Computers & Fluids xxx (2010) xxx–xxx

Contents lists available at ScienceDirect

Computers & Fluids

journal homepage: www.elsevier.com/locate/compfluid

Computational performance of a parallelized three-dimensional high-order spectral element toolbox

Christoph Bosshard [c], Roland Bouffanais [b], Michel Deville [a], Ralf Gruber [a], Jonas Latt [a,*]

[a] École Polytechnique Fédérale de Lausanne, STI–IGM–LIN, Station 9, 1015 Lausanne, Switzerland
[b] Massachusetts Institute of Technology, 77 Massachussetts Avenue, Bldg 5–326, Cambridge, MA 02139, USA
[c] Paul Scherrer Institut, Laboratory for Thermalhydraulics (LTH), 5232 Villigen PSI, Switzerland

ARTICLE INFO

Article history:
Received 30 November 2009
Received in revised form 16 September 2010
Accepted 18 November 2010
Available online xxxx

Keywords:
CFD
High-order method
Spectral element method
HPC
Parallelism

ABSTRACT

In this paper, a comprehensive performance review of an MPI-based high-order three-dimensional spectral element method C++ toolbox is presented. The focus is put on the performance evaluation of several aspects with a particular emphasis on the parallel efficiency. The performance evaluation is analyzed with the help of a time prediction model based on a parameterization of the application and the hardware resources. Two tailor-made benchmark cases in computational fluid dynamics (CFD) are introduced and used to carry out this review, stressing the particular interest for clusters with up to thousands of cores. Some problems in the parallel implementation have been detected and corrected. The theoretical complexities with respect to the number of elements, to the polynomial degree, and to communication needs are correctly reproduced. It is concluded that this type of code has a nearly perfect speedup on machines with thousands of cores, and is ready to make the step to next-generation petaFLOP machines.

1. Introduction

The *Spec*tral *u*nstructured E*L*ements *O*bject-*O*riented *S*ystem (SpecuLOOS) is a toolbox written in C++ [1], which is available free of charge under the terms of an open-source software license. SpecuLOOS is a spectral and mortar element analysis software for the numerical solution of partial differential equations and more particularly for solving incompressible unsteady fluid flow problems. The main architecture choices and the parallel implementation were elaborated and implemented by Van Kemenade and Dubois-Pèlerin, and published in [2]. Subsequently, SpecuLOOS' C++ code has been further developed, see [3].

It is well known that spectral element methods [4] are easily amenable to parallelization, as the domain decomposition into spectral elements can be made to correspond in a natural way to an attribution to parallel nodes [7].

The ongoing simulations based on SpecuLOOS highlight the achieved versatility and flexibility of this C++ toolbox. Nevertheless, 10 years have passed between the first version of SpecuLOOS' code and the present time and tremendous changes have occurred at both hardware and software levels. SpecuLOOS correspondingly improved to incorporate the benefits of fast dual DDR memory, RISC architectures, 64-bit memory addressing, compilers improvement, libraries optimization, libraries parallelization, and increase in inter-connecting switch performance.

Here we discuss adaptation of SpecuLOOS to thousands of multi-core nodes. Performance measurements on a one-core node, on an Intel-based cluster with hundreds of nodes, and on a 2048 quad-core nodes of a Blue Gene/P (BG/P) are presented. The obtained complexities are compared with theoretical predictions. Initial results show that small cases gave good complexities, but very large cases gave poor efficiencies. These results led to the detection of a poor parallel implementation which limited the usage of the code to approximately 1000 cores. As it is pointed out in [15], this threshold value of 1000 cores is frequently observed and puts often a limit to the scalability of modern software in the domain of computational fluid dynamics. Once this problem was corrected, the complexity of SpecuLOOS could however be adjusted and corresponds now to the theoretical one up to the 8192 cores used.

To our knowledge, there exist two other spectral-element-based codes for computational fluid dynamics which, like SpecuLOOS, are adapted to high performance computing and can be used under the terms of an open-source license. The first one, the Nektar code, is referenced in [8], and the second one, provided under the code-name Nek5000, is mentioned in [9].

Description of the test cases

The test cases belong to the field of CFD and consist in solving the 3D unsteady Navier–Stokes equations for a viscous Newtonian

* Corresponding author.
E-mail addresses: christoph.bosshard@epfl.ch (C. Bosshard), bouffana@mit.edu (R. Bouffanais), michel.deville@epfl.ch (M. Deville), ralf.gruber@epfl.ch (R. Gruber), jonas.latt@epfl.ch (J. Latt).

0045-7930/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.compfluid.2010.11.014

incompressible fluid. Based on the problem at hand, it is always physically rewarding to non-dimensionalize the governing Navier–Stokes equations which take the following general form:

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u} = -\nabla p + \frac{1}{\mathrm{Re}}\Delta\mathbf{u} + \mathbf{f}, \quad \forall(\mathbf{x},t)\in\Omega\times I, \tag{1}$$

$$\nabla\cdot\mathbf{u} = 0, \quad \forall(\mathbf{x},t)\in\Omega\times I, \tag{2}$$

where $\mathbf{u}$ is the velocity field, p the reduced pressure (normalized by the constant fluid density), $\mathbf{f}$ the body force per unit mass and Re the Reynolds number

$$\mathrm{Re} = \frac{UL}{\nu}, \tag{3}$$

expressed in terms of the characteristic length L, the characteristic velocity U, and the constant kinematic viscosity ν. The symbol Ω denotes the computational domain. The evolution of the system is studied in the time interval $I = [t_0, T]$. Considering particular flows, the governing Navier–Stokes equations (1) and (2) are supplemented with appropriate boundary conditions for the fluid velocity $\mathbf{u}$ and/or for the local stress at the boundary. For time-dependent problems, a given divergence-free velocity field is required as initial condition in the internal fluid domain.

The first test case corresponds to the fully three-dimensional simulation of the flow enclosed in a lid-driven cubical cavity. The cavity is cubical and imposes no-slip walls, including for the top wall which is driven by a constant velocity, tangential to the surface. This numerical setup offers an interesting context for the study of internal flows, and it offers a rich physical content. The simulations are run at a Reynolds number of 12,000 that corresponds to a locally-turbulent regime. This corresponds to the case denoted under-resolved DNS (UDNS) in Bouffanais et al. [10]. The reader is referred to Bouffanais et al. [10] for full details on the numerical method and on the parameters used.

