nSharma: Numerical Simulation Heterogeneity Aware Runtime Manager for OpenFOAM

Roberto Ribeiro[1][0000−0002−6593−4627], Luís Paulo Santos[2][0000−0003−4466−1129], and João M. Nóbrega[3][0000−0002−5303−6467]

[1] Algoritmi Research Center & Dept. of Informatics, University of Minho, Braga, Portugal
rribeiro@di.uminho.pt,
[2] INESC-TEC & Dept. of Informatics, University of Minho, Braga, Portugal
psantos@di.uminho.pt
[3] Institute for Polymers and Composites/I3N, University of Minho, Guimarães, Portugal
mnobrega@dep.uminho.pt

Abstract. CFD simulations are a fundamental engineering application, implying huge workloads, often with dynamic behaviour due to run-time mesh refinement. Parallel processing over heterogeneous distributed memory clusters is often used to process such workloads. The execution of dynamic workloads over a set of heterogeneous resources leads to load imbalances that severely impacts execution time, when static uniform load distribution is used. This paper proposes applying dynamic, heterogeneity aware, load balancing techniques within CFD simulations. nSharma, a software package that fully integrates with OpenFOAM, is presented and assessed. Performance gains are demonstrated, achieved by reducing busy times standard deviation among resources, i.e., heterogeneous computing resources are kept busy with useful work due to an effective workload distribution. To best of authors' knowledge, nSharma is the first implementation and integration of heterogeneity aware load balancing in OpenFOAM and will be made publicly available in order to foster its adoption by the large community of OpenFOAM users.

Keywords: computational fluid dynamics, OpenFOAM, heterogeneous systems, dynamic load balancing

1 Introduction

Computational Fluid Dynamics (CFD) simulations have become a fundamental engineering tool, witnessing an increasing demand for added accuracy and larger problem sizes, being one of the most compute intensive engineering workloads. The most common approaches to CFD, such as Finite Elements (FEs) and Finite Volumes (FVs), entail discretizing the problem domain into cells (or elements) and then solving relevant governing equations for the quantities of interest for each cell. Since each cell's state depends on its neighbours, solvers employ some form of nearest neighbour communication among cells and iterate until some convergence criteria are met. Typically, CFD problems are unsteady, requiring

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_33

an outer loop which progresses through simulation time in discrete steps. Such very compute intensive type of workloads are obvious candidates to exploit the multitude of resources available on parallel processing systems. Domain decomposition is used to make available a suitable degree of parallelism, i.e., the set of discrete cells is partitioned into subsets which can then be distributed among the computational resources.

Currently, the most widely available parallel systems are distributed memory clusters, which provide a cost-effective, extensible and powerful computing resource. A cluster can be fairly easily extended by adding more nodes with identical architectures, but often from newer generations offering more computing capabilities. This extensibility renders the system heterogeneous in the sense that different generations of hardware, with diverse configurations, coexist in the same system. An additional source of heterogeneity is the integration on current supercomputing clusters [16] of devices with alternative architectures, programming and execution models, such as the new highly parallel Intel KNLs and the massively parallel GPUs [5].

However, this heterogeneity results in different performances across nodes, potentially leading to severe load imbalances. Static and uniform workload distribution strategies, as typically used by CFD software, will result on the computational units waiting on each other and resources underutilization. Properly distributing the workload and leveraging all the available computing power is thus a crucial feature, which has been revisited in the latest years due to increasing systems' heterogeneity [10].

The load distribution problem is further aggravated in the presence of dynamic workloads. CFD solvers often refine the problem domain discretisation as the simulation progresses through time, allowing for higher accuracy in regions where the quantities of interest exhibit higher gradients. In the scope of this work, these applications will be referred to as *adaptive applications*. This refinement entails splitting and merging cells, resulting on a new domain discretisation. Given that the computational effort is in general proportional to the number of cells, its distribution across the problem domain also changes. Not accounting for this refinement and maintaining the initial mapping throughout the whole simulation would lead to load imbalances and huge performance losses.

