**Noname manuscript No.**
(will be inserted by the editor)

# On the Strong Scalability of Maritime CFD

**J. Hawkes · G. Vaz · A.B. Phillips · S.J. Cox · S.R. Turnock**

Received: date / Accepted: date

**Abstract** Since 2004, supercomputer growth has been constrained by energy efficiency rather than raw hardware speeds. To maintain exponential growth of overall computing power, a massive growth in parallelization is under way. To keep up with these changes, computational fluid dynamics (CFD) must improve its strong scalability – its ability to handle lower cells-per-core ratios and achieve finer-grain parallelization. A maritime-focused, unstructured, finite-volume code (ReFRESCO) is used to investigate the scalability problems for incompressible, viscous CFD using two classical test-cases. Existing research suggests that the linear equation-system solver is the main bottleneck to incompressible codes, due to the stiff Poisson pressure equation. Here, these results are expanded by analysing the reasons for this poor scalability. In particular, a number of alternative linear solvers and preconditioners are tested to determine if the scalability problem can be circumvented, including GMRES, Pipelined-GMRES, Flexible-GMRES and BCGS. Conventional block-wise preconditioners are tested, along with multi-grid preconditioners and smoothers in various configurations. Memory-bandwidth constraints and global communication patterns are found to be the main bottleneck, and no state-of-the-art solution techniques which solve the strong-scalability problem satisfactorily could be found. There is significant incentive for more research and development in this area.

**Keywords** High-Performance Computing, Strong Scalability, Software Profiling, Linear Solvers

J. Hawkes · S.J.Cox and S.R. Turnock
University of Southampton
Boldrewood Campus
Southampton
United Kingdom
E-mail: j.hawkes@soton.ac.uk

J. Hawkes · G.Vaz
Maritiem Research Instituut Nederlands (MARIN)
Wageningen
Netherlands

A.B. Phillips
National Oceanography Centre
Southampton
United Kingdom

## 1 Introduction

A recent report by Slotnick et al [28] attempted to create a "vision" of CFD in 2030, identifying some of the areas which must be improved to allow more widespread and successful use of CFD. Among these were improved turbulence and separation modelling; better automatic mesh generation and adaptivity; more capable multi-disciplinary simulations (for example, coupled CFD and structural simulations); improved post-processing, particularly of large simulations; greater accuracy through higher-order methods; and more practical design optimization. All of these goals require improvements to the underlying CFD algorithms – making them more efficient and more scalable – particularly considering the major changes in supercomputer architecture expected in the same era. Indeed, more scalable numerical methods are one of the areas highlighted by Slotnick et al [28], stating that development has been stagnant for too long. The particular numerical methods that require improvement are not clear, and depend upon the type of CFD code and application.

Here, the strong scalability of CFD is reviewed in detail, using ReFRESCO. ReFRESCO is an incompressible-flow, unstructured, finite-volume, SIMPLE-based, segregated solver specialized for maritime applications; similar in formulation to many open-source and commercial codes. A general scalability study of the whole code has been performed (section 4), which shows that the linear equation-system solver is the bottleneck in incompressible flow simu-

lations. This study details the reason for their poor scalability, and shows that there are new problems facing scalable CFD since earlier literature [12], due to increasing memory bandwidth issues. Two test cases are used: lid-driven cavity flow (LDCF) and the KRISO Very Large Crude Carrier (KVLCC2) [20], each with around 2.67-million cells – chosen to show the full range of intra-nodal and inter-nodal scalability bottlenecks on the University of Southampton supercomputer (Iridis4).

Following this, a variety of other linear solvers and preconditioners are tested to determine whether the scalability problems can be circumvented (section 5) and provide a comprehensive overview of the current 'best' solvers for strong scalability. These studies will aid CFD practioners in choosing suitable solvers and guide developers to find more scalable solutions. Widely-used solvers such as GMRES, Flexible-GMRES, BCGS and SOR are tested, along with state-of-the-art solvers such as Pipelined GMRES [11] which is designed to improve scalability. Similarly, multigrid preconditioners such as Sandia's ML [10] could bring a significant improvement when compared to block-wise preconditioners (such as Block Jacobi) or simple smoothers (such as SOR).

## 2 The Strong Scalability Problem

Over the last few decades, growth in supercomputing power has been exponential, with floating-point-operation (FLOP) rates doubling approximately every 14 months [29]. Whilst this growth is relatively constant, the underlying architectures which achieve such growth are not. Until 2004, the speed and electricity consumption of transistors was governed by Dennard's Scaling: as transistors shrank in size, their speed increased linearly and their electricity consumption dropped quadratically [7]. Unfortunately, the transistors used in modern processors are so small that electrons are able to 'leak' across the dielectric gates, and voltages must be increased to maintain stability. This limitation, known as the 'power wall', makes it more efficient for manufacturer's to provide multiple, slower cores in place of fewer, faster cores. Figure 1 shows the exponential growth rate of computing power and figure 2 shows how this computing power is provided – in terms of FLOP-rate-per-core and number of cores. The critical point in 2004 is clearly visible, where the shift towards a multi-core ('Chip-Level Multiprocessing') architecture occurred. This trend is relentless, and has lead to supercomputers with almost 200 cores-per-node. By 2020, it is expected that the cores-per-node ratio could be as high as 10-thousand; with the fastest supercomputers containing a total of 10- to 100-billion cores. Meanwhile, memory capacity is growing at an ever-slower rate, limiting the absolute size of simulations; and memory bandwidth (per core) is decreasing, creating new issues for CFD algorithms [27, 19, 17].

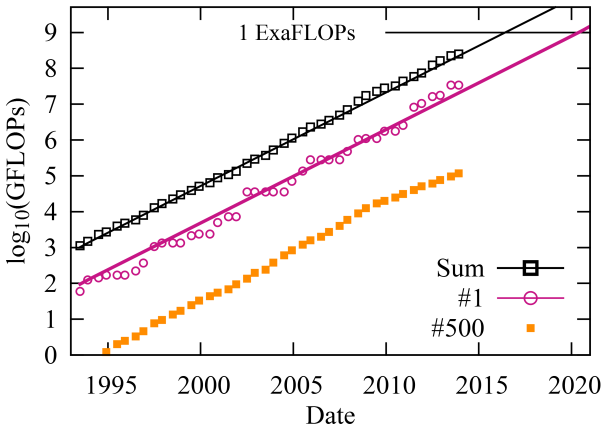


**Fig. 1** Total floating-point operation (FLOP) rate of the most powerful 500 supercomputers in the world, using data from the Top500 organization [29]. The data shows the benchmark FLOP rate of the #1 machine, the #500 machine and the sum of all 500 machines over time. The exponential growth rate corresponds to a doubling every 13.85 months.

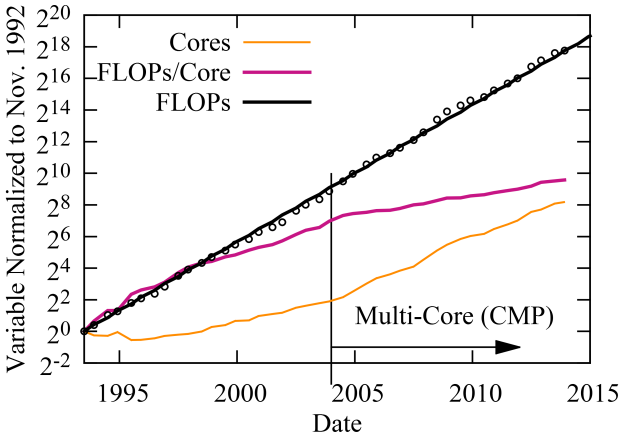


**Fig. 2** The FLOP rate, FLOP/core ratio and number of cores (average of the Top500 [29]) over time, normalized to a snapshot in November 1993. Since the advent of chip-level-multiprocessing (CMP) in 2004, FLOP/core ratio has grown by only a factor of approx. 4, whereas number of cores has grown by a factor of approx. 64.

