Potential of Chaotic Iterative Solvers for CFD

J. Hawkes^{a,c*}**, G. Vaz**^{b}**, S. R. Turnock**^{c}**, S. J. Cox** ^{d}**, A. B. Phillips**^{c}

^{a}MARIN Academy; ^{b}Research and Development Department; MARIN, Wageningen, NL.
^{c}Fluid Structure Interactions (FSI); ^{d}Computational Engineering and Design (CED);
University of Southampton, UK.

1 Introduction

Computational Fluid Dynamics (CFD) has enjoyed the speed-up available from supercomputer technology advancements for many years. In the coming decade, however, the architecture of supercomputers will change, and CFD codes must adapt to remain current.

Based on the predictions of next-generation supercomputer architectures it is expected that the first computer capable of 10^{18} floating-point-operations-per-second (1 ExaFLOPS) will arrive in around 2020. Its architecture will be governed by electrical power limitations, whereas previously the main limitation was pure hardware speed. This has two significant repercussions [14, 17, 25]. Firstly, due to physical power limitations of modern chips, core clock rates will decrease in favour of increasing concurrency. This trend can already been seen with the growth of accelerated "many-core" systems, which use graphics processing units (GPUs) or co-processors. Secondly, inter-nodal networks, typically using copper-wire or optical interconnect, must be reduced due to their proportionally large power consumption. This places more focus on shared-memory communications, with distributed-memory communication (predominantly MPI - "Message Passing Interface") becoming less important.

The current most powerful computer, Tianhe-2 [26], capable of 33 PFlops, consists of 3,120,000 cores. The first exascale machine, which will be 30 times more powerful, is likely to be 300-times more parallel – which is a massive acceleration in parallelization compared to the last 50 years. This concurrency will come primarily from intra-node parallelization. Whereas Tianhe-2 features an already-large O(100) cores per node, an exascale machine must consist of O(1k-10k) cores per node.

CFD has benefited from *weak scalability* (the ability to retain performance with a constant elements-per-core ratio) for many years; its *strong scalability* (the ability to reduce the elements-per-core ratio) has been poor and mostly irrelevant. With the shift to massive parallelism in the next few years, the strong scalability of CFD codes must be investigated and improved.

In this paper, a brief summary of earlier results [12] is given, which identified the linear-equation system solver as one of the least-scalable parts of the code. Based on these results, a chaotic iterative solver, which is a totally-asynchronous, non-stationary, linear solver for high-scalability, is proposed. This paper focuses on the suitability of such a solver, by investigating the linear equation systems produced by typical CFD problems. If the results are optimistic, future work will be carried out to implement and test chaotic iterative solvers.

2 ReFRESCO

The work presented in this paper focuses on the development of ReFRESCO – a typical viscous-flow CFD code. ReFRESCO solves multiphase, unsteady, incompressible flows with the Reynolds-Averaged Navier Stokes (RANS) equations, complemented with turbulence models, cavitation models and volume-fraction transport equations for different fluid phases [27]. ReFRESCO represents a general-purpose CFD code, with state-of-the-art features such as moving, sliding and deforming grids and automatic grid refinement – but has been verified, validated and optimized for numerous maritime industry problems.

The RANS equations are discretized in strong conservation form using a finite-volume approach with cell-centred collocated variables. The SIMPLE algorithm is used to ensure mass conservation, with pressure-weighted interpolation (PWI) to tackle pressure-velocity decoupling issue arising from the collocated arrangement [16].

Time integration is performed implicitly with first or second-order backward schemes. At each time step, the non-linear system for velocity and pressure is linearized with Picard's method – and a segregated method applied. All non-linearity is tackled by means of an iterative process so-called the outer loop. For each outer-loop iteration, and for each transport equation, an algebraic system of linear equations is solved iteratively until a prescribed residual decay is achieved.

All numerical schemes used to discretize the transport equations (convection schemes, diffusion, gradients, non-orthogonality corrections, eccentricity corrections) apply their low-order contributions implicitly, to the left-hand side of the equation system; and their higher-order contributions explicitly, to the right-hand side of the system, using values from the previous outer loop.

The code is parallelized using MPI and sub-domain decomposition. The grids are partitioned in sub-domains, each one having a layer of common cells so-called ghost-cells. Each of these sub-domains is calculated in its own MPI process. The ghost-cells are treated as normal cells, as far as the numerical algorithms are concerned, and are therefore handled implicitly.