The combination of the differences in computing power provided by the heterogeneous Computing Units (CUs) with the differences in computing requirements from dynamic workloads, results in a *two-fold computing imbalance*. The adoption of Dynamic Load Balancing (DLB) addresses this computing imbalance as a whole and allow for fully leveraging all the available computing power and improve execution time. This work will thus focus in combining DLB with Heterogeneous Systems (HS) in the context of CFD simulations by integrating DLB mechanisms in a widely used application: Open Source Field Operation and Manipulation (OpenFOAM).

OpenFOAM is a free and publicly available open-source software package, specifically targeting CFD applications [14]. It is an highly extensible package, allowing applied science experts to develop scientific and engineering numerical simulations in an expedite manner. OpenFOAM includes a wide range of

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_33

functionalities such as simulation refinement, dynamic meshes, particle simulations, among others. OpenFOAM large set of features and extensibility has made it one of the most used and leading open-source software packages across the CFD community. It has also been made available in multiple supercomputers and computing centres, along with technical support. OpenFOAM parallel distributed memory model is based on a domain decomposition approach, however, there is little to no support for either HS or DLB, which is addressed by this work by integrating and evaluating all proposed mechanisms into this package.

Providing such support is of crucial importance, however, this task is too complex to be handled by the CFD application developer. This complexity has two different causes: i) efficient mapping of the dynamic workload onto a vast set of heterogeneous resources is a research level issue, far from the typical concerns of a CFD expert, and ii) execution time migration of cells (particularly dynamically refined meshes of cells) across memory spaces requires a deep understanding of OpenFOAM's internal data structures and control flow among lower level code functions and methods. Integration of these facilities with OpenFOAM by computer science experts is proposed as the best solution to provide efficiency and robustness, while simultaneously promoting reuse by the CFD community.

This paper proposes nSharma – Numerical Simulation Heterogeneity Aware Runtime Manager – a runtime manager that provides OpenFOAM with heterogeneity aware DLB features. nSharma monitors the heterogeneous resources performance under the current load, combines this data and past history using a performance model to predict the resources behaviour under new workload resulting from the refinement process and makes informed decisions on how to re-distribute the workload. The aim is to minimize performance losses due to workload imbalances over HS, therefore contributing to minimize the simulation's execution time. DLB minimizes idle times across nodes by progressively and in an educated way assigning workload, which can be itself dynamic, to the available resources. nSharma package integrates in a straightforward manner with current OpenFOAM distributions, enabling the adoption of heterogeneity aware DLB. To best of authors' knowledge, this is the first implementation and integration of heterogeneous-aware DLB mechanism in OpenFOAM.

2 Related work

Libraries supporting the development of CFD simulations, include OpenFOAM[14], ANSYS Fluent[2], ANSYS CFX[1], STAR-CCM+[6], among others. OpenFOAM is distributed under the General Public Licence (GPL), allowing modification and redistribution while guaranteeing continued free use. This motivated the selection of OpenFOAM for the developments envisaged in this work. The authors see no reason why this document's higher level assessments and results can not be applied to other similar CFD libraries. This generalization should, however, be empirically verified on a per case basis.

Domain decomposition requires that the mesh discretization is partitioned into sub-domains. This is a challenging task impacting directly on the workload

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_33

associated with each sub-domain and on the volume of data that has to be exchanged among sub-domains in order to achieve global convergence. Frameworks that support mesh-based simulations most often delegate mesh partitioning to a third-party software. ParMETIS [15] and PTSCOTCH [7] are two widely used mesh partitioners, which interoperate with OpenFOAM. ParMETIS has been used within this work's context because it provides a more straightforward support for Adaptive Mesh Refinement (AMR).

ParMETIS includes methods to both partition an initial mesh and re-partition a mesh that is scattered across CUs disjoint memory address spaces, avoiding a potential full re-location of the mesh in runtime. The (re)partitioning algorithms optimize for two criteria: minimizing edge-cut and minimizing element migration. These criteria have been merged into a single user-supplied parameter (ITR), describing the intended ratio of inter-process communication cost over the data-redistribution cost. ParMETIS also provides an interface to describe the relative processing capabilities of the CUs, allowing more work units to be assigned to faster processors. nSharma calculates these parameters in order to control ParMETIS' repartitioning and thus achieve efficient DLB.