The second test case is the differentially heated cavity problem. Like for the first test case, the flow is confined in a cubical cavity. The fluid is heated over one vertical wall and cooled over its opposite wall at equal rates. For this case the flow is described by the Boussinesq equations:

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u} = -\nabla p + \frac{\mathrm{Pr}}{\sqrt{\mathrm{Ra}}}\Delta\mathbf{u} - PrT\frac{\mathbf{g}}{|\mathbf{g}|}, \quad \forall(\mathbf{x},t)\in\Omega\times I, \tag{4}$$

$$\nabla\cdot\mathbf{u} = 0, \quad \forall(\mathbf{x},t)\in\Omega\times I, \tag{5}$$

$$\frac{\partial T}{\partial t} + \mathbf{u}\cdot\nabla T = \frac{1}{\sqrt{Ra}}\triangle T, \quad \forall(\mathbf{x},t)\in\Omega\times I, \tag{6}$$

where T is the nondimensional temperature, Pr the Prandtl number, $\mathbf{g}$ the gravity vector and Ra the Rayleigh number

$$Ra = \frac{\beta g L^3 \Delta T}{\nu\alpha}. \tag{7}$$

In the last equation β is the thermal expansion coefficient, g is the gravitational constant, ΔT is the temperature difference between the hot and the cold wall and α is the thermal diffusivity. The differentially heated cavity problem is one of the classical heat and mass transfer problems and has become a popular numerical benchmark. The simulations are run at a Rayleigh number of 10^9. For further explanations and details of the differentially heated cavity problem see for example [13].

For the complexity analysis in the Sections 3 and 4 we used the lid-driven cavity as the test case. The benchmarks in Section 5 have been done for both test cases.

2. Legendre spectral element method

2.1. Space and time discretizations

The spectral element method is based on a weak formulation of the Navier–Stokes equations, with a choice of test and trial functions which are element-wise Lagrangian interpolation polynomials based on a Legendre grid [4]. The Navier–Stokes equations in their strong formulation are first expressed in a general weak form, which is also used in the context of finite elements. The solution for the velocity $\mathbf{u}$ for the Navier–Stokes problem in d-dimensional space is sought directly in the Sobolev space of test (resp. trial) functions $H_0^1(\Omega)^d$ (resp. $H^1(\Omega)^d$), defined as the space of differentiable vector functions having their first-order partial derivatives with respect the space of square integrable functions $L^2(\Omega)$ and vanishing on the domain boundary $\partial\Omega$ (resp. except on the top lid wall as in [10]), and where $d = 2, 3$ denotes the space dimension. The weak transient formulation reads as follows:

Find a solution $(\mathbf{u}(t), p(t)) \in H^1(\Omega)^d \times L^2(\Omega)$ *such that for almost every time* t*:*

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_\Omega \mathbf{u}\cdot\mathbf{v}\,\mathrm{d}\Omega + \int_\Omega (\mathbf{u}\cdot\nabla\mathbf{u})\mathbf{v}\,\mathrm{d}\Omega = \int_\Omega p\nabla\cdot\mathbf{v} - \frac{1}{\mathrm{Re}}\nabla\mathbf{u} : \nabla\mathbf{v}\,\mathrm{d}\Omega + \int_\Omega \mathbf{f}\cdot\mathbf{v}\,\mathrm{d}\Omega, \quad \forall\mathbf{v}\in H_0^1(\Omega)^d$$

and

$$-\int_\Omega q\nabla\cdot\mathbf{u}\,\mathrm{d}\Omega = 0, \quad \forall q\in L^2(\Omega), \tag{8}$$

The first step in the SEM discretization consists in subdividing the fluid domain $\bar{\Omega} = \Omega\cup\partial\Omega$ into E non-overlapping elements $\{\Omega^e\}_{e=1}^E$. Each element Ω^e involves a mesh constructed as a tensor product of one-dimensional grids. Although each space direction may be discretized independently of the others, for the sake of simplicity and without loss of generality, we consider in this paper only meshes obtained with the same number of nodes in each direction, within each element, denoted by $N+1$, and corresponding to the dimension of the space of Nth-order polynomials. In the following, we define the spaces

$$X \equiv H_0^1(\Omega)^d, \quad Z \equiv L^2(\Omega), \tag{9}$$

and apply the Galerkin approximation, in which finite dimensional polynomial sub-spaces X_N and Z_N are selected to represent X and Z respectively. In practice, some restrictions occur as far as the selection of polynomial degrees is concerned. In particular spurious oscillation phenomena in the pressure are avoided by means of a staggered-grid approach with elements based on sub-spaces of polynomial degree N and $N-2$ respectively for the velocity and the pressure field. With this formalism, the term $\frac{\mathrm{d}}{\mathrm{d}t}\int_\Omega \mathbf{u}\cdot\mathbf{v}\,d\Omega$ in the weak Navier–Stokes formulation is for example replaced by its semi-discrete analog $\frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{u}_N, \mathbf{v}_N)$, where the bracket denotes a scalar product within the corresponding polynomial subspace:

$$(\mathbf{u}_N, \mathbf{v}_N) = \sum_{e=1}^{E}\int_{\Omega^e}\mathbf{u}_N\cdot\mathbf{v}_N\,\mathrm{d}\Omega, \quad \forall\mathbf{v}_N\in X_N. \tag{10}$$

The functions $\mathbf{u}_N$ an $\mathbf{v}_N$ are approximated by Lagrangian interpolation polynomials of degree N based on a Legendre grid [4].

The integrals within each of the spectral elements $\{\Omega^e\}_{e=1}^E$ are calculated in a discrete manner using Gaussian quadrature rules. More specifically, all terms in the momentum equations are integrated using a Gauss–Lobatto–Legendre (GLL) quadrature rule, except for the pressure term. This term, as well as the divergence-free equation, are solved with help of a Gauss–Legendre (GL) quadrature rule. In order to formulate the semi-discrete version of the Navier–Stokes problem, the variables are approximated with

