VI International Conference on Computational Methods in Marine Engineering
MARINE 2015
F. Salvatore, R. Broglia and R. Muscari (Eds)

CHAOTIC LINEAR EQUATION-SYSTEM SOLVERS FOR UNSTEADY CFD

JAMES N. HAWKES[1,2], STEPHEN R. TURNOCK[1], GUILHERME VAZ[2], SIMON J. COX[1] AND ALEX B. PHILLIPS[3]

University of Southampton [1]
Boldrewood Campus, Southampton, United Kingdom
e-mail: J.Hawkes@soton.ac.uk

Maritime Institute Netherlands (MARIN) [2]
Wageningen, Netherlands

National Marine Facilities [3]
National Oceanography Centre, United Kingdom

Key words: Computational Methods, Computational Fluid Dynamics, Linear Solvers, Totally Asynchronous Methods, Chaotic Iterative Methods, PETSc, ReFRESCO.

Abstract. A Chaotic Iterative Method, which is a form of totally asynchronous linear equation-system solver, is implemented within an open-source framework. The solver is similar to simple Jacobi or Gauss-Seidel methods, but is highly optimized for massively-parallel computations. Processes or threads are free to run computations regardless of the current state of other processes, iterating individual equations with no limitations on the state of the variables which they use. Each individual iteration may pull variables from the same iteration, the previous iteration, or indeed any iteration. This effectively removes all synchronization from the Jacobi or Gauss-Seidel algorithm, allowing computations to run efficiently with high concurrency.

The trade-off is that the numerical convergence rate of these simple algorithms is slower compared to the classical Krylov Subspace methods, which are popular today. However, unique features of the computational fluid dynamics algorithm work in favour of Chaotic methods, allowing the fluid dynamics field to exploit these algorithms when other's cannot.

The results of the Chaotic solver are presented, verifying the numerical results and benchmarking performance against the Generalized Minimal Residual (GMRES) solver and a Pipelined GMRES solver. The results show that, under certain circumstances, Chaotic methods could be used as a standalone solver due to their superior scalability. The potential to use Chaotic methods as a pre-conditioner or hybrid solver is also revealed.

1 INTRODUCTION

One of the major objectives in high performance computing for computational fluid dynamics is to enable complex unsteady simulations. Whilst these computations benefit from concurrency in three spatial dimensions, time-stepping is strictly serial; and unsteady computations become bottle-necked by the amount of parallelism that can be utilized in the spatial dimensions. Combined with this, the architecture of modern supercomputers is changing rapidly, causing scientific applications to adapt to a many-core era.

Figure 1.a shows the history of the Top 500 supercomputers over the last 20 years. At current rates, it is expected that the first computer capable of 10^{18} floating-point-operations-per-second (1 ExaFLOPS) could be built in 2020. Its capability will be limited by electrical power consumption, rather than pure hardware speed, and this will lead to two significant architecture changes [9, 11, 14]. Firstly, core clock rates must decrease to save power, and total computing power must be improved by dramatically increasing the total number of cores. Secondly, inter-nodal networks must be reduced due to their relatively large power consumption. Overall, this means a huge growth in concurrency within nodes. This trend can already be seen in modern 'many-core' systems featuring GPUs or co-processors.

The current most powerful computer, Tianhe-2 [15], capable of 33 PFlops, consists of 3,120,000 cores. By contrast, the first exascale machine will be 30-times more powerful and 300-times more parallel. Whilst Tianhe-2 *already* consists of nodes with O(100) cores, using co-processors; an exascale machine is likely to feature O(1k-10k) cores per node.

In earlier work [6] the strong scalability of a typical CFD code (ReFRESCO) was assessed, to discover which parts of the algorithm required improvement for this paradigm-shift in architecture. Figure 1.b shows the breakdown of the code, which is based on the SIMPLE algorithm, into its key parts. Figure 1.c shows that the *assembly* (discretization, orthogonality & eccentricity corrections, and final assembly of the matrix structure) and *gradient computations* scale well, and figure 1.d shows that these routines account for a small proportion of the overall run-time. The *solve* routines, responsible for solving the linear-equation systems that have been assembled, exhibit poor scalability and take considerable amounts of total runtime. Similarly, the *exchange* routines (responsible for sharing of cell-values and gradient values across ghost cells) are a cause for concern, dominating total run-time when inter-nodal communication becomes large.

Considerable work is being performed to improve the *exchange* routines, primarily by overlapping communication with computation more efficiently or reducing global communications; and by utilizing a hybrid parallelization scheme. Cooperatively, the *solve* routines must be improved with the same goals.