ReFRESCO is currently being developed at MARIN (Netherlands) [8] in collaboration with IST (Portugal) [22], the University of Sao Paulo (Brazil) [24], the Technical University of Delft (the Netherlands) [16], the University of Groningen (the Netherlands) [4] and recently at the University of Southampton (UK) [12].

[*] *corresponding author's e-mail*: J.Hawkes@soton.ac.uk

3 Strong Scalability Investigation

In previous work [12], ReFRESCO was profiled using *Score-P* [28] to extract timings from the code and its relevant functions. In a CFD code, these functions can be grouped together into the following categories, giving a breakdown of the code (see figure 1.a for illustration):

- **Assembly** – the assembly of each inner-loop linear equation system, including discretization, linearization and face-value interpolation for each transport equation.
- **Solve** – the iterative solving of the linear equation system for each transport equation, in each outer loop.
- **Gradients** – the calculation of gradient values at cell centres, using Gauss theorem.
- **Exchange** – MPI data exchange of cell-centred variables and gradients across ghost cells.
- **Other** – the remainder of the above, including initialization routines.

The results of a typical breakdown, using the well-known KVLCC2 case[2], are shown in figure 1, where the "speedup" is defined as the relative decrease in wall-time with N cores compared to the serial or single-node (16 cores) runtime. The results of the nodal speedup (1.b) show relatively good scalability (up to ≈40k cells-per-core), but hide the intra-node inefficiencies which are important for next-generation machines (with high intra-node parallelization). Graphs which show the speed-up relative to the single-core run-time (1.c) are thus more useful.

