# A code for direct numerical simulation of turbulent boundary layers at high Reynolds numbers in BG/P supercomputers

Guillem Borrell *, Juan A. Sillero, Javier Jiménez

*School of Aeronautics, Universidad Politécnica de Madrid, 28040 Madrid, Spain*

A B S T R A C T

A new high-resolution code for the direct numerical simulation of a zero pressure gradient turbulent boundary layers over a flat plate has been developed. Its purpose is to simulate a wide range of Reynolds numbers from $Re_\theta = 300$ to 6800 while showing a linear weak scaling up to 32,768 cores in the BG/P architecture. Special attention has been paid to the generation of proper inflow boundary conditions. The results are in good agreement with existing numerical and experimental data sets.



## 1. Introduction

Turbulent boundary layers have an undeniable technological importance. Roughly one third of the energy dissipated by the movement of vehicles and transport through pipes is caused by the presence of a turbulent boundary layer. This is the reason why turbulent boundary layers were among the first flows to be simulated [1].

Our current research is focused on understanding the flow in the turbulent regions that are further from the wall, where range of scales is wider, and the Reynolds number plays a significant role. Our approach is to analyze data obtained from Direct Numerical Simulation (DNS). While high Reynolds number simulations exist for other wall bounded flows (mainly channels), similar data sets were not available for boundary layers. The present code was developed to generate them.

We needed a high resolution code that is able to perform a DNS of a boundary layer that has good performance and excellent scalability. The starting point was the parallel MPI-only code described in detail in [2]. It needed severe modifications to satisfy the additional constraints that the BG/P architecture introduces. The most relevant change was to add a second level of parallelism with OpenMP, necessary to achieve the desired degree of scalability and performance; and an auxiliary domain to extend the computational box.

Two different simulations using this code were completed in two BG/P supercomputers using 32,768 cores, a zero pressure gradient boundary layer over a flat plate with a Reynolds number based on the momentum thickness of $Re_\theta = 1100 - 6800$ on Intrepid at Argonne National Laboratory [3] and a forced boundary layer with artificial roughness with $Re_\theta = 300 - 4200$ [4] on Jugene at Juelich Forschungszentrum.

### 1.1. Previous DNS of turbulent boundary layers at similar Reynolds numbers

This is a domain-specific code designed to solve a zero pressure gradient turbulent boundary layer in a rectangular domain. It is not comparable to codes like nek5000 [5] that, while also achieving excellent scalability in the same supercomputer architecture, are general purpose and are designed to handle more complex geometries.

One could classify previous simulations of turbulent boundary layers on how they deal with the inhomogeneity of the streamwise direction.

It is worth mentioning the pioneering work of Spalart [1], despite covering a low range of $Re_\theta$. Periodicity in the streamwise direction was enforced with a multiple-scaling transform of the coordinates as well as approximate treatment of the Navier–Stokes equations. This simplification is accurate as long as the streamwise growth of the boundary layer is small; therefore, it is only valid when simulating a short domain.

Another technique to deal with the inhomogeneity is to enforce periodicity by adding a fringe region at the end of the domain,

where the flow is forced back to the laminar regime. This technique was applied, for instance, in [6–8], where the simulation ranges up to $Re_\theta = 4060$. While this method is useful to study the phenomenon of laminar-turbulent transition, it requires to start from a laminar flow condition. This may be a limiting factor when the region of interest is only the one further downstream. It also requires some perturbation to trigger turbulence, what makes the flow dependant on the tripping technique. On the other hand, it allows periodic treatment of the streamwise direction, and simplifies the algorithm significantly.

Finally, one can generate an inflow boundary condition that is already turbulent. While [2,9,10] rescale the flow with a scheme similar to the one proposed by Lund et al. [11] and Ferrante and Elghobashi [12] extended the cited method. The idea is to pick one cross-stream plane at an intermediate part of the domain, and to recycle it as inflow boundary condition. Two aspects must be taken into account: the separation between inflow and the recycled plane must be wide enough to ensure their independence, and the rescaling should take into account that the turbulent motion involves multiple scales, not only boundary layer thickness. In these simulations, the streamwise direction is non-periodic and a finite-difference scheme has to be used. While this approach permits the simulation to start at almost any given value of $Re_\theta$, the recycling process introduces an artificial inflow. All the scales have to evolve until they reach their asymptotic state; hence a portion of the simulation domain has to be discarded. A discussion about this accommodation length scale can be found in [13].