In this paper, a 'Chaotic' iterative solver is proposed as a solution to the strong scalability problem of the *solve* routines. In the following sections, a brief background is given, followed by the theory and implementation of a Chaotic solver. Finally, the Chaotic solver is verified and benchmarked against existing solvers.

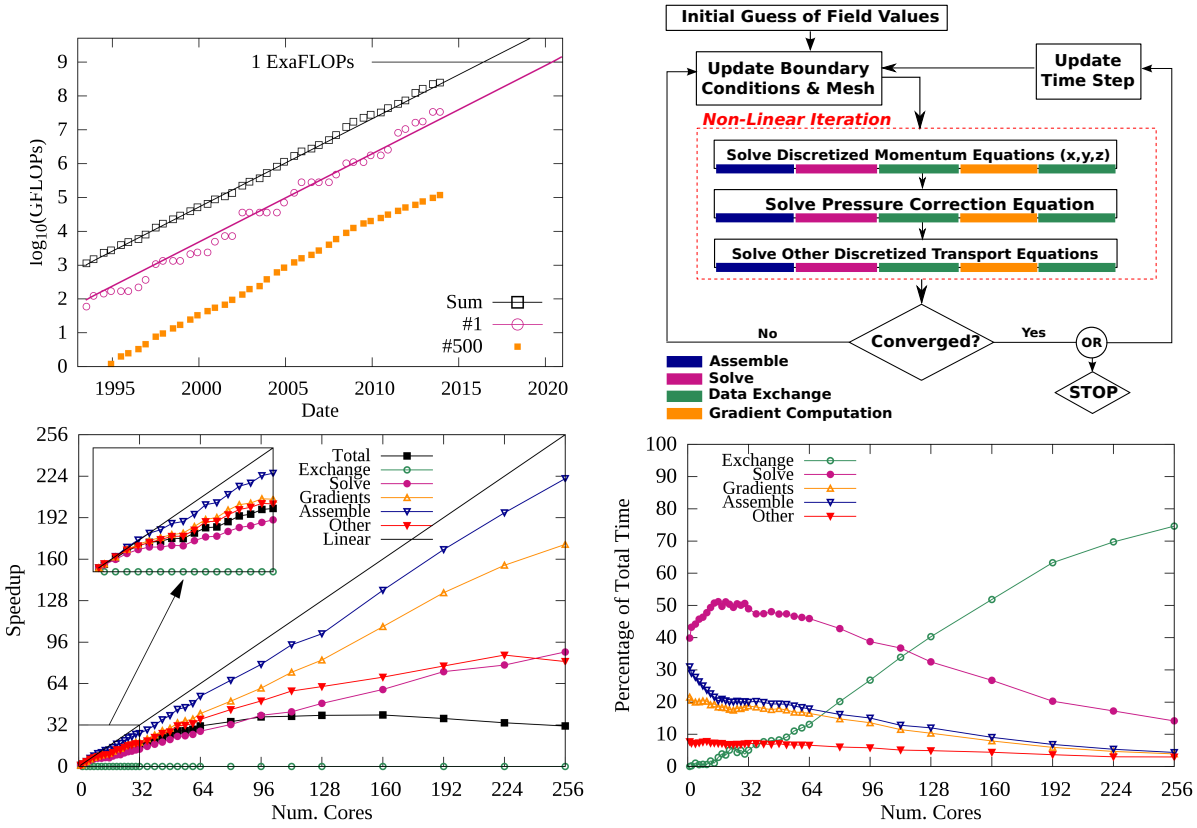


Figure 1: (a) Total floating-point operation (FLOP) rate of the Top500 supercomputers, showing the #1 machine, the #500, and the sum of the top 500. (b) An illustration of the SIMPLE algorithm, with colour-coding relating the various sections of the code. (c) Scalability breakdown of a typical simulation showing the parallel speed-up of the code compared to its serial operation and (d) the proportions of execution time spent in the various routines [6].

2 BACKGROUND TO LINEAR EQUATION-SYSTEM SOLVERS

The task of the linear equation-system solver, for each transport equation in each non-linear 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 created by the *assembly* routines, $\mathbf{b}$ is the constant right-hand-side (RHS) vector, and n is the number of elements.

It is possible to solve this system directly, but for large matrices this is expensive. For CFD, the equation systems 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. k is the iteration number. This is the Jacobi method, and the matrix $\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U})$ is the iteration matrix, $\mathbf{M}$.