Some frameworks providing DLB to iterative applications have been proposed. DRAMA [4] provides a collection of balancing algorithms that are guided by a cost model which aims to reduce the imbalance costs. It is strictly targeted for finite element applications. PREMA [3] is designed to explore an over-decomposition approach to minimize the overhead of stop-and-repartition approaches. This approach is not feasible in some mesh-based numerical simulations (due to, for instance, data dependencies) and no mention to HS support could be found. Zoltan [11] uses callbacks to interface with the application and integrates with DRUM [12], a resource monitoring system based on static benchmark measured in MFLOPS and averaged per node. The resource monitoring capabilities of nSharma are much more suitable to account for heterogeneous computing devices – see next section. Zoltan is not tied to any particular CFD framework. It does not enforce any particular cost functions and uses abstractions to maintain data structure neutrality. This however comes at the cost of requiring the CFD application developer to provide all data definitions and pack/unpack routines, which in a complex application like OpenFOAM is an programming intensive and error prone task.

nSharma integrates with OpenFOAM, accessing its data structures and migration routines. Although this option implies some code portability loss, it avoids the multiple costs of data (and even conceptual) transformations together with overheads of code binding between different software packages. This allows direct exploitation, assessment and validation of DLB techniques for OpenFOAM applications on HS. The results on conceptually more abstract design options, such as the performance model and the decision making mechanism, should still generalise to alternative software implementations, although empirical verification is required.

Some of the above cited works can handle HS. They do so by using high-level generic metrics, such as vendor announced theoretical peak performances or raw counters associated to generic events such as CPU and memory usage

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_33

[13, 12]. The associated performance models are however generic, ignoring both the characteristics of CFD workloads and emerging devices particular execution models and computing paradigms, and thus tend to be inaccurate [8]. This paper proposes a performance model which explicitly combines the workload particularities with the heterogeneous devices capabilities. The design of this performance model is strictly coupled with the requirements of the proposed DLB mechanisms.

FuPerMod [9] explores Functional Performance Models, extending traditional performance models to consider performance differences between devices and between problem sizes. It is based on speed functions built based on observed performances with multiple sizes, allowing the evaluation of a workload distribution [8]. Zhong applied these concepts to OpenFOAM [17] and validated it in multi-core and multi-GPU systems. This paper introduces a similar performance model tightly integrated with the remaining DLB mechanisms.

3 nSharma's Architecture

OpenFOAM simulations are organized as *solvers*, which are iterative processes evaluating, at each iteration, the quantities of interest across the problem domain. Each iteration includes multiple inner loops, solving a number of systems of equations by using iterative linear solvers. Within this work, *solver* refers to OpenFOAM general solvers, rather than the linear solvers. Since OpenFOAM parallel implementation is based on a zero layer domain decomposition over a distributed memory model, the solver's multiple processes synchronize often during each iteration, using both nearest neighbour and global communications.

nSharma is fully integrated into OpenFOAM and organized as a set of components, referred to as modules or models. The *Online Profiling Module (OPM)* acquires information w.r.t. raw system behaviour. The *Performance Model (PM)* uses this data to build an approximation of each CU performance and to generate estimates of near future behaviour, in particular for different workload distributions. The *Decision Model (DM)* decides whether workload redistribution shall happen, based on this higher level information and estimates. The *Repartitioning Module (RM)* handles the details of (re)partitioning subdomains for (re)distribution across multiple processors, while finally load redistribution mechanisms carry on the cells migration among computing resources, therefore enforcing the decisions made by nSharma.