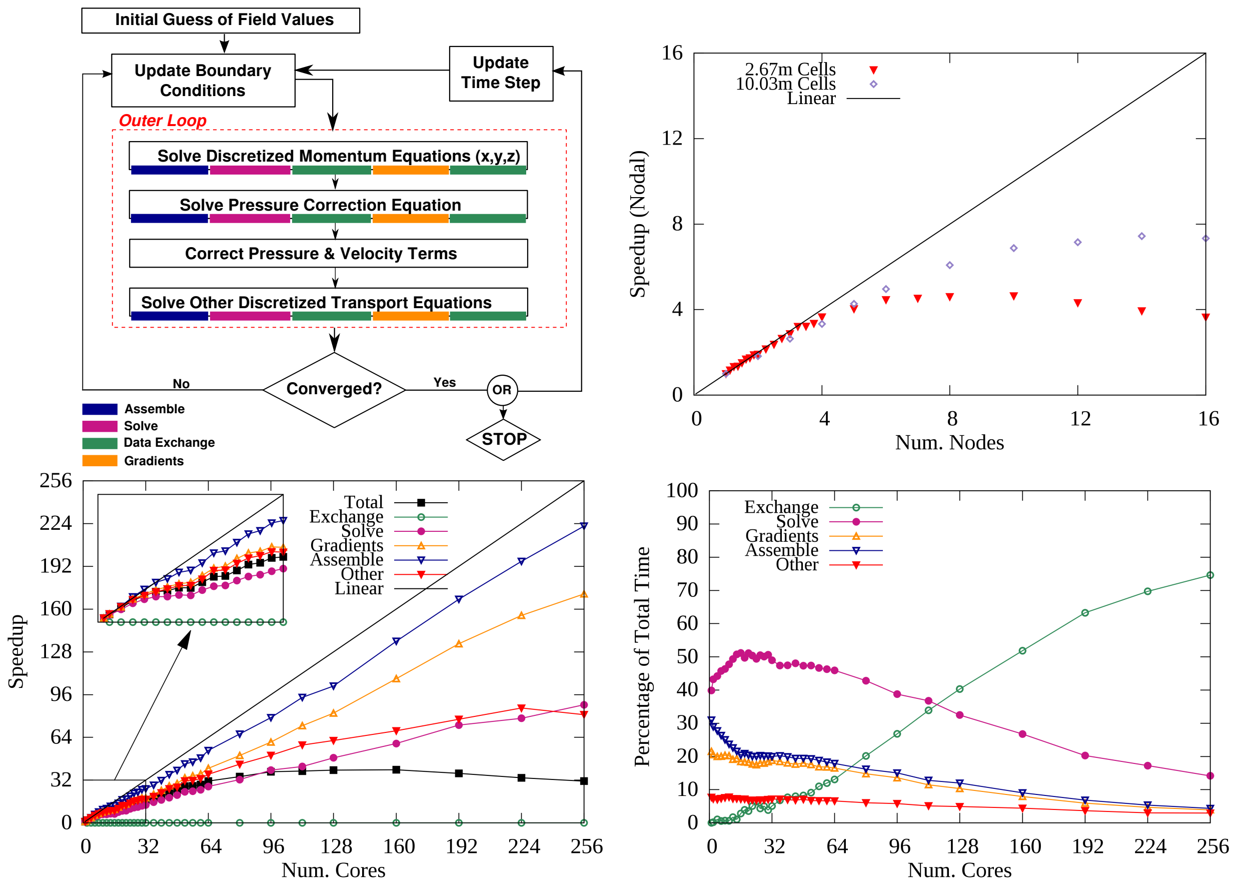

Figure 1: (a) An illustration of the SIMPLE algorithm, with colour-coding relating the various profiled functions of the code. (b) Total scalability of ReFRESCO normalized to single-node runtime, with two different grid sizes. (c) Scalability breakdown of a typical simulation showing the profiled functions individually and (d) the proportions of execution time spent in those routines [12].

The scalability graph (1.c) shows that *assembly* and *gradient* computations scale well, and the relative proportions graph (1.d) shows that these routines account for a very small proportion of run-time. The *other* routines do not scale so well, but are a small contribution to total run-time, so are of little concern.

The *solve* routines are an area for improvement. They exhibit poor scalability and take considerable amounts of total runtime, with particularly poor performance at the shared-memory level due to memory bandwidth or latency. Similarly, the *exchange* routines, which exhibit inverse scalability, are also a concern at the distributed-memory level. The data exchange is performed using MPI functions, and possibilities to improve this include switching to a hierarchical parallelization scheme (*i.e.* MPI + OpenMP) – as in [9, 10].

Up to ≈24k cells-per-core the iterative solver is the main limitation to scalability and should form the focus of future work, particularly as shared-memory parallelization grows. However, the data exchange issues are also an interesting area for further research and should not be neglected.

[2]KRISO Very Large Crude Carrier 2: half-body; two-equation κ-ω shear-stress transport (SST) turbulence model [21]; single-phase; based on wind-tunnel experiments [18]; 1000 outer loops; 2.67m structured grid.

4 Background to Iterative Solvers

The results shown in figure 1 used a Krylov subspace solver (GMRES - Generalized Minimal Residual method) with a Block Jacobi pre-conditioner [3]. Other Krylov methods such as BiCGStab (Bi-Conjugate Gradient Squared Stabilized) were tested with similar results. Other pre-conditioners were also tested, but Block Jacobi was (by far) the most scalable [12].

The Krylov solvers are powerful, but create a bottleneck due to the computation of inner products, which require global communication and synchronization in the form of MPI reductions. Efforts have been made to reduce the synchronization penalty of the Krylov solvers (down from two synchronized reductions to one, per iteration), with considerable improvements, but the bottleneck remains [20, 29].

By returning to simple, so-called stationary methods, it may be possible to obtain better performance in the limits of strong scalability. The task of a stationary solver (or, indeed, any iterative solver), for each transport equation in each outer loop, is to solve $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{x}$ is the unknown solution vector, $\mathbf{A}$ is an n-by-n sparse coefficient matrix, $\mathbf{b}$ is the constant right-hand-side (RHS) vector, and n is the number of elements.

Beginning with an initial guess for $\mathbf{x}$, the system can be solved iteratively:

$$\mathbf{x}^k = -\mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})\mathbf{x}^{k-1} + \mathbf{D}^{-1}\mathbf{b} \tag{1}$$

where $\mathbf{D}$ is the diagonal of $\mathbf{A}$, and $\mathbf{L}/\mathbf{U}$ are the lower- and upper-triangles respectively. The notation k represents the iteration number. This is the Jacobi method, and the matrix $\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U})$ is the iteration matrix, $\mathbf{M}$. Each equation (from 1 to n) can be solved (*a.k.a.* relaxed) independently as follows:

$$x_i^k = \left(-\sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij} x_j^{k-1} + b_i \right) / a_{ii}, \quad i = 1, \ldots, n. \tag{2}$$

where a, x and b are the individual components of $\mathbf{A}$, $\mathbf{x}$ and $\mathbf{b}$ respectively. At the end of each iteration the new values of $\mathbf{x}$ must be globally communicated before the next iteration can begin.

