FILMPAR: A parallel algorithm designed for the efficient and accurate computation of thin film flow on functional surfaces containing micro-structure

Y C Lee [a], H M Thompson [a], P H Gaskell [a,1]

[a] *School of Mechanical Engineering, The University of Leeds, Leeds, United Kingdom. LS2 9JT*

Abstract

A highly efficient and portable parallel multigrid algorithm for solving a discretised form of the coupled lubrication equations for three-dimensional, gravity-driven, continuous thin film free-surface flow over substrates containing micro-scale topography is presented. Although applicable to problems involving heterogeneous and distributed features, for illustrative purposes the algorithm is benchmarked for the case of flow over a single trench topography; this enables direct comparisons with complementary experimental data and existing serial multigrid solutions. Parallel performance is accessed and shown to lead to super-linear behaviour provided the finest multigrid mesh level employed is sufficiently so.

Key words: Multigrid, parallel computing, thin film flow, lubrication equations

PROGRAM SUMMARY

Manuscript Title: FILMPAR: A parallel algorithm designed for the efficient and accurate computation of thin film flow on functional surfaces containing micro-structure
Authors: P.H. Gaskell, Y.C. Lee, H.M. Thompson
Program Title: FILMPAR
Journal Reference:
Catalogue identifier:
Licensing provisions: none
Programming language: C++
Computer: Desktop, server

[1] Corresponding author: p.h.gaskell@leeds.ac.uk

Operating system: Unix/Linux, Mac OS X
RAM: 512 Mbytes
Number of processors used: 128
Supplementary material:
Keywords: Multigrid, parallel computing, thin film flow, lubrication equations, adaptive time-stepping
PACS:
Classification: 12
External routines/libraries: GNU C/C++, OpenMPI
Subprograms used:
*Catalogue identifier of previous version:**
*Journal reference of previous version:**
*Does the new version supersede the previous version?:**

Nature of problem:
Thin film flows over functional substrates containing well-defined single and complex topographical features are of enormous significance, having a wide variety of engineering, industrial and physical applications. However, despite recent modelling advances, the accurate numerical solution of the equations governing such problems is still at a relatively early stage. Indeed, recent studies employing a simplifying long-wave approximation have shown that highly efficient numerical methods are necessary to solve the resulting lubrication equations in order to achieve the level of grid resolution required to accurately capture the effects of micro- and nano-scale topographical features.
Solution method:
A portable parallel multigrid algorithm has been developed for the above purpose, for the particular case of flow over submerged topographical features. Within the multigrid framework adopted, a W-cycle is used to accelerate convergence in the respect of the time dependent nature of the problem, with relaxation sweeps performed using a fixed number of pre- and post- Red-Black Gauss-Seidel Newton iterations. In addition, the algorithm incorporates automatic adaptive time-stepping to avoid the computational expense associated with repeated time-step failure.
*Reasons for the new version:**

*Summary of revisions:**

Restrictions:

Unusual features:

Additional comments:

Running time:
1.31 minutes using 128 processors on BlueGene/P with a problem size of over 16.7

million nodes.
References:

Items marked with an asterisk are only required for new versions of programs previously published in the CPC Program Library.

1 Introduction

The accurate prediction of the free-surface disturbance arising from the flow of a continuous thin liquid film over functional substrates containing regions of micro- or nano-scale topography presents a considerable challenge given that the same can persist over length scales several orders of magnitude greater than the topography itself [1–3]. The problem becomes further exacerbated when the features concerned are: (i) small, thus requiring considerable mesh resolution to achieve the necessary level of solution accuracy and mesh independence; (ii) heterogeneous, covering a wide extent of the substrate's surface.

The above has motivated the development of an efficient, fully-implicit multigrid strategy for investigating such flows, by solving the lubrication equation(s) that result from using the long-wave approximation to simply the governing Navier-Stokes equations under the assumption that the ratio of the undisturbed asymptotic film thickness to that of the characteristic in-plane length scale is small, coupled with the neglect of inertia. The approach has been shown to be very robust and to return an order of magnitude, or more, improvement in the rate of convergence compared to ADI schemes [4], the benefits of which can be further enhanced by adopting error-controlled automatic adaptive time-stepping [5] and/or adaptive mesh refinement [6].

Described below is a new portable parallel multigrid lubrication flow solver that has been developed to solve continuous thin film flows over submerged topography. Benchmark results are presented, compared against complementary experimental data and results obtained from serial multigrid predictions of the free-surface disturbance, which demonstrate the accuracy of the parallel implementation. The parallel performance of the solver, executed on a distributed memory IBM BlueGene/P computing platform, is assessed and shown to lead to super-linear behaviour provided the finest mesh level employed in the multigrid approach is sufficiently so.

2 Theoretical background

2.1 Mathematical formulation and problem specification

Figure 1 is a sketch of the motion of a thin liquid film of undisturbed asymptotic film thickness, H_0, whose general thickness, defined using a three-dimensional

Cartesian coordinate system (X, Y, Z) at time, T, over a substrate of width, W_p, length, L_p, is $H(X, Y, T)$; the constant volumetric flow per unit width being Q_0. The substrate is inclined at an angle θ to the horizontal and contains, for illustration purposes, a single rectangular trench topography, $S(X, Y)$, of depth, S_0. The liquid is assumed Newtonian and incompressible with constant viscosity, μ, density, ρ, and surface tension, σ.

A generic set of governing equations, based on the long-wave lubrication approximation, for thin-film flow is derived. By proceeding as in [6], the governing Navier-Stokes equations simply considerably by neglecting inertia and assuming that $\epsilon = H_0/L_0$ is small, where L_0 is the characteristic in-plane length scale, proportional to the capillary length, L_c, given by:

$$L_0 = \beta \left(\frac{\sigma H_0}{3\rho g \sin\theta} \right)^{1/3} \quad \text{and} \quad \beta = L_0/L_c \ , \tag{1}$$

with $H_0 = (3\mu Q_0/\rho g \sin\theta)^{1/3}$, the undisturbed Nusselt film thickness; where g is the acceleration due to gravity.