The whole DLB mechanism is tightly coupled with OpenFOAM iterative execution model. This allows nSharma to learn about system behaviour and also allows for progressive convergence towards a globally balanced state - rather than trying to jump to such a state at each balancing episode. Dynamic workloads are also handled by OpenFOAM and nSharma iterative model, with impact on the whole system balanced state and simulation execution time being handled progressively.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_33

3.1 Online Profiling Module

The *OPM* instruments OpenFOAM routines to measure execution times, crucial to estimate the CUs relative performance differences. This has been achieved by thoroughly analysing OpenFOAM workflow and operations, and identifying a set of low-level routines that fundamentally contribute to the application execution time. It has been empirically verified that these times correlate well, enabling nSharma to monitor only the parts of the simulation that are relevant to the associated performance modelling, while simultaneously implying a low instrumentation overhead without any additional analytical models or benchmarking.

The *OPM* API allows for the registration of which routines to measure, and internally refers to these as Operations. Operations are classified as either *IDLE*, representing a synchronization or memory transfer, or *BUSY*, representing a computational task without any synchronizations or memory transfers. This categorization allows to measure performance individually, otherwise execution time would be cluttered by dependencies and communications.

3.2 Performance Model

The *PM* characterizes the system's – and its individual components, such as each CU – performance and provides estimates of future performances under different workload distributions. Workload and performance characterization requires the definition of a work unit, upon which problem size can be quantified. OpenFOAM uses Finite Volumes, with the problem domain discretisation being based on cells that are combined to define the computational domain. With this approach problem size is often characterized by the number of cells, which is, therefore, the work unit used by nSharma.

Each CU performance is characterized by the average time required to process one work unit, denoted by $TperCell_p$ (where p indexes the CUs). For each iteration i and CU p, the respective performance index ($TperCell_p^i$) is given by the ratio of the iteration's total busy time over the number of cells assigned to p, N_p^i: $TperCell_p^i = Tbusy_p^i / N_p^i$. The actual metric used for balancing decisions, $TperCell_p$, is a weighted average over a window of previous iterations, which smooths out outliers and, for dynamic workloads, takes into account different problem sizes (different numbers of cells assigned to each CU at each iteration).

Execution time estimates for arbitrary workload distributions, T_p, use the above described metric multiplied by the number of work units to assign to each CU, N_p, as given by Equation 1 – with P being the number of CUs.

$$T_p = TperCell_p \times N_p, \quad \forall p \in 0, 1, ..., P-1 \tag{1}$$

3.3 Decision Module

It is the *DM* role to assess the system balancing state and decide whether a load redistribution step should take place. It is also the *DM* who decides what load to redistribute. Assessing and making such decision is referred to as a *balancing*

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_33

episode. Since these episodes represent an overhead, it is crucial to decide when should they occur. nSharma allows them only at the beginning of a solver iteration, and defines a period, expressed in number of iterations, for their frequency. The unpredictability of dynamic workloads makes it unpractical to define an optimal balancing period, therefore it is auto-tuned in execution time, as described below.

At the beginning of a new solver's iteration i, the Relative Standard Deviation (RSD), among the CUs busy times for the previous iteration $i-1$ is calculated $\text{RSD}^{i-1} = \sigma^{i-1}/|\overline{T_{busy}{}^{i-1}}| * 100$; standard deviation, σ, is well known as a good, light-weight, indicator of a system's balancing state. A linear regression is then computed over the last few iterations RSD in order to estimate its rate of change, which is used to update the period. Also, a normalization of the magnitude of the RSD is added to the contribution to update the period. Therefore, the load balancing period is adjusted based on how fast the system's balancing state changes and how much it changes.

