High throughput software for direct numerical simulations of compressible two-phase flows

Babak Hejazialhosseini, Diego Rossinelli, Christian Conti and Petros Koumoutsakos
ETH Zurich
{hbabak, diegor, cconti, petros}@mavt.ethz.ch

***Abstract*—We present an open source, object-oriented software for high throughput Direct Numerical Simulations of compressible, two-phase flows. The Navier-Stokes equations are discretized on uniform grids using high order finite volume methods. The software exploits recent CPU micro-architectures by explicit vectorization and adopts NUMA-aware techniques as well as data and computation reordering. We report a compressible flow solver with unprecedented fractions of peak performance: 45% of the peak for a single node (nominal performance of 840 GFLOP/s) and 30% for a cluster of 47'000 cores (nominal performance of 0.8 PFLOP/s). We suggest that the present work may serve as a performance upper bound, regarding achievable GFLOP/s, for two-phase flow solvers using adaptive mesh refinement. The software enables 3D simulations of shock-bubble interaction including, for the first time, effects of diffusion and surface tension, by efficiently employing two hundred billion computational elements.**

***Index Terms*—Compressible flows, high performance computing, multiphase flows, supercomputers**

I. Introduction

Compressible, two-phase flows are encountered in a number of fundamental and engineering fluid mechanics problems. Their study is relevant to understanding the governing mechanisms of vortex breakdown and to engineering applications such as high speed aerodynamics, combustion and biomedical applications. The quantification of compressible two-phase flows requires the use of large scale numerical simulations to complement and extend experimental studies. While experimental studies are essential in validating flow simulations and providing us with qualitative understanding, they cannot elucidate detailed physical mechanisms, such as shock-induced vorticity production, due to current limitations of imaging techniques [1]. Furthermore efficient simulations are essential to optimization and uncertainty quantification of compressible two-phase flows. Hence high performance computing is increasingly becoming the established approach in studying complex, compressible two-phase flows.

Simulation challenges for these flows can arise due to high compressibility effects, the existence of multiphase interfaces, the enhanced production of vorticity at interfaces (see Figure 1) and the instabilities induced by capillary effects due to surface tension forces. The simulation of shock-bubble interaction (SBI) [2] is exemplary of such challenges with mixing and turbulence drastically enhanced by the vorticity produced during the passage of shock over the two-phase interface.

As reviewed in [1] the vorticity generation in SBI has been first analyzed in the uniform resolution, 2D simulations of Picone and Boris [3] employing 50 grid cells per bubble radius. Simulations of SBI employing Adaptive Mesh Refinement (AMR) [4] in 2D were performed by Quirk and Karni [5] and a new numerical method for two-phase flows was suggested by Marqiuna and Mulet [6]. Early uniform resolution simulations of SBI in 3D, in the context of astrophysics, have been performed by Stone and Norman [7], using a resolution of 60 grid cells per bubble radius. Recent 3D numerical simulations of SBI, using AMR, have employed up to 134 grid points across the radius of the bubble with symmetry boundary condition in two space dimensions [8] with an effective system size of 2 billion grid cells. Figure 1 depicts the shock waves and interface structure of SBI along with experimental result of Ranjan et al. [8] and the present 3D simulations. Another example of compressible flows featuring viscous dissipation is the reconnection of compressible vortex tubes (CVT) studied in simulations [9] and experiments [10]. Engineering relevant CVT simulations should target $Re > 10^4$ within isentropic flows with $M > 1$ (Reynolds number Re characterizes the ratio of inertial forces to viscous forces and Mach number M is a measure for the compressibility of a flow). In this perspective, next generation of simulations are likely to demand in the order of hundreds of billion computational elements, if not more.

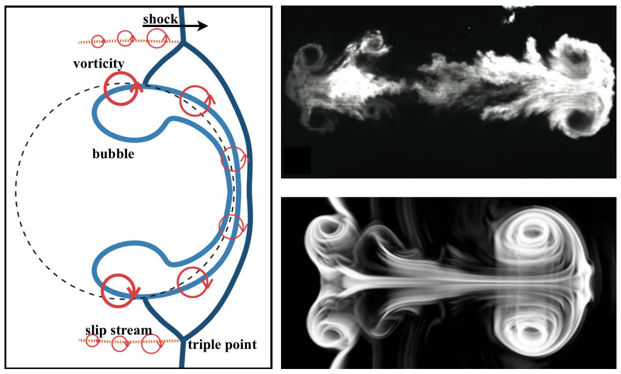

Fig. 1. Schematic of vorticity production in shock-bubble interaction (left), experimental flow visualization as shown in Ranjan et al. (right top, republished with permission of Annual Reviews Inc, from [1]; permission conveyed through Copyright Clearance Center, Inc.) and density gradient field from the present direct numerical simulations (right bottom).

In this work we present uniform resolution, 3D simulations of SBI, using up to 200 billion grid points and accounting for the first time for surface tension and diffusion effects. The

SC12, November 10-16, 2012, Salt Lake City, Utah, USA
978-1-4673-0806-9/12/$31.00 ©2012 IEEE

proposed software trades non-adaptivity of the computational elements for an efficient parallel implementations that aims to achieve high fractions of the peak performance. AMR algorithms rely on dynamically increasing the grid resolution in specified areas of interest in the flow field. Albeit their effective use of computational elements, the irregular patterns of AMR present challenges to reach high fractions of the peak, which could neutralize the algorithmic improvement gained from introducing the spatial (and possibly temporal) adaptivity. The reported state of the art calculations reach 7% of the peak performance [11] in geophysics simulations. AMR and wavelet-based solvers [12], although more economical in the number of computational points, might render the computation more irregular. We recognize the need for such algorithmic advantages and as such, the authors also focused on adaptive solvers on multicore and heterogeneous systems [13], [14]. The objective of this work is to assess a potential reference HPC software up to groundbreaking uniform resolutions, precisely motivated by the need to quantify the improvements provided by adaptive techniques.

The development of high throughput software for compressible two-phase flows necessitates software design choices and the effective mapping of the parallel work of the simulation onto the computing hardware. At the present time, we observe a “node-centric” performance increase of the computing resources: the notable growth in the vector widths boosting data-level parallelism (DLP), a convoluted memory organization as well as the enhancement of thread-level parallelism (TLP). The latter, on recent multicore platforms, is multifaceted since not all the computing resources scale with the number of cores (*e.g.* the Last Level Cache on most state-of-the-art CPUs and the symmetrical 128-bit floating-point pipelines within a module of the AMD Bulldozer). Software-visible features such as specialized arithmetic instructions and instructions to bypass cache-coherence protocols further contribute to the complexity of modern computing hardware. These advances provide a substantial increase in the available performance of the multicore platforms and present us with significant potential for high throughput software for flow simulations. However it has been shown that, in contrast to synthetic benchmarks, real world applications can hardly harness such potentials, reaching only about 1% of the peak performance [15]. A key hurdle is the imbalance between the peak performance and the memory bandwidth of the computing platforms as they grow towards 1-10 TFLOP/s and 0.1-1 TB/s. This means that software targeting peak performance should employ computing kernels that exhibit ratios of 10 FLOP/B (bytes of off-chip memory traffic) or more. As reported in this work, having such high ratios in the context of multiphase compressible flows does not seem to be feasible.

The contributions in the present work address a number of open issues regarding HPC of compressible flows. Firstly, to the best of our knowledge, there is currently no simulation software for compressible flows that conducts a-priori performance analysis. Secondly AMR and wavelet-based solvers have not been benchmarked against highly efficient uniform resolution solvers as a baseline within this context. Finally for compressible flow simulations, there is no acknowledged software whose performance can reach a significant fraction of the peak. Reported performance of existing software for compressible flows suggests hardware utilizations that are 2-3% of the peak performance [16], [17]. Additionally, this work enables us to evaluate the latest computing infrastructures (available since 2011 and 2012), ranging from desktops to supercomputers and assess recent optimization techniques. We are also able to perform unprecedented 3D simulations of SBI accounting for effects of surface tension and diffusion, thus bringing the simulations closer to the experiments [1], [2].

In this work we perform a-priori estimates on achievable performance bounds using the roofline model [18]. We focus on regular uniform resolution grids, two-phase flows with viscous dissipation and surface tension effects, therefore not considering astrophysical applications. The software presented in this work achieves an unprecedented 30% of the nominal peak performance on 47,000 cores, computing on 200 billion computational elements. For single node executions, the software reaches 45% of the nominal peak performance on 48 cores. We suggest that, due to the regularity and uniform resolution of the computational elements, the present work may serve as the performance benchmark of software for compressible flows, including AMR or wavelet based, simulations.