It is often regarded that the CFD algorithm benefits from good 'weak scalability' (the ability to maintain computational efficiency with a fixed cells-per-core ratio), thus realizing the benefits of supercomputing advances when the growth in core-count was moderate [3]. Although difficult to quantify, it can be assumed that the capabilities of CFD have grown more-or-less in proportion with supercomputing power because of this trait; especially given that the maximum problem size (limited by total memory) has more-or-less grown with the number of nodes.

The 'strong scalability' of CFD – the ability to decrease the cells-per-core ratio efficiently – is more important in a massively-parallel era; and is generally poor [6, 3]. By 2020, supercomputers are expected to contain approximately 3000-times more cores, but the size of CFD simulations can only increase by a factor of ≈11, thus the efficient cells-per-core ratio of CFD must drop by a factor of at least 250[1]. Furthermore, in the maritime industry the majority of state-of-the-art simulations are unsteady computations. Since it is difficult to parallelize the time domain, greater spatial domain-decomposition is required to improve CFD capabilities, requiring further improvements to strong scalability.

Whilst both forms of scalability have been investigated for incompressible CFD, the results are often subjective to the particular code and hardware, and difficult to generalize. In Bhushan et al [3] the linear-equation system solver, particularly for the Poisson pressure equation, is the main bottleneck to scalability. Culpo [6] reinforces this by considering the main elements of the linear-equation system solver and how they could be improved. However, neither paper investigates the reason for their poor scalability in detail, and neither considers alternative solver algorithms. Gropp et al [12] provides a good grounding in both these areas, but is now 15 years old – and the conclusions drawn could be somewhat outdated due to changes in hardware.

## 3 Experimental Setup Using ReFRESCO

In order to conduct scalability experiments, a sample CFD code (ReFRESCO) is used on two classical test cases. In order to get information on the run-time of ReFRESCO, the code is profiled by injecting timers into the code in key places. The test cases are run across a range of core-counts, from 1 to 512, on the University of Southampton supercomputer: Iridis4. The various aspects of the experimental setup are discussed below.

### 3.1 ReFRESCO

ReFRESCO[2] is a viscous-flow CFD code that solves multiphase, unsteady, incompressible flows for unstructured meshes [30]. It is complemented by various turbulence, cavitation and volume-fraction models. In many ways, 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 it has been verified, validated and optimized for numerous maritime industry problems. ReFRESCO is currently being developed at MARIN (Netherlands) and a number of universities around the world [8, 24, 26, 18, 2] including the University of Southampton [13, 14, 15].

In ReFRESCO, the governing equations are discretized in strong-conservation form using a finite-volume approach with cell-centred collocated variables. Simulations are parallelized with MPI (Message Passing Interface) and partitioned using METIS [9]. ReFRESCO is based on the SIMPLE (Semi-Implicit Method for Pressure-Linked Equations) solver with pressure-weighted interpolation (PWI) [18].

The SIMPLE algorithm is shown in figure 3. The coarsest loop in the SIMPLE algorithm is responsible for unsteady time-stepping. Time integration is performed implicitly with first- or second-order backward schemes. All nonlinearity is tackled in the 'outer loop', which is performed several times per time-step (until satisfactory convergence).

In each outer loop, a Picard-linearized version of each transport equation is *assembled* into a system of linear equations, based on the discretization of all the equation terms (time-derivatives, convection, diffusion, source terms) in the associated mesh. This process creates a sparse matrix of implicit terms ($\mathbf{A}$) and a vector of explicit terms ($\mathbf{b}$); which can be solved to find new values for the flow field ($\mathbf{x}$): $\mathbf{Ax} = \mathbf{b}$. *Iterative solvers* are used to solve for $\mathbf{x}$ to a given convergence tolerance (based on the $\ell^2$-norm of the residual) in an 'inner loop', with typical tolerances between 0.1 and 0.001. ReFRESCO uses PETSc (Portable Extensible Toolkit for Scientific Computing)[1] for its large range of linear solvers and preconditioners.

Since each MPI process has its own memory space and its own portion of the mesh, the updated values for $\mathbf{x}$ along partition boundaries must be shared via MPI *data exchange*. The *gradient* of the flow-field variable can then be computed using Gauss theorem, and the updated gradients exchanged across domain boundaries once again.

These four routines {assembly, solve, exchange and gradients}, which can be seen in figure 3, are the heart of the SIMPLE algorithm and are the most expensive part of the solution process. They are also the linchpin of other SIMPLE-derived algorithms and common alternatives such as SIMPLEC, SIMPLER and PISO.

In order to observe how these key routines scale, ReFRESCO has been profiled using Score-P [VI-HPS Acc. 2013]. Score-P is a compile-time wrapper which automatically logs various run-time events, such as function calls and durations. It has been carefully filtered such that only critical functions are measured, and the effect on total run-time is less than 2%. Score-P can be linked to PAPI (Performance

[1] Based on Tianhe-2, the current (January 2016) #1 supercomputer and conservative estimates of an exascale machine expected in 2020. Maximum capabilities of CFD are based on trends of total computational power, doubling every 13.85 months. Realistically, maximum CFD capabilities are limited by memory capacity which grows even more slowly [19], thus amplifying the problem.

[2] See www.refresco.org.

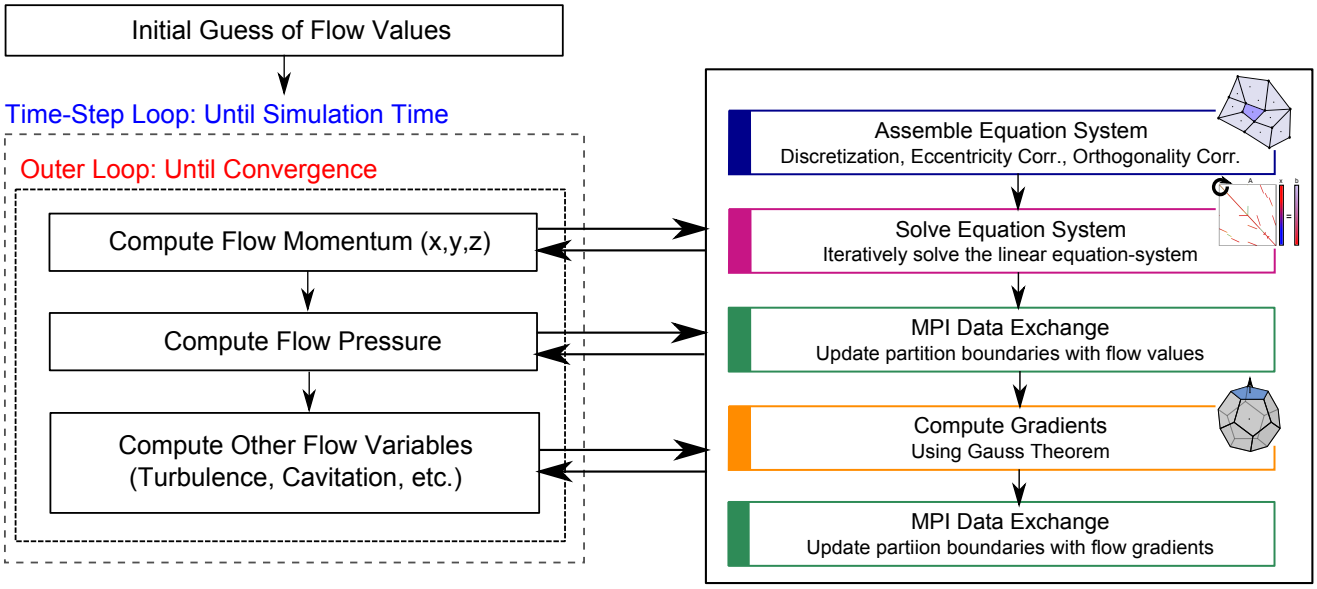


**Fig. 3** An overview of the SIMPLE algorithm. Time discretization is handled by the time-step loop at the coarsest level. Within each time-step a number of outer loops are performed in order to solve non-linearity and couple the governing equations. Within each outer loop, the solutions to the governing equations (momentum, pressure, etc.) are computed in their uncoupled, linearized, discretized form. The solution of each equation follows the same five steps as illustrated on the right.