When a load balancing episode is triggered the *DM* will compute, for each CU p, how many cells, N_p^*, to assign to it. It will devise a new load distribution, where all CUs will take, the same amount of time to process the assigned work units, according to each CU execution rate, $TperCell_p$. Since the total number of cells N is known, a well-determined system of P linear equations can be formulated (see Equation 2) and solved to find $N_0^*, \ldots, N_{P-1}^*$ – the number of cells to assign to each CU.

$$\begin{cases} TperCell_0 \times N_0^* = TperCell_1 \times N_1^* \\ TperCell_1 \times N_1^* = TperCell_2 \times N_2^* \\ \quad \ldots \\ TperCell_{P-2} \times N_{P-2}^* = TperCell_{P-1} \times N_{P-1}^* \\ N_0^* + N_1^* + \ldots + N_{P-1}^* = N \end{cases} \tag{2}$$

After computing this new distribution, a decision has to be made as to whether it will be applied or not, by taking into account the cells migration cost, m. The goal is that the remaining simulation execution time after the load redistribution must be smaller than not migrating. The next iteration i expected execution time without load redistribution is given by $t_i = \max_p(TperCell_p \times N_p)$, whereas with the new load distribution it is $t_i^* = TperCell_p \times N_p^*, \forall p$ (no need for max because t_i^* is the same for all p, according to Equation 2). Let n be the number of remaining iterations and δ represent some additional execution overheads independent on workload redistribution. Then the condition $n \times t_i + \delta > m + n \times t_i^* + \delta$ expresses that migration will only take place if it is expected to reduce the total remaining execution time, while taking into account the cost of actually enforcing the migration m. This cost is estimated by keeping track of the costs of previous migrations and using a linear regression to estimate the cost of any arbitrary decomposition.

$$t_i > \frac{m}{n} + t_i^* \tag{3}$$

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_33

System	SeARCH			Stampede2
Nodes	Tag **641** - Ivy Bridge E5-2650v2 @ 2.60GHz, **16** cores p/node			Tag **KNL7250** - Intel Xeon Phi 7250 @ 1.4GHz ("Knights Landing"), **68** cores p/ node
	Tag **662** - Ivy Bridge E5-2695v2@ 2.40GHz, **24** cores p/node			
	Tag **421** - Nehalem E5520 @ 2.27GHz, **8** cores p/node			
	Tag **KNL7210** - Intel Xeon Phi 7210 @ 1.3GHz, **64** cores p/ node			
Multi-node configurations	**Homogeneous**	**Heterogeneous I**	**Heterogeneous II**	**Homogeneous**
	Multiple 641's	Pair(s) of 641+421	Pair 662+KNL7210	Multiple KNL7250's
Network	Myrinet (**myri**)	Myrinet (**myri**)	Ethernet(**eth**)	Intel Omni-Path (**OPA**)

Table 1. Computing systems and system configurations used in evaluation

Equation 3 (a simplification of the condition equation above) makes it clear that a load redistribution should only be enforced if the cost of migrating cells can be properly amortized across the remaining n iterations. Consequently, towards the end of the simulation, as n gets smaller, the cells migration impact on execution times is progressively higher and load redistribution will become proportionally less likely.

3.4 Repartitioning Module

nSharma repartitioning module interfaces with ParMETIS (see Section 2), by carefully parametrising the relevant methods and by extending some functionality. ParMETIS' repartitioning method is used, which takes into account the current mesh distribution among CUs and balances cells' redistribution cost with the new cells' partition communication costs during the parallel execution of the next iterations. The relationship between these two costs is captured by the ITR parameter. nSharma learns this parameter by requesting multiple decompositions with different ITR values in initial iterations, assessing the most effective ones and converging to a single one. Besides ITR, this method also receives a list of each CU relative computing power, given by $\omega_p = N_p^*/N$, as evaluated by the Decision module (Section 3.3).

OpenFOAM does not natively support migration of refined meshes, which required integrating such support (based on Kyle Mooney's approach, see Acknowledgements). Since each refined cell is always a child of a single original (non-refined) cell and since the refined hierarchy is explicitly maintained, partitioning is applied to the original (non-refined) coarse mesh; after partitioning, the refined mesh is considered to perform migration. To ensure that the original non-refined coarse mesh reflects the correct workload, weights for each coarse cell are provided to ParMETIS based on the number of child cells, which will be used by ParMETIS in devising new partitions.