Please cite this article in press as: Bosshard C et al. Computational performance of a parallelized three-dimensional high-order spectral element toolbox. Comput Fluids (2010), doi:10.1016/j.compfluid.2010.11.014

expressions involving Lagrangian interpolation polynomials, with collocation points identical to the quadrature points. The SEM uses two tensor-product bases on the reference element $\widehat{\Omega} \equiv [-1,1]^d$. The semi-discrete equations are then expressed as

$$\mathbf{M}\frac{\mathrm{d}\underline{\mathbf{u}}}{\mathrm{d}t} = -\mathbf{A}\underline{\mathbf{u}} - \mathbf{C}\underline{\mathbf{u}} + \mathbf{D}^T\underline{p} + \mathbf{M}\underline{\mathbf{f}}, \tag{11}$$

$$-\mathbf{D}\underline{\mathbf{u}} = 0. \tag{12}$$

The operators appearing in these equations are: **M** the diagonal mass matrix, **A** the stiffness matrix, **C** the discrete convective operator, $\mathbf{D}^T$ the discrete gradient operator and **D** the discrete divergence. These matrices are all composed of three blocks associated with the discretization in each space direction. For instance, the diagonal mass matrix **M** is composed of d blocks, namely the mass matrices M which are identical when considering the same polynomial degree in the d space directions. The variables $\underline{\mathbf{u}}$ and $\underline{p}$ represent the degrees of freedom of the numerical model. They have a size of $(N+1)^d E$ variables in the former case, and $(N-1)^d E$ variables in the latter case. It is worth adding that nodal values for the velocity field, $\underline{\mathbf{u}}$, on subdomain interface boundaries are stored redundantly on each processor corresponding to the spectral elements having this interface in common. This approach is consistent with the element-based storage scheme which minimizes the inter-processor communications. The details of the parallelization of this scheme are provided in Section 8 of [4].

To discretize in time, we use a backward differentiation formula of order 2 (BDF2) for the Stokes operator and an extrapolation scheme of order 2 (EX2) for nonlinear terms. The resulting set of equations is solved via a generalized block decomposition with pressure correction (see chapter 5 in [4]), which can be summarized by the following steps (Δt represents the time step):

1. Computation of the tentative velocity vector $\underline{\mathbf{u}}^*$ by solving
$$\mathbf{H}\underline{\mathbf{u}}^* = \mathbf{D}^T\underline{p}^n + \frac{\mathbf{M}}{\Delta t}\left(2\underline{\mathbf{u}}^n - \frac{1}{2}\underline{\mathbf{u}}^{n-1}\right) + \mathbf{M}\mathbf{f}^{n+1} - (2\mathbf{C}\underline{\mathbf{u}}^n - \mathbf{C}\underline{\mathbf{u}}^{n-1}) \tag{13}$$
using the old time-level pressure $\underline{p}^n$, and defining the Helmholtz operator $\mathbf{H} = 3\mathbf{M}/(2\Delta t) + \mathbf{A}$.
2. Computation of the pressure at the new time-level by solving the following problem for the pressure correction $\delta\underline{p}^{n+1} = \underline{p}^{n+1} - \underline{p}^n$:
$$-\mathbf{D}\mathbf{Q}\mathbf{D}^T\delta\underline{p}^{n+1} = \frac{2}{\Delta t}\mathbf{D}\underline{\mathbf{u}}^*. \tag{14}$$
3. Computation of the final velocity at the time-level $n+1$ after a pressure correction
$$\underline{\mathbf{u}}^{n+1} = \underline{\mathbf{u}}^* + \mathbf{Q}\mathbf{D}^T\delta\underline{p}^{n+1}. \tag{15}$$

To entirely decouple the pressure from the velocity-field calculation, the matrix **Q** is an approximation of the inverse of the Helmholtz operator, and in this case is chosen as:

$$\mathbf{Q} = \frac{2\Delta t}{3}\mathbf{M}^{-1}. \tag{16}$$

In the case of the Boussinesq equations, the discretization of the temperature Eq. (6) is similar to the discretization of the momentum Eq. (1). The temperature is approximated by a Lagrangian interpolation polynomial of degree N with the same collocation points as for the velocity. The semi-discrete temperature equation is given by

$$\mathbf{M}\frac{\mathrm{d}\underline{T}}{\mathrm{d}t} = -\mathbf{A}\underline{T} - \mathbf{C}\underline{T}, \tag{17}$$

where **M**, **A**, and **C** are the mass matrix, the stiffness matrix and the convective operator respectively. To discretize in time, we use a backward differentiation formula of order 2 (BDF2) for the diffusion term and an extrapolation of order 2 (EX2) for the nonlinear convection term.

More information on this discretization method is provided in [5,6], which also provide a formal proof of the spectral order of accuracy of the method, for straight and for curved boundaries.

2.2. Computational complexity

The complexity of this algorithm is proportional to the total number of elements E in the three dimensional space. The geometric complexity of each physical element is handled by resorting to isoparametric mappings onto the unit cubic parent element $\widehat{\Omega} \equiv [-1,1]^d$. Since the Lagrangian interpolation basis functions based on the Gauss–Lobatto–Legendre grid points $\{\xi_j\}_{0\leqslant j\leqslant N}$ of degrees $N = N_x = N_y = N_z$, and defined as

$$h_j(r) = -\frac{1}{N(N+1)}\frac{1}{L_N(\xi_j)}\frac{(1-r^2)L_N'(r)}{(r-\xi_j)}, \quad -1 \leqslant r \leqslant +1, \quad 0 \leqslant j \leqslant N, \tag{18}$$

are orthonormal, the complexity for the velocity is $(N+1)^3$, while the complexity for the pressure is $(N-1)^3$. During the computations, the variables are frequently re-interpolated between the collocation nodes, an operation which has a leading complexity of N^4, due to the tensorization of the implied linear operations. At large values of N, these re-interpolations therefore dominate the total computation time. It should be remarked that from a complexity standpoint, a term like $(N-1)^3$ is equivalent to a term like $(N+1)^3$. In the following, a term $N-1$ has been applied systematically to read the complexity from experimental performance curves, while the notation of the equations is simplified by use of the term N.

The CPU time of the SpecuLOOS spectral code can then be estimated as

$$T(N_1, N_{\mathrm{CG}}, E, N) = a_1 N_1 N_{\mathrm{CG}} E^{a_2} N^{a_3}, \tag{19}$$

where N_{CG} is the number of conjugate gradient steps, N_1 is the number of time steps. The parameter a_1 represents the actual operations which are executed on an element at each CG iteration. It is independent on the simulated problem, but depends on the characteristics of the hardware, and is therefore found through simulation. The parameter a_3 has a leading complexity of 4, which appears with very large values of N. In practice, simulations are however run with relative modest polynomial degrees which are rarely higher than 12. In this case, benchmarks find a value of $3 < a_3 < 4$, which reflects the different complexity of the operations involved in the simulation process.