This work is structured as follows: in Section II we discuss the previous solvers for HPC CFD and efforts on software optimization techniques in the context of stencil computation. In Sections III-V we introduce the roofline model, describe the design of our software and specific optimization techniques considered together with the computing hardware. In Section VI, we present and discuss the achieved results of our software.

II. Related work

Most of the recent efforts on HPC CFD involves the use of adaptive mesh refinement (AMR) techniques after the work of Oliger and Berger [4]. *Chombo* [19] is a block-structured AMR package which features a number of solvers for time dependent system of PDEs with the support for compressible, two-phase flows and problems with solid boundaries as well as for elliptic PDEs. Its performance has been shown only for elliptic problems by Wen et al. [20] to scale up to 8000 processors.

FLASH [17] is a solver for hyperbolic system of equations on adaptive grids constructed by *PARAMESH* and it has been shown to scale up to 33k cores for terascale simulations of compressible turbulence [21]. Moreover, it has been used on 8k cores to simulate weakly compressible turbulence with efficient support for Lagrangian tracers [22]. This solver has been extensively used for the simulations of magneto-hydrodynamics (MHD, [23] and [24]) and astrophysics problems by Calder et al. [25].

Raptor developed by Greenough et al. [16] has been used for both uniform resolution and AMR-based simulations of compressible flows with two-phases (see [8]). The reported performance for uniform resolution simulations on a 32k-core IBM BG/L achieved 3 TFLOP/s, corresponding to 3.3% of peak performance.

Another AMR-based solver is presented by Burstedde et al. [11] for geophysics problems which scales on 223k cores of the Cray-XT5 Jaguar supercomputer and reaches 175.6 TFLOP/s *i.e.* 7% of the peak performance. There, high order continuous Galerkin method is employed and a space-filling curve is used to access the mesh created based on forest-of-octrees to ensure good data locality.

For fluid-structure interaction (FSI) problems, an AMR-based solver called *Uintah* by Berzins et al. [26], [27] is shown to scale up to 98k cores.

Other examples of optimized simulation software for multicores are presented by Madduri et al. [28] for gyrokinetic toroidal simulations on 49k cores of Cray-XE6 Hopper supercomputer and by Williams et al. [29] for simulations of MHD using Lattice-Boltzmann on structured grids. The optimization strategies of the former include adopting the structure-of-arrays (SoA) as opposed to of array-of-structures (AoS) format of the computational elements, aligned memory allocation, NUMA awareness and cache-bypass optimization. These techniques increase the performance of their solver by 60% on 6144 cores.

In the context of compressible multiphase flows, stencil computation is the building block of most of the computing kernels due to extensive use of finite volume reconstruction-evolution schemes. Specific techniques to enhance the performance of stencil computations are suggested by Kamil et al. [30], [31]. These works are based on the faster growth of the peak computing performance with respect to the peak memory bandwidth. Recently a framework for auto-tuning and auto-parallelization of stencil computations has been proposed, also featuring NUMA-aware optimization techniques [32].

One way to achieve higher fractions of the peak performance is to consider discretization schemes that lead to computational steps featuring higher FLOP/B ratio in computation. Examples are p-refinement in the context of finite element method (FEM), compact finite difference schemes or solving for both vorticity and its derivatives as in vortex methods [33]. This approach however imposes substantial modifications to existing flow simulation software and implies that algorithms, implementations and considered optimization techniques must be revisited to reach the production stage. Other efforts on performance improvement include the hardware/software co-design for domain specific problems [28], [34], [35].

Techniques based on the roofline model proposed by Williams et al. [18] can help increase the performance by suggesting appropriate software optimization strategies with the main advantage of being less drastic than those mentioned before. This has been proven to be suitable in quantifying the performance on different scientific problems in the context of flow simulations [13], [36], [37]. Software auto tuning tools [38] and dynamics schedulers based on work stealing [39]–[41] have proven to increase the achieved fraction of peak performance of multicore platforms. Other advancements related to memory placement can substantially improve the performance in the context of multiprocessors such as next-touch policies [42]–[44]. Automated code generation proposed in [45] and [46] is among other possibilities but has not yet been focused to the field of compressible flow simulation.

Due to their long vector width, GPUs have been extensively considered to accelerate the computation of flow simulations. Heterogeneous multi-core/multi-GPU computing proved to significantly reduce the time to solution for simulations of SBI with wavelet-based adaptive solvers [47]. Clusters of GPUs have recently enabled simulations of biofluidics [48] and metal solidification [49] at peta-scale. A new framework for stencil computation on large-scale clusters of GPUs is also presented by Maruyama et al. [50]. Even though heterogeneous computing provides significant acceleration factors, optimizing CPU code prior to comparison with GPUs is still considered to be of crucial importance [51]. The design and study of heterogeneous GPU-CPU solvers for compressible flow simulations is beyond the goals of this work and will be treated in the future.

TABLE I
SUPERCOMPUTING CLUSTERS

Server Platform	**Magny-Cours**	**Interlagos**
Per-core performance	17.6 GFLOP/s	33.6 GFLOP/s
Cores (modules)/die	6	8(4)
Dies/node	8	4
DRAM Bandwidth	96 GB/s	60 GB/s
Ridge point	8.8 FLOP/B	9 FLOP/B
Cluster name	Brutus	Monte Rosa
Cluster nodes	64	1482
Cluster SP	54 TFLOP/s	797 TFLOP/s
Cluster Network	Infiniband QDR	Gemini

III. THE ROOFLINE MODEL

In this work, we employ the roofline model [18] to predict and assess the performance of the simulation steps. This model is constructed upon the peak performance and the DRAM memory bandwidth of the underlying hardware as well as the “operational intensity” – OI, defined as the amount of FLOP or relevant operations per Byte of off-chip memory transfer – of the given computation. The roofline model is a plot of the achievable performance (GFLOP/s) versus the OI (FLOP/B). The plot consists of a horizontal line denoting the compute-bound region and a diagonal line which denotes the region where performance is memory-bound. The intersection of these two lines is referred to as the “ridge point” of the machine. Given a compute kernel with an operational intensity of 0.1 FLOP/B – for a machine with 800 GFLOP/s and 100 GB/s – the maximum achievable performance is estimated by $\min(800, 0.1 \cdot 100) = 10$ GFLOP/s. The min(.) function reflects the parallelism between computation and memory transfer. Any kernel with OI<8 FLOP/B cannot achieve peak performance and it is thus memory-bound. In addition to OI, we refer to the “arithmetic intensity” (AI), which takes into account all the memory traffic from and to the CPU. OI and

AI of the computing kernels are used to estimate the upper and lower bound of the achievable performance, respectively.

IV. Experimental setup

In this work we consider two high-end desktop platforms and two supercomputing clusters. Desktops are considered here since our software targets single-node computation for engineering optimizations. A summary of the considered computing hardware is provided in Tables I and II, with their performance reported for single precision.

A. Intel Sandy Bridge (Desktop)

Intel Core i7-2600K (released in January 2011) is a quad-core processor running at 3.4 GHz on a single socket that features HyperThreading technology. Its micro-architecture is Sandy Bridge, which supports AVX instructions. The measured peak performance and DRAM memory bandwidth are 217.6 GFLOP/s and 20.5 GB/s, respectively.

B. AMD Zambezi (Desktop)

The AMD Zambezi (released in December 2011) is built on AMD's Bulldozer micro-architecture and has four modules with a frequency of 3.6 GHz. A module is composed of two cores which share two 128-bit FPUs and an L2 cache. When using 256-bit AVX instructions, the two FPUs of a module are used together. This platform supports FMA4 instructions for fused multiply-add operations. The cache hierarchy is composed by the L3 cache (LLC) shared on die, the L2 cache shared on module and two dedicated L1 write-through data caches per module (corresponding to the two cores). The measured peak performance and DRAM memory bandwidth are 230.4 GFLOP/s and 18 GB/s, respectively.

C. AMD Interlagos (Server)

The AMD Interlagos (released in September 2011) is also built on AMD's Bulldozer micro-architecture. Each socket is composed of 4 cache-coherent Non-Uniform Memory Access (ccNUMA) nodes running at 2.1 GHz on 2 sockets. The nominal peak performance is 537.6 GFLOP/s. The DRAM memory bandwidth of the XE6 reaches 30 GB/s per socket (60 GB/s in total). To assess the performance of large-scale simulations, we consider "Monte Rosa", the cluster of the Swiss National Supercomputing Center (CSCS) and employ up to 1482 Cray XE6 compute nodes (about 47k cores) with a peak performance of 0.8 PFLOP/s.