Numerically, the Jacobi method, or its derivatives such as Gauss-Seidel or Successive Over-Relaxation methods, are weak. Their convergence rate is limited by the spectral

radius of the iteration matrix, $\rho(\mathbf{M})$, which is dependent on the matrix size and stiffness. For CFD, this means the pressure equation is hardest to solve, as it is close to singular [7] and the spectral radius is close to unity.

It is possible to add a form of 'memory' to these solvers, where some information about the solution path is stored in a vector subspace, known as the Krylov Subspace (KSP). For example, in the popular (Bi-)Conjugate Gradient Squared (BCGS) algorithm, the solver remembers the previous gradients that it followed, to prevent it from going back on itself. In the GMRES (Generalized Minimal Residual Method) the subspace is based on the Arnoldi iteration, and is used to find an approximate solution based on the minimal residual of these vectors. See Barrett et al. [2] for more information on a variety of iterative solvers.

There is an initial grace period where high frequency errors can be reduced more quickly than the spectral radius would normally allow, using the simpler methods, but KSP solvers typically over-power the simpler solvers when lower frequencies must also converge.

Though numerically powerful, the KSP methods are computationally inefficient on a larger number of cores due to certain communication patterns. Figure 2 shows an illustration of a GMRES iteration. Two significant communication patterns are required, with very different scaling characteristics. Firstly, the sparse-matrix-vector-multiplication (SpMV) requires neighbour-to-neighbour communication to obtain boundary data from other processors. This communication pattern is highly scalable, as different neighbours can communicate in parallel.

The second communication pattern is a global reduction (and associated broadcast). Many small messages are sent down a hierarchical tree in a non-concurrent manner. The time for a reduction *increases* as the number of cores increases, whereas every other routine receives a speed-up from increasing the number of cores.

In the GMRES or BCGS algorithm there are two global reductions per iteration. As the number of cores rises and the scalable routines reduce to a small proportion of overall time, these reductions dominate wall-time and prevent the KSP solvers from scaling. Improvements have been made to GMRES, in the form of 'pipelined' GMRES (P-GMRES) which attempt to overlap the first reduction with serial computations [5] – results show better scalability than GMRES, especially when the so-called 'latency-factor' is large, but as the cells-per-core ratio decreases global reduction latency will once again dominate.

Compared to other applications, where solving linear equation systems to machine accuracy is important (for example, solving Markov chains for risk analysis), CFD is not so strict. As the 'real' problem is a non-linear one, much of the work done in the linear loops is discarded as under-relaxation is applied to the non-linear iteration. In fact, for many cases, it is possible to reach non-linear convergence with a relative convergence factor of the linear system as high as 60%, although this is not usually beneficial. It highly depends on the problem and the type of non-linear iterative process; and is only worthwhile if the non-linear iterative process is efficient, as many more non-linear iterations are required.

With this in mind, it may be possible to take advantage of the initial 'grace period' of the simpler solvers such as Jacobi or Gauss-Seidel; however, to compete with a well-

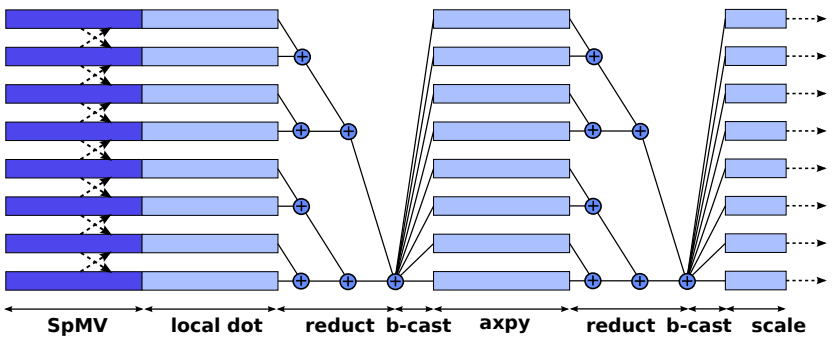


Figure 2: Illustrative parallel trace of a GMRES iteration, showing the computation and communication patterns when utilizing 8 cores. Note that as the serial computations and SpMV become small, the reduction dominates the total wall-time. Not to scale.

preconditioned GMRES solver there must be a significant computational benefit.

3 THEORY AND IMPLEMENTATION OF A CHAOTIC SOLVER

The Jacobi solver is particularly poor numerically, using old values of the solution vector in each iteration rather than using newly-computed variables from the current iteration. By contrast, Gauss-Seidel is stronger numerically as it uses newly-computed variables; but is impossible to parallelize. Forms of Block Gauss-Seidel exist, which are a good compromise; but both Jacobi and Block Gauss-Seidel can be improved computationally.