See Barrett et al. [5] for more information on a variety of iterative solvers – Krylov, stationary and otherwise.

5 Chaotic Iterative Solver

In 1969, Chazan & Miranker [7] proposed the concept of *Chaotic Relaxation* whereby several processes (distributed-memory processes or shared-memory threads) never synchronize. Instead, the processes freely pull values of off-diagonal $\mathbf{x}$ from memory whenever they are required, and push new values for the diagonal x_i whenever they have been relaxed. In this way, each relaxation uses the values of $\mathbf{x}$ from the latest iteration that is available – this could be several iterations behind the current relaxation iteration (s = 1,. . . ,n), or even ahead of it ($s < 0$):

$$x_i^k = \left(-\sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij} x_j^{k-s} + b_i \right) / a_{ii}, \quad s < s_{max}, \quad i = 1, \ldots, n. \tag{3}$$

The order in which the n equations are relaxed is completely arbitrary. With this scheme, processes never need to wait for each other. Although communication must still occur, it can be entirely *asynchronous*, thereby making efficient use of memory bandwidth and computational resources. Processes may even iterate multiple times on the same data if memory bandwidth is completely saturated, making the best use of the available hardware. Whilst this method is based on the stationary Jacobi method, s can vary between iterations implying that chaotic methods are actually non-stationary.

Chazan & Miranker proved that this iterative scheme will converge for any real iteration matrix if $\rho(|\mathbf{M}|) < 1$ so long as s_{max} is bounded. $\rho(\cdot)$ denotes the *spectral radius* (the absolute value of the maximum eigenvalue) and $|\cdot|$ represents a matrix where all the components have been replaced with their absolute values. The implications of s_{max} being bounded is simply that if two relaxations take different amounts of time (either due to imbalanced hardware or relaxation complexity), they cannot be left completely independent indefinitely, such that s could potentially become infinite. Baudet [6] went on to prove that $\rho(|\mathbf{M}|) < 1$ is a necessary condition for convergence for any $s_{max} \leq k$. Baudet denoted the relaxation method where $s = 0, ..., k$ as an *asynchronous method* but the terms "chaotic" and "asynchronous" are often used interchangeably. Bahi [2] further showed that $\rho(|\mathbf{M}|) = 1$ is also valid, if $\mathbf{M}$ is singular and s_{max} is bounded.

Preconditioning of $\mathbf{A}$ usually serves to reduce the condition number and spectral radius of the equation system; however, preconditioning is rarely applied to simpler solvers, since almost all preconditioners are more complex than the solver itself.

At their conception, chaotic methods were considerably ahead of their time. Although created specifically for parallel computing, the concurrency of state-of-the-art supercomputers in 1969 was too small to utilize the methods efficiently. With new architectures, the true potential of chaotic methods may be realized. Anzt et al. [1] begins to show the use of chaotic or asynchronous methods on a modern architecture, using the GPU to perform block-relaxations. Despite the intrinsic loss in global convergence rates, Anzt et al. showed that chaotic iterative methods provided a boost in real-time convergence rates compared to standard stationary methods (*i.e.* Jacobi) – although comparisons to the more advanced Krylov methods were not performed, and the chosen matrices were not derived from CFD applications.

Chaotic methods provide a means to exploit massive parallelism due to the absence of synchronization points. They are also implicitly heterogeneous, allowing seamless cooperation between CPUs, GPUs or co-processors running at different speeds – allowing all computational resources to be used to maximum capacity with little concern for load-balancing. The convergence criteria, $\rho(|\mathbf{M}|) < 1$, is stricter than that of standard stationary methods which only require $\rho(\mathbf{M}) < 1$, and stricter still than the oft-used Krylov methods (which have no such requirements). The following section aims to determine if a range of standard CFD test cases will produce matrices that satisfy the criteria – the results of which will determine whether chaotic solvers are worth implementing and investigating.

6 Suitability of CFD Equation Systems

In this section, un-preconditioned matrices are extracted from a number of test cases. The matrices are analyzed to obtain key statistics and determine their suitability to chaotic methods.