D. AMD Magny-Cours (Server)

The AMD Magny-Cours (released in March 2010) is a 4P 12-core AMD Opteron 6174 running at 2.2 GHz totaling in 48 cores and is built on the Barcelona micro-architecture. The 12-core processors are composed of two ccNUMA nodes (exa-cores) sharing a socket with a quad channel memory controller (two per NUMA node). The measured peak performance and memory bandwidth is 844 GFLOP/s and 96 GB/s for one compute node. We use up to 64 Magny-Cours nodes with a nominal peak performance of 27 TFLOP/s on the Brutus cluster at ETH Zurich.

TABLE II
High-end Desktop Platforms

Desktop Platform	**Intel Sandy Bridge 2600K**	**AMD Zambezi FX-8150**
Per-core performance	54.4 GFLOP/s	57.6 GFLOP/s
Cores (modules)/die	4	4
Dies/node	1	1
DRAM Bandwidth	20.5 GB/s	18 GB/s
Ridge point	10.6 FLOP/B	12.8 FLOP/B

V. The software

The design adopted in our software is based on a stack with three software layers: core, node and cluster. These separations allow for layer-specific optimization and provide the necessary abstraction levels for enhancing code re-use. Furthermore they facilitate future developments on new computing hardware and can easily accommodate novel numerical methods.
The high-level parts of the software rely on object-oriented patterns such as the Singleton, Decorator, Facade and Strategy [52]. The high performance parts of the software are also organized into classes, and make extensive use of AVX and SSE intrinsics. The polymorphic calls in these classes are forced to be resolved at compile time with the help of generic programming. In terms of productivity, adopting an object-oriented programming language allowed us to quickly evaluate different computational strategies. It also led to a very cost-effective software development, requiring the efforts of less than 9 man months. In terms of performance, the overhead incurred by the object-oriented patterns is practically negligible as it is amortized over the computational work.

A. Numerical methods and algorithm

The considered governing equations in this work are the compressible Navier-Stokes equations with diffusion and surface tension terms:

$$\begin{aligned} \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) &= \mathbf{0}, \\ \frac{\partial (\rho \mathbf{u})}{\partial t} + \nabla \cdot (\rho \mathbf{u}\mathbf{u}^{\mathbf{T}} + \mathbf{p}\mathbf{I}) &= \nabla \cdot \tau + \nabla \cdot \mathbf{T}, \\ \frac{\partial (\rho E)}{\partial t} + \nabla \cdot ((\rho E + p)\mathbf{u}) &= \nabla \cdot (\tau \cdot \mathbf{u}) + \nabla \cdot (\mathbf{T} \cdot \mathbf{u}), \\ \frac{\partial \phi}{\partial t} + \mathbf{u} \cdot \nabla \phi &= \mathbf{0}, \end{aligned} \quad (1)$$

where $\tau = (\mu(\nabla \mathbf{u} + \nabla \mathbf{u}^{\mathbf{T}} - \frac{\mathbf{2}}{\mathbf{3}}\nabla \cdot \mathbf{u}\mathbf{I}))$ and $\mathbf{T} = \sigma(\mathbf{I} - \mathbf{n}^{\mathbf{T}}\mathbf{n})|\nabla \mathbf{C}|$ are the viscous stress and surface tension tensors [53]. μ and σ are the dynamic viscosity and the surface tension coefficient, respectively. C is the volume fraction and is computed by applying a smoothed heaviside filter to the marker function, ϕ and $\mathbf{n}$ is the interface normal computed from C. These equations require solving for 6 unknowns: density ρ, momenta ρu, ρv, ρw, total energy per unit density E and the multiphase interface marker ϕ. Pressure, as an intrinsic unknown, is calculated from ideal gas law.

Since we employ a finite volume discretization, the computation is organized into two parts: the evaluation of the right hand side (RHS) and the update of the flow quantities.

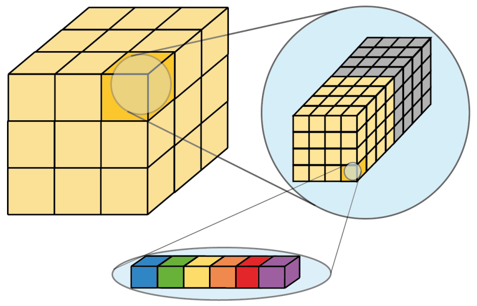
Fig. 2. Grid composed of blocks (top left) and blocks made of grid points and organized in AoS format (top right, bottom). Every block also holds extra storage for the RHS (gray).

One simulation step requires 3 evaluations of the RHS and 3 updates since we employ a third order memory efficient version of the Runge-Kutta time stepping scheme [54].
The *dt* stage computes an adaptive advection-/diffusion-limited time step by computing the maximum characteristic velocity of the flow before every time step. The evaluation of the RHS consists of computing the convective, diffusion and capillary terms. The *convection* stage first converts the conserved flow quantities to primitive quantities (density ρ, velocities $\mathbf{u}$ and pressure p) as this has been shown to maintain the zero velocity and pressure jump across the multiphase interface in time [55]. A WENO stencil [56], [57] is then applied to reconstruct the primitive quantities after [55] given by the previous step on discretization cell boundaries. Finally the HLLE numerical flux [58] is used to approximate the integral of fluxes on cell boundaries in time. The *diffusion* stage computes the diffusive flux and the work of shear stresses for the equations of momenta and total energy, respectively. The stress tensor on finite volume cell boundaries is computed with averages of the velocity derivatives from the neighboring cells. The *surface tension* stage computes the capillary effect in a conservative form from the interface marker function [59], similarly to the diffusive fluxes. In this work, we modify the interface evolution equation for the HLLE flux using the method introduced by Saurel and Abgrall [60]. In order to perform the stencil computation steps and apply the desired boundary conditions, we need to temporarily construct the values of auxiliary points, hereafter called ghosts.

In a previous work [14], this approach has been shown to be capable of resolving the flow features of SBI at different Mach numbers when compared to the previous experiments [2] and numerical studies [3], [5], [61]. Alternatives to this approach are the spatial reconstructions based on TVD/MUSCL [62], PPM [63] and the more advanced HLLC numerical flux [64]. These reconstruction schemes have lower OI and are formally less accurate than those considered in this work. HLLC flux although more accurate than HLLE, involves several fine-grained conditional branches. Viscous dissipation and surface tension are computed explicitly in conservative form with second order accurate stress tensors. In order to ensure high order accuracy for the critical convection step, we employ a fifth order WENO scheme [14].

TABLE III
ARITHMETIC AND OPERATIONAL INTENSITIES IN FLOP/B

Kernel	SoA needed for vectorization	Conditional branches	AI (FLOP/B)	OI (FLOP/B)
dt	no	no	1.7	5.1
Conv.	yes	yes	1.5	45.4
Diff.	yes	no	0.3	4.8
S. Ten.	yes	yes	0.3	2.8
Upd.	no	no	0.2	0.2

B. Data structures

The 6 unknowns are organized into a structure referred to as "grid point". To increase the data locality and hence the OI of the computation, we introduce an intermediate data structure between the individual grid points and the global grid, called hereafter *block* (Figure 2) which contains 16-32 grid points per dimension and is allocated contiguously in memory. Data inside a block is organized in an AoS format as it provides abstraction for a block with any given grid point. This enables us to straightforwardly represent grid points with classes, which in turn enhance the flexibility of the software. Moreover, a block stores 6 extra variables for computed right hand sides required by the Runge-Kutta time stepper. The main disadvantage of introducing blocks is that the global unit stride access over the grid is lost.

C. Core layer

This layer is responsible for harnessing the available DLP and instruction-level parallelism (ILP) to process blocks by executing the different computing kernels. It is noteworthy that within this layer, there is no TLP since every single block is processed by only one thread.

Table III presents the AI and OI and other relevant properties of the individual computing stages. The update kernel is a data-parallel operation and the estimated values for AI and OI is 0.17 FLOP/B. The time step computation (*dt* kernel) involves a data-parallel operation that computes the specific heat ratio by applying a step function filter to the marker ϕ, and it is followed by a reduction operation. For this kernel, we approximate AI≈1.7 FLOP/B and OI≈5.1 FLOP/B. In the convection kernel, the conversion from conserved to primitive quantities is done by AoS/SoA shuffling and is data-parallel. WENO reconstruction is inherently data-parallel. HLLE numerical flux involves conditional branches in order to choose the right flux based on the signal velocities. This kernel can be expressed as vector operations only if the data is in SoA format and every flow quantity is computed separately. We use different HLLE kernels for diagonal and non-diagonal momenta so as to avoid conditional branches. For the convection stage, the estimated values for AI and OI are 1.5 FLOP/B and 45.4 FLOP/B, respectively. The diffusion kernel requires another conversion to translate momenta to velocities. For the diffusion stage, AI≈0.3 FLOP/B and OI≈4.8 FLOP/B. The surface tension kernel is very similar to the diffusion kernel in computing the gradients on the cell interfaces and the divergence of the tensors. For the surface tension, we estimate AI≈0.3 FLOP/B and OI≈2.8 FLOP/B.

