could in fact only be successfully compiled for this particular case by using the pathscale compiler. This compiler is no longer supported on HECToR, and it is likely that it no longer produces optimal code for phase 3 of this machine. Some of the observed performance improvement is probably due to its replacement by the Cray compiler. Recall, however, that large performance benefits (> 100%) were obtained by replacing the same FFT routines in the SS3F code, even when the Cray compiler was used in both cases.

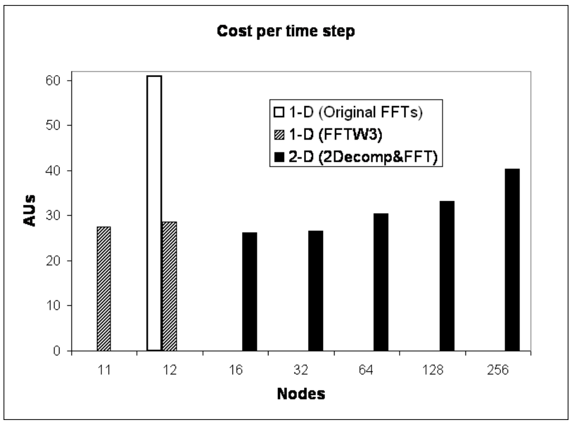

Figure 7: SWT, efficiency for 3072 x 321 x 1024 modes.

The implementation of a 2-D decomposition using the 2DECOMP&FFT library has made it possible to reduce by at least an order of magnitude the wall clock time required to run the most challenging case possible with the original code version, without giving up efficiency gains resulting from the modification of the FFTs. Note that even using 256 nodes (8192 cores), although scaling less efficiently than at lower node counts, the new version still requires about 34% fewer AUs to perform the same amount of work as the original (see figure 7), and does so about 30x faster. More important, however, is probably the fact that still larger grids are now thinkable, something that was not necessarily true prior to this work (eventually the memory required per core would have exceeded the total memory per node, at which point progress becomes impossible rather than merely very expensive - unless out of core algorithms or simply paging to disk are considered acceptable). The scaling of the simulation program with problem size, as well as processor count, is therefore of interest.

The four symbol styles shown in figure 8 illustrates the scaling of the SWT code with the total number of gridpoints N, for four different simulation sizes, each over a range of decompositions. Low values on the x-axis correspond to a large number of cores relative to the N, while high values imply a lesser degree of parallelism. The y-axis variable is a measure of efficiency - the number of gridpoints that can be advanced by one time step by expending one allocation unit. Perfect parallel scaling, therefore, would mean that all like symbols would align horizontally - i.e. efficiency independent of decomposition. In practice, we would expect to see a positive slope on the left hand side of the graph, and this is indeed the case. Above approximately 10^6 modes per core, however, scaling is quite good, independently of the total problem size N.

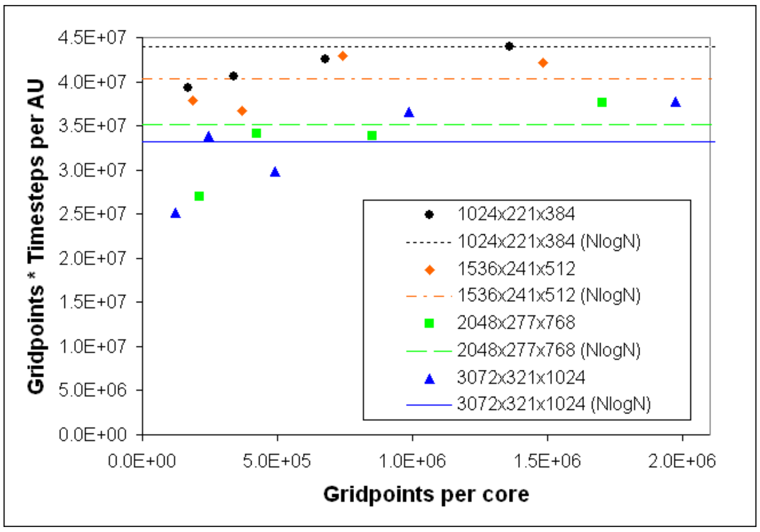

Figure 8:

The horizontal lines in figure 8 show the expected maximum efficiency for each N, based on the best efficiency obtained for the smallest N (the 1024x221x384 grid), and assuming $O(N\log(N))$ scaling. $O(N)$ scaling would correspond to collapse of all symbols onto one curve (at least towards the right hand side of the graph), while $O(N\log(N))$ - the computational complexity scaling of the fast Fourier transform - would correspond to good alignment of each set of symbols with the horizontal line of the same colour. It appears that the real N scaling is intermediate between the two - the best efficiency obtained at a given N shows a tendency to decline a little more slowly than implied by $O(N\log(N))$ complexity. This is very encouraging, as it suggests that - apart from the superlinear scaling in N that results from the increasing range of time scales (equation 7) - approximate Gustafson scaling holds. The wall clock time required per time step will therefore increase only very slowly if both N and the processor count p are increased in the same proportion; although the number of time steps required will increase.

5 Acknowledgments

This project was funded under the HECToR Distributed Computational Science and Engineering (CSE) Service operated by NAG Ltd. HECToR - A Research Councils UK High End Computing Service - is the UK's national supercomputing service, managed by EPSRC on behalf of the participating

Research Councils. Its mission is to support capability science and engineering in UK academia. The HECToR supercomputers are managed by UoE HPCx Ltd and the CSE Support Service is provided by NAG Ltd. http://www.hector.ac.uk.

References

[1] P.J. Archer, T.G. Thomas, and G.N. Coleman. Direct numerical simulation of vortex ring evolution from the laminar to the early turbulent regime. *Journal of Fluid Mechanics*, 598:201–226, 2008.

[2] P.J. Archer, T.G. Thomas, and G.N. Coleman. The instability of a vortex ring impinging on a free surface. *Journal of Fluid Mechanics*, 642:79–94, 2010.

[3] G.N. Coleman, D. Fedorov, P.R. Spalart, and J. Kim. A numerical study of laterally strained wall-bounded turbulence. *Journal of Fluid Mechanics*, 639:443–478, 2009.

[4] G.N. Coleman, R. Johnstone, C.P. Yorke, and I.P. Castro. DNS of aircraft wake vortices: the effect of stable stratification on the development of the Crow instability. In V. Armenio, B. Geurts, and J. Frolich, editors, *ERCOFTAC Workshop: Direct and Large-Eddy Simulation 7*, pages 519–525. Springer, 2008.

[5] R. Corral and J. Jiménez. Fourier/Chebyshev methods for the incompressible Navier-Stokes equations in infinite domains. *Journal of Computational Physics*, 121:261–270, 1995.

[6] M. Frigo and S.G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93:216–231, 2005.

[7] J.L. Gustafson. Reevaluating Amdahl's law. *Communications of the ACM*, 31:532–533, 1988.

[8] N. Hutchins and I. Marusic. Evidence of very long meandering features in the logarithmic region of turbulent boundary layers. *Journal of Fluid Mechanics*, 579:1–28, 2007.

[9] R. Johnstone, G.N. Coleman, and P.R. Spalart. The resilience of the logarithmic law to pressure gradients: evidence from direct numerical simulation. *Journal of Fluid Mechanics*, 643:163–175, 2010.

[10] J. Kim, Moin. P., and R. Moser. Turbulence statistics in fully developed channel flow at low reynolds number. *Journal of Fluid Mechanics*, 177:133–166, 1987.

[11] N. Li. 2DECOMP&FFT - library for 2d pencil decomposition and distributed fast fourier transform. http://www.2decomp.org/.

[12] N. Li and S. Laizet. 2DECOMP&FFT - a highly scalable 2d decomposition library and FFT interface. Cray User Group 2010 conference, Edinburgh, 2010.

[13] L. Snyder. Type architectures, shared memory, and the corollary of modest potential. *Annual Review of Computer Science*, 1:289–317, 1986.

[14] P.R. Spalart, G.N. Coleman, and R. Johnstone. Retraction: Direct numerical simulation of the Ekman layer: A step in Reynolds number, and cautious support for a log law with a shifted origin [phys fluids 20, 101507, 2008]. *Physics of Fluids*, 21:109901, 2009.

[15] A. Wray. Vectoral: only by trying new things will we ever get computer languages right. http://merrimac.stanford.edu/brook/vectoral.pdf.