Neglecting terms of $\mathrm{O}(\epsilon^2)$ and smaller and imposing no-slip conditions and zero tangential shear stress at the substrate and free-surface, respectively, and using the following scalings [7], $(x, y) = (X, Y)/L_0$, $h = H/H_0$, $t = U_0 T/L_0$, $U_0 = 3Q_0/2H_0$, $s = S/H_0$ and $p = 2P/\rho g L_0 \sin\theta$, results in the following equation for h:

$$\begin{aligned} \frac{\partial h}{\partial t} = & \frac{\partial}{\partial x}\left\{ \frac{h^3}{3}\frac{\partial}{\partial x}\left[-\frac{6}{\beta^3}\left(\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} \right) + \frac{2}{\beta} 6^{1/3} N \psi \right] - \frac{2}{3} h^3 \right\} \\ & + \frac{\partial}{\partial y}\left\{ \frac{h^3}{3}\frac{\partial}{\partial y}\left[-\frac{6}{\beta^3}\left(\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} \right) + \frac{2}{\beta} 6^{1/3} N \psi \right] \right\} \ , \end{aligned} \tag{2}$$

where $\psi = h + s$. Alternatively, the above can be written as a coupled set of equations for h and p, the pressure throughout the film, which from the standpoint of computational efficiency has been shown to be preferable [8]:

$$\frac{\partial h}{\partial t} = \frac{\partial}{\partial x}\left[\frac{h^3}{3}\left(\frac{\partial p}{\partial x} - 2 \right) \right] + \frac{\partial}{\partial y}\left[\frac{h^3}{3}\left(\frac{\partial p}{\partial y} \right) \right] \ , \tag{3}$$

$$p = -\frac{6}{\beta^3}\nabla^2(\psi) + \frac{2}{\beta} 6^{1/3} N(\psi) \ , \tag{4}$$

with the pressure datum is set to zero; $N = Ca^{1/3} \cot\theta$ measures the influence of the normal component of gravity on the flow profile and $Ca =$

$(9\mu^2 Q_0^2/8\rho g\sigma^3 \sin\theta)^{1/3}$ is the Capillary number.

2.2 Topography specification

Following [9], the topography depth, s, is defined via an arctangent function which enables the side steepness to be controlled easily, thus providing the flexibility needed to create simple primitive shapes [6]. For example, the rectangular topography shown in Figure 1, of length $l_t = L_T/L_0$, width $w_t = W_T/L_0$ and height, $s_0 = S_0/H_0$, centred at point (x_t, y_t), is specified by defining:

$$s(x,y) = \frac{s_0}{b_0}\left[\tan^{-1}\left(\frac{-a_x - l_t/2}{\gamma l_t}\right) + \tan^{-1}\left(\frac{a_x - l_t/2}{\gamma l_t}\right)\right] \times \left[\tan^{-1}\left(\frac{-a_y - w_t/2}{\gamma w_t}\right) + \tan^{-1}\left(\frac{a_y - w_t/2}{\gamma w_t}\right)\right] , \quad (5)$$

where γ is an adjustable steepness parameter, $b_0 = 4\tan^{-1}(1/2\gamma)\tan^{-1}(A/2\gamma)$, $A = w_t/l_t$ is the aspect ratio and $a_x = x_t - x$ and $a_y = y_t - y$ are the local topography co-ordinates in the x- and y-directions, respectively.

Equation (5) can be used to specify other simple primitive topography shapes such as elliptic or lozenge shapes by modifying a_x and a_y appropriately. It is then relatively straightforward to add and subtract these primitives to create complex topographical patterns to represent realistic engineering functional substrates [10].

2.3 Boundary conditions

The problem is closed by specifying appropriate boundary conditions and assuming developed flow conditions to exist far upstream, namely:

$$h(x = 0, y) = 1 , \quad (6)$$

while imposing zero normal flux conditions at the other streamwise and spanwise boundaries yield:

$$\frac{\partial h}{\partial n} = \frac{\partial p}{\partial n} = 0. \quad (7)$$

2.4 Spatial discretisation

The lubrication equations (3) and (4) are discretised using a central finite-difference scheme with uniform grid spacing, Δ, in both the x- and y-directions; leading to the following second order accurate spatial analogues for h and p:

$$\begin{aligned}\frac{\partial h_{i,j}}{\partial t} = \frac{1}{\Delta^2}\Big[&\frac{h^3}{3}\Big|_{i+\frac{1}{2},j}(p_{i+1,j}-p_{i,j}) - \frac{h^3}{3}\Big|_{i-\frac{1}{2},j}(p_{i,j}-p_{i-1,j}) + \\ &\frac{h^3}{3}\Big|_{i,j+\frac{1}{2}}(p_{i,j+1}-p_{i,j}) - \frac{h^3}{3}\Big|_{i,j-\frac{1}{2}}(p_{i,j}-p_{i,j-1})\Big] - \\ &\frac{2}{\Delta}\Big(\frac{h^3}{3}\Big|_{i+\frac{1}{2},j} - \frac{h^3}{3}\Big|_{i-\frac{1}{2},j}\Big)\,, \end{aligned} \tag{8}$$

$$\begin{aligned}p_{i,j} = -\frac{6}{\beta^3\Delta^2}\Big[&(h_{i+1,j}+s_{i+1,j}) + (h_{i-1,j}+s_{i-1,j}) + (h_{i,j+1}+s_{i,j+1}) + \\ &(h_{i,j-1}+s_{i,j-1}) - 4(h_{i,j}+s_{i,j})\Big] + \frac{2\sqrt[3]{6}N}{\beta}(h_{i,j}+s_{i,j})\,, \end{aligned} \tag{9}$$

at each node, (i,j), in the computational domain. The pre-factor terms, $\frac{h^3}{3}|_{i\pm\frac{1}{2},j}$ and $\frac{h^3}{3}|_{i,j\pm\frac{1}{2}}$, are obtained from linear interpolation between neighbouring points.

2.5 Temporal discretisation

Time integration is performed using the second-order accurate Crank-Nicholson method to approximate the time-derivative of equation (8) and which, rewriting the right-hand-side as a function, $\mathcal{F}(h_{i,j}, p_{i,j}, h_{i\pm1,j}, p_{i\pm1,j}, h_{i,j\pm1}, p_{i,j\pm1})$, leads to an equation of the form:

$$\begin{aligned}&h^{n+1}_{i,j} - \frac{\Delta t^{n+1}}{2}\mathcal{F}(h^{n+1}_{i,j}, p^{n+1}_{i,j}, h^{n+1}_{i\pm1,j}, p^{n+1}_{i\pm1,j}, h^{n+1}_{i,j\pm1}, p^{n+1}_{i,j\pm1}) \\ &= h^{n}_{i,j} + \frac{\Delta t^{n+1}}{2}\mathcal{F}(h^{n}_{i,j}, p^{n}_{i,j}, h^{n}_{i\pm1,j}, p^{n}_{i\pm1,j}, h^{n}_{i,j\pm1}, p^{n}_{i,j\pm1}), \end{aligned} \tag{10}$$

where $\Delta t^{n+1} = t^{n+1} - t^n$; the right hand side of the above equation is expressed in terms of known variables at the end of the nth time step, $t = t^n$.

Automatic adaptive time-stepping is achieved by employing a temporal error control algorithm based on predictor-corrector stages, as reported in [5]. The

method employed provides an efficient alternative to existing schemes such as [11] by using time-stepping based on local error estimates, one that is implicit and second order accurate, obtained from the difference between the current solution and predicted one and which act as an indicator of whether to increase or decrease the time step in a controlled manner whilst concurrently minimising the computational expense associated with repeated time step failure.