For each matrix, it is possible to plot the connectivity graph, following the methods of Hu [15] – this gives a qualitative, visual insight into the sparse matrices by graphically connecting the elements of the matrix. The largest eigenvalues of $|\mathbf{M}|$ can be found using ARPACK [19] routines, and plotted in an Argand diagram – for $\rho(|\mathbf{M}|) < 1$ all eigenvalues must lie within a unit circle. The sparsity pattern of the matrix $\mathbf{A}$ may also be plotted directly to give qualitative clues on the matrix structure.

Assuming a satisfactory spectral radius, the *condition number* (the ratio of the largest to smallest eigenvalue) of the original matrix $\mathbf{A}$ can be used to assess the difficulty of convergence ($\propto$ number of iterations required). The condition number is computed using a 1-norm condition estimator [11].

As explained in section 2, all higher-order influences on the linear equation system are shifted to the RHS (the $\mathbf{b}$ vector) – which strongly decouples many common user settings (such as discretization scheme) from the format of the iteration matrix $\mathbf{M}$.

Changes in element-count and geometry may have a more profound effect on the matrices, and special equations (such as volume-fraction and cavitation equations) should also be examined. Thus the following test cases are chosen for increasing geometric complexity and their additional equations.

- **Lid-Driven Cavity Flow** (LDCF) on a number of 2D structured grids (225, 14.4k, 1m elements); no turbulence model; see [16].
- **NACA0015** hydrofoil (15° angle-of-attack) on a two-dimensional multi-block structured grid (28k elements) with a two-eqn. κ-ω SST turbulence model [21]; see [23].
- **KVLCC2** (half-body, no free surface, single-phase) on a three-dimensional multi-block structured grid (317k elements) and a hexahedral unstructured grid (**U**) with hanging nodes (167k elements); κ-ω SST turbulence; see [12].
- **NACA0015(C)**, as before, but with a Sauer-modified cavitation model; κ-ω SST turbulence; see [13].
- **Dambreak**, a homogeneous two-phase, three-dimensional problem with a simple structured grid (16k cells) with a volume-fraction equation; no turbulence; see [27].
- **Cylinder**, low Reynold's number, unsteady (10 timesteps) on a structured grid (4.3k elements); no turbulence; poor initial flow estimation; see [24].

The matrices were extracted at outer loops 1, 5, 10, 50 and 100. The momentum equations (in x, y and z) are identical, due to the way in which they are assembled. The differences between $\rho(|\mathbf{M}|)$ and $\rho(\mathbf{M})$ were negligible, implying that $\mathbf{M}$ is mostly positive.

Figure 2 shows the qualitative view of the simple 2D LDCF, the more complex 2D hydrofoil, and the 3D KVLCC2 case at the 5^{th} outer loop. The connectivity graphs show resemblance to the underlying geometry and mesh structure – for example, the NACA0015 connectivity resembles the O-grid from which it arose.

The sparsity patterns are interesting from a computational perspective, since they highlight communication patterns when the matrix is split into parallel blocks. Where off-diagonal components of an equation are spread out, more cross-communication between processes is required – since the variables will be stored in parts of memory not directly accessible. The sparsity is closely related to the structure of the grid and the cell-numbering – the more complex KVLCC2 case is highly complex compared to the cartesian, structured LDCF grid.

Table 1 shows the quantitative results from all the test cases, taking the maximum values of all the extracted outer loops. The condition number appeared to be higher for more complex cases, such as the 3D KVLCC2 case, although there was little correlation with flow features or mesh structure. In all cases, $\rho(|\mathbf{M}|) \leq 1$, meeting the requirements for chaotic solvers.

The pressure equation, which takes a Poisson-equation format, was singular for the LDCF and Dambreak case, where only Neumann boundary conditions are applied. In all the other cases, a Dirichlet condition on the outflow reduced the spectral radius, although it was still close to one. In the LDCF-225 case and the first timestep of the unsteady cylinder, the spectral radius was also very high – these cases both feature complex flows with simple initial-flow estimations and coarse meshes; this large discrepancy and poor resolution could be the reason for the the near-singular matrices (however, the condition number was still low).