3. Complexity on one node

In the software SpecuLOOS, the linear equations involved in steps 1 and 2 of the procedure shown in the previous section are solved with help of an iterative conjugate gradient solver. The relative expense of solving the pressure problem (step 2) compared to the Helmholtz problem (step 1) increases when Δt is decreased. The choice of Δt is restricted by a Courant–Friedrichs–Lewy (CFL) condition in step 1,

$$\frac{u_{\max}\Delta t}{\Delta x_{\min}} \leqslant 1, \tag{20}$$

where $u_{\max}$ is the physical flow velocity and $\Delta x_{\min}$ the smallest distance between two grid points. In a GLL grid, the smallest distance between two collocation points is located close to the mesh boundary, and scales like $1/N^2$. Due to the high grid resolution, the time step is small in the present problem, and the computation of the pressure

Please cite this article in press as: Bosshard C et al. Computational performance of a parallelized three-dimensional high-order spectral element toolbox. Comput Fluids (2010), doi:10.1016/j.compfluid.2010.11.014

problem becomes dominant, taking as much as 90% of the total computation time. A complexity analysis of the code therefore concentrates on a complexity analysis of this part of the problem. It should be remarked however that a complexity analysis of the Helmholtz problem is very similar, and the overall conclusions are therefore the same in face of a problem with a larger discrete time step.

A time iteration of the solver is performed by executing the iterative algorithm of the conjugate gradient (CG) solver. At each CG iteration, the problem is preconditioned, after which the linear operators such as **D** or **Q** are evaluated. Thus, the total complexity of a time step is described by the leading complexity of either preconditioning the problem or evaluating the linear operators, whichever is more expensive, multiplied by the number of CG iterations.

For the case of the Boussinesq Eqs. (4)–(6), the linear system of equations coming from the discretized temperature equation is also solved with a conjugate gradient algorithm. This linear system is always solved in a few iterations independently from the problem size. Therefore, time spent in code for solving the temperature Eq. (6) is truly negligible for the complexity analysis.

All measurements shown in the current Section 3 are from the lid-driven cavity problem described in Section 1.

3.1. Number of CG iterations for the Helmholtz and the pressure problem

The number of conjugate gradient iterations needed to solve a linear problem depends on the quality of the preconditioning. With good preconditioning, it is observed that an optimal complexity is achieved, with a number of iterations which is proportional to both the polynomial degree N, and to the number of elements $E^{1/3}$ along a space direction.

3.2. Preconditioning of the pressure calculation

Two preconditioners are used in the following. The first preconditioner, referred to as the Couzy preconditioner [11], holds a full matrix representation of the preconditioning matrix. Applying this preconditioner therefore requires a number of operations proportional to N^6. Because of this unfavorable complexity, the Couzy preconditioner is a bad option for large values of the polynomial degree N. On the other hand, it dramatically reduces the number of required CG steps thanks to a high quality preconditioning, and is therefore a good option for modest values of N. The second preconditioner, referred to as the diagonal preconditioner, is represented by a diagonal matrix. It is therefore applied with an optimal complexity of N^3 and is an ideal choice at large values of N, although its quality of preconditioning is distinctly inferior to the abilities of the Couzy preconditioner. Typically, the Couzy preconditioner leads to an improved overall performance for $N < 12$.

3.3. Evaluation of the linear operators

During the computation of the pressure term, step 2 in the algorithm, the operators **D**, $\mathbf{D}^T$, and **Q** are evaluated. Furthermore, the variables of the vector $\underline{p}$ need to be re-interpolated from the GL to the GLL grid, to be combined with the other terms. All mentioned operators, as well as the operator responsible for reinterpolation, are expressed as tensor products of one-dimensional operators. It can be shown that such tensor-product operators can be applied to a vector with a leading complexity of N^4. This is much faster than applying a general linear operator in a space of dimension N^3, which would require an operational complexity N^6.

Table 1 presents the results of SpecuLOOS on one node of a home-grown cluster which consists of quad-core Xeon X3350 CPUs, running at 2.66 GHz, and with a Gigabit Ethernet interconnect [12]. The CPU time measurements, $T_{CG,(1\ iter)}$, for one CG iteration step ($N_1 = N_{CG} = 1$) are used to minimize $\sum(T - T_{CG,(1iter)})^2$, where

Table 1
SpecuLOOS on one node of an Intel Xeon cluster. The number of conjugate gradient iterations N_{CG} is an average value over all time steps for the pressure.

N_1	E	N	T_{exec} (s)	N_{CG} # iter	T_{CG} (s)	$\frac{T_{CG}}{T_{exec}}$	$T_{CG,(1iter)}$ (s)
1	256	8	40.1	198	32.8	0.818	0.17
1	256	10	119.3	247	103.8	0.870	0.42
1	512	6	43.2	205	33.9	0.785	0.17
1	512	8	116.4	268	106.7	0.917	0.40
1	512	10	394.3	344	342.3	0.868	1.00
1	1024	6	105.2	259	83.4	0.793	0.32
1	1024	8	311.0	339	265.4	0.853	0.78

$$T(N_{CG} = 1) = a_1 E^{a_2} N^{a_3}, \tag{21}$$

This minimization procedure gives the scaling law.

$$T(E, N) = 2.01 \cdot 10^{-6} E^{0.97} \times N^{3.3}. \tag{22}$$

One realizes that this complexity law corresponds well to the theoretical one, (Eq. (19)).

Generally, the number of iteration steps is not known. If in the optimization procedure one includes N_{CG} in the parameters E and N, the power law (19) becomes

$$T(N_1, E, N) = 1.15 \times 10^{-5} \cdot N_1 \cdot E^{1.30} \cdot N^{4.19}. \tag{23}$$

As a consequence, the estimated number of $N_{CG,est}$ is

$$N_{CG,est} = 5.72 \times E^{0.33} \cdot N^{0.89}. \tag{24}$$

This prediction is also close to the expected theoretical complexity $N_{CG,theo}$:

$$N_{CG,theo} \approx E^{\frac{1}{3}} \cdot N. \tag{25}$$

The same type of studies has been made for a diagonal preconditioner varying the polynomial degree. The complexity found is $N_{CG,\ est} \approx N^{1.47}$. Thus, for $N \geqslant 12$ the diagonal preconditioner is faster, while for $N < 12$ the Couzy preconditioner is faster. Since we treat cases with $N = 12$ or smaller, we concentrate on the Couzy preconditioner.