2.6 Multigrid strategy

In line with the multigrid algorithm employed in [6], a sequence of progressively finer grids ($\mathcal{G}_k$: $k = 0, 1, ..., K$) is defined with uniform grid spacing, Δ_k. Each grid level, $\mathcal{G}_k$, has $n_k = 2^{k+c+1} + 1$ nodes per unit length in each co-ordinate direction, where c is a constant defining the resolution of the coarsest grid level such that mesh size, $\Delta_k = 2^{-(k+c+1)}$.

The associated time-dependent, nonlinear, coupled set of governing lubrication equations (9) and (10) are solved using a full approximation storage (FAS) multigrid scheme [12]. By combining and re-writing the above set of discretised equations as follows:

$$\mathcal{N}_k u_k^{n+1} = f_k(u_k^n) \tag{11}$$

where $u_k = (h_k, p_k)^T$ with $\mathcal{N}_k = (\mathcal{N}_k^h, \mathcal{N}_k^p)^T$ and $f_k = (f_k^h, f_k^p)^T$ corresponds to the left- and right-hand sides of the both the film thickness and pressure equations on $\mathcal{G}_k$, respectively, the multigrid method can be summarised schematically, as in Algorithm 1.

Note that the cycle-index, κ, which represents the number of iterations of the multigrid process at each intermediate grid level, determines the type of coarse grid correction cycle. Here, the W-cycle ($\kappa = 2$) is employed to optimise the convergence properties provided by the scheme for the time-dependent lubrication solution.

The inter-grid transfer operators used in the multigrid algorithm consist of restriction operator R_k^{k-1} (from $\mathcal{G}_k$ to $\mathcal{G}_{k-1}$) and prolongation operator, I_{k-1}^k (grid $\mathcal{G}_{k-1}$ to $\mathcal{G}_k$), whose orders depend on the order of derivatives being solved. For the second order lubrication equations under consideration, the standard half-weighting restriction and bi-linear interpolation are appropriate.

In order to avoid convergence problems that may arise from using an arbitrary initial guess, the Full Multigrid (FMG) technique [12] is used together with the FAS algorithm, see Algorithm 2. Note that Π_{k-1}^k is a prolongation operator

for transferring information from $\mathcal{G}_{k-1}$ to $\mathcal{G}_k$ and its order may not necessary be equal to that of the prolongation operator I_{k-1}^k.

2.7 Relaxation

Relaxation is performed using a fix number, (ν_1, ν_2), of pre- and post- Red-Black Gauss-Seidel Newton iterations with the linearised Newton iterative step written in the form:

$$\frac{\partial \mathcal{N}_k^h}{\partial h_k^{n+1}} \Delta h_k + \frac{\partial \mathcal{N}_k^h}{\partial p_k^{n+1}} \Delta p_k = f_k^h - \mathcal{N}_k^h \left(h_k^n, p_k^n\right) \ , \tag{12}$$

$$\frac{\partial \mathcal{N}_k^p}{\partial h_k^{n+1}} \Delta h_k + \frac{\partial \mathcal{N}_k^p}{\partial p_k^{n+1}} \Delta p_k = f_k^p - \mathcal{N}_k^p \left(h_k^n, p_k^n\right) \ , \tag{13}$$

for the increments Δh and Δp at point (i, j) on level k. The expressions are then solved simultaneously to obtain a new approximate solution on $\mathcal{G}_k$:

$$\tilde{h}_k^{n+1} = h_k^{n+1} + \Delta h_k \ , \tag{14}$$

$$\tilde{p}_k^{n+1} = p_k^{n+1} + \Delta p_k \ . \tag{15}$$

2.8 Parallel implementation

Parallel implementation of the above automatic adaptive time-stepping multigrid solver is achieved through the use of a message passing interface (MPI) framework, which facilitates portability across different (distributed and shared memory) high performance computing platforms. Parallelism is achieved via a geometric partitioning at each of the grid levels, using a strategy that automatically constructs virtual topologies arranged so as to minimise communication cost. This is illustrated in Figure 2, where the partitions, depending on the number of processors and shape of the domain, are split into, or as close as is possible, equally divisible rectangular blocks. Note that when the number of columns or rows is not an exact multiple of the number of processors used, the left over points are allocated to the last processor associated with the respective rows and columns.

Each of the processors is responsible for implementing the FMG-FAS multigrid algorithm on its own subdomain, making use of additional halo nodes (points located within and adjacent to the dotted lines, Figure 2) to store exchanged computed values between neighbouring processors. The inter-processor communication provides updated values at the halo points during relaxation and

after the inter-grid transfer operations. Additionally, a global communication exchange is performed at the end of each multigrid cycle to ascertain whether a converged solution has been achieved.

The above approach allows standard sequential algorithms, such as the FMG-FAS multigrid scheme to be parallelised without any fundamental alteration, as the assembly of the underlying discrete system of equations is undertaken by the initial domain decomposition process; maintaining this strategy for the parallel solver minimises the need for data movement within a distributed memory parallel code. Nonetheless, obtaining good parallel efficiencies for such a scheme is challenging due to the fact that several computations must be undertaken on relatively coarse meshes which may result in an undesirably large communication overhead.

3 Benchmark numerical results

The benchmark test problem considered is that of the gravity-driven flow of a thin film of water (viscosity, $\mu = 0.001$ Pa s, density, $\rho = 1000$ kg m^{-3} and surface tension, $\sigma = 0.07$ N m^{-1}), having an asymptotic film thickness $H_0 = 100\,\mu$m, over a rigid substrate (of size $l_p = w_p = 100$ and tilted at 30^o to the horizontal) containing a single localised, square trench topography ($\gamma = 0.05$, $s_o = -0.25$, and $l_t = w_t = 1.54$), centred at $(x_t, y_t) = (30.77, 50)$. The above fluid properties and associated flow parameters are consistent with the experiments carried out by [1] enabling direct comparison with the same and with complementary numerical predictions obtained using a serial multigrid algorithm [6] employing automatic mesh adaption. The values for $L_c = L_0$ and N are 0.78 mm and 0.12, respectively; the latter indicting the normal component of gravity has little effect on the resultant free-surface shape [2].

The steady-state free-surface disturbance illustrated in Figure 3 was obtained with a fine mesh containing 1025×1025 nodes, guaranteeing an accurate grid independent solution. It shows the gross characteristic "horse-shoe" bow-wave and a "comet-tail" pattern that results. Streamwise free-surface profiles at different spanwise cross-sectional locations are depicted in Figure 4, comparing those obtained with FILMPAR against their experiment counterparts and ones obtained using a serial multigrid solver. As can be seen, the FILMPAR profiles shows very good qualitative agreement with experimental data, to within the 2% r.m.s experimental error reported in [1], and are indistinguishable from the corresponding serial multigrid solutions [6].