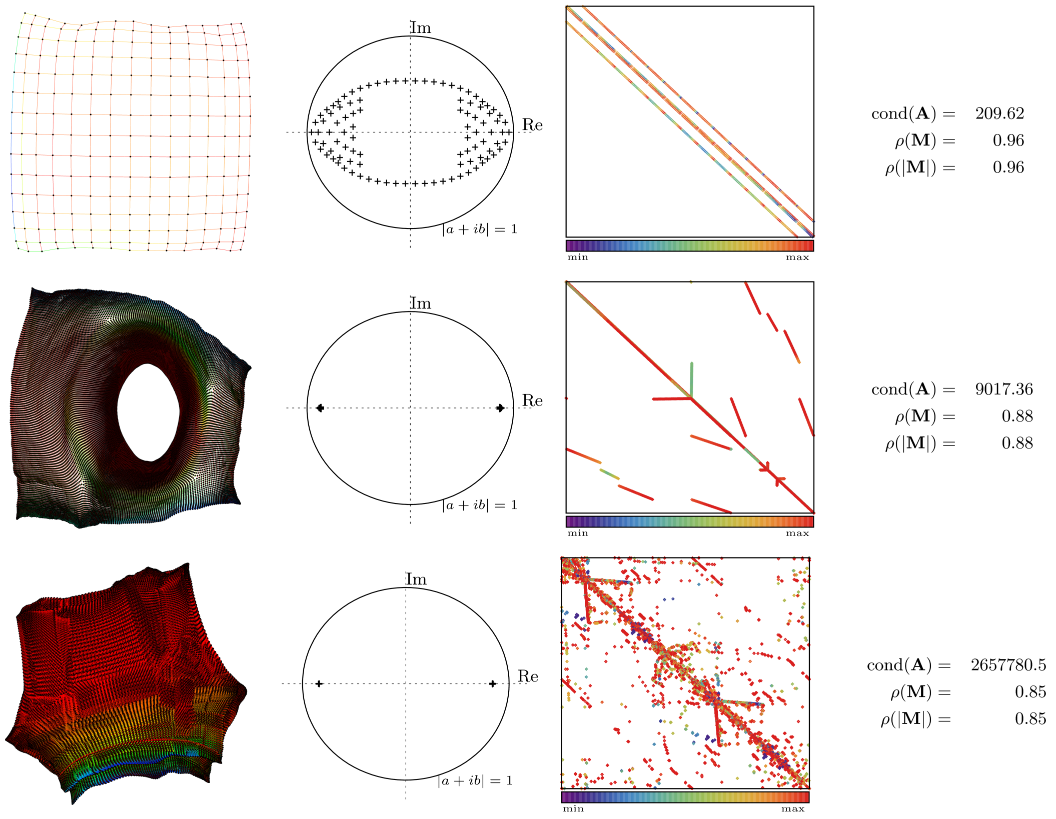

Figure 2: Connectivity, 100 largest eigenvalues, sparsity and statistics for the momentum equation (outer loop 5). [Top] LDCF-225, [Middle] NACA0015-28k and [Bottom] KVLCC2-317k.

Additional matrices for the KVLCC2 case (up to 2.67m elements) and a 21.7m-element INSEAN E779A propeller (with sliding interface) were tested, but could not be fully post-processed due to memory requirements – nonetheless, the original matrices (**A**) were diagonally dominant: a sufficient condition for $\rho(|\mathbf{M}|) < 1$.

Table 1: Quantitative results for a range of test matrices, showing condition number and spectral radius for momentum-, pressure-, turbulence- and free-surface-/cavitation-equations.

	Mom.		Pres.		Turb. 1		Turb. 2		V.F./Cav.	
	cond(**A**)	$\rho(\|\mathbf{M}\|)$	cond(**A**)	$\rho(\|\mathbf{M}\|)$	cond(**A**)	$\rho(\|\mathbf{M}\|)$	cond(**A**)	$\rho(\|\mathbf{M}\|)$	cond(**A**)	$\rho(\|\mathbf{M}\|)$
LDCF-225	210	0.955	9.4·e17	1.000						
LDCF-14.4k	7	0.500	2.1·e18	1.000						
LDCF-1m	4	0.500	2.9·e19	1.000						
NACA0015	9.0·e3	0.884	6.53·e6	0.999	4.5·e3	0.849	4.8·e3	0.851		
NACA0015(C)	6.3·e3	0.734	4.67·e6	0.999	5.1·e3	0.688	5.2·e3	0.690	2.3·e8	0.240
KVLCC2	2.6·e6	0.845	2.63·e8	.9999	1.2·e7	0.588	1.2·e7	0.594		
KVLCC2 (unst.)	5.2·e6	0.810	5.4·e7	.9999	8.3·e6	0.726	7.7·e6	0.728		
Dambreak	105	0.048	5.6·e18	1.000					7	0.048
Cylinder (t=1)	111	.9999	1.1·e6	.9999						
Cylinder (t=2–10)	4.4·e2	0.520	1.1·e6	.9999						