Compared to the previous related simulations, this code is focused on achieving the highest Reynolds number possible with the given computational resources. For example, the target for the smooth-wall case was to reach a friction Reynolds number $Re_\tau = 2000$, so that it could be compared with an existing simulation [14]. That comparison introduces additional constraints regarding box size and resolution. At such $Re_\tau$ the flow is fully turbulent, there is no need to simulate the transition, and the recycling scheme of the previous implementation is kept.

Another key difference between the current code and the previous ones is that, despite running efficiently on any distributed memory supercomputer, it was tuned for a specific supercomputing architecture that imposes severe constraints on domain decomposition, communications and I/O.

It was also our intention to design an application as flexible as possible that was able to generate data sets at even higher $Re_\tau$ without introducing further design modifications in the next generation of supercomputers. Therefore, one of our goals is also to share implementation details that can be useful to design similar large-scale simulations.

The organization of the paper is as follows: A basic description of the code is given in Section 2, followed in Sections 2.1–2.5 by the most relevant modifications to its previous version. Scalability is addressed in Section 3; and parallel Input/Output, a new feature, is commented in Section 4. Finally, validation and conclusions are in Sections 5 and 6 respectively.

## 2. The numerical code

The boundary layer is simulated in a parallelepiped over a smooth wall, with spatially periodic boundary conditions in the spanwise direction, but with non-periodic inflow and outflow in the streamwise direction. The code uses a well-established fractional-step method [15,16] to solve the incompressible Navier–Stokes equations expressed in primitive variables, using spectral expansions in the spanwise direction, and compact finite differences [17] in the other two. A three sub-step, semi-implicit low storage Runge–Kutta scheme, in which wall-normal second derivative terms use a Crank–Nicholson scheme to increase the time step, is used to evolve the equations in time. A full description of the algorithm can be found in [2].

For the problem considered, spatial derivatives are tightly coupled operations. Our code is constructed in such a way that only single data lines along one of the coordinate directions at a time have to be accessed globally. However, all the three directions have to be treated in every sub-step.

The code is single precision in the I/O operations and communications and double precision in the differentiation and interpolation operations where the implicit part of the compact finite differences and the fast Fourier transform can cause loss of significance.

### 2.1. Computational setup

A schema of the computational domain can be seen in Fig. 1. The $x$, $y$, and $z$ axes correspond to the streamwise, wall-normal and spanwise directions, respectively. The simulation is split in two concatenated domains with different boundary conditions. The planes $\pi_i$ and $\pi_i'$ are given inflow boundary conditions, and outflow boundary conditions are assigned to $\pi_e$ and $\pi_e'$. The boundary conditions at the top of the boxes, $\pi_t$ and $\pi_t'$, impose a zero pressure gradient on the domain. Finally, the spanwise direction is considered periodic. The purpose of the first boundary layer ($BL_1$) is to provide accurate inflow boundary conditions to the second one ($BL_2$). The inflow of $BL_1$ is obtained from its own plane $\pi_1$ that is rescaled using a method based on the one proposed by [11]. The physical length of $BL_1$ is chosen to be long enough to let the large scales recover from an unrealistic initial condition, and once this asymptotic state has been reached, the plane $\pi_2$ is used to give $BL_2$ its inflow boundary condition. As a consequence, a small portion of the $BL_1$ simulation is thrown away.

Given that the goal of $BL_1$ is to allow the large scales to reach their asymptotic state and, given that the smaller scales reach a similar condition far more rapidly, $BL_1$ is run at a coarser resolution than $BL_2$. This setup permits computing a single boundary layer with significantly less computational work.

The separation between adjacent collocation points is determined by the resolution of the spatial discretization scheme and the local Kolmogorov scale. This scale changes depending on the distance to the wall, so using a non-uniform mesh in the wall

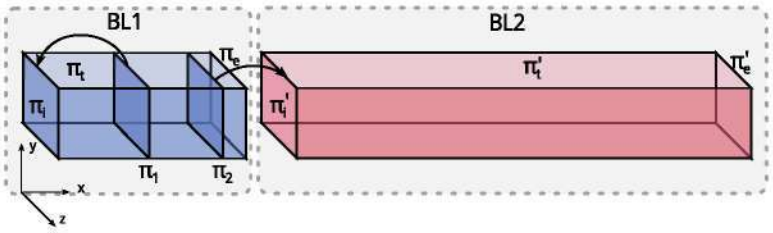


**Fig. 1.** Schema of the computational domain and boundary conditions.


**Fig. 2.** Elemental domains of the domain decomposition.

**Table 1**
Computational setup for the auxiliary $BL_1$ and main $BL_2$ boundary layers: $N_t$ is the total number of degree of freedoms in giga points; Time/DoF is the amount of total CPU (core) time spent to compute a degree of freedom for every step.