The parallel performance of the portable FILMPAR algorithm was assessed, for illustrative purposes, on a distributed memory IBM BlueGene/P computing platform comprised of four 850 MHz PowerPC 450 cores per processor

and 2 GB of local memory. Figure 5 shows the speedup and parallel efficiency of the parallel multigrid solver when executed on between 4 and 128 processors, together with the theoretically expected values. The results are presented relative to the smallest number of processors used, which in this case is 4, employing 4 multigrid levels with W(2,2)-cycles. Figure 5(a) indicates that a relatively good level of speedup is achieved, as one would expect, giving better returns when finer meshes (1025×1025 and above) are employed, as opposed to coarser grids due to the considerable inter-processor communication cost incurred during coarse grid computations that lead to large computational overheads. This is more clearly shown in Figure 5(b) where parallel efficiency is seen to approach the theoretical limit when very fine grids are employed as a result of large relative computation to communication ratios especially when a larger number of processors are used. Indeed, super-linear parallel performance is observed with 16 processor for a fine mesh having 2049×2049 points, and with 16, 32 and 64 processors when the fine mesh contains 4097×4097 points. The reason for this can be attributed to the likely fact of the efficiency of communication and cache effects, which come into play when the size of the sub-domains become small and the variables accessed fit into the cache dramatically reducing memory access times and hence computational times.

The precise choice made for size of the coarsest grid level has a bearing on the overall parallel performance. A comparison of the efficiency of FILMPAR for fine grid levels of 2049×2049 and 4097×4097, using different number of multigrid levels is shown in Figure 6. It reveals that the cost of performing a greater number of multigrid levels diminishes as a finer mesh is used, as indicated by the improved consistency in speedup efficiencies depicted in Figures 6(a) and 6(b). Speedup becomes more pronounced as the number of multgrid levels increases, particularly when a large number of processors is employed. This is more evident for the case when the finest mesh contains 2049×2049 points, which is to be expected since the coarse grid levels utilised are coarser and thus the amount of computation per processor is smaller.

Given that the computational complexity of the underlying sequential multigrid algorithm is of O(N) [5], with N the total number of unknowns, it is clear that a perfectly scalable algorithm should yield the same performance for all runs with a fixed number of points per processor. This is demonstrated plainly in Figure 6 which shows that as the number of points per processor is halved, the speedup efficiency doubles, suggesting that the parallel implementation is very efficient.

4 Conclusions

A highly accurate and efficient portable parallel multigrid lubrication solver is presented for the solution of three-dimensional gravity-driven continuous thin film free-surface flow over substrates containing topography. It enables results to be obtained on extremely fine meshes, ones that cannot be attempted using a serial solver; the solver having a parallel efficiency that is both scalable and cost effective. Benchmark results, compared against known experimental data and with existing serial multigrid solutions reveal that the parallel solver generates accurate steady-state free-surface disturbances for flow over a small trench.

The performance characteristics of the parallel solver indicate that its implementation is highly efficient specifically when, as is the case of flow over small topographical features, a fine mesh is required to produce accurate mesh independent solutions; small losses in efficiency that can be attributed to the overhead associated with inter-processor communication. The latter becomes apparent in the case of coarser grid levels where the amount of computation per processor is relatively small. Accordingly, given that multigrid schemes are designed do most of their work on coarser grid levels, it is important to select a fine grid level that does not compromise parallel performance.

5 Overview of the software structure

FILMPAR is written entirely using C++, making full use of its object-oriented framework to simplify the creation of arrays and variable structures as well as to allow for code flexibility and future extensibility. The program has a modular form, with routines and functions defined in separate files, as per the following structure:

`./input` – stores user provided input data files
`./src` – location of the source files
`./scr/core` – location of the core source files
`./src/include` – location of the header files
`./src/lib` – location of general library routines
`./src/template` – location of template files
`./output` – stores the output of the program

6 Description of the individual software components

Core source files located at `./src/core`:

- `ats_predict.cpp` – Computes and controls the adaptive time-stepping routine.
- `ats_set.cpp` – Initialises and defines variables h and p for the new time step.
- `engine_clean.cpp` – Uninitialise the grid arrays used and free up memory.
- `engine_init.cpp` –Initialise the grid arrays, topography and initial profiles of the thin film flow problem.
- `engine_mpi_mg.cpp` – Performs a full multigrid (FMG) iteration and determine if additional multigrid iterations are required.
- `engine_mpi_solver.cpp` – Performs the adaptive or non-adaptive time-stepping routine on the multigrid algorithm.
- `engine_mpi_write.cpp` – Determine types of output data.
- `engine_parameters.cpp` – Calls read, process and write input parameter data.
- `evalpressure.cpp` – Calculates initial p from h and s values.
- `func_mpi_lop.cpp` – Calculates $\mathcal{N}_k u_k^{n+1}$.
- `func_mpi_rhs.cpp` – Calculates f_k^n.
- `grids_init.cpp` – Initialises arrays and grid structure for parallel implementation.
- `grids_mpi_reset.cpp` – Resets temporary arrays.
- `mg_mpi_cycle.cpp` – Performs the parallel multigrid cycle with calls to appropriate relaxation and interpolation operators.
- `mg_mpi_lop.cpp` – Sets up domain boundaries and calls `func_mpi_lop.cpp`.
- `mg_mpi_relax.cpp` – Set up domain boundaries and calls `newtitr.cpp`.
- `mg_mpi_solve_coarsest.cpp` – Performs multiple relaxation iterations for coarsest grid solution.
- `mg_mpi_solve_rhs.cpp` – Setup domain and calls `func_mpi_rhs.cpp`.
- `mpi_init.cpp` – Initialises and sets up the parallel environment variables.
- `mpi_uninit.cpp` – Uninitialise the parallel environment variables.
- `newtitr.cpp` – Performs relaxation using Newton iteration on the coupled system of equations h and p.
- `nonats_set.cpp` – Sets up appropriate variables and arrays for non-adaptive time-stepping.
- `parameters_mpi_write.cpp` – Creates and sets up appropriate directories for parameter output files.
- `parameters_process.cpp` – Calculates and processes the input parameters used.
- `parameters_read_tag.cpp` – Defines the tags that are read from the input parameter file.
- `parameters_read.cpp` – Reads the input parameter file.