4 Results Analysis

For experimental validation, the *damBreak* simulation was selected among those distributed with OpenFOAM tutorials. It uses the *interDyMFoam* solver to simulate the multiphase flow of two incompressible fluids – air and water – after the break of a dam. Adjustable time step was disabled and the nSharma configuration dictionary was introduced – all other parameters are the same as distributed

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_33

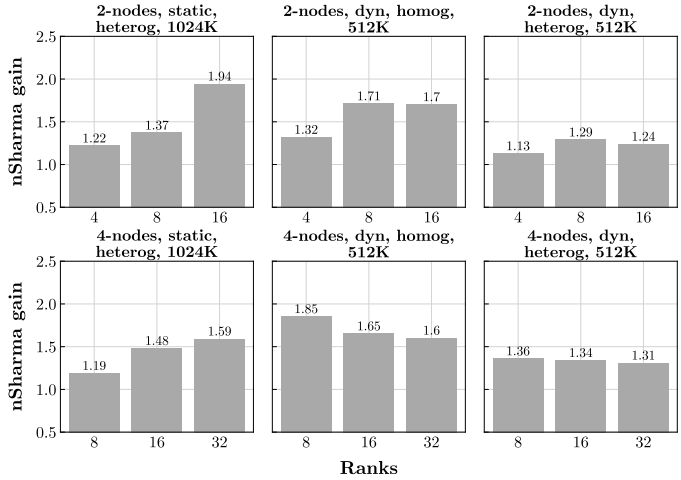

Fig. 1. nSharma gain with SeARCH Homogeneous and Heterogeneous I

in the package. For dynamic workloads, AMR subdivides a cell into 8 new cells according to the interface between the water and air; cells will thus be refined (and unrefined) following the evolution of the two phases' interface.

This solver requires frequent local and global communications. As the degree of parallelism is increased, more sub-domains are created, increasing the number of cells in sub-domains boundaries and, consequently, increasing communications among sub-domains, with network bandwidth and latency impacting significantly in the simulation's performance.

Four hardware configurations were used from two different clusters – SeARCH cluster (Universidade do Minho, Portugal) and Stampede2 (Texas Advanced Computing Center, USA). Configurations are described in Table 1. OpenFOAM 2.4.0 was used, compiled with GNU C Compiler in SeARCH and with Intel C Compiler in Stampede2. Each MPI process is associated to one CU or processing core: the number of used cores is equivalent to the number of processes. MPI terminology refers to processes as ranks, and this terminology is maintained throughout this section.

4.1 Performance Gain

Performance gain is hereby defined as the reduction in execution time achieved by using nSharma and quantified as the ratio between the execution times without and with nSharma, respectively. Figure 1 illustrates such gain for 200 iterations of the damBreak simulation in SeARCH. The first row depicts results obtained with 2 nodes, the second row results obtained with 4 nodes. Results in the first column were obtained with a static workload and problem size of 1024K cells (Heterogeneous I configuration), whereas in the second and third columns dynamic workloads were used with 512K cells (Homogeneous and Heterogeneous I configurations, respectively).

nSharma achieves a significant performance gain for all experimental conditions. For static workloads, the gain increases with the number of ranks, with a

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_33

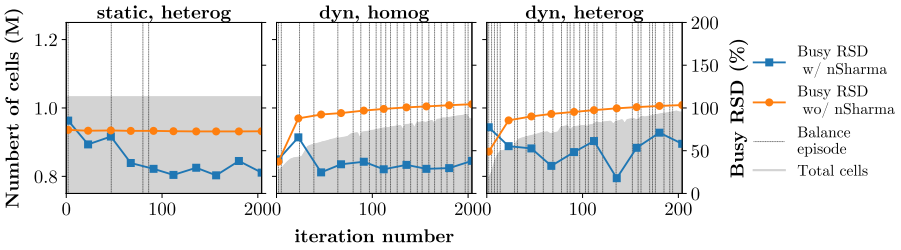

Fig. 2. Busy RSD with and without nSharma for 4 nodes and 32 ranks.

maximum gain of 1.94 gain with 2 nodes and 16 ranks and 1.59 with 4 nodes and 32 ranks. This gain is basically a consequence of nSharma's heterogeneous awareness, which allows remapping more cells to the 641 more powerful cores, which would otherwise be waiting for the 421 processing cores to finish execution.