| Case | $Re_\theta$ | Nodes | $N_x \times N_y \times N_z$ | $N_t$ ($Gp$) | Time/DoF |
|---|---|---|---|---|---|
| $BL_1$ | 1100–3000 | 512 | $3585 \times 315 \times 2560$ | 2.89 | 13.98 μs |
| $BL_2$ | 2800–6650 | 7680 | $15{,}361 \times 535 \times 4096$ | 33.66 | 18.01 μs |

normal direction is essential to save memory. To achieve a Reynolds number based on the friction velocity of up to $Re_\tau = 2000$ a computational box for the second domain of size $15{,}360 \times 535 \times 4096$ for a total of $35 \times 10^9$ points per variable.

### 2.2. Domain decomposition and MPI communications

To take advantage of the distributed memory architectures, the computational domain must be partitioned. The only possible decomposition that guaranteed portability to the Blue Gene/P architecture was to use cross-stream planes schematized in Fig. 2 as $\pi_i$.

To compute interpolations and derivatives over the x coordinate it is necessary to transpose the whole variable. This operation creates another elemental domain partition formed by lines in the streamwise direction, labeled in the Fig. 2 as $\varpi_i$. Once these computations are finished the result is transposed back to planes $\pi_i$. A more traditional plane-to-plane transpose would be much simpler but it is not possible on the present supercomputer architecture. The low available amount of memory per node and the need for a large computational domain mandate that no essential domain decomposition based on planes that includes the streamwise direction can be stored as a whole. The $\varpi_i$ pencils can be considered as a secondary partition of such plane.

Each of these two boundary layers is mapped to an MPI group. The first group runs the auxiliary simulation at coarse resolution and it consists of 512 nodes while the second MPI group comprises 7680 nodes and runs the main one in high resolution. The first MPI group is only about 8.5% of the total computational cost. This information is shown in Table 1.

The two computational domains communicate with each other only twice per sub-step, to send the $\pi_2$ plane from $BL_1$ to $BL_2$ and to synchronize the time step, using an additional MPI group that includes all the processes.

The work done by each group must be balanced since each MPI group must wait for the other one in global operations, otherwise one group will slow down the other one that must remain idle waiting for the other group. The worst-case scenario occurs when the auxiliary simulation slows down the main one. The time taken by communication for the auxiliary simulation has been improved using a customized node topology described in Section 2.4.

### 2.3. Global transposes and collective communications

Roughly 45% of the overall execution time is spent transposing the variables from planes to pencils and back; therefore, it was mandatory to optimize the global transpose as much as possible. Preliminary tests revealed that the most suitable communication strategy was to use the MPI_ALLTOALLV routine and the BG/P torus network. This method is twice as fast than our previous custom transpose routine based on point-to-point communication over the same network implemented in [2].

The global transpose is split into three sub-steps. The first one changes the alignment of the buffer containing a variable and casts the data from double to single precision to reduce the amount of information to be communicated. If more than one $\pi$ plane is stored in every node then the buffer comprises the portion of contiguous data belonging to that node in order to keep message sizes as big as possible.

The second sub-step is a call to the MPI_ALLTOALLV routine. In this case the possibility of performing collective communications with derived datatypes would simplify the algorithm, but unfortunately it is not a feature of the present MPI standard. This is the reason why the global transpose is split into three sub-steps.

The third and last sub-step transpose the resulting buffer aligning the data $\varpi$-wise. This last transpose has been optimized using a blocking strategy because the array to be transposed has many times more rows than columns. The whole array is split into smaller and squarer arrays that are transposed separately. The aspect ratio of those smaller arrays is optimized for cache performance using collected data from a series of tests. Finally the data is cast to double precision again.

The procedure to transpose from $\varpi_i$ pencils to $\pi_i$ planes is similar and is split in three sub-steps too.

### 2.4. Blue Gene/P node mapping

Mapping virtual processes onto physical processors is one of the essential issues in parallel computing, a field of intense study in the last decade. Proper mapping is critical to achieve sustainable and scalable performance in modern supercomputing systems.

Blue Gene/P has a torus network topology except for allocations smaller than 512 nodes, in which the torus degenerates to a mesh. Therefore, each node is connected to six nodes by a direct link. The location of a node within the torus can be described by three coordinates $[X, Y, Z]$.