4. Wrong complexity on the Blue Gene/P

The SpecuLOOS code has been ported to the IBM Blue Gene/P machine at EPFL with 4096 quad-core nodes (for practical reasons, only 2048 nodes of the BG/P could be used). Table 2 presents the results obtained with the original version of the SpecuLOOS code, before important adaptations described below were performed. One element is running in a core. The polynomial degree is fixed to $N = 12$ for the velocity components, and to $N - 2 = 10$ for the pressure. The resulting complexity given by the pressure computation is illustrated in Table 2, and is identified as [12]

$$T \approx E^2. \tag{26}$$

This result shows that the complexity of the original parallel code is far from the $E^{1.3}$ law, which is expected for theoretical reasons, as shown in Eq. (23), and verified numerically in a serial program execution. The reasons for this bad result could be identified as follows. In fact, an "IF" conditional statement over all elements had been introduced in the code, in order to check the attribution of elements to computational nodes dynamically, at each iteration of the conjugate gradient method. Such an instruction is typical for rapid corrections in a code, which are made to parallelize a pro-

Please cite this article in press as: Bosshard C et al. Computational performance of a parallelized three-dimensional high-order spectral element toolbox. Comput Fluids (2010), doi:10.1016/j.compfluid.2010.11.014

Table 2
SpecuLOOS on the Blue Gene/L machine up to $P = 8192$ nodes. The number of elements per core has been fixed to one. The polynomial degree for the pressure is equal to $N - 2 = 10$.

N_1	$E = E_x E_y E_z$	N	P	$\frac{\#\text{elem}}{\text{node}}$	T_{exec}
1	$8 \times 8 \times 16$	12	1024	1	17.22
1	$8 \times 16 \times 16$	12	2048	1	29.91
1	$16 \times 16 \times 16$	12	4096	1	57.05
1	$16 \times 16 \times 32$	12	8192	1	140.50

gram rapidly without realizing its impact on future program executions. This did not affect the CPU time for less than 100 cores P (the acronym P stands for "processes"), but became dominant at a large number of cores $P > 1000$. This "IF" instruction has now been replaced by the use of a precomputed list, pointing to the elements which are active on a core.

5. Benchmarks

In the following benchmarks, the SpecuLOOS code is executed on two different parallel platforms, and the scaling laws which were derived theoretically in the previous section are verified in practice. Furthermore, the negative impact of inter-process communication on performance is quantified. In this way, it is possible predicting to which extent spectral element-based codes scale to very large problems, or to very large parallel machines, or both.

All benchmarks have been done for both test cases described in Section 1 with almost identical results. Unless stated otherwise in the text, only the results from the differentially heated cavity problem are shown in this section.

5.1. Scaling on constant-size problems

In this first benchmark case, a problem with a constant polynomial degree N and a constant number of elements E is chosen and executed on a varying number of cores. To clearly identify how the inter-process communication affects the efficiency of a code like SpecuLOOS, the ratio of the number of elements to the number of cores is kept as small as possible. Within the chosen range of cores, the number of elements is therefore chosen in such a way as to have only a few elements, or even just a single element per core when all cores are used. This is an untypical regime to use in high performance computing, because of the unfavorable efficiency figures which are usually obtained. One can however think of this as a typical use case in a near future, as it becomes interesting, with the advent of cheap many-core hardware, to run relatively small problems on many cores. To emphasize the challenging nature of such a benchmark, one speaks of a *strong scaling* for the behavior of constant-size problems with an increasing number of cores, while the case of a problem with linearly increasing size, as compared to the number of cores, is said to be benchmarked with *weak scaling*.

The program was first executed on the commodity cluster Pleiades2 at the EPFL, the hardware characteristics of which are presented in Section 3.3. The result is shown in Fig. 1, with a number of cores varying between 1 and 64. It is seen that while the speedup is excellent within the shared memory environment of a quad-core, it drops dramatically when inter-process communication comes into play. Such a performance figure is typical for measurements of strong scaling on commodity clusters. Nevertheless, the excellent speedup in a shared-memory environment allows optimistic predictions for a code like SpecuLOOS on future many-core platforms.

Next, the same program was executed on an IBM Blue Gene/P, on a number of cores ranging from 64 to 8192. It was impossible

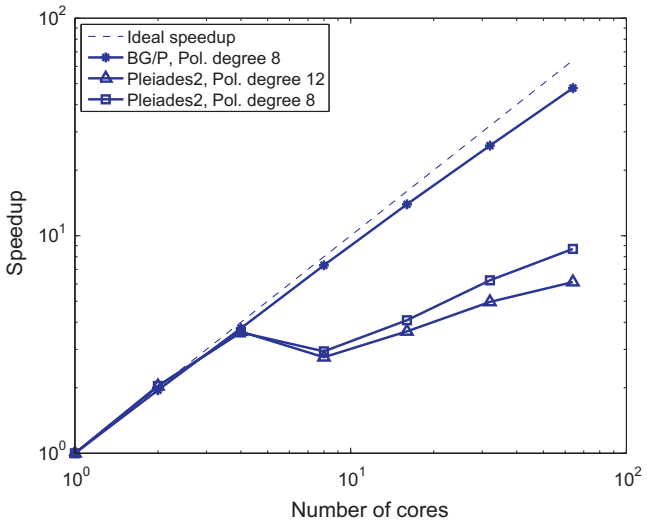

Fig. 1. Speedup of a constant-size problem on the BG/P and on the Pleiades2 commodity cluster, on a number of cores varying from 1 to 64. The number of elements is $4 \times 4 \times 4 = 64$. It is seen that the scaling is practically ideal on the BG/P. On Pleiades2, while the scaling is very good within the four cores of a quadri-core node, it is severely affected by the inter-node communication. The scaling improves when the polynomial degree is increased, because this increases the relative computational load compared to the communication time.

to use fewer cores, as the available memory was insufficient. The problem size ranges from an extreme regime of 8192 elements (which is identical to the maximal number of cores) to 32,768 elements (corresponding to four times the maximal number of cores). As it is seen in Fig. 2, the speedup is close to ideal at a usage of up to approximately 2000 cores. Beyond this value, a loss of performance due to inter-process communication becomes visible. However, in spite of the small problem size, the program scales up to 8192 cores. At 8192 cores, the parallel efficiency, measured as the ratio of the measured speedup over the ideal speedup, takes a value of 34% with 8192 elements, 50% with 16,384 elements, and 65% with 32,768 elements. It should be emphasized that such a good perfor-