TABLE IV
OPERATIONAL INTENSITIES IN FLOP/B WITH AND WITHOUT BLOCKS

Representation	dt	Conv.	Diff.	S. Ten.	Upd.
with Blocks	5.1	45.4	4.8	2.8	0.2
w/o Blocks	5.1	3.6	0.5	0.5	0.2

Table IV shows how the increased locality provided by the blocks (with 32 cells in each dimension, in contrast to the full grid) improves the OI of the convection, diffusion and surface tension kernels, leading to higher upper bounds in achievable performance for these kernels. The values for the AI remain unchanged and can be seen on Table III.

We note that except for the convection kernel, all the kernels have an OI which is less than any ridge point of the considered platforms and are therefore memory-bound. Our efforts on this layer are oriented towards increasing the OI of individual kernels by minimizing the off-chip memory traffic and maximizing the effective use of the memory bandwidth. The latter is achieved by enforcing 16 (or 32) Byte alignment for the data inside every block and using (vectorized) aligned memory read instructions.

We increase the spatial and temporal locality, by means of data and computation reordering techniques. By introducing the blocks, the spatial locality of the adjacent grid cells residing on a block is readily increased as they become closer in memory. To further enhance the spatial locality while computing the right hand side, we introduce intermediate data structures, referred to as "slices" that are organized in SoA format and are lightweight in memory. This format is helpful as the vectorization of many kernels demands for it and it is supposed to achieve better performance than working in the original AoS format [28], [65].

Data flow and AoS/SoA shuffling into/from slices are depicted in Figure 3. Conserved quantities from the AoS-formatted blocks are extracted, converted and stored into 2D slices and after the computation of the RHS, the results are stored back to the temporary storage of the block (region in gray). The cost of the AoS/SoA transposition is hidden by the relatively high FLOP/B ratio of the conversion. The slice can be then readily processed by WENO and HLLE kernels in x- and y-directions.

We process 2D xy-slices one after another. Performing the directional sweeps in succession on one slice (and not on individual cells or stripes of cells within a slice) relaxes the data dependency, thus potentially making better use of the ILP.

In order to increase the OI of the kernels, transposition of the computed RHS back to AoS format is merged with the first stage of the Runge-Kutta time stepper. To avoid potential ILP issues, we replace the conditional branches within the kernels with arithmetic expressions and blend instructions (when AVX intrinsics are used), the latter however are not included in the FLOP count. Moreover, divisions and square roots are replaced by 1.5/2.5 ulps accurate alternatives [66] for the computation of the stencil weights of WENO reconstructions, the speed of sound and the flux of the energy.

It is worth mentioning that the flux computations are only shared within blocks. Inter-block fluxes are not shared among different blocks as this would require additional synchronizations and memory operations.

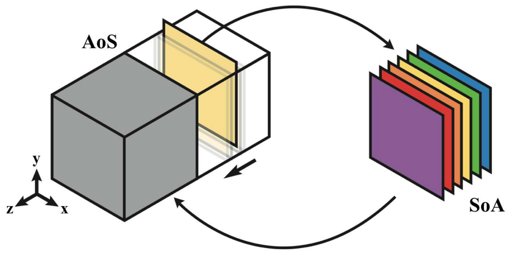

Fig. 3. The march on the blocks is done in the z-direction. Data is extracted and converted from the AoS-formatted block into slices with SoA format. The color code of the slices is as in Figure 2.

D. Node layer

This layer is responsible for exploiting the TLP available on the CPU and relies on the core layer. The computing on individual blocks is delegated to the core layer while the node layer performs the ghost reconstruction and coordinates the kernel executions within the simulation steps. A major issue of this layer is the load imbalance which is caused by either choosing a parallel grain size that is too large or experiencing non-constant cost of memory accesses. The choice of the parallel grain size in our software inherently corresponds to a single block whose size is large enough such that the computing time (0.1-10 milliseconds) hides the multithreading overhead. This grain size is small enough if we consider that the total number of blocks in the grid is typically chosen to be one order of magnitude higher than the number of available threads. Another source of load imbalance is the variable number of required ghosts since it depends on the prescribed boundary conditions in each direction.

Another task of this layer is to minimize the off-chip memory traffic, which necessitates an increase in spatial and temporal locality. This is achieved in two ways: by providing thread-data affinity and employing space filling curves (SFCs). The former is achieved via the open source `libnuma` library to always bind logical threads to the same cores, forcing the threads to access the same memory locations and also guaranteeing that threads running on the same socket work on blocks that are close in memory. Due to the diversity of the caches in the considered platforms, it is appealing to organize the blocks according to an SFC and march over the blocks in a cache-oblivious fashion. In this work, we use Morton indexing for data reordering and the marching over the blocks (we do not use any SFC within a block).

A crucial issue for the design of this layer is the choice of the concurrency level. Since most of the kernels in this work are memory-bound, we enforce a 1-1 mapping between the logical and the hardware threads (or integer cores in the case of Bulldozer micro-architecture) as the resource contention is hidden by the cost of the memory accesses.

Another issue is the cost associated with the DRAM access on a ccNUMA compute node. On the considered NUMA platforms the cost of memory access may not be constant. For example on Interlagos, the inter-node distance is 60% more than the intra-node. It is possible that some data needed from a given thread resides on a remote NUMA node, for example in the ghosts reconstruction stage. This is alleviated by carefully placing the data on the right NUMA node compliant to first-touch policy at the initialization stage, by using OpenMP pragmas with static scheduling. Because of the typical distance between NUMA nodes, this strategy can yield significant gain for those kernels with lower FLOP/B ratio. Alternatively, we delegate this issue to the cluster layer which assigns one MPI process to every NUMA node for proper data placement (see below).

A number of issues can potentially arise from different kernels during a simulation. Firstly, since the convection kernel takes roughly 1 millisecond per block, any persistent source of load imbalance can lead to a noticeable performance degradation. Secondly, diffusion and surface tension kernels require tensorial ghost exchange and have several spread accesses and therefore can suffer from accessing longer NUMA distances.

TABLE V
CORE LAYER MEASURED PERFORMANCE

(a) Server platforms: M=measured performance in GFLOP/s, R=predicted range in GFLOP/s, %P=fraction of the peak

Kernel	Interlagos (32 threads)			Magny-Cours (48 threads)		
	M	R	%P	M	R	%P
dt	8.9	6.4-19	26	4.6	3.4-10.2	26
Conv.	11.3	5.7-33.6	34	6	3-17.6	34
Diff.	4.4	1-16.1	14	2.7	0.6-8.6	15
S. Ten.	3.1	1-10.5	9	1.8	0.6-5.6	10
Upd.	0.7	0.6-0.6	2	0.3	0.3-0.3	2

(b) Desktop platforms: M=measured performance in GFLOP/s, R=predicted range in GFLOP/s, %P=fraction of the peak

Kernel	Sandy Bridge (8 threads)			Zambezi (8 threads)		
	M	R	%P	M	R	%P
dt	22.8	8.7-26	42	9.7	7.7-22.9	17
Conv.	26	7.8-54.4	48	18.6	6.8-57.6	32
Diff.	7.3	1.4-22	13	6.1	1.3-19.3	12
S. Ten.	5.3	1.5-14.4	10	3.9	1.3-12.6	7
Upd.	0.9	0.9	2	0.8	0.8	1

E. Cluster layer

This layer exploits the parallelism available on supercomputing clusters. The main issues of this layer include the load imbalance, high internode communication costs and the synchronization overhead. By design, load imbalance is minimized with a Cartesian domain decomposition. A "pseudo-block" of blocks is assigned to every node and the number of blocks is the same for every node.

This layer is responsible to synchronize the inter-node ghosts several times during a simulation step. We separate the synchronization stage of the convection kernel from diffusion (and surface tension) kernel since it is non-tensorial and it requires a thicker layer of ghosts.

Our design aims to hide the communication costs with computation. The pseudo-block is decomposed into an "inner block" region and a "halo" region. While the halo region is sent and received by neighboring nodes in a non-blocking fashion, we process the inner block, which does not need to wait for the inter-node ghosts to arrive. In a second pass, we process the blocks residing on the halo region as the inter-node ghosts are completely exchanged. The total expected cost of this approach is equal to $\max(T^{\text{kernel}}_{\text{inner block}}, T^{\text{comm.}}_{\text{halo}}) + T^{\text{kernel}}_{\text{halo}}$.