Application Programming Interface)[4] which gathers additional information from physical hardware-counters.

Two hardware counters are enabled for these tests. The first measures floating point operations per second, which helps to determine whether the processing units are saturated. For an unstructured CFD code this is an unlikely situation, as the indirect memory access bottlenecks the processor. Structured-mesh codes are more likely to reach these limits due to better memory layout and vectorization.

Hardware counters for pure memory bandwidth are not available, but level one (L1) cache misses give a good indication of memory-fetching issues (since a cache miss must result in a memory or next-level cache transaction). However, there are two issues. Firstly, the compiler will often prefetch memory into the cache, resulting in memory bandwidth usage which is not detected by this hardware counter. Secondly, an L1 cache-miss will be registered even when the data resides in L2 or shared cache (which is still very fast compared to off-chip memory). Despite these imperfections, it is possible to see certain bottlenecks due to memory bandwidth, particularly when combined with other hardware counters or profiling.

Other hardware counters are available (such as L2 cache misses), but they cannot be enabled at the same time due to register sizes or competing circuitry.

### 3.2 Iridis4

ReFRESCO is run on the University of Southampton's latest supercomputer. Iridis4 has 750 nodes, consisting of two Intel Xeon E5-2670 Sandybridge processors (8 cores, 2.6 Ghz), for a total of 12,200 cores. Each 16-core node is diskless, but is connected to a parallel file system, and has 64GB of memory. The nodes run Red Hat Enterprise Linux version 6.3. Nodes are grouped into sets of 30, which communicate via 14 Gbit/s Infiniband. Each of these groups is connected to a leaf switch, and inter-switch communication is then via four 10 Gbit/s Infiniband connections to each of the core switches. Management functions are controlled via an ethernet network.

Iridis4 ranked #179 on the Top500 list of November 2013 with a peak performance of 227 TFLOPS [29]. Iridis4 cannot be classified as a next-generation, many-core machine, with only 16 cores per node. Indeed, it is several years behind the state-of-the-art. Nonetheless, it should be able to give sensible insight into the limitations of the CFD algorithm.

### 3.3 LDCF & KVLCC2

Two test cases are used in the experiments. The first is a laminar-flow, canonical, unit-length, three-dimensional lid-driven cavity flow (LDCF). A uniform structured mesh of 2.68-million cells ($139^3$) is used, and only momentum and pressure equations are solved. The simulation mimics an infinite domain, with two cyclic boundary conditions. The remaining four boundaries are constrained with Dirichlet boundary conditions, one of which specifies a tangential, non-dimensional velocity of 1.

The second test case is the KRISO Very Large Crude Carrier (KVLCC2) double-body wind-tunnel model [20]. The mesh is a three-dimensional multi-block structured mesh consisting of 2.67-million cells. A $k$-$\omega$, two-equation shear stress transport turbulence model is used [22]. The domain and mesh are shown in figures 4 and 5 respectively.

The two test-cases are designed to have a similar number of cells. The size of the mesh has been chosen to show the full range of scalability issues. On 512 cores, the cells-per-core ratio reaches approximiately 5200 which helps demonstrate the parallel bottlenecks on a large number of cores; yet the problem is substantial enough to show memory bandwidth issues on a single node (with approximately 167k cells-per-core).

In both cases, 400 outer-loops are performed with no time-stepping, with an inner-loop (linear) relative convergence tolerance of 0.1 (0.01 and 0.001 later). Relaxation is applied to all outer-loops to stabilize the non-linear iterative process. QUICK (Quadratic Upstream Interpolation for Convective Kinematics) and first-order upwind schemes are used to discretize the convective terms of the momentum and turbulence equations respectively. A GMRES (Generalized Minimal Residual method) solver is used with a Block Jacobi preconditioner, as in Gropp et al [12].

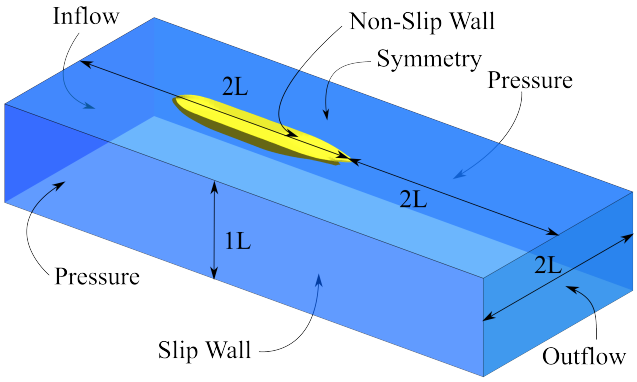


**Fig. 4** The domain used for the KVLCC2 double-body wind-tunnel simulation. Symmetry boundary conditions are applied at the waterplane, but port- and starboard-sides of the centre-line are both simulated.

## 4 General Scalability Study

The scalability of the SIMPLE algorithm has been examined by measuring the run-time of the two test cases as successively more cores are added to the simulation. A scalability factor $S$ can be defined as $S = T_1/T_C$ where $T_C$ is the wall-time using $C$ cores. Ideal scalability is when $S = C$, although this is rarely achieved. This scalability factor can be found for various subroutines in the SIMPLE algorithm in order to identify potential bottlenecks. Here, the scalability

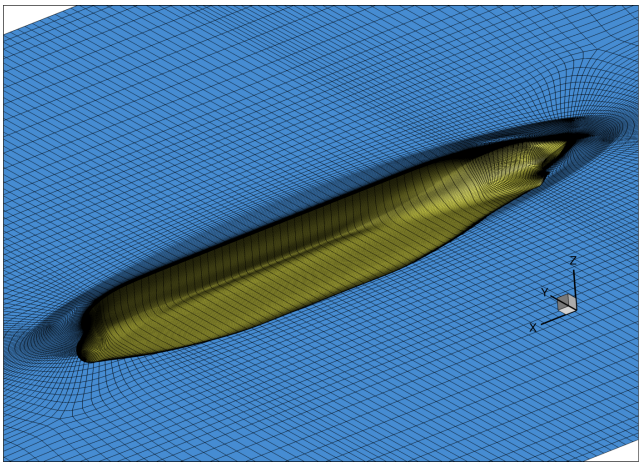

**Fig. 5** The KVLCC2 mesh is a multi-block structured mesh consisting of 2.67-million cells.

factor is normalized to serial operation ($C = 1$), and is often called a parallel 'speed-up' factor for this reason.

Figure 6.a and 6.b show the scalability of the code as the number of cores increases. The embedded bar charts show absolute core-hours ($C \times T_C$, which would ideally remain constant) in serial operation ($C = 1$), single-node operation ($C = 16$) and highly-parallel operation ($C = 512$). Total core-hours is shown with black bars; the composite parts are coloured and keyed with respect to the enclosing scalability plot. This shows the denormalized costs of various routines; and also highlights where the scalability bottlenecks occur – at the intra-nodal or inter-nodal level. The results from the hardware counters are shown in figure 7, for the LDCF test case.

Total scalability is poor overall, suffering significantly on just 16 cores due to intra-nodal bottlenecks and worsening to the point at which almost no speed-up is gained from adding additional nodes. On 512 cores the parallel speed-up is just 128 (KVLCC2) and 100 (LDCF). This scalability is similar to other codes, such as OpenFOAM [25, 6], STAR-CCM+ [5] or Ansys Fluent [HP 2014] – although some published results are normalized to nodal performance (i.e. $C = 16$, which hides intra-nodal inefficiency and gives overly-optimistic results) or are truncated (hiding inter-nodal bottlenecks). Exact comparisons between various codes on identical hardware were not feasible, but only minor differences are expected since all of these packages are based off the same algorithm and share similar implementations. Major differences should only be observed if coupled solvers or inferior partitioning schemes are used.

The routines outlined earlier {assembly, solve, exchange and gradients}, for each equation in each outer loop, account for the large majority of overall run-time. The remaining time (other) is spent (mostly) in one-off functions such as file IO or MPI initialization, thus is subjective to the length of the simulation and amount of IO required. These other rou-

tines may also increase significantly if additional features such as moving, deforming and adapative grids are used – again, this is highly subjective.