7 Conclusion

Chaotic iterative solvers have been proposed as a means to improve the scalability of a typical CFD code, in preparation for a paradigm-shift towards massive parallelization in supercomputing architectures. The removal of synchronization points from one of the major bottlenecks in the code (the *solve* routines) could lead to a vast improvement in scalability, although the loss in convergence rate which compromises the speed-up is difficult to predict. The matrices which the chaotic solver would be required to solve have been extracted and evaluated, and it has been determined that necessary and sufficient conditions for convergence have been satisfied.

Future work will focus on implementing a chaotic solver in a hierarchical parallel environment (MPI & OpenMP & GPU/coprocessor). At the shared-memory (OpenMP) level, issues associated with double-precision atomic operations and intra-core cache-coherency will be encountered; and methods must be investigated to hide any cache latency that may occur. Maintaining true asynchronicity across memory-boundaries (for example, across nodes, using MPI) will also be a unique challenge – since most MPI communications require at least two nodes to synchronize. One-sided MPI operations may be investigated to allow direct access to non-shared memory. Similarly, true asynchronicity between host processes and attached GPUs/coprocessors must be achieved – perhaps by assigning processes on the host processor purely for these communications.

From a numerical perspective, it is best if each equation in the system is iterated the same number of times/at the same speed (*i.e.* the equations do not become too out-of-synch). Regulating this, by moving equations to different processes dynamically, may be beneficial for overall performance and may be necessary for convergence when the pressure equation is singular.

Above all, however, is the intrinsic difficulty of debugging a chaotic scheme. As soon as instrumentation points are added, the chaotic order of the process changes – indeed, every time the process is run, different numerical results and convergence rates may be obtained.

Chaotic methods introduce a number of unique challenges, but could create a significant speed-up for CFD on current and next-generation supercomputers. Whilst it has been shown that the chaotic methods will converge for the relevant equations, it remains to predict their real-world performance for CFD applications; and to implement and test them successfully.

References

[1] Anzt, H., Tomov, S., Dongarra, J. & Heuveline, V. (2013, December). A Block-Asynchronous Relaxation Method for Graphics Processing Units. *Journal of Parallel and Distributed Computing*, 73(12):1613–1626.

[2] Bahi, J. (2000). Asynchronous Iterative Algorithms for Nonexpansive Linear Systems. *Journal of Parallel and Distributed Computing*, 60:92–112.

[3] Balay, S., Abhyankar, S., Adams, M. F., Brown, J., Brune, P., Buschelman, K., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Rupp, K., Smith, B. F. & Zhang, H. (2013). PETSc Users Manual. Technical Report ANL-95/11 - Revision 3.4, Argonne National Laboratory. `http://www.mcs.anl.gov/petsc`.

[4] Bandringa, H., Verstappen, R., Wubbs, F., Klaij, C. & Ploeg, A. (2012, October). On Novel Simulation Methods for Complex Flows in Maritime Applications. *Numerical Towing Tank Symposium (NUTTS)*, Cortona, Italy.

[5] Barrett, R., Berry, M., Chan, T. F., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C. & der Vorst, H. V. (1994). *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition.* SIAM, Philadelphia, PA.

[6] Baudet, G. M. (1978, April). Asynchronous Iterative Methods for Multiprocessors. *J. ACM*, 25(2):226–244.

[7] Chazan, D. & Miranker, W. (1969, April). Chaotic Relaxation. *Linear Algebra and its Applications*, 2(2):199–222.

[8] Eca, L. & Hoekstra, M. (2012, August). Verification and Validation for Marine Applications of CFD. *29th Symposium on Naval Hydrodynamics*, Gothenburg, Sweden.

[9] Gorobets, A., Trias, F. & Oliva, A. (2013, December). A Parallel MPI+OpenMP+OpenCL Algorithm for Hybrid Supercomputations of Incompressible Flows. *Computers & Fluids*, 88:764–772.

[10] Gropp, W. D., Kaushik, D. K., Keyes, D. E. & Smith, B. F. (2001, March). High-Performance Parallel Implicit CFD. *Parallel Computing*, 27(4):337–362.