The Jacobi or Block Gauss-Seidel methods have no reduction routines, using purely an SpMV to perform the iterations. This is a huge benefit to scalability, and should allow these solvers to scale far higher than KSP methods. However, the Jacobi/Block G-S routine still has room for improvement. In these methods, the $k + 1^{th}$ iteration cannot start until the k^{th} iteration is complete. This means that the whole solution can only proceed as fast as the slowest core. A core may be slow due to background operating-system load, poor load-balancing, heterogeneity in the hardware, irregular communication times or Non-Uniform Memory Access (NUMA). These irregularities are likely to be more of a concern in the many-core era, and will significantly slow down computations when trying to utilize thousands of cores.

In 1969, Chazan & Miranker [3] proposed the concept of *Chaotic Relaxation* whereby several processes 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. By exploiting data races in this way, each relaxation uses the values of $\mathbf{x}$ from the latest iteration that is available – this could be several iterations behind ($s = 1, ..., k$), or even ahead of it ($s < 0$):

$$\mathbf{x}^k = -\mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})\mathbf{x}^{k-s} + \mathbf{D}^{-1}\mathbf{b} \tag{2}$$

In the average case, this scheme should produce numerical convergence similar to that of Gauss-Seidel.

In previous work [7] it was shown that the strict convergence criteria for Chaotic methods are met by all CFD equations, which prompts the implementation and testing of a Chaotic solver, as follows.

The solver is written in a hybrid MPI + OpenMP (Open Multi-Processing – a multi-threading standard) environment, using PETSc (Portable, Extensible Toolkit for Scientific Computation) [1] as a platform. The MPI processes and OpenMP threads are split in a NUMA-friendly manner, with one MPI process per socket and one thread per core within each socket. The test machine is Iridis 4, the University of Southampton's latest super-computer, which consists of 750 compute nodes - each with 2 Intel Xeon E5-2670 Sandy-bridge processors (8 cores, 2.6 Ghz). Within each MPI process one thread is dedicated to halo-data communication and the remaining seven perform the chaotic matrix-vector multiplications.

In the following sections, the results of the Chaotic solver will be verified by comparing the solutions produced by them against KSP methods, and the performance (in terms of scalability and absolute wall-time) will be assessed.

4 VERIFICATION

The solution produced by the Chaotic solver was compared against the solution produced by GMRES (with a restart of 3, 30 and 100), P-GMRES and BCGS. Pre-conditioners were not used and there was no attempt to compare performance, these tests were performed purely to compare the absolute solution and verify the implementation. The solvers were required to iterate until a desired absolute residual in the l^2-norm.

Three analytical matrices from the matrix marketplace [4] were tested ($pde225$, $pde900$ and $orsirr\_1$), along with a pressure equation matrix extracted from a 14.4k-cell lid-driven cavity flow simulation ($LDCF$). The results for the smallest case ($pde225$) are shown in figure 3. The results were similar for all cases and a range of convergence tolerances from 10^{-1} to 10^{-7} (not shown).

There are no concerns for any of the solvers, although there are some differences in the solution vector. There is a tendency for the Chaotic solver to produce a much lower l^1-norm, presumably as the stiffer equations in the system require many iterations to solve, the easier equations can progress much more quickly, whereas the KSP methods have more inter-dependency between the equations.

Due to the squared-weighting of the l^2-norm this has little effect on the reliability of the solution, and differences in the overall flow field are likely to be negligible.

5 PERFORMANCE TESTING

The aim of Chaotic solvers is to improve scalability, and it is expected that at lower core-counts that KSP solvers will out-perform the Chaotic methods. In order to predict how these solvers will perform on many-core machines, scalability tests are the main focus of the following tests. It should be possible to predict the break-even point of Chaotic methods, where their worse numerical performance overtakes the KSP solvers based on improved scalability.