The improvements provided by this technique may not be appreciable in many cases *e.g.* when $T^{\text{comm.}}_{\text{halo}} \gg T^{\text{kernel}}_{\text{inner block}}$ or $T^{\text{kernel}}_{\text{halo}} \gg \max(T^{\text{kernel}}_{\text{inner block}}, T^{\text{comm.}}_{\text{halo}})$. We therefore opt for a multi-pass approach where the halo is further decomposed into "faces", "edges" and "corners". The last two are necessary only if the underlying operator requires tensorial ghosts, as for the diffusion and surface tension kernels. This approach allows us to relax the data dependency across the computation. As soon as a portion of inter-node ghosts arrives, the corresponding sub-region in the halo is "dispatched" for computation to the node layer. We implement this mechanism using OpenMP. Although being advantageous with respect to the plain inner/halo regions approach, this strategy could suffer from reduced degree of parallelism. If one of the necessary faces on one node arrives a little bit earlier than other faces, the node layer proceeds with computing only one face of the halo, disregarding the amount of parallel work available immediately after.

To properly address this issue in the cluster layer, we have designed a parallel pattern that features "genuine" computation/transfer (C/T) overlap. This approach turns the master thread into a "listener" for the incoming ghosts, while the other threads are put in a thread pool. The listener immediately dispatches the computation of the inner region to the thread pool. As soon as the inter-node ghosts arrive, the blocks in the halo are dispatched for the computation. When the last necessary data arrives, the master thread joins the thread pool to help process the last blocks by stealing some work. This is implemented with the Intel's Threading Building Blocks (TBB) library due to its simplicity in expressing such patterns. This provides us with the advantage of dynamic load balancing which is desirable when processing faces, edges and corners as they inherently have different sizes. Dynamic load balancing compensate for the irregular communication cost incurred by ghost reconstructions across different blocks: the domain boundaries need less communication, inner blocks of a subdomain require some intra-node communication, whereas blocks in the subdomain halo perform costly internode communication. On this layer we do not use any SFC as we rely on the 3D torus network provided by the cluster.

VI. RESULTS AND DISCUSSION

A. Core layer

We aim to quantify the efficiency of the core layer by measuring the performance of the individual kernels for the evaluation of the RHS and the update of the flow quantities in single precision. The benchmark consists of executing every kernel by subscribing one thread per hardware thread/integer core, so as to capture possible resource contentions within the sockets. The per-thread memory footprint is kept constant at 128 MB. To compute the AIs and OIs of the kernels, we only count "clean" FLOPs, *i.e.* only floating point operations appearing in the algorithm formulation.

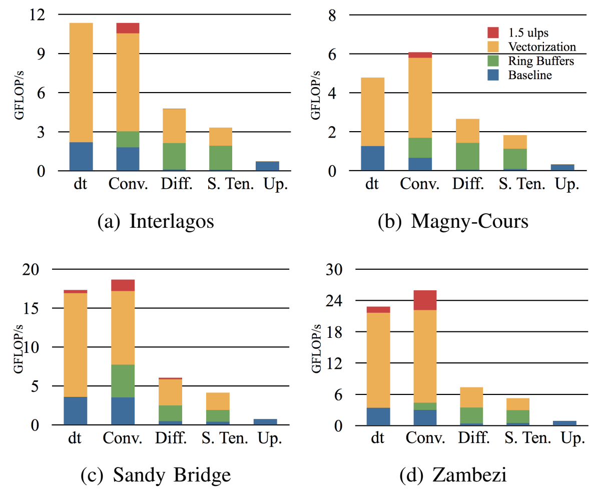

Fig. 4. Performance gain provided by the different optimization techniques on the core layer.

Tables V(a) and V(b) report the measured performance, the expected range and the fraction of the peak. The expected range is calculated with the roofline model using the AI and the OI of the kernels. Since the ridge point of the considered machines is always higher than the OI of most of the kernels, we do not expect to reach the peak. However unprecedented fractions of the peak performance are achieved: on desktop platforms, convection kernel reaches 48% (for Sandy Bridge) and 32% (for Zambezi) of peak performance. On both server platforms this kernel achieves 34% of the peak. The worst fraction is achieved by the update kernel which reaches only 2% of the peak, as expected from the roofline predictions. Although the update kernel has a lower FLOP count compared to the other kernels, its optimization is necessary since it can take up to 10% of the simulation time. We notice that the surface tension and the diffusion kernels reach similar fractions of peak performance. This behavior is expected as they share most of the computation. Overall, the results confirm that the measured performance for each kernel and platform is within the predicted bounds of the roofline model, except for the update kernel on Interlagos.

Figure 4 depicts the performance gains of individual optimizations on the considered platforms. Organizing the intermediate data in SoA format and low memory size ring buffers provide in general a significant performance gain while enabling fully vectorizable code. For the convection kernel, the performance increase is 50% at best while for memory-bound kernels such as diffusion or surface tension, this factor becomes as high as 10. The use of explicit vectorization provides factors up to 5 (over 8 in single precision, using AVX intrinsics on the Sandy Bridge platform). Vectorization is significantly more effective on the dt and convection kernels as they feature higher FLOP/B ratios. Concerning the lower precision arithmetics, Sandy Bridge benefits the most from 1.5/2.5 ulps with a performance increase of 15% while Magny-Cours and Interlagos gain 5% and 7%, respectively.

TABLE VI
NODE LAYER PERFORMANCE

(a) Server platforms: M=measured performance in GFLOP/s, E=expected maximum in GFLOP/s, %P=fraction of the peak

Kernel	Interlagos			Magny-Cours		
	M	E	%P	M	E	%P
dt	100	304	19	234	487	28
Conv.	174	537	32	279	844	33
Diff.	61	257	11	89	411	11
S. Ten.	45	168	8	66	268	8
Upd.	10	10	2	18	16	2

(b) Desktop platforms: M=measured performance in GFLOP/s, R=predicted range in GFLOP/s, %P=fraction of the peak

Kernel	Sandy Bridge			Zambezi		
	M	E	%P	M	E	%P
dt	85	102	40	39	79	17
Conv.	98	218	45	61	230	26
Diff.	26	86	12	23	77	10
S. Ten.	18	56	8	17	43	7
Upd.	3.4	3.4	2	2.8	3.1	1

B. Node layer

On this layer, we investigate the additional cost incurred by the necessary copying of the ghost values across the blocks and the increased memory footprint in performing a simulation where all the kernels are executed within simulation steps. We perform simulations of SBI and CVT with a maximum problem size of 0.1 billion elements ($512 \times 512 \times 512$) per node.

To allow comparisons, the surface tension kernel is computed on the entire domain (instead of only working on tagged blocks that contain the interface).

TABLE VII
STRONG EFFICIENCY (%) OF THE KERNELS ON THE NODE LAYER FOR SIMULATIONS WITH 0.1 BILLION ELEMENTS

Kernel	Sandy Bridge	Zambezi	Interlagos	Magny-Cours
dt	54	68	78	94
Conv.	55	56	59	94
Diff.	30	44	47	52
S. Ten.	36	51	54	61
Upd.	23	29	32	35

Tables VI(a) and VI(b) show the best measured performance, the maximum expected performance and the fraction of the peak on the considered server and desktop platforms, respectively. The highest measured performance reach 45%, 33%, 32% and 26% fraction of the peak for Sandy Bridge, Magny-Cours, Interlagos and Zambezi, respectively.

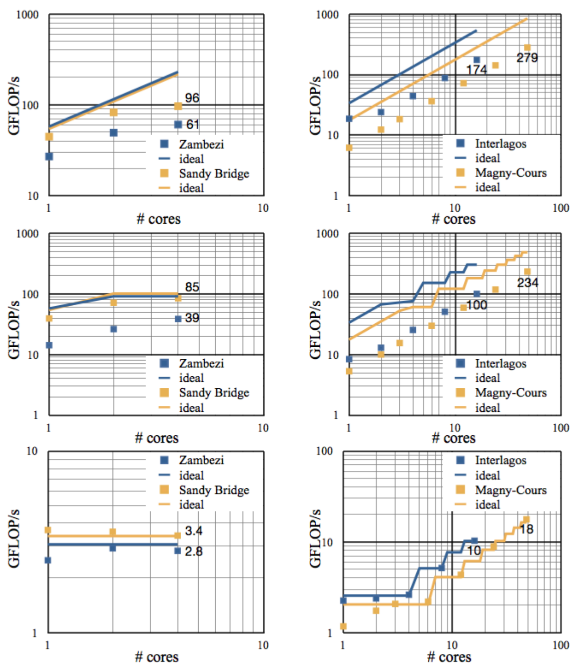

Fig. 5. Weak scaling of the software versus the number of cores on a single node of the desktop platforms (left) and on servers (right) for convection (top), surface tension (middle) and update (bottom) kernels.