The assembly and gradients routines scale favourably, reaching almost 90% parallel efficiency. The high, and increasing, cache-miss rate of the gradients routines is curious, as it does not seem to affect performance (FLOP rate is maintained). It is likely that the data required resides in L2 or shared-cache, rather than off-chip memory, so the impact of these L1 cache misses is much lower.

The data exchange routines are not visible in the scalability plots, because normalizing against ($T_1 \approx 0$) gives negative scalability (no communications are required in serial operation). In reality, these routines scale reasonably – as the number of cores increases the size of the messages become smaller, and these messages can be sent concurrently. With inadequate load-balancing these data exchanges can become costly, due to the implicit synchronization of MPI processes. However, the results show that these communications account for a small proportion of overall run-time.

As consistent with literature, the solve routines have poor scaling and are a major contributor to total run-time, thus are the main concern for scalability. The hardware performance counters show a high cache-miss rate between 16 and 128 cores, corresponding to saturated memory bandwidth at the intra-nodal level.

Memory-bandwidth-per-node has been growing at approximately half the rate of processing-power-per-node leading to today's problems with memory bandwidth [27]. In Gropp et al [12], conducted in 2000 on single-core processors, memory-bandwidth problems were apparent but not as concerning – changes in architecture over the last decade have made memory bandwidth issues more critical.

Beyond 128 cores, cache misses in the solve routines become less frequent but FLOP rate continues to decrease and scalability worsens. This is due to the oft-observed global communication bottleneck. An illustration of a single GMRES iteration is shown in figure 8. This pattern is computationally similar to most Krylov Subspace (KSP) solvers, including conjugate gradient methods – although some differences will be mentioned in section 5. In particular, two distinct communication routines are required by KSP solvers.

Firstly, sparse-matrix-vector-multiplication (SpMV) requires concurrent neighbour-to-neighbour communication as in the data exchange routines – scaling reasonably well.

Secondly, two global reduction-broadcast routines are required for orthogonalization and normalization of the Krylov vectors. These require a hierarchical global communication pattern which scales poorly, usually with $T_C \overset{\propto}{\sim} \log_2(C)$ (where the proportionality factor depends primarily on network latency). These communication patterns are blocking, and cannot easily be overlapped or hidden by other useful work. On a large number of cores spread over an inter-nodal network with relatively high latency, these global communications become a bottleneck to scalability of the linear solvers. Whilst most routines reduce in wall-time as more cores are added (with less-than-ideal efficiency), wall-time for global communications *increases*, as each time the number of cores doubles, an extra set of messages must be sent (incurring the latency cost of the network).

These global communications create a scalability bottleneck when a high number of nodes are used, and has been well-documented in the literature [6]. The memory-bandwidth problems previously noted are often overlooked, but are an important bottleneck to overcome for next-generation supercomputing.

Figure 6.c and 6.d shows the breakdown of time spent in the various equations (pressure, momentum, turbulence). In both cases, the single pressure equation took considerable time to compute – similar to all three momentum equations combined. In incompressible-flow simulations the pressure equation is much harder to solve than other transport equations, due to its elliptic Poisson form.

This can be illustrated by considering the spectral radius (maximum eigenvalue) of the Jacobi iteration matrix: $\rho(\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U}))$, where $\mathbf{D}$, $\mathbf{L}$ and $\mathbf{U}$ are the diagonal, lower and upper triangles of $\mathbf{A}$ respectively. The rate of convergence of the Jacobi method is proportional to the spectral radius, and must be less than unity for convergence. Using a smaller version of the KVLCC2 mesh (317k cells), the spectral radius of the KVLCC2 pressure equation was >0.9999, compared to 0.8450 for the momentum equation and 0.5888 for the turbulence equation[3]. Thus it would take at least 1700-times more iterations of the pressure equation to reach the same convergence as in the momentum equation, if using a naïve Jacobi solver. The pressure equation also gets stiffer as the mesh gets larger, amplifying the problem. KSP methods, particularly with good preconditioning, close this gap considerably, but there is still a large difference between the pressure equation and other transport equations [14].

The above results were performed using an inner-loop (linear) relative convergence tolerance of 0.1 for all equations. The results were repeated using a tolerance of 0.01 and 0.001 (see figure 9). Note the significant increase in wall-time, entirely due to the solve routines for the pressure equation.

These results were also re-run using alternative convective discretization schemes [13], which had no significant effect on scalability. Higher order methods, such as QUICK, apply their high-order terms explicitly (into the $\mathbf{b}$ vector); with only the low-order terms affecting the matrix ($\mathbf{A}$). It is

[3] The spectral radii were found by extracting the matrices from the fifth outer loop in a separate batch of simulations. The maximum eigenvalue, and thus spectral radius, of the corresponding Jacobi iteration matrix could be found using ARPACK routines [21] based on Arnoldi iterative methods. The size of the matrix that could be tested was limited by memory capacity.

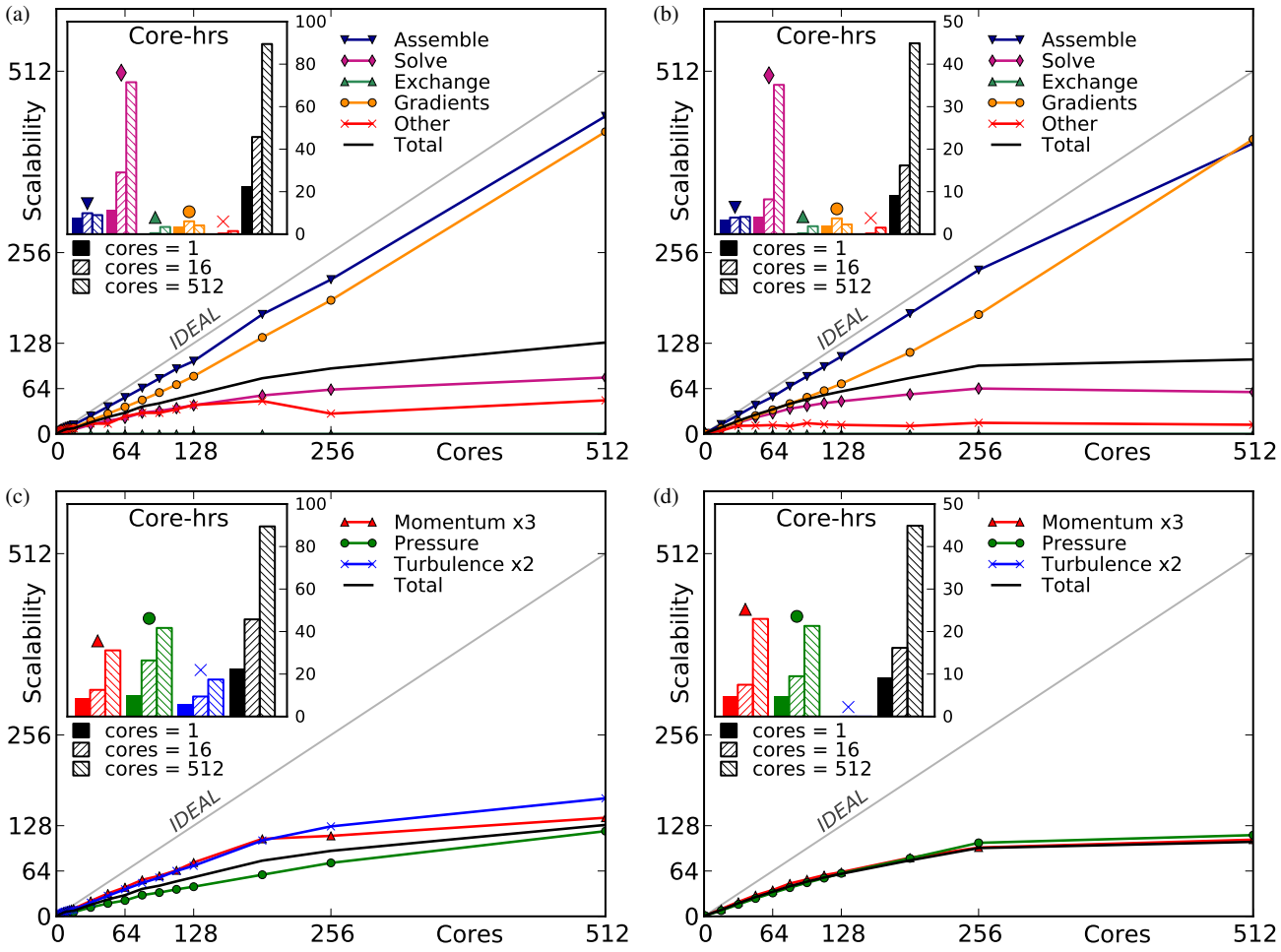


**Fig. 6** Scalability of the code, and the profiled routines within, as the number of cores increases. These results used GMRES as the linear solver with a Block Jacobi preconditioner, and an inner-loop relative convergence tolerance of 0.1. (a) KVLCC2 breakdown by routine, (b) LDCF breakdown by routine, (c) KVLCC2 breakdown by equation, (d) LDCF breakdown by equation.