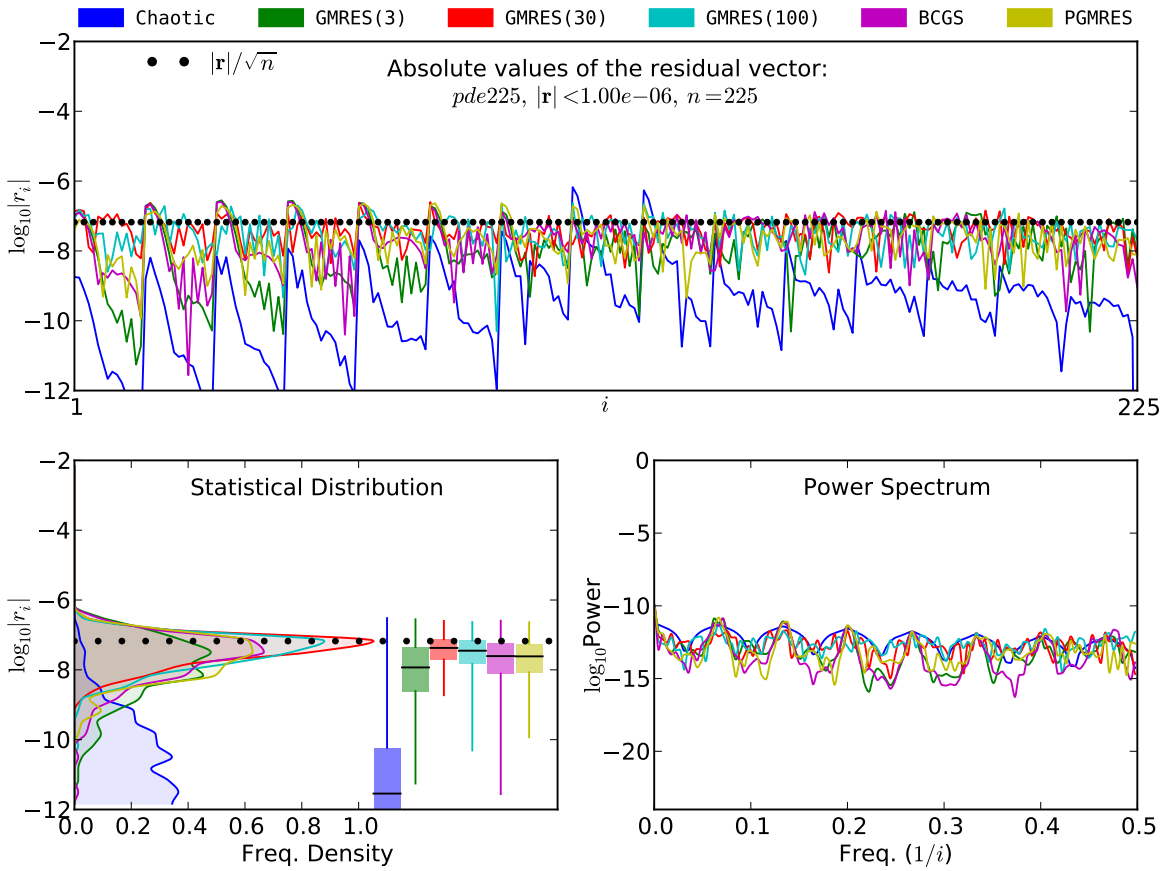


Figure 3: Full verification results from the *pde225* case, with a requested absolute l^2-norm of 10^{-6}. [Top] A plot of the raw residual vector ($\mathbf{r} = \mathbf{Ax} - \mathbf{b}$). [Bottom-Left] Statistical Distribution of the residual vector, showing the mean, upper/lower quartiles, and minimum/maximum values. [Bottom-Right] Spectral analysis of the residual vector, with peaks corresponding to frequencies of error in the converged system. The dotted lines in the results mark the *mean value of the residual required in order to satisfy the l^2-norm*. This line divides the elements of the residual that positively or negatively contribute to the l^2-norm.

A number of benchmark cases are used in order to capture a range of problem sizes:

- **KVLCC2 Tanker** (half-body, single-phase) on three-dimensional, multi-block structured grids (317k, 1.11m, 2.67m, 5.99m, 10.0m & 15.8m elements), using a two-equation κ-ω SST turbulence [12]. See [6].
- **Lid-Driven Cavity Flow** (LDCF) on 2D structured grids (14.4k & 1m elements). No turbulence model. See [10].
- **NACA0015C** hydrofoil (15° angle-of-attack) on a two-dimensional multi-block structured grid (28k elements), using a κ-ω SST turbulence model, and a Sauer-modified cavitation model. See [8].
- **Dambreak**, a homogeneous two-phase, three-dimensional problem with a simple structured grid (16k cells) and a volume-fraction equation. No turbulence model.

See [16].

- **Cylinder** at a low Reynold's number. An unsteady simulation (10 non-linear iterations per timestep) on a structured grid (4.3k elements) with a poor initial flow estimation. No turbulence model. See [13].