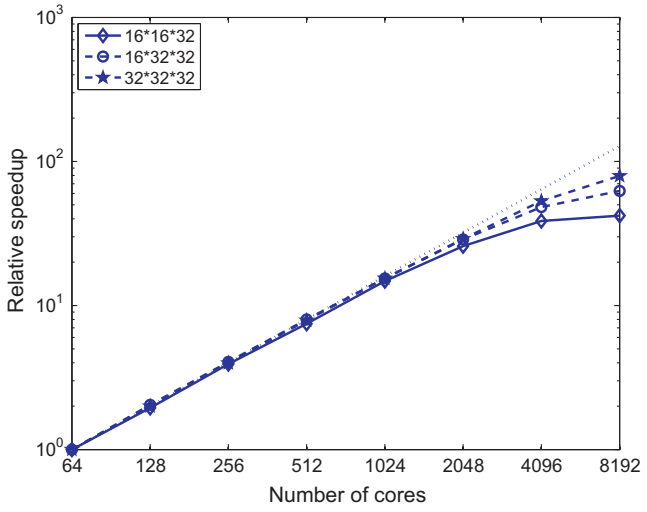

Fig. 2. Performance of a constant-size problem on the BG/P, on a number of cores varying from 64 to 8192. The speedup represents the relative efficiency of the program on N nodes, as compared to the efficiency on 64 nodes. The number of elements is selected among the choices of $E = 16 \times 16 \times 32 = 8192$, $E = 16 \times 32 \times 32 = 16{,}384$, and $E = 32 \times 32 \times 32 = 32{,}768$. It is seen that the scaling is excellent in spite of the small element-per-core ratio in the many-core limit.

Please cite this article in press as: Bosshard C et al. Computational performance of a parallelized three-dimensional high-order spectral element toolbox. Comput Fluids (2010), doi:10.1016/j.compfluid.2010.11.014

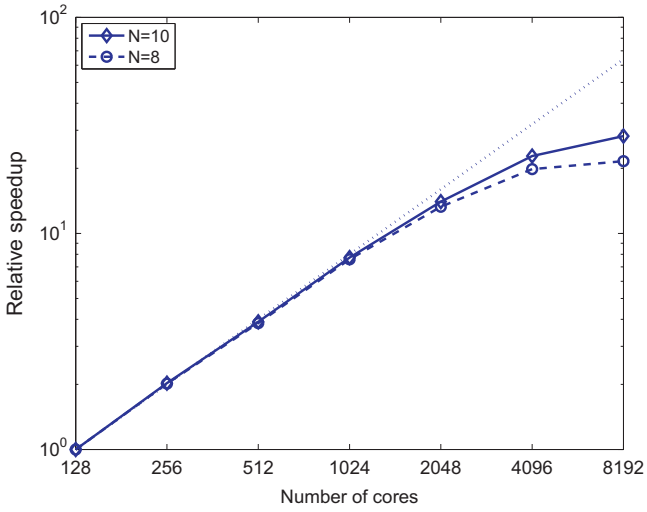

Fig. 3. Relative speedup of a constant-size problem on the BG/P, on a number of cores varying from 128 to 8192. The number of elements is chosen as $E = 16 \times 16 \times 32 = 8192$, and two different polynomial degrees $N = 8$ and $N = 10$ are used.

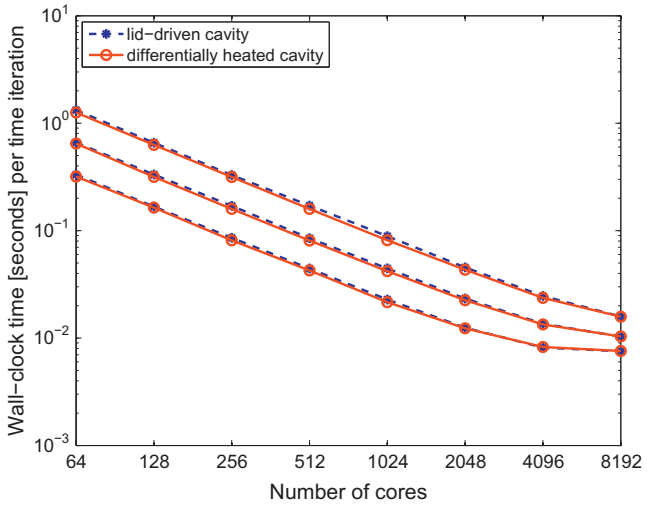

Fig. 4. Wall-clock time per conjugate iteration on the BG/P for both the differentially heated cavity problem and the lid-driven cavity problem. The number of elements is selected among the choices of $E = 16 \times 16 \times 32 = 8192$ (under two lines), $E = 16 \times 32 \times 32 = 16{,}384$ (middle two lines), and $E = 32 \times 32 \times 32 = 32{,}768$ (upper two lines). It is seen that the time per conjugate gradient iteration, and with this also the speedup, is essentially the same for both problems.

mance is rare for benchmark measurements in the limit of strong scaling. This is a proof of concept for the fact that even very small problems can be solved efficiently with a spectral element method on thousands of cores, if the hardware is well adapted and a performing interconnection network is used.

Knowing that a spectral element code based on the algorithm described in this paper is entirely exempt of non-parallel program components, it is clear that the performance loss observed in the many-core limit in Fig. 2 is entirely due to inter-process communication. This observation also applies to the SpecuLOOS code in which, as mentioned in Section 4, remaining non-parallel components were carefully eliminated. The program performance is therefore affected when the communication time between MPI threads approaches or reaches the time needed for the computations. In Eq. (22), it is argued that the computation time is proportional to $N^{3.3}$. The communication time on the other hand is proportional to the number of variables exchanged between two elements at each CG iteration. This corresponds to the number of degrees of freedom present on a face of an element, and is proportional to N^2. The communication-to-computation ratio is therefore proportional to $N^{-1.3}$, and decreases as N increases. This argument is validated in Fig. 3, which shows that the speedup of the program improves when a polynomial degree of $N = 10$ instead of $N = 8$ is used.

In Fig. 4 we show the wall-clock time per conjugate gradient iteration on the Blue Gene/P for both the differentially heated cavity and the lid-driven cavity problem and for different numbers of elements. The corresponding curves for the differentially heated cavity and the lid-driven cavity problem are very close. This shows that, as we expected, the time per conjugate gradient iteration and with this also the speedup do not depend on the specific test case.