For homogeneous hardware and dynamic workloads (second column), performance gain is due to moving cells from overloaded cores to underloaded ones, with such fluctuations due to AMR. Significant gains are still observed for all experimental conditions, but this gain suffers a slight decreases as the number of ranks increases for 4 nodes. This is due to an increase in migration and repartitioning costs (see Figure 3), proportional to the increased number of balance episodes required in a dynamic workload scenario (see Figure 2). The communication overheads also increase from 2 to 4 nodes sustaining more sub-domains and more communications over a limited bandwidth network.

The last column illustrates the combination of dynamic workload with HS. The gain is mostly constant with the number of ranks. It is lower than with static workloads or homogeneous hardware, because the decision making process is much more complex requiring a much higher level of adaptability, i.e more frequent balancing episodes and larger volumes of data migration (see Figures 3 and 2).

Figure 2 illustrates the accumulated busy RSD (as described in Section 3.3) with and without nSharma for the same experimental conditions, 4 nodes and 32 ranks. The grey area represents the total number of cells and the vertical lines are balance episodes. Clearly nSharma results in a large RSD reduction, i.e. reduced busy times variation across ranks, thus enabling significant performance gains. This can be clearly seen around iteration number 50 for the static case, where a large RSD reduction occurs.

Figure 3 illustrates, for the 4 nodes cases of Figure 1, the percentage of execution time spent in different algorithmic segments: *Profiler* represents time used by the OPM, *nSharma* time for decision making, *parMetis* represents repartitioning, *redistribute* is cells migration cost and *simulation* represents the time dedicated to the actual simulation. The side slim bars represent the performance gain, which is the same as in Figure 1. The vertical axis goes up to only 20%, the remaining 80% are simulation time and add-up to the illustrated.

The overheads associated with profiling and decision making are negligible in all experimental conditions. Repartitioning (ParMETIS) and redistribution costs increase with the number of ranks. The former is negligible as a percentage

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_33

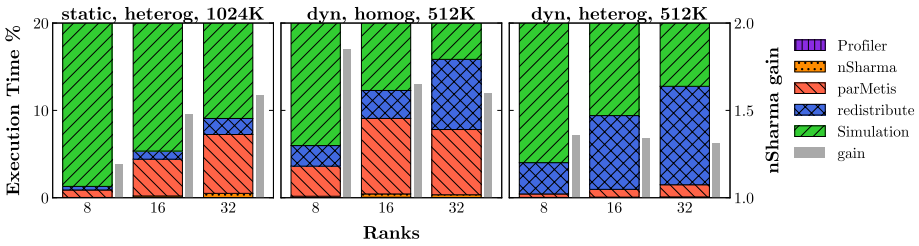

Fig. 3. Execution time percentage breakdown for 4 nodes

for the dynamic plus heterogeneous case, but the latter represents an increasing overhead in all cases. This is tightly related to the fact that the numbers of migrated cells and balancing episodes (see Figure 2) increase with the hardware configuration and the workload complexities (homogeneous versus heterogeneous and static versus dynamic, respectively). Nevertheless the overheads associated with DLB are below 15%, allowing for very significant performance gains.

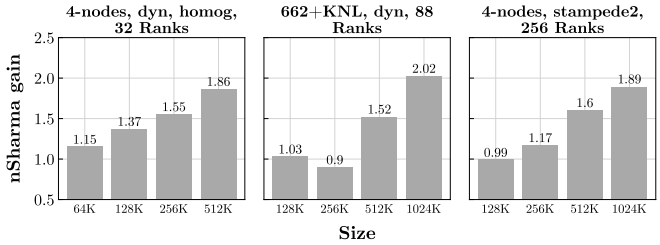

Fig. 4. Increasing problem size for four 641 SeARCH nodes, 662+KNL and four Stampede2 nodes