- `parameters_write_tag.cpp` – Writes out the processed information and parameters used.
- `parameters_write.cpp` – Creates and writes the parameters used into a file.
- `pre.cpp` – Performs the exchange of halo data from initial h, p and s.
- `profiles_film.cpp` – Specifies and sets up the initial profile for the thin film flow problem.
- `profiles_init.cpp` – Initialises the definition of the thin film profile.
- `topography_init.cpp` – Sets up the topography input file for reading.
- `topography_read.cpp` – Reads in the parameters from the input topography file.
- `topography_set.cpp` – Creates and processes the topography information read from the input file.
- `write_data_array.cpp` – Writes parallel output data to file.
- `write_mpi_create_mesh.cpp` – Determines the types of output files.
- `write_mpi_data.cpp` – Creates and prepares the appropriate filenames and data for parallel output.
- `xmg_main.cpp` – The main file that performs the time-adaptive parallel multigrid solver.

Library source files located at `./src/lib`:

- `anorm2_mpi.cpp` – A function that returns the $\mathcal{L}^2$-norm value.
- `copy_mpi.cpp` – Duplicates a grid array variable.
- `interp_mpi.cpp` – Performs the bilinear interpolation of grid variables from a coarse to fine grid.
- `matadd_mpi.cpp` – Calculates the addition of two grid array variables.
- `matsub_mpi.cpp` – Calculates the subtraction of two grid array variables.
- `rstrct_mpi.cpp` – Performs the half-weighting restriction of grid variables from a fine to coarse grid.
- `rstrctinject_mpi.cpp` – Performs the injection of grid variables from a fine to coarse grid.

Header files located at `./src/include`:

- `incl_core_header.h` – Provides the namespace for all the core source files.
- `incl_grids.h` – Defines the structure of the array variables and parallel parameters.
- `incl_lib_header.h` – Provides the namespace for all the library source files.
- `incl_parameters.h` – Defines all the parameters used in the program.
- `incl_template.h` – Defines the one- and two-dimensional array template structures.

Template files located at `./src/template`:

- `tmpl_array_1d.h` – Defines the template and class for one-dimensional arrays.
- `tmpl_array_2d.h` – Defines the template and class for two-dimensional arrays.
- `tmpl_io_check.h` – Defines the template class for input and output file checking.
- `tmpl_level_define.h` – Defines the template class for various multigrid array structures.

Input files located at `./input`:

- `xmg-parameter.input` – Defines the input flow, multigrid and domain variables.
- `xmg-topography.input` – Defines the shape and size of the topography.

7 Installation instructions

The code can be compiled on any system with a GNU C++ compiler and OpenMPI libraries installed, by invoking the following command:

```
$ make
```

All the files will be compiled and linked to produce an output binary executable file named `filmpar`.

Invoking the command:

```
$ make clean
```

will remove all compiled object files while

```
$ make clean-all
```

removes both the compiled object files and the executable binary.

8 Test run description

The binary executable file can be executed on a parallel system with multiple processors by invoking, as an example:

```
$ mpirun -np 8 ./filmpar
```

in which, 8 processors are allocated and utilised for the execution of the program based on the user defined inputs provided in files `./input/xmg-topography.input` and `./input/xmg-paramters.input`, respectively.

All data outputs are stored in the `./output` directory with each set of results generated by the program automatically stored in their respective subdirectories; the first set of results is stored under the sub-directory `./output/result` while subsequent results generated from the execution of the program are stored in sub-directories labelled `result0`, `result1`, ..., increments.

The output files created in the result sub-directories consist of an output file `./output/result/xmg-parameters.output`, which prints out the parameter variables used by the program, and the associated output data files computed on each processor labelled in their respective sub-directories beginning with `./output/result/rank0`, `./output/result/rank1`, ..., `./output/result/rank7`, if 8 processors are used.

9 Acknowledgements

The authors wish to record their gratitude to the Engineering and Physical Sciences Research Council (EPSRC) for supporting this work via grant reference EP/F010745/1. The authors also wish to thank Dr. XJ Gu and Professor DR Emerson (Daresbury Laboratory, Warrington, UK) for informative discussions surrounding parallel computing issues and for access to and use of the Bluegene/P high performance computing platform for benchmarking purposes.

References

[1] M.M.J. Decre, C-J. Baret, J. Fluid Mech. 487 (2003) 147.
[2] P.H. Gaskell, P.K. Jimack, M. Sellier, H.M. Thompson, M.C.T. Wilson, J. Fluid Mech. 509 (2004) 253.
[3] M. Sellier, Y.C. Lee, H.M. Thompson, P.H. Gaskell, Comput. & Fluids 38 (2009) 171.
[4] L.W. Schwartz, R.R. Eley, J. Colloid Interface Sci. 202 (1998) 173.
[5] P.H. Gaskell, P.K. Jimack, M. Sellier, H.M. Thompson, Int. J. Num. Meth. Fluids 45 (2004) 1161.
[6] Y.C. Lee, H.M. Thompson, P.H. Gaskell, Comput. & Fluids 36 (2007) 838.
[7] A. Oron, S.H. Davis, S.G. Bankoff, Reviews of Modern Physics 69 (1997) 931.
[8] N. Daniels, P. Ehret, P.H. Gaskell, H.M. Thompson, M.M.J. Decre, in: Proceedings of the 1st international conference on CFD. Springer, 2001, 279-284.

[9] L.M. Peurrung, G.G. Graves, IEEE Trans. Semi. Man. 6 (1993) 72.
[10] Y.C. Lee, H.M. Thompson, P.H. Gaskell, Int. J. Num. Meth. Fluids 56 (2008) 1375.
[11] J.A. Diez, L. Kondic, J. Comput. Phys. 183 (2002) 274.
[12] A. Brandt, in: Lecture notes in mathematics 960. Springer, Berlin, 1982, 220-312.

$\tilde{u}_k^{n+1} = \text{MGFASCYC}(u_k^{n+1}, f_k^{n+1})$

1: pre-relaxation

perform ν_1 relaxation sweeps on u_k^{n+1}

2: coarse grid correction

compute residual $d_k^{n+1} = f_k^{n+1} - \mathcal{N}_k^{n+1} u_k^{n+1}$ on $\mathcal{G}_k$

restrict residual $d_{k-1}^{n+1} = R_k^{k-1} d_k^{n+1}$ onto $\mathcal{G}_{k-1}$

restrict fine grid solution $u_{k-1}^{n+1} = R_k^{k-1} u_k^{n+1}$ onto $\mathcal{G}_{k-1}$

compute right-hand side $f_{k-1}^{n+1} = d_{k-1}^{n+1} + \mathcal{N}_{k-1}^{n+1} u_k^{n+1}$ on $\mathcal{G}_{k-1}$