Different physical layouts of MPI tasks onto physical processors are predefined depending of the number of nodes to be allocated. The predefined mapping for a 512 node partition is a [8, 8, 8] topology, while for 8192 nodes it is [8, 32, 32] as it is shown in Fig. 3.

Changing the node topology completely changes the graph embedding problem and the path in which the MPI message travels. This can increase or decrease the number of hops needed to connect one node to another, and as a result, alter the communication time to send a message. Fine tuning for specific problems can considerably improve the time spent in communications. Table 2 shows different mappings that have been evaluated for our specific problem size. The custom mapping reduces the communication

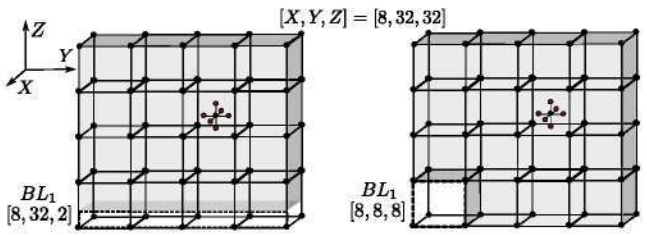


**Fig. 3.** Predefined (left) and custom (right) node mapping for a 8192 node partition in a [8, 32, 32] topology. The predefined mapping assigns to $BL_1$ the nodes in a [8, 32, 2] sub-domain. Custom mapping assigns the nodes to a [8, 8, 8] sub-domain. $BL_2$ is mapped to the rest of the domain till complete the partition.

**Table 2**
Time spent in communication during global transposes. Different node topologies are presented for 10 time steps and for each boundary layer. Times are given in seconds.

| Topology | Nodes | Comm. $BL_1$ | Comm. $BL_2$ |
|---|---|---|---|
| Predefined [8, 8, 8] | 512 | 27.77 | – |
| Custom [32, 32, 8] | 8192 | 79.59 | 86.09 |
| Predefined [32, 32, 8] | 8192 | 160.22 | 85.44 |

time for $BL_1$ by a factor of two. The work load for $BL_1$ is estimated using this new communication time while the load for $BL_2$ is fixed. Balance is achieved minimizing the time in which $BL_1$ or $BL_2$ are idle in the global communications.

The choice of a user-defined mapping is motivated due to the particular distribution of nodes and MPI groups. The first boundary layer $BL_1$ runs in 512 MPI processes mapped onto the first 512 nodes, while $BL_2$ runs in 7680 MPI processes mapped onto the nodes ranging form 513 to 8192. The optimum topology for our particular problem would be the one in which the number of hops for each MPI group is minimum since collective communications occur locally for each group. For a single 512 node partitions the optimum is the use of [8, 8, 8] topology, in which messages travel within a single communication switch. We have found the optimum mapping for $BL_1$ to be [8, 8, 8] sub-domain within the predefined [8, 32, 32], as shown in the right side of Fig. 3. $BL_2$ is mapped to the remaining nodes using the predefined topology and no other mappings have been further tested. Although a [8, 8, 8] topology is used for $BL_1$ by analogy with the single 512 node partition, communication time is nevertheless greater. This is due to the sub-optimal performance of using a 2D mesh instead of a 3D torus network, as already discussed. Finally, the reason can be found in the new hardware connection, since the 512 nodes and 8192 nodes of the 3D torus network are physically connected in a different way. This leads to the increase in the number of hops for $BL_1$ collective communications, since messages cannot travel within a single communication switch anymore.

The methodology to optimize communications for another size partitions would be similar to the one just described: mapping virtual processes to nodes that are physically as close as possible so the number of hops is minimized.

### 2.5. The hybrid MPI-OpenMP approach

Introducing OpenMP adds a second domain decomposition to the $\pi_i$ and $\varpi_i$ used for MPI. The most important non trivial uses of OpenMP are the parallelization of the compact finite differences operators for wall-normal derivatives, that require a tridiagonal solver, and the Fast Fourier Transforms in the spanwise direction. While threaded versions of both band-diagonal solvers and FFT exist, our decision was to handle the OpenMP parallel regions by hand to ensure portability between the different available platforms.