suggested that higher-order discretization will be a prominent development in the next decade [28], so this is a promising result. However, there are difficulties when going to even higher-order methods, and this remains an area of active linear-solver development [23].

A fully unstructured mesh was also tested using the KVLCC2 test case [13]. The unstructured mesh was much larger (12.5m) and did not directly match a structured mesh, so the scalabiliy of two structured meshes was interpolated (10.0m and 15.8m). The interpolated scalability of the structured meshes was virtually identical to the unstructured mesh. Note that ReFRESCO treats all meshes as unstructured meshes, in terms of data structure – and therefore does not take advantage of structured meshes and structured memory layout.

This preliminary study concludes that in their basic form, the linear equation-system solvers are the primary bottleneck to strong scalability of CFD, in agreement with recent literature. In particular, detailed profiling of ReFRESCO has revealed that memory bandwidth contention and expensive hierarchical communication patterns are the main bottleneck. However, the scalability may be significantly altered if different solvers and preconditioners are used.

## 5 Effect of Linear Solvers & Preconditioners on Scalability

Thus far, the results have used a basic GMRES solver with a Block Jacobi preconditioner. More modern solvers or preconditioners could provide very different scalability characteristics. For example, a powerful preconditioner could reduce the number of KSP iterations (and global communications) required; but may be unscalable in itself due to communication or high setup costs. CFD is unique in that a solution to the linear system is only approximated, since the solution to the linear system is a small part of a nonlinear system. Compared to applications which require ma-

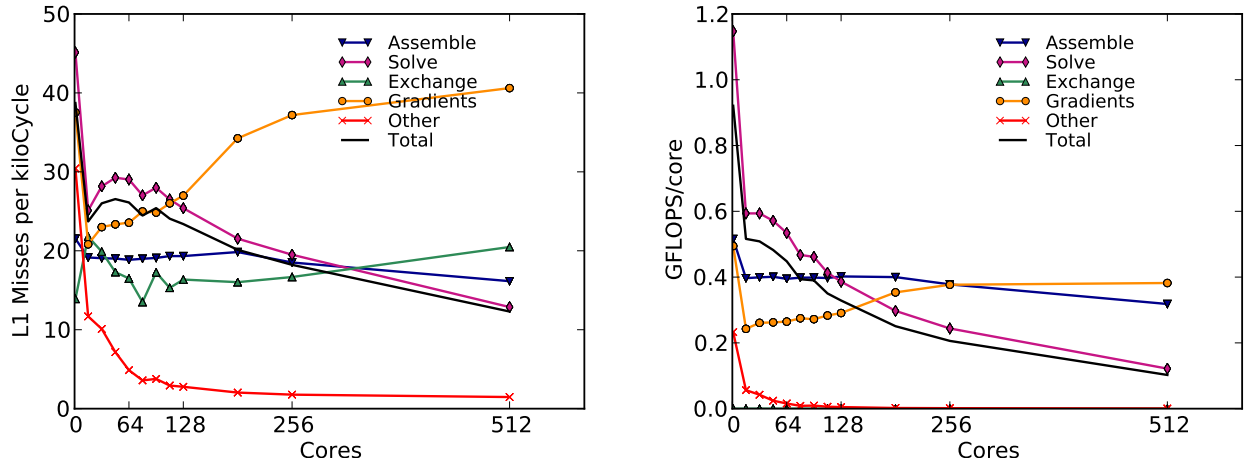


**Fig. 7** An example of the information gleaned from PAPI hardware counters. For all routines in the LDCF test case, the number of first-level (L1) cache-misses per thousand clock cycles [left] and the total floating-point operation rate (FLOPs) [right] are shown.

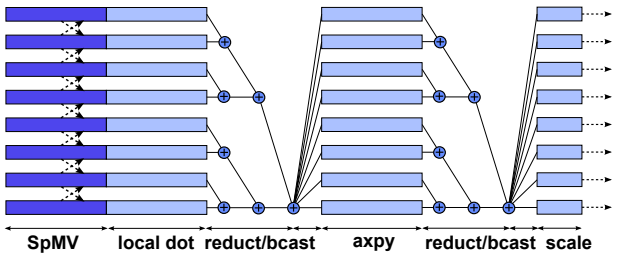


**Fig. 8** An illustrative trace of a GMRES iteration on 8 cores (not to scale). Note the local neighbour-to-neighbour communications performed asynchronously in the SpMV routine, and the two reduction-broadcast patterns. There are many variations of the reduction-broadcast algorithm which cannot be illustrated clearly. The most common is the 'butterfly' algorithm which completes in $\log_2(C)$-time, combining the reduction and broadcast into a single hierarchy of latency-bound messages. As the number of cores increases, this reduction-broadcast takes longer, whilst other routines take less time.

chine accuracy of linear systems, CFD is very sensitive to start-up (initialization of linear solvers, memory allocation, etc.) costs which may rule out the most advanced solvers or preconditioners. Indeed, it may be that simpler preconditioners than Block Jacobi provide better scaling. In this section, the scalability of the linear solvers will be tested. Following this, a number of preconditioning techniques will be investigated – including block preconditioning techniques, multigrid methods and simple smoothers.

### 5.1 Solvers

A recent improvement to GMRES has been developed, so-called Pipelined GMRES (PGMRES) [11], which removes one of the global communications from the standard GMRES iteration – replacing it with a correction routine, and allowing the remaining reduction to be overlapped with other useful work. The scalability of GMRES and PGMRES are compared in figure 10. Flexible GMRES (FGMRES) has been tested, as it allows a wider range of preconditioning techniques to be used later. A Bi-Conjugate Gradient Squared (BCGS) method has also been tested. All of the KSP methods use Block Jacobi as a preconditioner. A successive over-relaxation (SOR) method is also shown, demonstrating the differences between KSP and non-KSP methods.

GMRES performs as previously noted, with poor performance at the intra-nodal level due to memory-bandwidth contention and poor performance on a large number of cores due to global communication patterns. FGMRES exhibits slightly worse inter-nodal scalability than GMRES. PGMRES scales worse in the memory-bandwidth zone than either GMRES or FGMRES, but the gradient of the scalability factor, $dS/dC$, between 256 and 512 cores is approximately double that of GMRES – consistent with the algorithmic improvement (half the number of global communications). Overall, the wall-time gains from PGMRES on 512 cores are minimal.

BCGS gives strong numerical performance in serial operation and similar memory-limited scaling at the intra-nodal level. BCGS requires four global communications per iteration, thus inter-nodal scaling suffers.

As expected, SOR performs much worse than the KSP methods, but has far superior scalability. SOR is limited by memory bandwidth briefly, but quickly recovers this as the simulation fits into cache. The SOR algorithm still uses one global communication pattern to compute a residual at the end of each iteration; thus its final gradient is similar to that of PGMRES. Residuals could be calculated less frequently to improve scalability of SOR considerably.