5.2. Scaling on problems with linearly growing size

The limit of weak scaling is explored on problems in which the domain size grows linearly with the number of available cores. In this case, the time spent in inter-process communication is mainly dominated by the increasing computation time. As it can be seen in Fig. 5, the required wall-clock time per iteration step of the CG solver remains essentially constant as one increases both the number of cores and the domain size on the Blue Gene/P. The parallel performance is only affected in the many-core limit, in which the

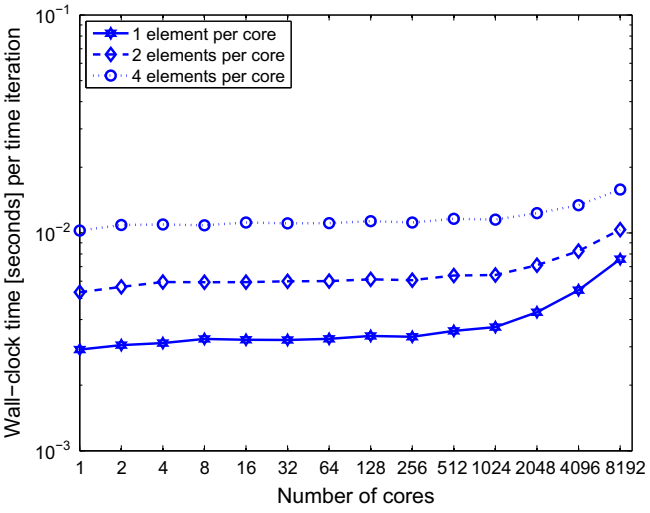

Fig. 5. Speedup of a growing-size problem on the BG/P, on a number of cores varying from 1 to 8192. In each case, the total number of elements is equal to the number of cores. The figure plots the computation time for iteration step of the CG solver, normalized with respect to the computation time on one core.

program is observed to slow down slightly. This effect vanishes however when the number of elements, and thus the overall computational load of the program, is increased. This loss of performance can be attributed to a saturation of the interconnection network as the overall load of communication on the machine increases. It might be possible to avoid such a saturation by mapping the communication structure of the algorithm more closely to the interconnection structure of the network, but this path was not explored in the present work.

5.3. Maximum measured performance

In this section, we provide an estimate of the number of floating-point operations per second (FLOPs per second, or FLOPS) executed by SpecuLOOS on the Blue Gene/P. This number does

Please cite this article in press as: Bosshard C et al. Computational performance of a parallelized three-dimensional high-order spectral element toolbox. Comput Fluids (2010), doi:10.1016/j.compfluid.2010.11.014

not provide a measure of how fast the program solves a given physical problem, because it does not take into account the quality of the numerical method or the quality of the algorithm used to implement this method. Instead, the FLOP-rate is a measure of how well the program takes advantage of the resources provided by a computer.

An overwhelming part of SpecuLOOS' computational time is used for the iterative resolution of systems of linear equations. These are implemented in terms of matrix–matrix and matrix–vector multiplications which are executed by corresponding routines of a BLAS library. The FLOP-rate of the program can therefore be evaluated accurately by counting the number of additions and multiplications executed at each call of the BLAS routines. With this method, the highest performance measured with SpecuLOOS on the Blue Gene/P was found to be 1.96 teraFLOPS, during the execution of a problem with $E = 32 \times 32 \times 32 = 32{,}768$ elements and a polynomial degree of $N = 8$ on 8192 cores.

With this FLOP-rate, SpecuLOOS reaches 7% of the Blue Gene/P's theoretical peak performance on 8192 cores reported in [14]. Such a seemingly modest performance should be regarded as quite common on modern computers on which a major bottleneck stems from the limited bandwidth between the main memory and the CPU. Other components which are thought to affect SpecuLOOS' performance in this case are an incomplete usage of the full instruction set offered by the CPUs of the Blue Gene/P, and the time spent in inter-core communication. As a future work, we therefore consider to replace the BLAS libraries by dedicated implementations of matrix–matrix and matrix–vector multiplications which make better use of the specific hardware characteristics of the Blue Gene/P.

5.4. Global scaling figure on the Blue Gene/P

While in the previous sections specific limits of scaling of a spectral element code were investigated, the present section proposes a global view on the performance figure of SpecuLOOS, presented on a single graph. Fig. 6 represents the measurements of wall-clock time, taken from a vast range of simulations of the lid-driven cavity problem, with a varying number of cores and size of the problem. It should be pointed out that all axes are logarithmic, and that both the number of cores and the number of elements span over several orders of magnitude. It is rare in computational science that such a vast range of scales can be treated with a single algorithm and with a single machine. This fact therefore underlines the exceptional scalability of both the SpecuLOOS code and the Blue Gene/P hardware.

A few lines on the plane in Fig. 6 are emphasized by the use of a bold line-style (and the use of colors in the online version of the paper), because of their particular signification. The front borderline of the surface represents for example a regime in which the number of elements is equal to the number of cores, and is identical to the curve plotted in Fig. 5. As discussed in Section 5.2, this curve has not a constant height, because of a saturation of the interconnection network in the many-core limit. The top borderline of the surface on the other hand is nearly constant, because in this case the number of element is proportional to the number of cores, with a proportionality constant much larger than 1. In this case, the communication time is hidden by the large computational load.

Furthermore, two diagonally oriented curves are emphasized on the surface of the plane. The curve evolving from the bottom-left to the top-right of the plane (blue in the colored version) represents a limit of strong scaling in which the number of elements is independent of the number of cores. The curve evolving from bottom-right to top-left (magenta in the colored version) represents a regime in which the number of cores is constant, and the execution time increases linearly with the overall computational load.

A global conclusion is drawn from Fig. 6 by observing that the represented surface is nearly plane in most of its parts. This basically means that the size of a problem is, to a large extent, decorrelated from the size of the utilized parallel machine. Any given problem can be, at choice, solved slowly on a small machine, or rapidly on a large machine. This argument is only limited by the atomic nature of an element which, in the algorithm presented here, cannot be parallelized. The number of cores of the parallel machine can therefore never exceed the number of elements used to discretize the problem.

6. Conclusions

The performance review presented in this paper for the high-order spectral and mortar element method C++ toolbox, SpecuLOOS, has shown that good performances can be achieved on small commodity clusters as well as on high-end parallel machines. The results support the original choices made in SpecuLOOS parallel implementation by keeping it at a very low-level.