It is important to state that the reason to mix concurrency and parallelism was not driven by the need for more performance but because the small memory capacity of the Blue Gene/P node, which does not allow a physically-significant block of data to be allocated to each core. For instance, in the forced boundary layer case, a single $\pi$ plane is stored in every node that has been assigned to $BL_2$. While very special attention was payed to the collective transpose, that takes almost half of the runtime, the goal of using OpenMP was to use all the available resources of the node. Once we achieved the required scalability and performance, no further tuning was explored.

Some tests were run in a 512 node configuration after porting the code to OpenMP. The results are shown in Table 3. These samples suggest that almost no penalty is paid when the computations are parallelized with OpenMP.

**Table 3**
OpenMP scalability test performed on 512 nodes. Two efficiencies are given: $E$ is based on the computation time only (*Comp. T.*) and $\eta$ is based on the total time per step (*Total T.*) and is lower given that only one of the OpenMP threads is able to transfer data to other processes. Times are given in seconds.

| $N_{threads}$ | *Comp. T* | $E$ | *Total T* | $\eta$ |
|---|---|---|---|---|
| 1 | 60.820 | 1 | 70.528 | 1 |
| 2 | 30.895 | 0.984 | 38.951 | 0.905 |
| 4 | 16.470 | 0.923 | 24.438 | 0.721 |

**Table 4**
Data collected from the profiled test cases. Time/*DoF* is the amount of total CPU (core) time spent to compute a degree of freedom for every step; $N_t$ is the size in GiB of a buffer of size $N_x \times N_y \times N_z$; Comm. is the percentage of the time spent on MPI communications respect the total.

| Nodes | $N_x \times N_y \times N_z$ | $N_t$ | Time/*DoF* | Comm. | Symbol |
|---|---|---|---|---|---|
| 512 | $1297 \times 331 \times 768$ | 0.33 | 10.6 µs | 17.9% | ▶ |
| 1024 | $3457 \times 646 \times 1536$ | 3.43 | 17.6 µs | 44.7% | ◀ |
| 2048 | $6145 \times 646 \times 1536$ | 6.10 | 17.4 µs | 46.0% | ▲ |
| 4096 | $8193 \times 711 \times 1536$ | 8.94 | 17.6 µs | 44.6% | ▼ |
| 8192 | $8193 \times 711 \times 2048$ | 11.93 | 19.4 µs | 37.4% | ♦ |
| 8192 | $16{,}385 \times 801 \times 4608$ | 60.47 | 19.3 µs | 39.7% | ■ |

## 3. Scalability

Extensive data about scalability was collected during the test runs in a BG/P system. The most relevant cases are listed in the Table 4.

All the simulations run keep a linear weak scaling up to 8192 nodes (32,768 cores). The same code is expected to scale further without modifications, although larger node partitions have been not tested yet.

Fig. 4b. Communication time is typically 40% of the total run time, and that both computation and communication are scaling as expected. The global transpose implementation shows excellent scalability in all the test cases shown in Fig. 4a. It is important to mention that, in the BG/P supercomputer architecture, the linear scaling is kept even when the estimated message size is about 1 kB in size. All our previous implementations of the global transpose in more conventional high performance networks broke the scalability near the 3 kB estimated message size limit.

## 4. Parallel I/O

Intermediate stages of the simulation in the form of flow fields (velocities and pressure) are an important result and are saved even more often than would be required for checkpointing. Another mandatory feature to maintain the scalability with a large node count is the support for parallel collective I/O operations when a parallel file system is available. A handful of alternatives have been tested on parallel file systems, such as the use of raw posix calls enforcing the file system block size, sionlib (developed at Juelich) and parallel HDF5 [19].

HDF5 is a more convenient choice for storing and distributing scientific data than the alternatives tested because, despite having better performance [18], they require translating the resulting files to a more useful format. Unfortunately, sufficient performance could not be achieved without tuning the I/O process. HDF5 performance depends on the availability of a cache in the file system. The