A large number of core-counts are tested, beginning from 32 cores, to which the scalability graphs are normalized:

$$Scalability = \frac{Time(\text{32-cores})}{Time(\text{n-cores})} \tag{3}$$

The linear system convergence tolerance is set to 60% and 50 non-linear iterations are performed. Chaotic solvers were compared against GMRES and P-GMRES, both with a restart of 30. The respective solver was applied to every equation (*i.e.* momentum, pressure, turbulence, volume-fraction and cavitation), and their average taken for each case.

It was necessary to use pre-conditioning on the GMRES solvers in order to realize their true potential. Unfortunately these solvers then base their convergence on the pre-conditioned residual $\mathbf{P^{-1}(Ax - b)}$ which is not proportional to the true residual. If the pre-conditioner is not smooth, and the matrix is close to singular (as in the pressure equation) a 10% relative convergence tolerance in the preconditioned residual may be equivalent to a 50% or higher convergence tolerance in the true residual. To test fairly, the pre-conditioner was limited to a simple diagonal scaling (Jacobi) and the equivalent residual was computed in the Chaotic solver. This is not the best-case scenario for the KSP solver, as pre-conditioning can drastically change their performance. However, there is some overhead in the Chaotic solver in computing the diagonally-scaled residual, so it seems the best compromise. Regardless, the effects of pre-conditioning should not have a huge impact on the scalability results, especially given that Jacobi pre-conditioning is highly scalable compared to more complex pre-conditioners.

Figure 4 shows the performance results for each solver. Firstly, the tabular results show the absolute wall-time comparison on 32-cores and 256-cores. The Chaotic solvers are expectedly slower on all reasonably-sized test cases and up to 6.88 times slower than GMRES in the largest test case. The differences between GMRES and Chaotic solvers in terms of absolute time mostly depend on the convergence tolerance (tolerances of 40% and 20% were partially tested – not shown), however, scalability is mostly unchanged. P-GMRES performs approximately 50% slower where strong scalability is not a concern.

The scaling graphs require more careful analysis. There are three factors at play: (1) the cells-per-core ratio, which governs how much serial work and scalable ghost-cell exchange each process/thread performs; (2) the absolute number of processes, which governs how much time is spent performing reductions; and (3) cache effects. The combination of all three, and their relativity to each other, determines scalability.

The cells-per-core ratio is dominant in terms of scalability, since it changes most dramatically with the number of cores (doubling the number of cores halves the time due to factor (1)); whereas doubling the number of cores has little change on factor (2), as

the reduction time increases at a rate of $\log_2(cores)$. Cache effects (3) occur when the main parts of an algorithm fit into a high-speed memory buffer, located on the CPU. The cache is responsible for the super-linear scalability demonstrated by each solver. Its effect is more qualitative, as one could do extensive cache-optimization for each solver and each CPU to obtain the best results. No such optimization was performed for these solvers, and one can make the argument that the Chaotic solver more naïvely exploits the cache, leading to performance benefits at the intra-node level, where memory contention is a bottleneck. For the Chaotic solver, it is possible to see the point where the working memory fits into cache (≈20k-30k cells-per-core).

The scalability of GMRES and P-GMRES collapses when a certain cells-per-core ratio is reached - typically around 10k cells-per-core. Chaotic solvers continue to scale beyond this point. For example, with the KVLCC2-1.11m case on 512 cores, the scalability of the Chaotic solver is a factor of 601 compared to GMRES at 104. In absolute terms, the Chaotic solver is approximately twice as fast at ≈2.2k cells-per-core.

This scalability does not continue indefinitely, as demonstrated by the very small test cases. This is due to the expense of setting up the 8 OpenMP threads each time the solver is called, as well as a few other serial bottlenecks in the initialization, finalization and interface with PETSc's data structures; there is also some 'lag' involved with the residual computations which can cause over-convergence of very small equations. The reduction required to compute the residual also blocks the SpMV MPI communications, which is not ideal.