If $k = 0$, solve for $\tilde{u}_0^{n+1}$ on $\mathcal{G}_0$

else perform κ iterations on $\tilde{u}_{k-1}^{n+1} = \text{MGFASCYC}(u_{k-1}^{n+1}, f_{k-1}^{n+1})$

endif

compute correction $e_{k-1}^{n+1} = \tilde{u}_{k-1}^{n+1} - u_{k-1}^{n+1}$ on $\mathcal{G}_{k-1}$

interpolate $e_k^{n+1} = I_{k-1}^k e_{k-1}^{n+1}$ onto $\mathcal{G}_k$

update solution $u_k^{n+1} = u_k^{n+1} + e_k^{n+1}$ on $\mathcal{G}_k$

3: post-relaxation

perform ν_2 relaxation sweeps on u_k^{n+1}

Algorithm 1. FAS multigrid cycle

1: **for** $k = 0, 1, 2, \ldots, K$

2: **if** $k = 0$, solve for $\tilde{u}_0^{n+1}$ on $\mathcal{G}_0$

3: **else** interpolate $u_k^{n+1} = \Pi_{k-1}^k \tilde{u}_{k-1}^{n+1}$ onto $\mathcal{G}_k$

4: compute $\tilde{u}_k^{n+1} = \text{MGFASCYC}(u_k^{n+1}, f_k^{n+1})$

5: **endif**

6: **done**

Algorithm 2. FMG iteration

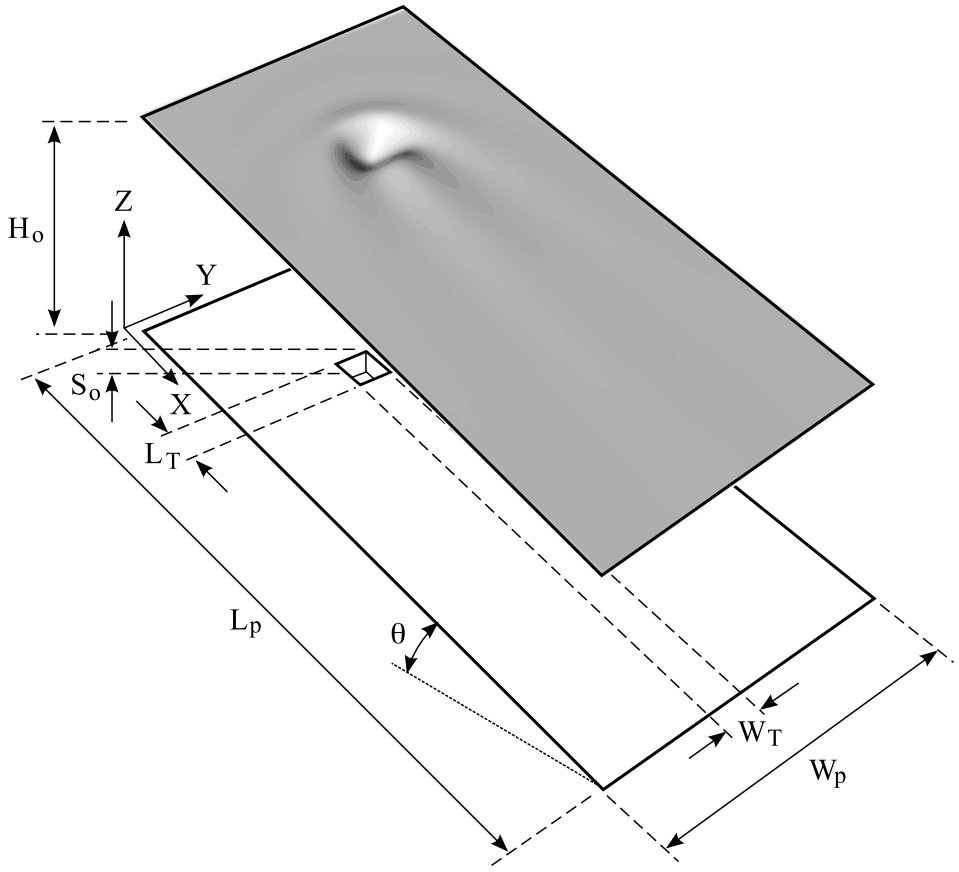


Fig. 1. Schematic diagram of gravity-driven thin film flow (from left to right) over a square trench topography.

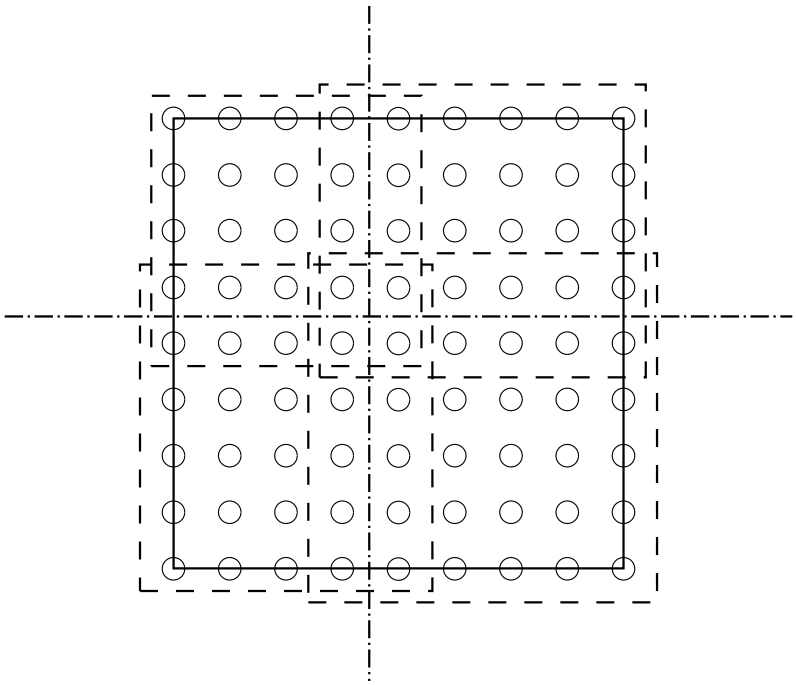


Fig. 2. Partition of a (9×9) coarse grid solution domain across 4 processors. The solid line denote the domain boundary; dash lines indicate the subdomains which are split as closely as possible into equally divisible rectangular blocks based on the total number of processors used.