Table VII shows the strong efficiency of the individual kernels. This efficiency is computed with respect to the time measurement on a fully subscribed core/module. The best speedups are observed with the dt and convection kernels: 2.2X (over 4), 2.7X (over 4), 12.5X (over 16) and 45X (over 48) for Sandy Bridge, Zambezi, Interlagos and Magny-Cours, respectively. While a good scaling is observed on the server platforms, the strong efficiency on the desktop platforms is relatively poor, assuming values of 50%-70%.

Figure 5 illustrates the weak scaling of the convection, update and surface tension kernels. The ideal scaling is calculated with the roofline model, by considering the aggregate memory bandwidth of the active NUMA nodes. The dt and diffusion kernels behave very similar to the convection and surface tension kernels and therefore are not shown here.

From Figure 5, we observe a good scaling on the server platforms for all the kernels. However the ideal performance of the surface tension kernel does not scale with the number of cores, but rather with the number of NUMA nodes. This behavior is even more pronounced for the update kernel, caused by low FLOP/B ratios. The poor strong scaling of desktop platforms is also reflected in the weak scaling study. In Figure 5 (bottom, left) we note that the performance is slightly above the roofline prediction for a small number of threads on Sandy Bridge. We attribute this to a caching effect in the LLC.

For the simulation of SBI, the time spent in one time step is measured to be 11.6 seconds with 87% of the time spent in the convection kernel, 11% in the surface tension kernel, 1% in the update and the dt kernel.

C. Cluster layer

On this layer, we investigate the overhead incurred by the inter-node communication and the increased synchronization costs. We also want to quantify how effective are the different C/T overlap strategies considered in this work. Moreover we are interested in identifying possible performance bottlenecks when the software is scaled up to thousands of cores/nodes. The benchmark consists of a simulation of SBI with 9 GB of memory footprint per compute node. Figure 6 illustrates the volume rendered density field of this simulation at M=3 at non-dimensional time $\tilde{t} = 2$. The shock passage compresses the bubble and deposits vorticity on the air/helium interface. The counter-rotating vorticity advects the bubble downstream and turns it inwards forming a primary vortex ring (PVR) and a smaller secondary vortex ring (SVR). The time spent in one simulation step is measured to be 13.3 seconds with 73% of the time spent in the convection kernel, 18% in the surface tension kernel, 7% in the update kernel and 1% in the dt kernel.

On 1482 compute nodes, the simulation features a system size of 9’728×6’656×3’072 cells, totaling 198 billion computational elements. Within this benchmark, we consider three different alternatives for the application placement and the C/T overlap: 1 MPI process per node and OpenMP-based software on the node layer, one MPI process per NUMA node and OpenMP-based software on the node layer, and one MPI process per NUMA node and TBB-based software on the node layer (featuring the genuine C/T overlap).

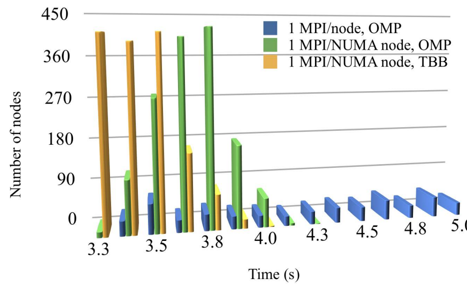

Fig. 7. Distribution of the time spent in the convection kernel using 1 MPI per node and OpenMP (blue), 1 MPI per NUMA node and OpenMP (green) and using 1 MPI per NUMA node and TBB (orange) C/T overlap strategies. Every other set is labeled.

On this layer, we could not effectively use the roofline model for precise performance predictions as it is not clear which bandwidth to consider (*e.g.* bisection and injection bandwidths). Weak scaling plots in Figure 8 (top) show an achieved performance of 0.24 PFLOP/s for the convection kernel, corresponding to an unprecedented 30% fraction of the peak. On the cluster layer, this kernel becomes memory-bound since the memory bandwidth is limited by the network interconnect. For this kernel, the achievable fraction of the peak is estimated to be 85%. The surface tension kernel achieves only 5% of the peak, out of the expected 6%. The scaling on the cluster layer is almost perfect for all the three evaluations and no relevant bottlenecks are identified. The best

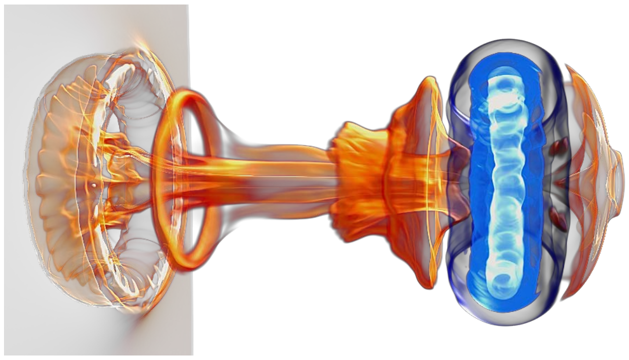

Fig. 6. Volume rendering of the density field of SBI at M=3 and $\tilde{t} = 2$ showing the PVR and the SVR. High/low density is shown in orange/blue.

C/T overlap strategy is found to be the one implemented with TBB, which outperforms the alternative strategies by up to 10%. As this improvement is barely visible in the log-log plot of Figure 8, we report in Figure 7, an histogram of the distribution of time spent in the convection kernel per MPI process for an execution on 1482 compute nodes for the different C/T overlap strategies, where the slowest samples define the performance of the kernel. A first observation is

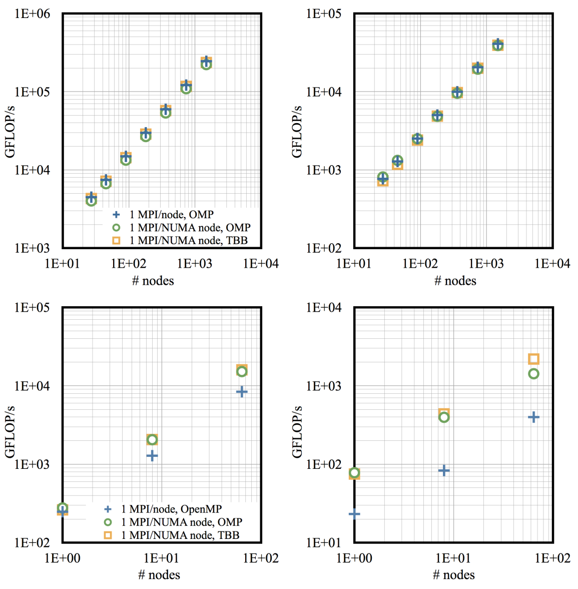

Fig. 8. Weak scaling of the software for the simulation of SBI versus the number of nodes on the cluster of Interlagos (top) and Magny-Cours (bottom) for the convection (left) and surface tension (right) kernels.

that thread binding effectively decrease the execution time and its associated variance (clearly shown by the differences between using 1 MPI process per node and the OpenMP implementation and using 1 MPI process per NUMA node with the same OpenMP code). A second observation is that employing C/T overlap by means of TBB further decreases the average computing time.

Figure 8 (bottom) also shows the weak scaling plots for the convection and surface tension kernels on the cluster of Magny-Cours, using OpenMP and TBB and up to 64 compute nodes. Best measured performance is 16 TFLOP/s for the convection kernel, corresponding to roughly 30% of the nominal peak. For this kernel the maximum achievable performance is estimated to be 55%. No scaling issues are observed for the convection kernel, whereas the surface tension kernel suffers from performance degradation when OpenMP is used on the node layer.

VII. Conclusion

We presented an open source, object-oriented software for high throughput direct numerical simulations of compressible, vortical, two-phase flows. We used the roofline model to a-priori predict the achievable performance of the three software layers. The measured performance have been found to be in good agreement with the estimated range.

We have reported simulations that achieve unprecedented fractions of the peak on the recently released CPU micro-architectures, namely Bulldozer and Sandy Bridge. For single-node simulations, the best performance on Bulldozer has been measured to be 32% of the peak, and 45% on Sandy Bridge.

On the CSCS Monte Rosa supercomputer, the software has proved to reach 0.24 PFLOP/s, corresponding to 30% of the nominal peak for the simulation of shock-bubble interaction. The observed performance makes this software a suitable benchmark for AMR-based solvers.

Furthermore, due to the block-based nature of the software, the core and node layers can serve as a foundation for building effective AMR and wavelet adapted compressible flow solvers [14]. The software is released as open source on GitHub: `git://github.com/cselab/CUBISM-MPCF.git`. Ongoing work include the addition of anti-diffusion schemes based on [67], [68] to prevent the numerical diffusion on the interface, the implementation of a wavelet-adapted solver based on this work and the use of the software developed to study physical problems such as bubble collapse and compressible vortex reconnections.