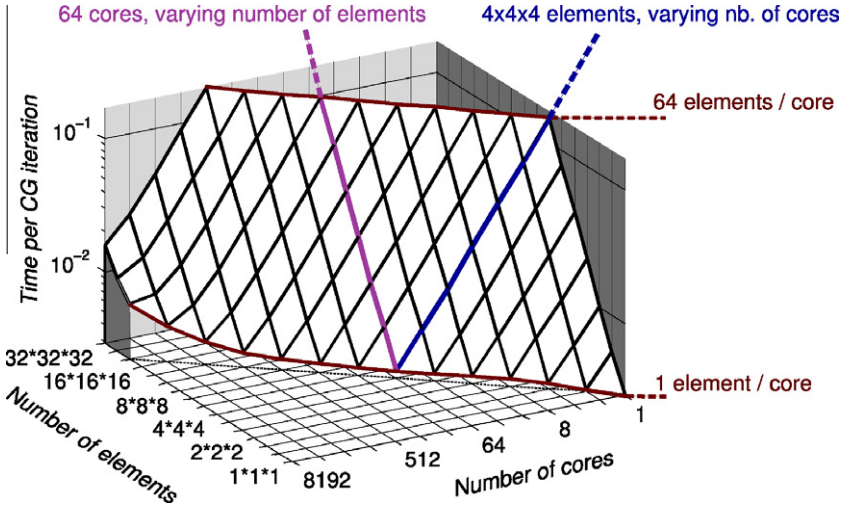

Fig. 6. Performance of SpecuLOOS on the Blue Gene/P at a polynomial degree of $N = 8$, depending on the size of the problem and the number of cores, for the lid-driven cavity problem. Each intersection of lines on the surface represents a measured benchmark value, corresponding to the wall-clock time, in seconds, needed to compute an iteration of the conjugate gradient solver. This surface is nearly plane, which shows that the algorithm, the parallelization, and the utilized hardware all scale nearly ideally. The performance is only affected, as expected, in the limit of small problem size and large number of cores, represented by the lower-left corner of the plane.

Please cite this article in press as: Bosshard C et al. Computational performance of a parallelized three-dimensional high-order spectral element toolbox. Comput Fluids (2010), doi:10.1016/j.compfluid.2010.11.014

One of the goal of this study was to estimate if SpecuLOOS could run on a massively parallel computer architecture comprising thousands of computational units, specifically on the IBM Blue Gene/P machine at EPFL with 4096 quad-core processor units. After detection and correction of a poor implementation choice in the parallel version, perfect scalabilities on up to 8192 cores have been achieved and measured.

A key to achieving this goal was to enforce a perfect scaling at the level of both the numerical method and the actual implementation. At the level of the numerical method, we made sure that the solver is preconditioned in order to converge rapidly, even asymptotically for large problems. The number of iterations required by the Conjugate-Gradient method scales linearly with the number of elements along a space dimensions, which means, as the cube root of the total number of elements. As for the implementation, we eliminated even the smallest non-parallelizable program components in order to guarantee scalability to any number of cores.

The presented performance figures clearly show that a spectral element method can efficiently exploit parallel architectures with thousands of cores, for large as well as for small problems. As many-core platforms become more and more common, this method is therefore an excellent choice if one wants to keep increasing the computational power available to computational fluid dynamics at the rate imposed by technological progress in supercomputing. Thanks to the scalability figures shown in this paper, we are confident that it will be possible to use the SpecuLOOS code on the next generation of IBM Blue Gene machines as well, with a number of cores ranging from 10^5 to 10^6. The results of corresponding test runs is the subject of a future publication.

Acknowledgments

This research is being partially funded by a Swiss National Science Foundation Grant (No. 200020-101707) and by the Swiss National Supercomputing Center CSCS, whose supports are gratefully acknowledged. We thank R. Puragliesi for providing us an initial solution for the test case of the differentially heated cavity problem.

References

[1] The OpenSPECULOOS project, <http://sourceforge.net/projects/openspeculoos/>.
[2] Dubois-Pèlerin Y, Van Kemenade V, Deville M. An object-oriented toolbox for spectral element analysis. J Sci Comput 1999;14:1–29.
[3] Bouffanais R, Fiétier N, Latt J, Deville M. High performance computing with spectral element methods. In: Buchlin JM, Rambaud P, Planquart Ph, editors. VKI lecture series 2009-05 on high-performance computing of industrial flows. Belgium: von Karman Institute for Fluid Dynamics; 2009. ISBN: 13 978-2-930389-93-1.
[4] Deville MO, Fischer PF, Mund EH. High-order methods for incompressible fluid flow. Cambridge: Cambridge University Press; 2002.
[5] Bodard N, Bouffanais R, Deville MO. Solution of moving boundary problems by the spectral element method. Appl Numer Math. 2008;58:968–84.
[6] Maday Y, Rønquist EM. Optimal error analysis of spectral methods with emphasis on non-constant coefficients and deformed geometries. Comput Methods Appl Mech Eng 1990;80:91–115.
[7] Fischer PF, Patera AT. Parallel spectral element solution of the Stokes problem. J Comput Phys 1991;92:380–421.
[8] Karniadakis GE, Sherwin SJ. Spetral/hp methods for computational fluid dynamics. Oxford University Press; 2005.
[9] Fisher P, Lottes J, Pointer D, Siegel A. Petascale algorithms for reactor hydrodynamics. J Phys: Conf Ser 2008;125:012076.
[10] Bouffanais R, Deville MO, Leriche E. Large-eddy simulation of the flow in a lid-driven cubical cavity. Phys Fluids 2007;19. Art. 055108.
[11] Couzy W, Deville MO. Spectral-element preconditioners for the Uzawa pressure operator applied to incompressible flows. J Sci Comput 1994;9:107–12.
[12] Gruber R, Keller V. HPC@Green IT. first ed. Berlin: Springer; 2010.
[13] Puragliesi R. Numerical investigation of particle-laden thermally driven turbulent flows in enclosure. PhD thesis, EPF Lausanne; 2010. <http://library.epfl.ch/theses/?nr=4600>.
[14] IBM Journal of Research and Development Staff. Overview of the IBM Blue Gene/P project. IBM J Res Dev 2008;52:199–220.
[15] Grinberg L, Karniadakis GE. A new domain decomposition method with overlapping patches for ultrascale simulations: application to biological flows. J Comput Phys 2010;229:5541–63.

Please cite this article in press as: Bosshard C et al. Computational performance of a parallelized three-dimensional high-order spectral element toolbox. Comput Fluids (2010), doi:10.1016/j.compfluid.2010.11.014