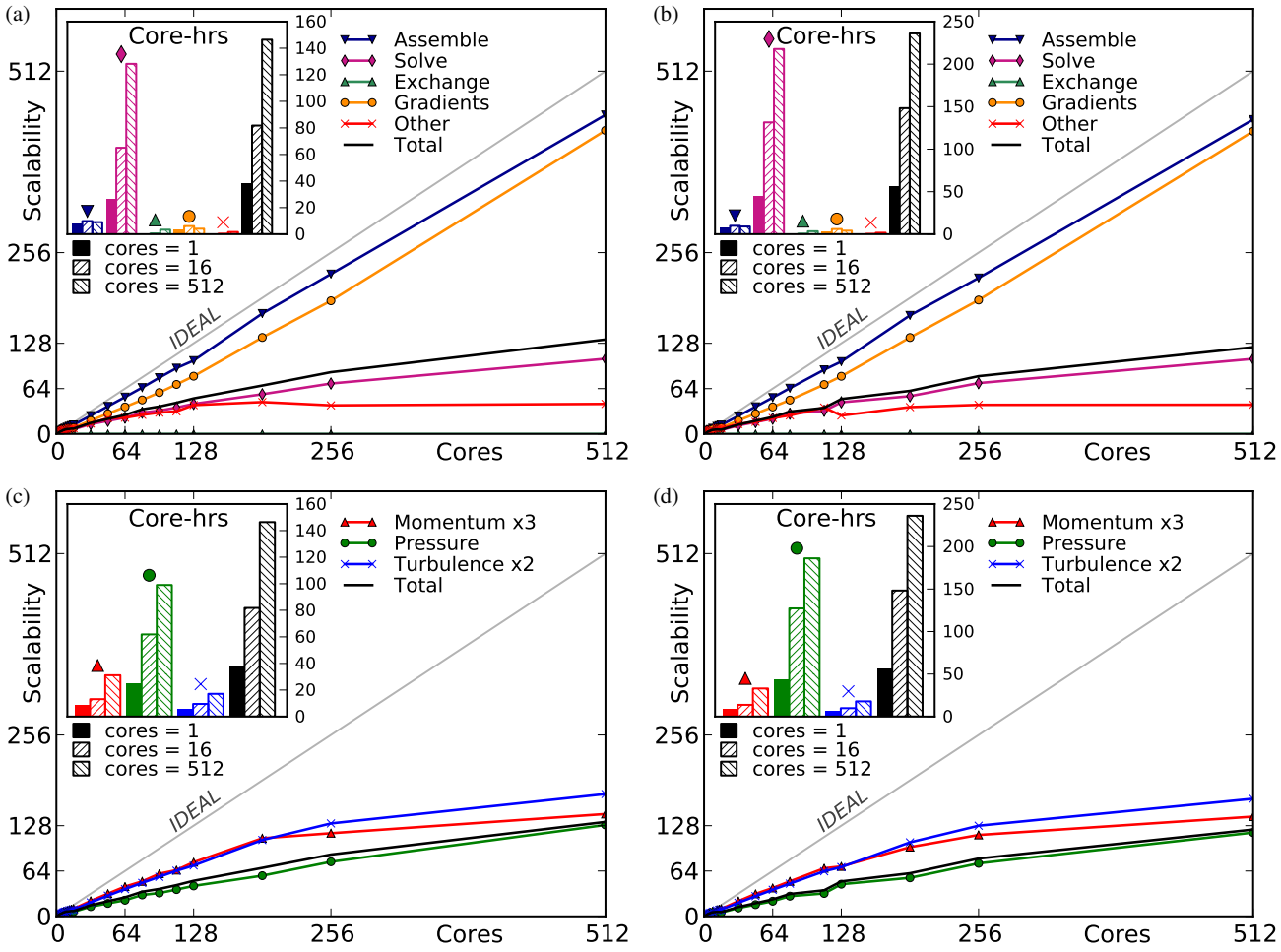


**Fig. 9** Scalability of the code with an inner-loop convergence tolerance of (a,c) 0.01 and (b,d) 0.001 for the KVLCC2 test case as in figure 6, showing scalability of (a,b) the various routines and (c,d) the various equations, using GMRES with a Block Jacobi preconditioner. The results show poor scaling of the solve routines, which particularly influences the cost of the pressure equation. Similar results were obtained for the LDCF test case.

ReFRESCO also has access to a large range of preconditioners through its use of PETSc, many of which have been tested as follows.

## 5.2 Block Preconditioners

The Block Jacobi algorithm used thus far implements a block-wise Incomplete LU (ILU) factorization with zero-level fill, and sets a high benchmark for other preconditioners. ILU(0) is performed on each MPI process's local portion of the matrix, leading to an interesting problem: convergence deteriorates as the number of cores increases, as the local portion becomes less significant to the global solution[4]. An Additive Schwarz Method (ASM) was tested, with the same block-wise ILU(0) solver. ASM is similar to Block Jacobi, but allows communication between neighbouring blocks to augment the process. The results are shown in figure 11. The differences between Block Jacobi and ASM were small, with ASM fairing worse overall due to additional communications. ILU(0), ILU(1) and ILU(2) were also tested as preconditioners in their own right, with poor results in all regards (not shown).

[4] Indeed, this made it difficult to distinguish between convergence-loss-problems and global-communication-problems in the previous section. Profiling of an unpreconditioned GMRES reveals that globals communications are the leading problem. However, the number of iterations required with Block Jacobi increased when a tolerance of 0.001 was requested.

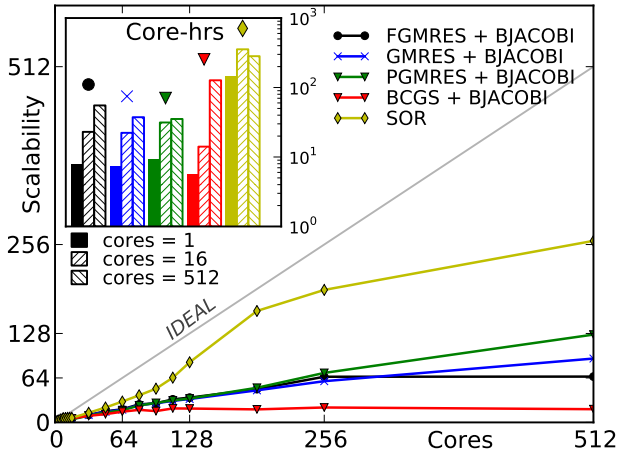


**Fig. 10** Scalability of the solve routines in the pressure equation, using the KVLCC2 test case with inner-loop convergence tolerance set to 0.1. The results compare three Krylov Subspace solvers and a Successive Over-Relaxation (SOR) algorithm operating as a Block Gauss-Seidel method. Note the exponential scale of the inset core-hours chart.

### 5.3 Multi-grid Preconditioners

For elliptic equations (such as the pressure equation) multi-grid methods such as ML [10] should be very powerful. Multi-grid methods cover a broad category, with multiple formulations and many opportunities for fine-tuning. They are all based on the principle that multiple scales of the problem can be solved efficiently by solving coarse-grid approximations to the actual (fine) grid. The coarsest grid will have a much lower spectral radius than the finest, allowing low-frequency errors to be reduced quickly. Meanwhile, the fine grid solves high-frequency errors, and the results are combined. Since each grid is much easier to solve, 'smoothers' are used instead of complete solvers at each level. A typical smoother may just be one iteration of SOR or an ILU factorization, for example, although the coarsest grid is often solved directly. There are many variations of multi-grid methods: different methods for coarse-grid construction; different methods for coarse-grid interpolation; various methods of communicating (or not) on coarse grids; and so on. All of these variations will have a large effect on scalability.

ML is a state-of-the-art smoothed-aggregation algebraic multi-grid method from Sandia's National Labaratories, and is one of the most commonly-used multigrid packages [10]. ML automatically creates coarse grids until a minimum size is reached using a smoothed aggregation process. For the KVLCC2 test case, ML automatically decided to create six grids when running on one core, and four grids on 512 cores. FGMRES was necessary to accommodate the multi-grid preconditioner, because the preconditioning matrix could change between iterations.