Acknowledgment

The authors would like to thank Mauro Bianco of the Swiss National Supercomputing Center (CSCS) and Dr. Olivier Byrde, Adrian Ulrich, Teodoro Brasacchio and Eric Müller of Brutus cluster at ETH Zurich for their exceptional and crucial assistance.

REFERENCES

[1] D. Ranjan, J. Oakley, and R. Bonazza, “Shock-bubble interactions,” *Annual Review of Fluid Mechanics*, vol. 43, no. 1, pp. 117–140, 2011.

[2] J. Haas and B. Sturtevant, “Interaction of weak shock-waves with cylindrical and spherical gas inhomogeneities,” *Journal of Fluid Mechanics*, vol. 181, pp. 41–76, Aug 1987.

[3] J. Picone and J. Boris, “Vorticity Generation by Shock Propagation Through Bubbles in a Gas,” *Journal of Fluid Mechanics*, vol. 189, pp. 23–51, Apr 1988.

[4] M. J. Berger and J. Oliger, “Adaptive mesh refinement for hyperbolic partial differential equations,” *J. Comput. Phys.*, vol. 53, no. 3, pp. 484–512, 1984.

[5] J. Quirk and S. Karni, “On the dynamics of a shock-bubble interaction,” *Journal of Fluid Mechanics*, vol. 318, pp. 129–163, 1996.

[6] A. Marquina and P. Mulet, “A flux-split algorithm applied to conservative models for multicomponent compressible flows,” *Journal of Computational Physics*, vol. 185, no. 1, pp. 120 – 138, 2003.

[7] J. M. Stone and M. L. Norman, “The three-dimensional interaction of a supernova remnant with an interstellar cloud,” *APJL*, vol. 390, pp. L17–L19, May 1992.

[8] D. Ranjan, J. H. J. Niederhaus, J. G. Oakley, M. H. Anderson, J. A. Greenough, and R. Bonazza, “Experimental and numerical investigation of shock-induced distortion of a spherical gas inhomogeneity,” *Physica Scripta Volume T*, vol. 132, no. 1, p. 014020, Dec. 2008.

[9] D. Virk, F. Hussain, and R. Kerr, “Compressible vortex reconnection,” *Journal of Fluid Mechanics*, vol. 304, pp. 47–86, Dec 10 1995.

[10] J. M. Ortega, R. L. Bristol, and O. Savas, “Experimental study of the instability of unequal-strength counter-rotating vortex pairs,” *Journal of Fluid Mechanics*, vol. 474, pp. 35–84, 2003.

[11] C. Burstedde, O. Ghattas, M. Gurnis, T. Isaac, G. Stadler, T. Warburton, and L. Wilcox, “Extreme-scale amr,” in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC ’10. IEEE Computer Society, 2010, pp. 1–12.

[12] K. Schneider and O. V. Vasilyev, “Wavelet methods in computational fluid dynamics*,” *Annual Review of Fluid Mechanics*, vol. 42, no. 1, pp. 473–503, 2010.

[13] D. Rossinelli, C. Conti, and P. Koumoutsakos, “Mesh-particle interpolations on graphics processing units and multicore central processing units,” *Royal Society of London Philosophical Transactions Series A*, vol. 369, pp. 2164–2175, 2011.

[14] B. Hejazialhosseini, D. Rossinelli, M. Bergdorf, and P. Koumoutsakos, “High order finite volume methods on wavelet-adapted grids with local time-stepping on multicore architectures for the simulation of shock-bubble interactions,” *J. Comput. Phys.*, vol. 229, no. 22, pp. 8364–8383, Nov. 2010.

[15] K. Cameron, R. Ge, and X. Feng, “High-performance, power-aware distributed computing for scientific applications,” *Computer*, vol. 38, no. 11, pp. 40 – 47, 2005.

[16] J. A. Greenough, B. R. De Supinski, R. K. Yates, C. A. Rendleman, D. Skinner, V. Beckner, M. Lijewski, and J. Bell, “Performance of a block structured, hierarchical adaptive mesh refinement code on the 64k node ibm bluegene/l computer,” *Computer*, pp. 1–12, 2005.

[17] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, J. W. Truran, and H. Tufo, “Flash: An adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes,” *The Astrophysical Journal Supplement Series*, vol. 131, no. 1, p. 273, 2000.

[18] S. Williams, A. Waterman, and D. Patterson, “Roofline: an insightful visual performance model for multicore architectures,” *Commun. ACM*, vol. 52, pp. 65–76, 2009. [Online]. Available: http://doi.acm.org/10.1145/1498765.1498785

[19] P. Colella, D. T. Graves, T. J. Ligocki, D. F. Martin, D. Mondiano, D. B. Serafini, and B. Van Straalen, “Chombo software package for amr applications design document,” Lawrence Berkeley National Laboratory, Tech. Rep., 2003.

[20] T. Wen, J. Su, P. Colella, K. Yelick, and N. Keen, “An adaptive mesh refinement benchmark for modern parallel programming languages,” in *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, ser. SC ’07. ACM, 2007, pp. 1–12.

[21] R. T. Fisher, L. P. Kadanoff, D. Q. Lamb, A. Dubey, T. Plewa, A. Calder, F. Cattaneo, P. Constantin, I. T. Foster, M. E. Papka, S. I. Abarzhi, S. M. Asida, P. M. Rich, C. C. Glendenin, K. Antypas, D. J. Sheeler, L. B. Reid, B. Gallagher, and S. G. Needham, “Terascale turbulence computation using the flash3 application framework on the ibm blue gene/lsystem,” *IBM Journal of Research and Development*, vol. 52, no. 1-2, pp. 127–136, 2008.

[22] A. Dubey, K. Antypas, and C. Daley, “Parallel algorithms for moving lagrangian data on block structured eulerian meshes,” *Parallel Computing*, vol. 37, no. 2, pp. 101 – 113, 2011.

[23] S. M. Couch, D. Pooley, J. C. Wheeler, and M. Milosavljević, “Aspherical supernova shock breakout and the observations of supernova 2008d,” *The Astrophysical Journal*, vol. 727, no. 2, p. 104, 2011.

[24] A. S. Hill, M. R. Joung, R. A. Benjamin, L. M. Haffner, C. Klingenberg, M. M. Mac Low, K. Waagan, and K. A. Wood, “Mhd simulations of a supernova-driven ism and the warm ionized medium,” in *American Astronomical Society Meeting Abstracts No.217*, vol. 43, 2011, p. 251.06.

[25] A. C. Calder, B. Fryxell, T. Plewa, R. Rosner, L. J. Dursi, V. G. Weirs, T. Dupont, H. F. Robey, J. O. Kane, B. A. Remington, R. P. Drake, G. Dimonte, M. Zingale, F. X. Timmes, K. Olson, P. Ricker, P. MacNeice, and H. M. Tufo, “On validating an astrophysical simulation code,” *The Astrophysical Journal Supplement Series*, vol. 143, no. 1, p. 201, 2002.

[26] M. Berzins, J. Luitjens, Q. Meng, T. Harman, C. A. Wight, and J. R. Peterson, “Uintah: a scalable framework for hazard analysis,” in *Proceedings of the 2010 TeraGrid Conference*, ser. TG ’10. ACM, 2010, pp. 3:1–3:8.

[27] Q. Meng, M. Berzins, and J. Schmidt, “Using hybrid parallelism to improve memory use in the uintah framework,” in *Proceedings of the 2011 TeraGrid Conference: Extreme Digital Discovery*, ser. TG ’11. ACM, 2011, pp. 24:1–24:8.

[28] K. Madduri, K. Z. Ibrahim, S. Williams, E.-J. Im, S. Ethier, J. Shalf, and L. Oliker, “Gyrokinetic toroidal simulations on leading multi- and manycore HPC systems,” in *SC*, 2011, p. 23.

[29] S. Williams, L. Oliker, J. Carter, and J. Shalf, “Extracting ultra-scale lattice boltzmann performance via hierarchical and distributed auto-tuning,” in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC ’11. ACM, 2011, pp. 55:1–55:12.

[30] S. Kamil, P. Husbands, L. Oliker, J. Shalf, and K. Yelick, “Impact of modern memory subsystems on cache optimizations for stencil computations,” in *Proceedings of the 2005 workshop on Memory system performance*, ser. MSP ’05. ACM, 2005, pp. 36–43.

[31] S. Kamil, K. Datta, S. Williams, L. Oliker, J. Shalf, and K. Yelick, “Implicit and explicit optimizations for stencil computations,” in *Proceedings of the 2006 workshop on Memory system performance and correctness*, ser. MSPC ’06. ACM, 2006, pp. 51–60.

[32] S. Kamil, C. Chan, L. Oliker, J. Shalf, and S. Williams, “An auto-tuning framework for parallel multicore stencil computations,” in *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, April 2010, pp. 1 –12.