[11] Hager, W. W. (1984). Condition Estimates. *SIAM Journal on Scientific and Statistical Computing*, 5:311–316.

[12] Hawkes, J., Turnock, S. R., Cox, S. J., Phillips, A. B. & Vaz, G. (2014, October). Performance Analysis Of Massively-Parallel Computational Fluid Dynamics. Submitted to *The 11th International Conference on Hydrodynamics (ICHD)*, Singapore.

[13] Hoekstra, M. & Vaz, G. (2009, August). The Partial Cavity on a 2D Foil Revisited. In proceedings of the *7th International Symposium on Cavitation*, Ann Arbor, MI, USA.

[14] Horst, S. (2013, May). Why We Need Exascale And Why We Won't Get There By 2020. *Optical Interconnects Conference*, Santa Fe, New Mexico, USA.

[15] Hu, Y. (2005). Efficient, High-Quality Force-Directed Graph Drawing. *Mathematica Journal*, 10(1):37–71.

[16] Klaij, C. & Vuik, C. (2013). Simple-Type Preconditioners for Cell-centered, Collocated, Finite Volume Discretization of Incompressible Reynolds-averaged Navier-Stokes Equations. *International Journal for Numerical Methods in Fluids*, 71(7):830–849.

[17] Kogge, P., Bergman, K., Borkar, S., Campbell, D., Carlson, W., Dally, W., Denneau, M., Franzon, P., Harrod, W., Hill, K., Hiller, J., Karp, S., Keckler, S., Klein, D., Lucas, R., Richards, M., Scarpelli, A., Scott, S., Snavely, A., Sterling, T., Williams, S. & Yelick, K. (2008, September). ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems. *DARPA IPTO*.

[18] Lee, S., Kim, H., Kim, W. & Van, S. (2003). Wind Tunnel Tests on Flow Characteristics of the KRISO 3,600 TEU Container Ship and 300K VLCC Double-Deck Ship Models. *Journal of Ship Research*, 47(1):24–38.

[19] Lehoucq, R. B., Sorensen, D. C. & Yang, C. (1998). *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems by Implicitely Restarted Arnoldi Methods.* SIAM, Philadelphia, PA, USA.

[20] Maheswaran, M., Webb, K. & Siegel, H. (1998, August). Reducing the Synchronization Overhead in Parallel Nonsymmetric Krylov Algorithms on MIMD Machines. In proceedings of *International Conference on Parallel Processing*, Minneapolis, Minnesota, USA.

[21] Menter, F., Kuntz, M. & Langtry, R. (2003, October). Ten Years of Industrial Experience with the SST Turbulence Model. In *Turbulence, Heat and Mass Transfer 4*. Antalya, Turkey.

[22] Pereira, F., Eca, L. & Vaz, G. (2013, June). On the Order of Grid Convergence of the Hybrid Convection Scheme for RANS Codes. In proceedings of *CMNI*, Bilbao, Spain.

[23] Rijpkema, D. R. (2008, November). Numerical Simulation of Single-Phase and Multi-Phase Flow over a NACA 0015 Hydrofoil. M.Sc. Thesis, Delft University of Technology, Faculty of 3ME.

[24] Rosetti, G., Vaz, G. & Fujarra, A. (2012, July). URANS Calculations for Smooth Circular Cylinder Flow in a Wide Range of Reynolds Numbers: Solution Verification and Validation. *Journal of Fluids Engineering, ASME*, page 549.

[25] Shalf, J. (2013, October). The Evolution of Programming Models in Response to Energy Efficiency Constraints. *Oklahoma Supercomputing Symposium*, Norman, Oklahoma, USA.

[26] Top 500 List (Acc. 2013, February). `http://www.top500.org`.

[27] Vaz, G., Jaouen, F. & Hoekstra, M. (2009, May–June). Free-Surface Viscous Flow Computations: Validation of URANS Code FRESCO. *28th International Conference on Ocean, Offshore and Arctic Engineering (OMAE)*, Honolulu, Hawaii.

[28] VI-HPS, Score-P, v.1.2.3 (Acc. 2013, November). `http://www.vi-hps.org/projects/score-p`.

[29] Zuo, X.-Y., Zhang, L.-T. & Gu, T.-X. (2014, December). An Improved Generalized Conjugate Residual Squared Algorithm Suitable for Distributed Parallel Computing. *Journal of Computational and Applied Mathematics*, 271:285–294.