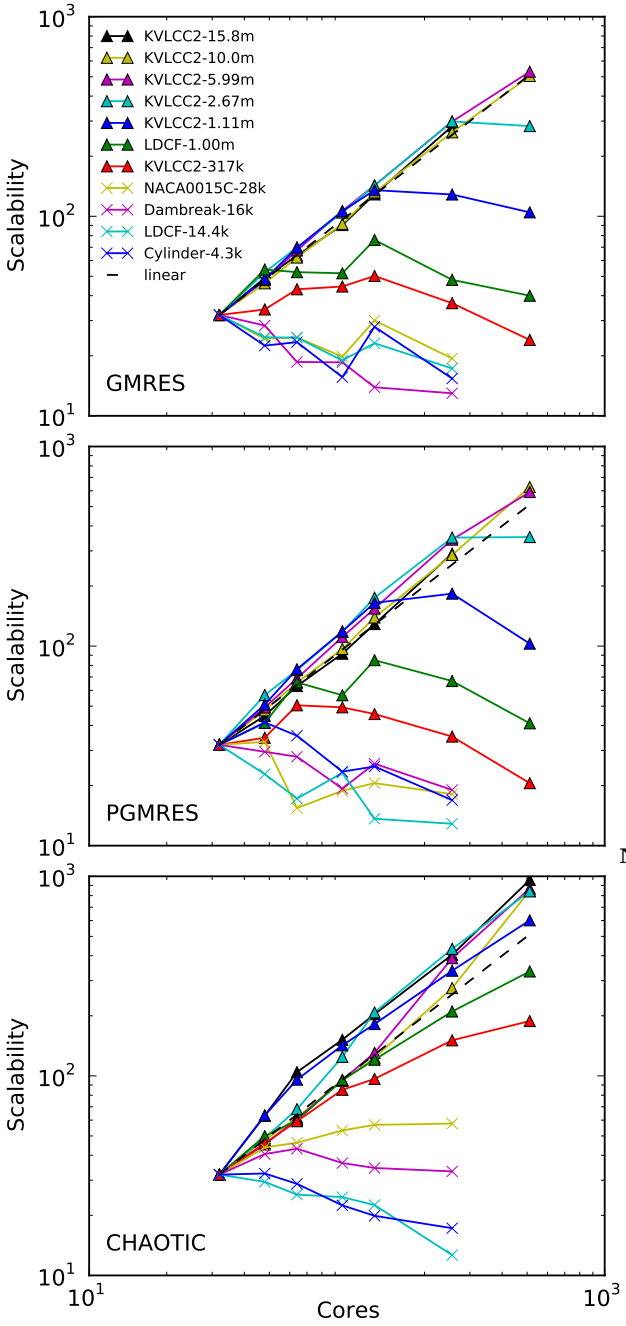
Some scalability is also lost due to the MPI communications themselves, which, although asynchronous with the computations, ultimately proceed in lock-step with each other. This means two MPI processes cannot benefit from faster communication between themselves.

The scalability of P-GMRES is lacklustre for this application. It is suspected that the very lax convergence tolerance prevents P-GMRES from showing its true benefit (since the number of total iterations is very small and more time is spent in setting up the solver), and very naïve pre-conditioning is used. It is also known that the scalability can be improved by increasing the l-factor of the solver, and this could be further investigated [5].

It is important to note that the results show the average of all the equation systems, although the pressure equation dominates the others in terms of total wall-time. Splitting the analysis between each equation individually is important, but the relative comparison between solvers is virtually unchanged in these cases.

6 CONCLUSION

The results show that the Chaotic solver provides a correct solution and that the scalability is better than a typical Krylov solver. The comparison of absolute wall-time is subjective - utilizing such high convergence tolerance of the linear equation system may not be viable, depending on the non-linear iterative process, the particular test case, and the efficiency of the non-linear iteration. Furthermore, more effective pre-conditioning