[33] G. Winckelmans and A. Leonard, “Contributions to vortex particle methods for the computation of three-dimensional incompressible unsteady flows,” *Journal of Computational Physics*, vol. 109, no. 2, pp. 247 – 273, 1993.

[34] J. Krueger, D. Donofrio, J. Shalf, M. Mohiyuddin, S. Williams, L. Oliker, and F.-J. Pfreund, “Hardware/software co-design for energy-efficient seismic modeling,” in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC ’11. ACM, 2011, pp. 73:1–73:12.

[35] J. Shalf, D. Quinlan, and C. Janssen, “Rethinking hardware-software codesign for exascale systems,” *IEEE Computer*, vol. 44, no. 11, pp. 22–30, 2011.

[36] G. Chen, L. Chacón, and D. C. Barnes, “An efficient mixed-precision, hybrid CPU-GPU implementation of a fully implicit particle-in-cell algorithm,” *ArXiv e-prints*, Nov. 2011.

[37] C. Conti, D. Rossinelli, and P. Koumoutsakos, “GPU and APU computations of Finite Time Lyapunov Exponent fields,” *Journal of Computational Physics*, vol. 231, no. 5, pp. 2229–2244, 2011.

[38] C.-K. Luk, R. Newton, W. Hasenplaugh, M. Hampton, and G. Lowney, “A Synergetic Approach to Throughput Computing on x86-Based Multicore Desktops,” *IEEE Softw.*, vol. 28, pp. 39–50, 2011.

[39] R. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou, “Cilk - An Efficient Multithreaded Runtime System,” *Sigplan Notices*, vol. 30, no. 8, pp. 207–216, Aug 1995,

5Th Acm Sigplan Symposium On Principles And Practice of Parallel Programming, Santa Barbara, Ca, Jul 19-21, 1995.

[40] R. Blumofe and C. Leiserson, "Scheduling multithreaded computations by work stealing," *Journal of the ACM*, vol. 46, no. 5, pp. 720–748, SEP 1999.

[41] A. Robison, M. Voss, and A. Kukanov, "Optimization via reflection on work stealing in TBB," in *2008 IEEE International Symposium on Parallel & Distributed Processing, Vols 1-8*, IEEE. IEEE, 2008, Proceedings Paper, pp. 598–605.

[42] B. Goglin and N. Furmento, "Enabling high-performance memory migration for multithreaded applications on linux," in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, may 2009, pp. 1 –9.

[43] C. Terboven, D. an Mey, D. Schmidl, H. Jin, and T. Reichstein, "Data and thread affinity in openmp programs," in *Proceedings of the 2008 workshop on Memory access on future processors: a solved problem?*, ser. MAW '08. ACM, 2008, pp. 377–384.

[44] F. Broquedis, N. Furmento, B. Goglin, R. Namyst, and P.-A. Wacrenier, "Dynamic task and data placement over numa architectures: An openmp runtime perspective," in *Evolving OpenMP in an Age of Extreme Parallelism*, ser. Lecture Notes in Computer Science, M. Müller, B. de Supinski, and B. Chapman, Eds. Springer Berlin / Heidelberg, 2009, vol. 5568, pp. 79–92.

[45] M. Puschel, J. Moura, J. Johnson, D. Padua, M. Veloso, B. Singer, J. Xiong, F. Franchetti, A. Gacic, Y. Voronenko, K. Chen, R. Johnson, and N. Rizzolo, "SPIRAL: Code Generation for DSP Transforms," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 232 –275, 2005.

[46] C. W. Antoine, A. Petitet, and J. J. Dongarra, "Automated empirical optimization of software and the atlas project," *Parallel Computing*, vol. 27, p. 2001, 2000.

[47] D. Rossinelli, B. Hejazialhosseini, D. Spampinato, and P. Koumoutsakos, "Multicore/multi-gpu accelerated simulations of multiphase compressible flows using wavelet adapted grids," *SIAM J. Scientific Computing*, vol. 33, no. 2, 2011.

[48] M. Bernaschi, M. Bisson, T. Endo, S. Matsuoka, M. Fatica, and S. Melchionna, "Petaflop biofluidics simulations on a two million-core system," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC'11, no. 4. IEEE Computer Society, 2011, pp. 1–12.

[49] T. Shimokawabe, T. Aoki, T. Takaki, T. Endo, A. Yamanaka, N. Maruyama, A. Nukada, and S. Matsuoka, "Peta-scale phase-field simulation for dendritic solidification on the tsubame 2.0 supercomputer," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11, no. 3. IEEE Computer Society, 2011, pp. 1–11.

[50] N. Maruyama, T. Nomura, K. Sato, and S. Matsuoka, "Physis: An implicitly parallel programming model for stencil computations on large-scale gpu-accelerated supercomputers," in *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, ser. SC'11, no. 11, Nov 2011, pp. 1 –12.

[51] N. G. Dickson, K. Karimi, and F. Hamze, "Importance of explicit vectorization for cpu and gpu software performance," *Journal of Computational Physics*, vol. 230, no. 13, pp. 5383 – 5398, 2011.

[52] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design patterns: Abstraction and reuse of object-oriented design," in *ECOOP' 93 — Object-Oriented Programming*, ser. Lecture Notes in Computer Science, O. Nierstrasz, Ed. Springer Berlin / Heidelberg, 1993, vol. 707, pp. 406–431.

[53] G. Tryggvason, R. Scardovelli, and S. Zaleski, *Direct Numerical Simulations of Gas-Liquid Multiphase Flows*. Cambridge University Press, 2011.

[54] J. Williamson, "Low-Storage Runge-Kutta Schemes," *Journal of Computational Physics*, vol. 35, no. 1, pp. 48–56, 1980.

[55] E. Johnsen and T. Colonius, "Implementation of WENO schemes in compressible multicomponent flow problems," *Journal of Computational Physics*, vol. 219, no. 2, pp. 715–732, Dec 10 2006.

[56] X. Liu, S. Osher, and T. Chan, "Weighted Essentially Nonoscillatory Schemes," *Journal of Computational Physics*, vol. 115, no. 1, pp. 200–212, Nov 1994.

[57] G. Jiang and C. Shu, "Efficient implementation of weighted ENO schemes," *Journal of Computational Physics*, vol. 126, no. 1, pp. 202–228, Jun 1996.

[58] B. Wendroff, "Approximate Riemann solvers, Godunov schemes and contact discontinuities," in *Godunov Methods: Theory and Applications*, Toro, EF, Ed., London Math Soc. 233 Spring St, New York, NY 10013 USA: Kluwer Academic/Plenum Publ, 2001, Proceedings Paper, pp. 1023–1056.

[59] X. Hu and N. Adams, "A multi-phase sph method for macroscopic and mesoscopic flows," *Journal of Computational Physics*, vol. 213, no. 2, pp. 844 – 861, 2006.

[60] R. Saurel and R. Abgrall, "A simple method for compressible multifluid flows," *SIAM Journal on Scientific Computing*, vol. 21, no. 3, pp. 1115–1145, Dec 6 1999.

[61] X. Y. Hu, B. C. Khoo, N. A. Adams, and F. L. Huang, "A conservative interface method for compressible flows," *Journal of Computational Physics*, vol. 219, no. 2, pp. 553–578, Dec 10 2006.

[62] W. Anderson, J. Thomas, and B. Vanleer, "Comparison Of Finite Volume Flux Vector Splittings For The Euler Equations," *AIAA Journal*, vol. 24, no. 9, pp. 1453–1460, Sep 1986.

[63] P. Colella and P. Woodward, "The Piecewise Parabolic Method (PPM) For Gas-Dynamical Simulations," *Journal of Computational Physics*, vol. 54, no. 1, pp. 174–201, 1984.

[64] E. Toro, M. Spruce, and W. Speares, "Restoration of the contact surface in the HLL-Riemann solver," *Shock Waves*, vol. 4, no. 1, pp. 25–34, 1994.

[65] Intel Corporation, *Intel® 64 and IA-32 Architectures Optimization Reference Manual*. Intel Corporation, 2009.

[66] AMD Inc., *Software Optimization Guide for the AMD 15h Family*, 2011.

[67] K. K. So, X. Y. Hu, and N. A. Adams, "Anti-diffusion interface sharpening technique for two-phase compressible flow simulations," *J. Comput. Phys.*, vol. 231, no. 11, pp. 4304–4323, Jun. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.jcp.2012.02.013

[68] R. K. Shukla, C. Pantano, and J. B. Freund, "An interface capturing method for the simulation of multi-phase compressible flows," *J. Comput. Phys.*, vol. 229, no. 19, pp. 7411–7439, Sep. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.jcp.2010.06.025