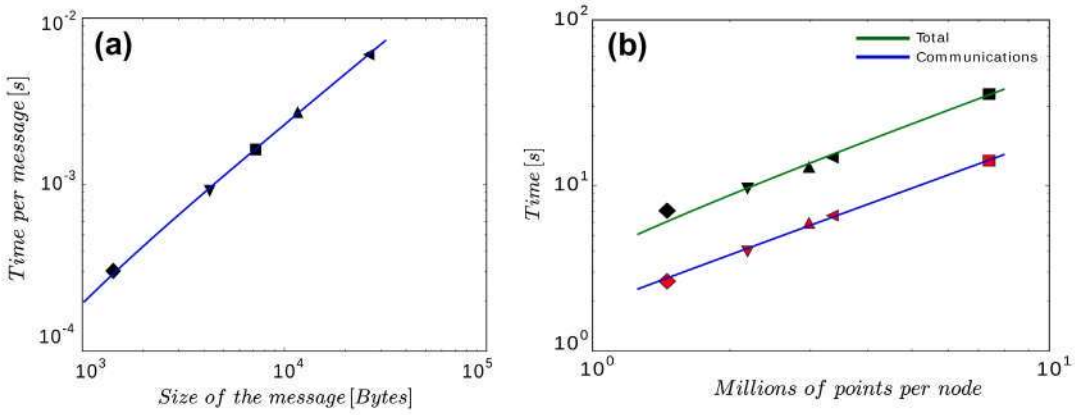


**Fig. 4.** Latency analysis (a) and scalability of the total and communication time for different test cases. (b) Solid lines are linear regressions computed before taking logarithms of both axis.

observed behavior in the BG/P systems was that writing was one, and sometimes two, orders of magnitude slower than reading because in the GPFS used the write cache was turned off. To overcome this issue, when the MPI I/O driver for HDF5 is used, the sieve buffer size parameter of HDF5 can be set to the file system block size. The resulting write bandwidth for 8192 nodes in the Jugene BG/P system was increased up to 16 GiB/s, which is similar to the read bandwidth 22 GiB/s and closer to the estimated maximum.

## 5. Validation

The numerical scheme is identical to the previous version of the code, which was appropriately validated in [2] and in [20], where it was also compared with other experiments and simulations at comparable Reynolds numbers. However, some basic one-point statistics are presented for the present high Reynolds number simulation, showing excellent agreement with numerical and experimental data sets too.

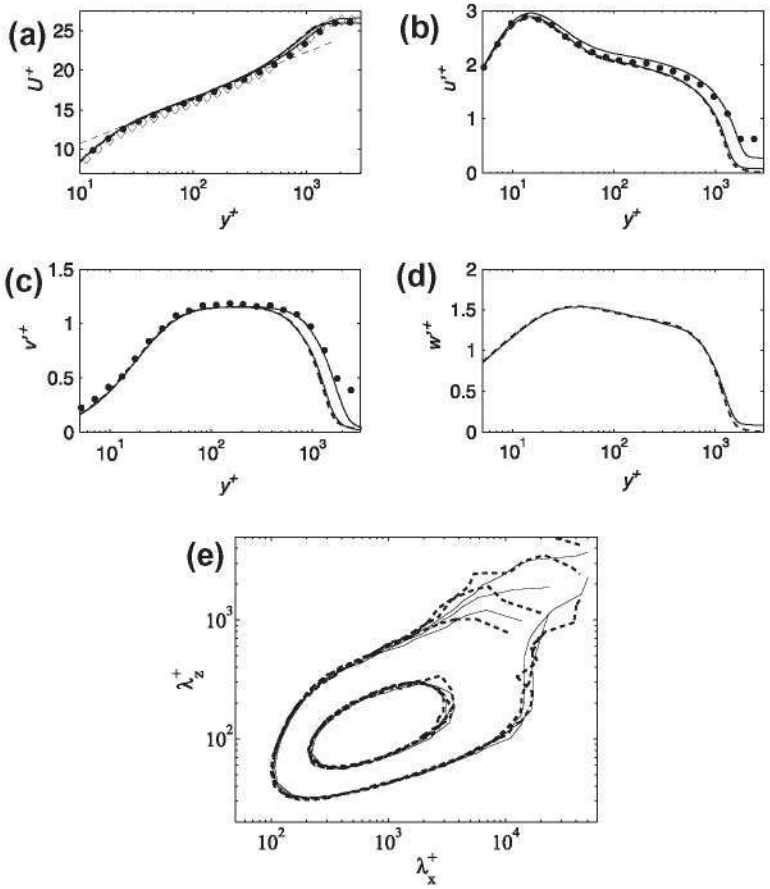


**Fig. 5.** Experiments by [21], •, $Re_\theta = 5261$; and [22], ♦, $Re_\theta = 5156$. Simulations by [8], - -, $Re_\theta = 4060$; present, —, $Re_\theta = 4060$, 5261. The law $\log(y^+)/0.4 + 5$ is the discontinuous straight line in (a). (e) Two-dimensional premultiplied energy spectra $k_x k_z E_{uu}(k)$ at three Reynolds numbers and 15 wall units height for channels [14] (solid) and boundary layers (dashed), present and [20].