The results shown in figure 11 show that ML is highly capable at the intra-nodal level. It exhibits strong serial performance and moderate scaling to 16 cores – better than Block Jacobi. Unfortunately it rapidly breaks down beyond 64 cores where global communications dominate. It was suspected that this was due to the direct solver used on the coarsest grid, but replacing it with a smoother (5 iterations of SOR) resulted in worse scalability. Another recommendation is to restrict the number of levels created by ML, thus reducing expensive start-up costs. Restricting ML to three levels worsened the scalability; and two levels (not shown) gave similar results. It is expected that less sophisticated multigrid methods (such as non-smoothed aggregation) may provide better results for CFD, since start-up is cheaper. It is also possible to re-use the coarse-grid mappings between non-linear iterations, which would improve the results shown here. Furthermore, the multigrid method could be used as a standalone solver rather than a preconditioner, omitting FGMRES entirely. Clearly a much deeper study of multi-grid methods is required as they certainly cannot be used as a black-box for scalable CFD.

### 5.4 Smoothing as a Preconditioner

Finally, it is worth considering a much simpler preconditioner than even Block Jacobi. Instead of preconditioning in the classic sense, a smoother can be used before the main solver (FGMRES). Ten iterations of SOR as a smoother was optimal (compared to 1, 100 or 1000). Although it performed worse than Block Jacobi in serial operation, where Block Jacobi has good convergence, it was able to utilize the super-linear scalability as noted in section 4 due to caching of memory, thus providing excellent scalability. Since a fixed number of smoother iterations were performed, residual computation in the SOR algorithm was unnecessary, improving scalability further. This combination has minimal setup costs, since SOR requires no additional memory or pre-computation, so could be a viable option for scalable CFD. However, the solver is still unavoidably limited by memory bandwidth contention, and global reductions from the overruling KSP solver. Furthermore, the numerical performance is not good, so it is only competitive on a large number of cores.

This section has looked at various linear solver and preconditioners. The most promising result was from a SOR smoother instead of a classical preconditioner. Although it was the slowest configuration for serial computation, superior scaling meant that it was often the best performer at $C = 512$. A multi-grid preconditioner was tested, which performed well on a low number of cores, but had poor scaling. A more detailed study of multi-grid preconditioning may yield better results. A more scalable version of GMRES was also tested (Pipelined GMRES) with mixed results

– global communication problems were halved; but at the cost of more memory-bandwidth problems.

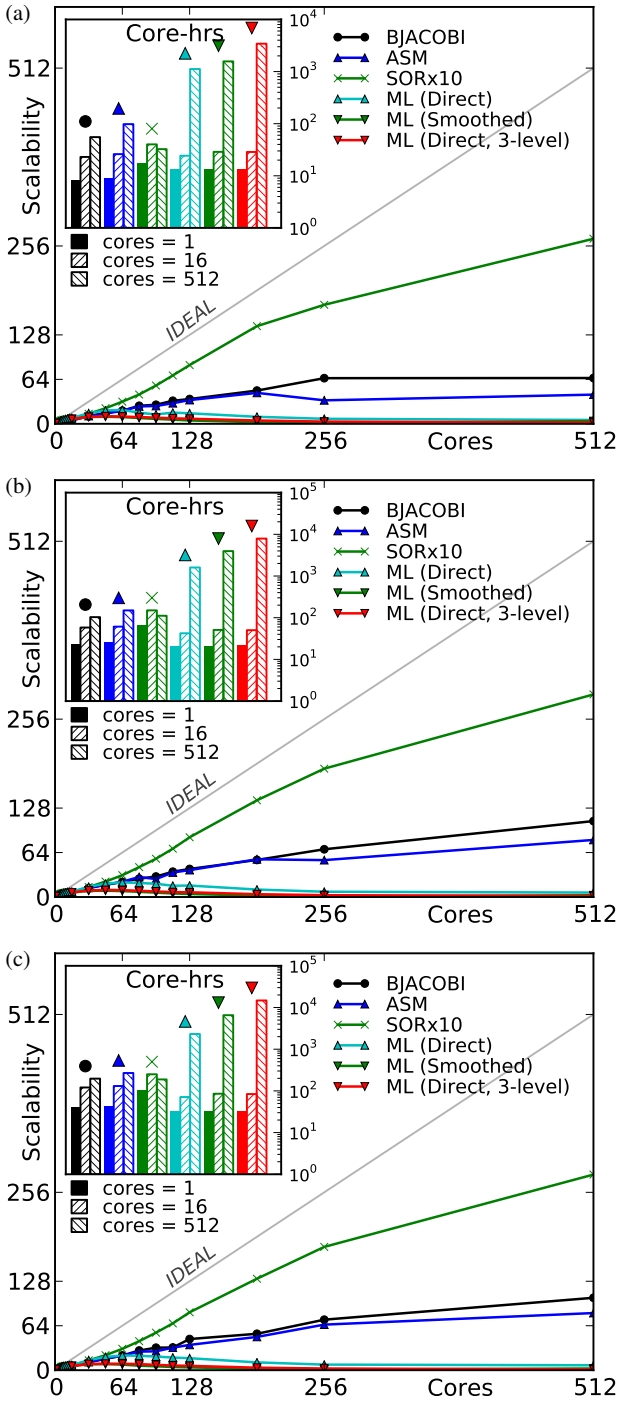


**Fig. 11** Scalability of the solve routines in the pressure equation, using the KVLCC2 test case with inner-loop convergence tolerance set to (a) 0.1, (b) 0.01 and (c) 0.001. The results compare different preconditioners including Block Jacobi, Additive Schwarz (ASM), SOR smoothing (SORx10) and a multi-grid method (ML). FGMRES is used as the solver to allow more flexible preconditioning. Note the exponential scale of the inset core-hours chart.

Overall, the best setup required using 10 iterations of SOR as a smoother/preconditioner. The initial study is re-illustrated using this configuration in figure 12. Inter-nodal scaling is significantly improved (compared to figure 6.a), which is encouraging, but parallel efficiency is still poor and global communications are still limiting. Overall wall-time on 512 cores has been improved by approximately 30%, but with stricter convergence tolerances these gains are lost due to the poor numerical properties of SOR.

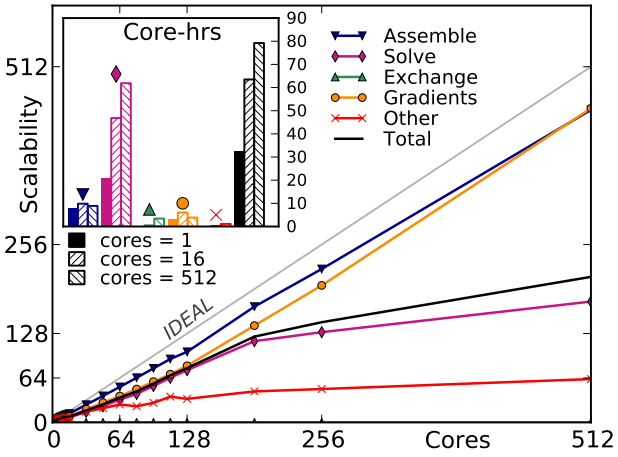


**Fig. 12** Scalability of the code for the KVLCC2 test case, and the profiled routines within, as the number of cores increases. These results used FGMRES with 10 iterations of SOR as a smoother, with an inner-loop relative convergence tolerance of 0.1. Similar results were found for the LDCF test case.

## 6 Conclusions

The main causes of inefficiency and poor scalability of the SIMPLE method have been analyzed by profiling the performance of a state-of-the-art CFD code from 1 to 512 cores. The results show that the main bottleneck is the linear equation-system solvers, particularly for the Poisson pressure equation. The main problem with the linear equation-system solvers is the large amount of expensive, unscalable global communications that are performed. Profiling with hardware counters has also revealed further problems at the intra-nodal level due to memory-bandwidth contention.