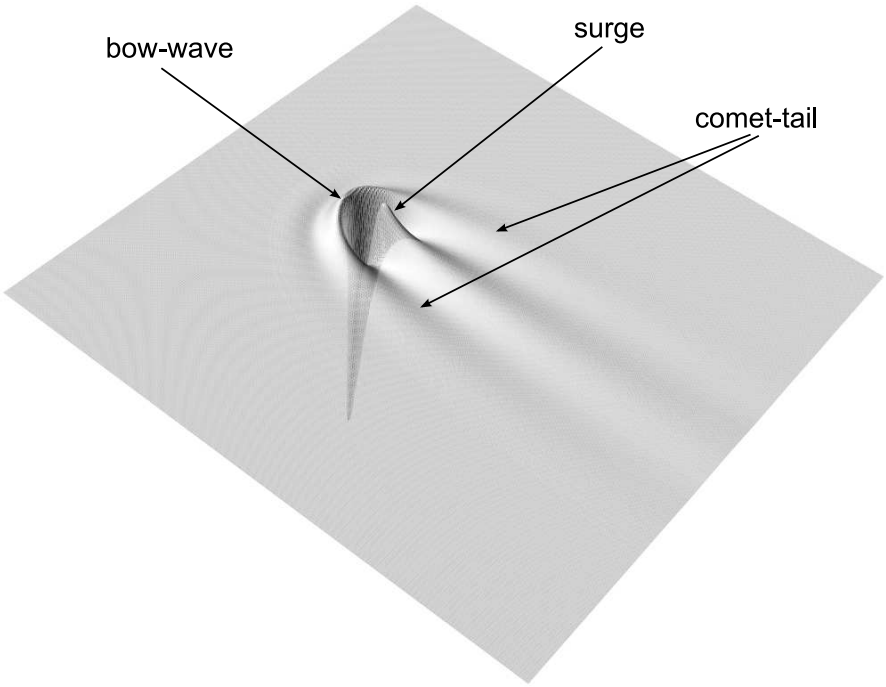
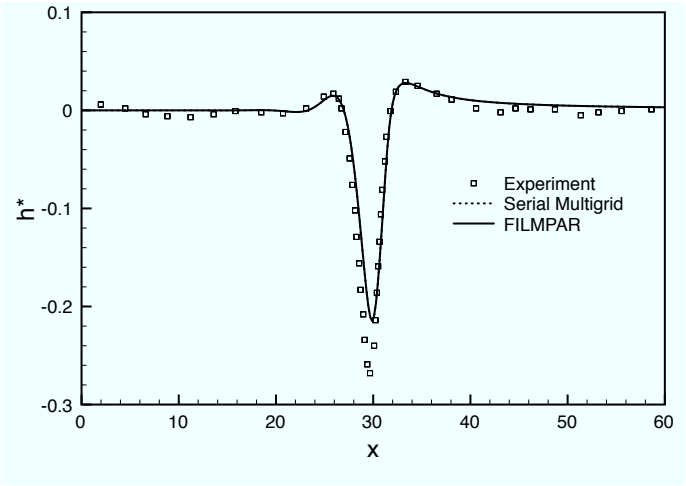
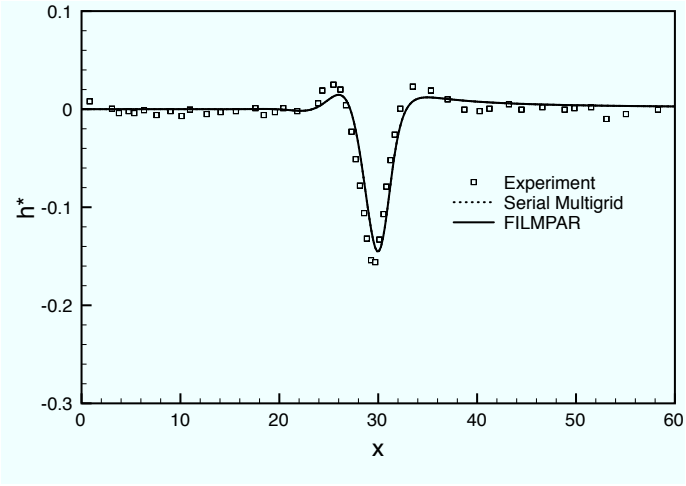


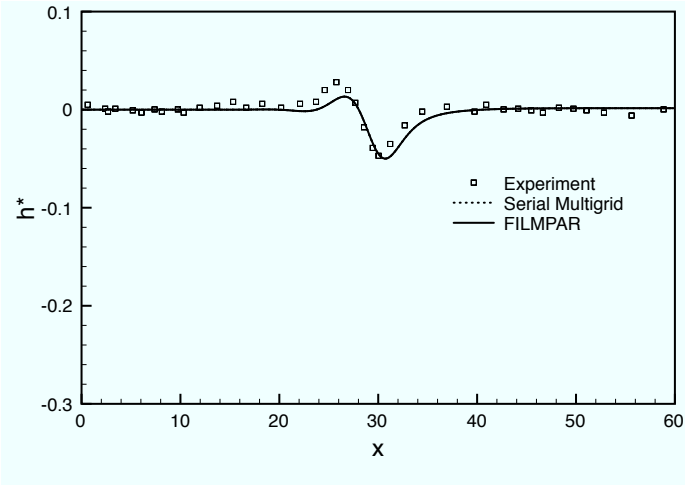
Fig. 3. Characteristic three-dimensional resultant steady-state free-surface disturbance, showing the presence of: an upstream "bow-wave"; a "comet-tail"; a downstream surge. The direction if flow is from top left to bottom right.



(a)

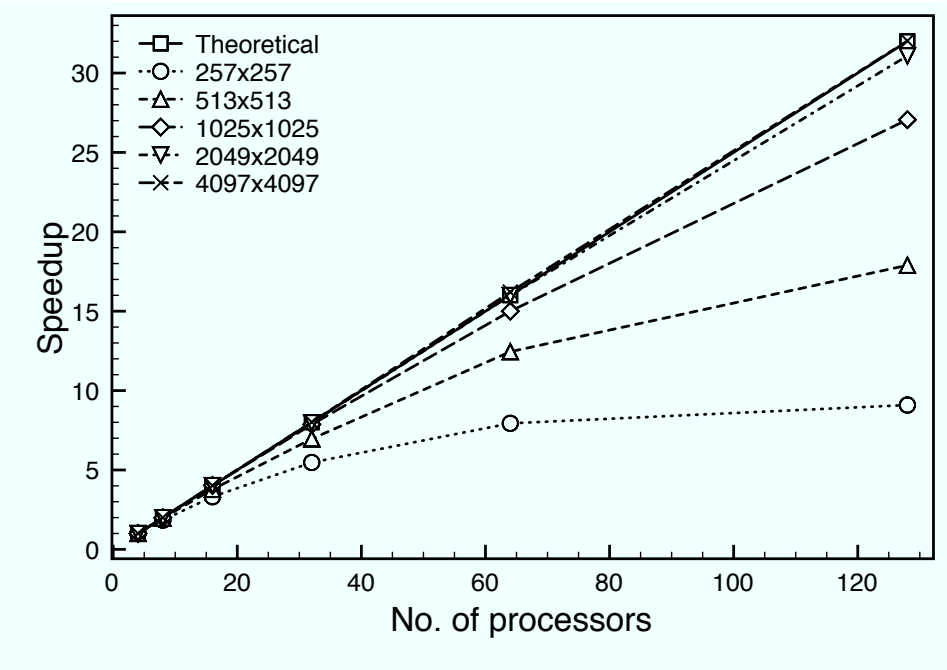


(b)