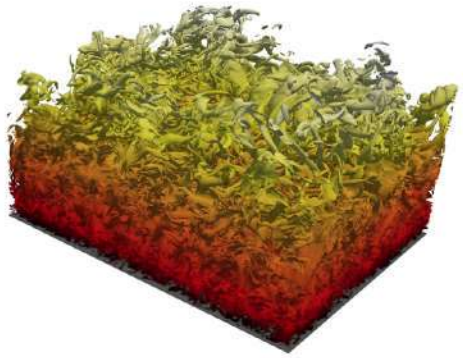

**Fig. 6.** Non dimensional enstrophy contour $|\omega^+| = 0.4$ at $Re_\theta = 6500$, $Re_\tau \simeq 2000$. The contour is colored with the distance to the wall. The stream goes from bottom-right to top-left. The box corresponds to a small portion of the simulation at the mentioned Reynolds number, approximately 1.1 boundary layer thickness wide and 1.5 boundary layer thickness long.

In Fig. 5a–d are shown the mean and fluctuations of the velocity profiles of the present simulation compared with other available experimental [21,22] and numerical [8] data sets for two different Reynolds numbers. The agreement is excellent.

Fig. 5e presents the premultiplied energy spectrum $k_x k_z E_{uu}$ where $k$ stands for the wavenumber $k = 2\pi/\lambda$ associated to the wavelength $\lambda$. Three different Reynolds numbers ($Re_\tau$ = 550, 980, and 2000) at height 15 wall units have been considered. It is a more complete check than one-point statistics because it shows the energy contained in eddies of any size at a given distance to the wall.

The inner contour are the wavelengths whose energy is the 54% of the most energetic modes, while the outer corresponds to the 14% of this peak. The energy spectrum contains information for all the scales, ranging from the smallest ones at the Kolmogorov scale, to the largest ones of the order of the edge of the boundary layer thickness. For example, the new simulation has enough resolution (fine mesh) to resolve the smallest energy-containing eddies as well as the largest ones where the energy reside. Close to the wall both channels and boundary layers (including the new simulation) are very similar at small scales, where they are Reynolds independent. The footprint of the largest structures, which depends on the Reynolds number, appears in the spectrum as a handle in the top-right corner, and the length of those eddies increases with the Reynolds number (see Fig. 6).

## 6. Conclusions and future work

A hybrid OpenMP-MPI code has been developed from its original MPI version to perform direct numerical simulations of boundary layers over smooth walls at high Reynolds numbers. The code has been tested in a Blue Gene/P computer using up to 8192 MPI processes, and four threads per process for OpenMP, showing good scalability for both MPI and OpenMP.

Some of the changes were necessary because of the architecture, like hybrid parallelism, all-collective communications and parallel I/O. Others were introduced to correct the somehow unpredictable influence of the inflow boundary conditions at large Reynolds numbers in turbulent boundary layers. This coupled the problem of defining the simulation and tuning the code. The solution here presented is the result at the end of this process.

The simultaneous use of OpenMP and MPI was relatively straightforward in our case, and is becoming a common feature in modern scalable codes. Collective communications are a similar case; once the global transposed was modified according to the suggestions of the Blue Gene handbook and system administrators, performance and scalability were improved significantly.

The approach of simulating two different computational domains, each at a different resolution, has proven to be effective and can be used in other spatially developing turbulent flows. However, it became an issue for our communications scheme. The solution was to separate the auxiliary low-resolution and the main high-resolution simulation in two different MPI groups and to define a customized mapping of processes onto physical processors. While this particular kind of tuning is not necessary when the process count is low, it is crucial when one is using thousands of nodes over torus networks.

Parallel I/O had a large impact too, despite its performance changes depending on the particular hardware configuration of the platform.

At the time of publication of this paper two simulations using it have been successfully completed, each one producing valuable data for the study of wall bounded turbulence and boundary layers in particular.

Some features of this new code are considered mandatory for the new generation of supercomputers. We hope that this experience can be a guideline for porting similar codes. Some implementation details that are described are particular to the Blue Gene/P and their applicability to other present supercomputing architectures is arguable. However, it is probable that the next generation of supercomputers will share important architectural characteristics with the BG/P and that some features of this implementation are mandatory to push the Reynolds number limit further. This code is designed to generate large datasets. A snapshot of the bigger case, that achieves $Re_\tau$ = 2000, requires approximately 0.6 TiB of disk space. Post-processing this kind of result is a challenge on itself, for instance, to generate the enstrophy contour of Fig. 6, a visualization cluster had to be used.