	32-cores		256-cores	
KVLCC2-15.8m	**Time (s)**	**Factor**	**Time**	**Factor**
GMRES	2.72E-01	-	3.08E-02	-
P-GMRES	4.06E-01	1.49	4.47E-02	1.45
Chaotic	1.87E+00	6.88	1.48E-01	4.81
KVLCC2-10.0m	**Time (s)**	**Factor**	**Time**	**Factor**
GMRES	1.48E-01	-	1.79E-02	-
P-GMRES	2.37E-01	1.61	2.65E-02	1.48
Chaotic	6.07E-01	4.11	7.05E-02	3.94
KVLCC2-5.99m	**Time (s)**	**Factor**	**Time**	**Factor**
GMRES	8.04E-02	-	8.63E-03	-
P-GMRES	1.23E-01	1.53	1.15E-02	1.34
Chaotic	2.83E-01	3.52	2.33E-02	2.70
KVLCC2-2.67m	**Time (s)**	**Factor**	**Time**	**Factor**
GMRES	2.98E-02	-	3.19E-03	-
P-GMRES	4.69E-02	1.58	4.29E-03	1.35
Chaotic	9.00E-02	3.02	6.65E-03	2.09
KVLCC2-1.11m	**Time (s)**	**Factor**	**Time**	**Factor**
GMRES	8.39E-03	-	2.09E-03	-
P-GMRES	1.20E-02	1.43	2.09E-03	1.00
Chaotic	2.03E-02	2.42	1.93E-03	0.93
LDCF–m	**Time (s)**	**Factor**	**Time**	**Factor**
GMRES	3.95E-03	-	2.63E-03	-
P-GMRES	5.57E-03	1.41	2.67E-03	1.01
Chaotic	1.05E-02	2.65	1.59E-03	0.61
KVLCC2-317k	**Time (s)**	**Factor**	**Time**	**Factor**
GMRES	1.57E-03	-	1.37E-03	-
P-GMRES	1.91E-03	1.21	1.73E-03	1.27
Chaotic	2.08E-03	1.32	4.41E-04	0.32
NACA0015C-28k	**Time (s)**	**Factor**	**Time**	**Factor**
GMRES	9.68E-04	-	1.60E-03	-
P-GMRES	8.30E-04	0.86	1.47E-03	0.92
Chaotic	3.22E-04	0.33	1.79E-04	0.11
Dambreak-16k	**Time (s)**	**Factor**	**Time**	**Factor**
GMRES	6.24E-04	-	1.54E-03	-
P-GMRES	8.86E-04	1.42	1.49E-03	0.97
Chaotic	3.13E-04	0.50	3.01E-04	0.20
LDCF-14.4k	**Time (s)**	**Factor**	**Time**	**Factor**
GMRES	7.69E-04	-	1.42E-03	-
P-GMRES	7.48E-04	0.97	1.86E-03	1.31
Chaotic	1.99E-04	0.26	5.03E-04	0.35
Cylinder-4.3k	**Time (s)**	**Factor**	**Time**	**Factor**
GMRES	8.06E-04	-	1.68E-03	-
P-GMRES	8.78E-04	1.09	1.66E-03	0.99
Chaotic	1.96E-04	0.24	3.64E-04	0.22

Figure 4: Performance results for the three solvers (GMRES, PGMRES and Chaotic). Scalability graphs show speed-up relative to 32 cores. The tabular results show absolute wall-time and how this benchmarks against GMRES (lower factor implies faster than GMRES). Results are based on average solve times for all equations (momentum, pressure and additional transport equations over 50 non-linear iterations).

of the KSP methods may make them more favourable – assuming scalability can be maintained.

For other applications, where the convergence of the linear system is important, Chaotic solvers could be used in conjunction with more complex methods – either as a pre-conditioner or a hybrid solver. It is very simple to detect when the Chaotic methods begin to stagnate (corresponding to the point where high-frequency errors have been smoothed); and the solver could dynamically switch at this point. For CFD, this could also be viable, and further analysis will be performed along with improvements to the Chaotic solver itself.

The performance of the Chaotic solver could be improved further by modifying the stopping criteria evaluation, in order to prevent over-convergence of very small cases, and also to remove the blocking reduction used to compute the residual. One concept is to attach the local component of the residual to the SpMV communication, and have a delayed residual computed for free as these messages are passed through all the processes. Another option is to use non-blocking collectives from the MPI-3.0 standard to allow other communications to continue (however, early attempts at this were slow). Perhaps the best solution would be a hybrid, whereby the former option is used, but an early-stop routine could be built which forces an immediate residual-check when the local portion of the solution is well below the required convergence.

Ensuring the OpenMP threads stay 'alive' between the individual calls to the solver should be simple, and will reduce serial bottlenecks further.

Improving MPI communication routines for higher asynchronicity is likely to be a good area for improvement. The best way to achieve this would be through one-sided communication and hardware-level RDMA (Remote Direct Memory Access). This would be particularly hard to implement, especially without invoking high initialization costs - and requires significant changes to PETSc and ReFRESCO.

The implementation of the Chaotic solver, using a hybrid MPI + OpenMP parallelization scheme, is well-suited to acceleration using Graphics Processings Units (GPUs) or co-processors, and is also an area for investigation.

REFERENCES

[1] 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`.

[2] 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.

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

[4] Davis, T. A. & Hu, Y. (2011). The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25. ISSN 0098-3500. doi:10.1145/2049662.2049663.

[5] Ghysels, P., Ashby, T. J., Meerbergen, K. & Vanroose, W. (2013). Hiding global communication latency in the GMRES algorithm on massively parallel machines. *Journal of Scientific Computing*,

35(1):48–71.

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

[7] Hawkes, J., Turnock, S. R., Cox, S. J., Phillips, A. B. & Vaz, G. (2014, September). Potential of Chaotic Iterative Solvers for CFD. The 17th Numerical Towing Tank Symposium (NuTTS 14), Marstrand, Sweden.

[8] 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.

[9] 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.

[10] 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.

[11] 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*.

[12] 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.

[13] 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.

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

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

[16] 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.