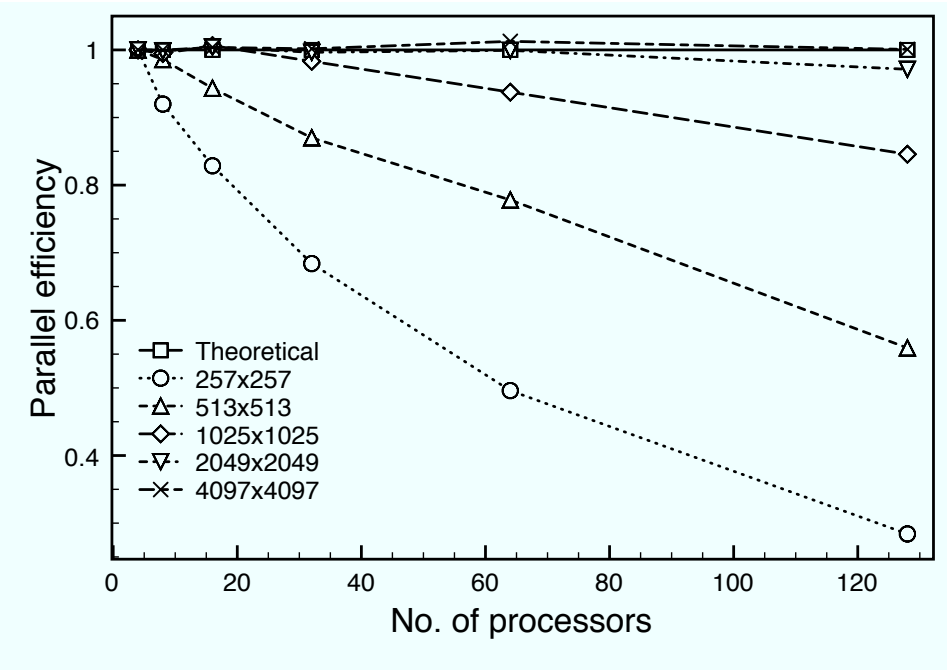


(c)

Fig. 4. Comparison of experimentally obtained [1] and predicted FILMPAR and serial multigrid solutions [6] streamwise free-surface film thickness profile, $h^* = (h + s)/|s_0|$, at span-wise locations: (a) $y - y_t = 0$; (b) $y - y_t = w_t$; (c) $y - y_t = 2w_t$. The direction if flow is from left to right and the predicted profiles are indistinguishable.

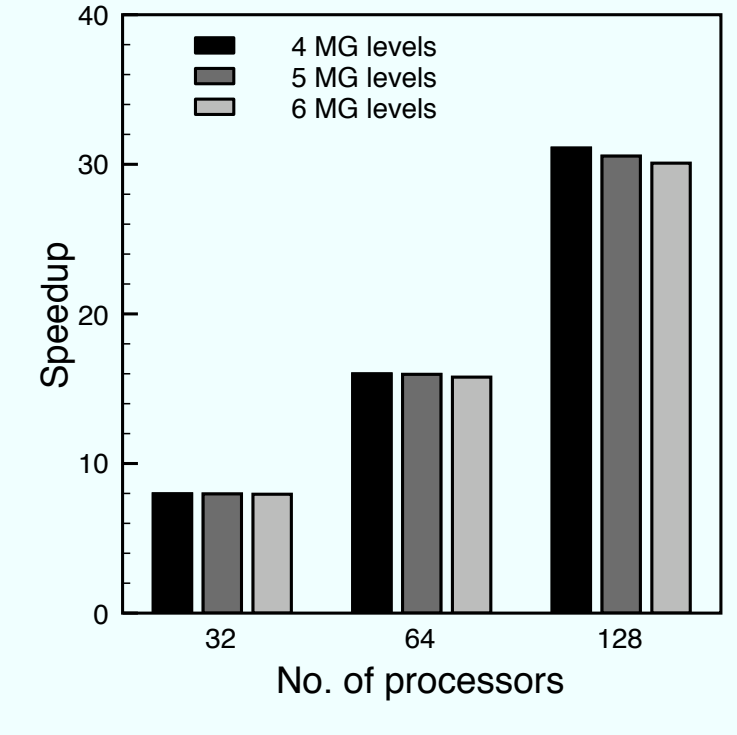


(a)

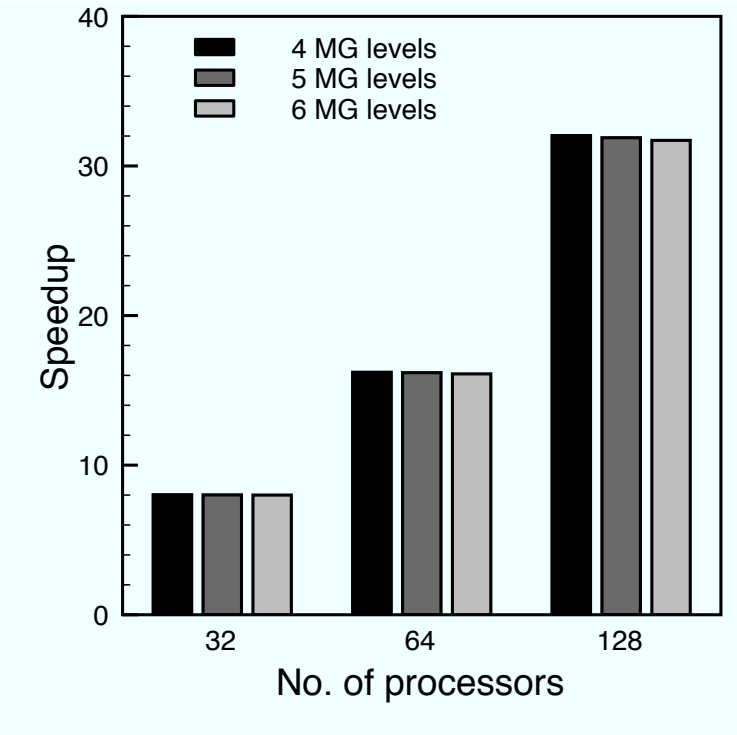


(b)

Fig. 5. Performance of the parallel multigrid solver: (a) relative speedup; (b) parallel efficiency. Comparison is drawn with the theoretical result that might be expected for 5 different mesh densities performed on 4 multigrid W(2,2) levels.



(a)



(b)

Fig. 6. Parallel performance of the multigrid W(2,2) solver obtained using different numbers of grid levels by varying the mesh densities of the coarsest grid employed with different numbers of processors, the finest mesh size being: (a) 2049×2049; and (b) 4097×4097.