## Acknowledgements

This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the US Department of Energy under contract DE-AC02-06CH11357. The resources in Juelich Forgschungszentrum were provided by the PRACE initiative. The work by Guillem Borrell was funded by CICYT under grant TRA2009-11498. Juan A. Sillero was supported by an FPU fellowship from the UPM. This work was also funded by Consolider CSD2007-00050. Fig. 6 was generated with the help of the Barcelona Supercomputing Center.

## References

[1] Spalart PR. Direct simulation of a turbulent boundary layer up to $Re_\theta$ = 1410. J Fluid Mech 1988;187:61–98.
[2] Simens MP, Jiménez J, Hoyas S, Mizuno Y. A high-resolution code for turbulent boundary layers. J Comput Phys 2009;228:4218–31.
[3] Sillero JA, Borrell G, Gungor AG, Jiménez J, Moser RD, Oliver TA. Direct simulation of the zero-pressure-gradient boundary layer up to $Re_\theta$ = 6000. In: Proceedings of the divisional fluid dynamics, EB-4. College Park, MD: Am. Phys. Soc.; 2010.
[4] Borrell G, Gungor AG, Jiménez J. Direct numerical simulation of a high-entrainment turbulent boundary layer. APS divisional fluid dynamics meeting. Baltimore USA, November 20–22 2011.
[5] Fischer PF, Lottes JW, Kerkemeier SG. nek5000 Web page; 2008. <http://nek5000.mcs.anl.gov>.
[6] Khurajadze J, Oberlack M. DNS and scaling laws from new symmetry groups of zpg turbulent boundary layer flow. Theor Comput Fluid Dynam 2004;18:391441.
[7] Wu X, Moin P. Direct numerical simulation of turbulence in a nominally zero-pressure-gradient flat-plate boundary layer. J Fluid Mech 2009;630:5–41.
[8] Schlatter P, Örlü R. Assessment of direct numerical simulation data of turbulent boundary layers. J Fluid Mech 2010;695:116–26.
[9] Lee JH, Sung HJ. Direct numerical simulation of a turbulent boundary layer up to $Re_\theta$ = 2500. Int J Heat Fluid Flow 2011;32:1–10.

[10] Pirozzoli S, Bernardini M, Grasso F. Direct numerical simulation of transonic shock/boundary layer interaction under conditions of incipient separation. J Fluid Mech 2010;657:361–93.
[11] Lund TS, Wu X, Squires KD. Generation or turbulent inflow data for spatially-developing boundary layer simulations. J Comput Phys 1998;140:233–58.
[12] Ferrante A, Elghobashi S. A robust method for generating inflow conditions for direct simulations of spatially-developing turbulent boundary layers. J Comput Phys 2004;198:372–87.
[13] Sillero JA, Jimnez J, Moser RD, Malaya NP. Direct simulation of a zero-pressure-gradient turbulent boundary layer up to $Re_\theta = 6650$. J Phys: Conf Ser 2011;218:022023.
[14] Hoyas S, Jiménez J. Scaling of the velocity fluctuations in turbulent channels up to $Re_\tau = 2003$. Phys Fluids 2006;18:011702.
[15] Harlow FH, Welch JE. Numerical calculation of time dependent viscous incompressible flow of fluid with free surface. Phys Fluids 1965;8:2182.
[16] Perot JB. An analysis of the fractional step method. J Comput Phys 1993;108:51–8.
[17] Lele SK. Compact finite difference schemes with spectral-like resolution. J Comput Phys 1992;103:16–42.
[18] Frings W, Wolf F, Petkov V. Scalable massively parallel I/O to task-local files. In: Proceedings of the conference on high performance computational networking, storage and analysis, vol. 17; 2009. p. 17:1–11.
[19] The HDF Group. (2000–2010) Hierarchical data format version 5. <http://www.hdfgroup.org/HDF5>.
[20] Jiménez J, Hoyas S, Simens MP, Mizuno Y. Turbulent boundary layers and channels at moderate Reynolds numbers. J Fluid Mech 2010;657:335–60.
[21] De Graaf DB, Eaton JK. Reynolds number scaling of the flat-plate turbulent boundary layer. J Fluid Mech 2000;422:319–46.
[22] Österlund JM, Johansson AV, Nagib HM, Hites M. A note on the overlap region in turbulent boundary layers. Phys Fluids 2000;12:1–4.