Experiments were performed in order to measure performance differences between various state-of-the-art linear equation-system solvers and preconditioners. Recent developments such as a ‘pipelined’ version of GMRES showed improved inter-nodal scalability but gave worse absolute speed and intra-nodal scalability – overall giving only a minor performance increase. Multigrid methods offer some hope, but their performance is nuanced and difficult to predict. The results showed that multigrid preconditioners were

able to offer better absolute speed, but did not scale as well as simpler preconditioners such as Block Jacobi. Depending on the convergence tolerance of the linear equation system, simple smoothers often gave the best performance. Replacing the classical Block Jacobi preconditioner with ten iterations of Successive Overrelaxtion improved overall walltime on 512 cores by 30%.

By 2020, supercomputers are expected to be 3000-times more parallel [19], with total power (and practical simulation size) growing by a factor of just ≈11. Based on these hardware predictions, the cells-per-core ratio must drop by a factor of at least 250 in order to maintain a practical simulation time. Today, a practical simulation of 50-million elements, using 512 cores of Iridis4, would achieve a cells-per-core ratio of approximately 100-thousand. Dividing this by 250 gives a predicted cells-per-core ratio of just 391. The results have shown that scalability at 5200 cells-per-core is already poor, with a maximum parallel efficiency of 25-50%. Beyond 512 cores, negative scalability is likely, with simulation time increasing as more cores are added. With the current state of the CFD algorithm, extrapolating to less than 500 cells-per-core is almost inconceivable.

There are currently developments to improve on the presented SOR results using 'chaotic' iterative methods, which provide even better scalability by removing the implicit synchronization in the sparse-matrix-vector communications; improving cache-use at the intra-nodal level; and slightly improving convergence rates. These chaotic methods could also be used to accelerate multigrid methods [14, 15]. Regardless of the specific methods, more research is needed to carry incompressible CFD codes into the next era of supercomputing, where many-core machines (including GPU or co-processor architectures) will be commonplace. Minor improvements can be expected from alternative software models (such as hybrid parallelization), but significant changes are required at the algorithmic level to keep up with rapidly-evolving hardware.

## Acknowledgements

Over 3000 simulations have been performed to obtain the results presented herein. The authors acknowledge the use of the IRIDIS High Performance Computing Facility, and associated support services at the University of Southampton, in the completion of this work. The authors would also like to thank C.M. Klaij (MARIN) for his guidance and expertise.

## References

1. Balay S, Abhyankar S, Adams MF, Brown J, Brune P, Buschelman K, Eijkhout V, Gropp WD, Kaushik D, Knepley MG, McInnes LC, Rupp K, Smith BF, Zhang H (2013) PETSc Users Manual. Tech. Rep. ANL-95/11 - Revision 3.4, Argonne National Laboratory, http://www.mcs.anl.gov/petsc
2. Bandringa H, Verstappen R, Wubbs F, Klaij C, Ploeg A (2012) On Novel Simulation Methods for Complex Flows in Maritime Applications, *Numerical Towing Tank Symposium (NUTTS)*, Cortona, Italy
3. Bhushan S, Carrica P, Yang J, Stern F (2011) Scalability Studies and Large Grid Computations for Surface Combatant Using CFDShip-Iowa. IJHPCA 25(4):466–487
4. Browne S, Dongarra J, Garner N, Ho G, Mucci P (2000) A portable programming interface for performance evaluation on modern processors. Int J High Perform Comput Appl 14(3):189–204, DOI 10.1177/109434200001400303
5. CD-Adapco (2010) Star-CCM+ Performance Benchmark and Profiling, *HPC Advisory Council (Best Practices)*
6. Culpo M (2011) Current Bottlenecks in the Scalability of OpenFOAM on Massively Parallel Clusters. Tech. rep., Partnership for Advanced Computing in Europe
7. Dennard R, Gaensslen F, Rideout V, Bassous E, Leblanc A (1999) Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions. Proceedings of the IEEE 87(4):668–678
8. Eca L, Hoekstra M (2012) Verification and Validation for Marine Applications of CFD, *29th Symposium on Naval Hydrodynamics*, Gothenburg, Sweden.
9. G Karypis (Acc. 2014) METIS: Serial Graph Partitioning And Fill-Reducing Matrix Ordering, v5.1.0, department of Computer Science and Engineering, University of Minnesota, MN, USA. http://glaros.dtc.umn.edu/gkhome/views/metis
10. Gee M, Siefert C, Hu J, Tuminaro R, Sala M (2006) ML 5.0 Smoothed Aggregation User's Guide. Tech. Rep. SAND2006-2649, Sandia National Laboratories
11. Ghysels P, Ashby TJ, 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
12. Gropp W, Kaushik D, Keyes D, Smith B (2000) Analyzing the Parallel Scalability of an Implicit Unstructured Mesh CFD Code. In: Valero M, Prasanna V, Vajapeyam S (eds) High Performance Computing HiPC 2000, Lecture Notes in Computer Science, vol 1970, Springer Berlin Heidelberg, pp 395–404
13. Hawkes J, Turnock SR, Cox SJ, Phillips AB, Vaz G (2014) Performance Analysis Of Massively-Parallel Computational Fluid Dynamics, *The 11th International Conference on Hydrodynamics (ICHD)*, Singapore
14. Hawkes J, Turnock SR, Cox SJ, Phillips AB, Vaz G (2014) Potential of Chaotic Iterative Solvers for CFD, *The 17th Numerical Towing Tank Symposium (NuTTS*

*2014)*, Marstrand, Sweden
15. Hawkes J, Turnock SR, Cox SJ, Phillips AB, Vaz G (2015) Chaotic Linear Equation-System Solvers for Unsteady CFD, *The 6th International Conference on Computational Methods in Marine Engineering (MARINE 2015)*, Rome, Italy
16. Hewlett-Packard Development Company (2014) Scalability of ANSYS 15.0 Applications and Hardware Selection. Tech. rep.
17. Horst S (2013) Why We Need Exascale And Why We Won't Get There By 2020, *Optical Interconnects Conference*, Santa Fe, New Mexico, USA
18. 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
19. 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) ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems, *DARPA IPTO*
20. 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
21. Lehoucq RB, Sorensen DC, Yang C (1998) ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems by Implicitely Restarted Arnoldi Methods. SIAM, Philadelphia, PA, USA
22. Menter F, Kuntz M, Langtry R (2003) Ten Years of Industrial Experience with the SST Turbulence Model. In: Turbulence, Heat and Mass Transfer 4, Antalya, Turkey
23. Olson LN, Schroder JB (2011) Smoothed Aggregation Multigrid Solvers For High-order Discontinuous Galerkin Methods For Elliptic Problems. Journal of Computational Physics 230(18):6959 – 6976
24. Pereira F, Eca L, Vaz G (2013) On the Order of Grid Convergence of the Hybrid Convection Scheme for RANS Codes, in proceedings of *CMNI*, Bilbao, Spain.
25. Pringle G (2010) Porting OpenFOAM to HECToR, EPCC, The University of Edinburgh
26. Rosetti G, Vaz G, Fujarra A (2012) URANS Calculations for Smooth Circular Cylinder Flow in a Wide Range of Reynolds Numbers: Solution Verification and Validation. Journal of Fluids Engineering, ASME p 549
27. Shalf J (2013) The Evolution of Programming Models in Response to Energy Efficiency Constraints, *Oklahoma Supercomputing Symposium*, Norman, Oklahoma, USA.
28. Slotnick J, Khodadoust A, Alonso J, Darmofal D, Gropp W, Lurie E, Mavriplis D (2014) CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences. Tech. Rep. March, NASA Langley Research Center, Hampton, VA, URL http://ntrs.nasa.gov/search.jsp?R=20140003093
29. Top 500 List (Acc. 2013) http://www.top500.org
30. Vaz G, Jaouen F, Hoekstra M (2009) Free-Surface Viscous Flow Computations: Validation of URANS Code FRESCO, *28th International Conference on Ocean, Offshore and Arctic Engineering (OMAE)*, Honolulu, Hawaii, USA.
31. VI-HPS, Score-P, v123 (Acc. 2013) http://www.vi-hps.org/projects/score-p