Figure 4 presents nSharma performance gain for dynamic workload, 4 nodes, fixed number of ranks and increasing problem size for 3 alternative hardware configurations: SeARCH homogeneous, SeARCH Heterogeneous II and Stampede2 homogeneous (see Table 1). The x-axis shows the rank count (particularly, for 662+KNL configuration, 64 plus 24 ranks are used from KNL and 662 respectively), which corresponds to the use of all available CUs. The performance gain associated with the introduction of DLB increases consistently with the problem size. Larger problems have the potential to exhibit more significant imbalance penalties with dynamic workloads, due to larger local fluctuations in the number of cells. nSharma is capable to effectively handle this increased penalty, becoming more efficient as the problem size increases. Based on the observed data, this trend is expected to continue. No inflection point should be reached and nSharma performance gain will keep increasing with the workload, i.e. exactly when the potential for load imbalances becomes higher.

4.2 Efficiency Gain

Strong and weak scalability based on parallel efficiency are evaluated in this section. Parallel efficiency is evaluated with respect to the timing results achieved with only 1 rank and without nSharma (DLB is senseless for a single rank).

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_33

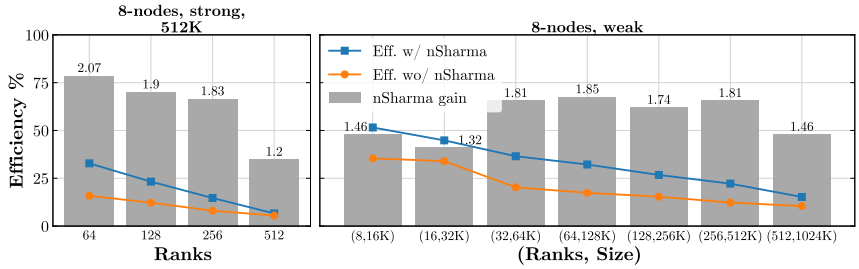

Fig. 5. Efficiency (w/ and wo/ nSharma) with dynamic loads for Stampede2 nodes

Figure 5 presents performance gain with nSharma (grey bars) and parallel efficiency with and without nSharma (blue and orange lines), using 8 KNL nodes of Stampede2 (up to 512 ranks). For the strong scaling case – left plot – nSharma performance gain is around 2, except for 512 ranks. In this latter case, the workload per rank is so low (the number of cells ranges from 1000 to 2000 per rank) that incurred overheads (partitioning and cells migration) significantly impact on the load redistribution benefits. For the weak scaling case – right plot – problem size increases at the same rate as number of ranks, thus the workload per rank is kept constant; performance gain is quite consistent, since increasing DLB costs are compensated by the added workload.

The scalability curves in Figure 5 illustrate that OpenFOAM without DLB exhibits very low efficiency even for increasing problem size. Two major penalties contribute to this: aforementioned parallel communications costs and load imbalance due to dynamic workloads. nSharma addresses the load imbalance penalty in a very effective manner, roughly doubling efficiency for most configurations – the (512,512K) case of strong scalability can not be taken into account due to the very scarce load per rank. This clearly illustrates that introducing DLB mechanisms results in a very significant reduction of execution time, sustained by an increase in efficiency, i.e. a better utilization of the parallel computing resources.

4.3 Heterogeneity and Dynamic Load Balancing

Effective exploitation of the raw computing capabilities available on heterogeneous systems is hard, with load balancing being one of the main challenges, specially for dynamic workloads.

Figure 6 details the performance speedup when combining a KNL node – with two different core configurations, one with the full 64 cores (*knl*) and another with only 32 cores (*half-knl*) – with a 24-core 662 node. Speedup is illustrated w.r.t to the execution time obtained with the node 662 for static (left) and dynamic (right) workloads. By adding a KNL node to a 662 node (*662+knl* and *662+half-knl*) yields no significant performance gain, with a severe deterioration for the dynamic workloads. This is due the imbalance introduced by the large computing power differences between the nodes (as illustrated by the white bars). By enabling nSharma, the whole system capabilities will be assessed and

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_33

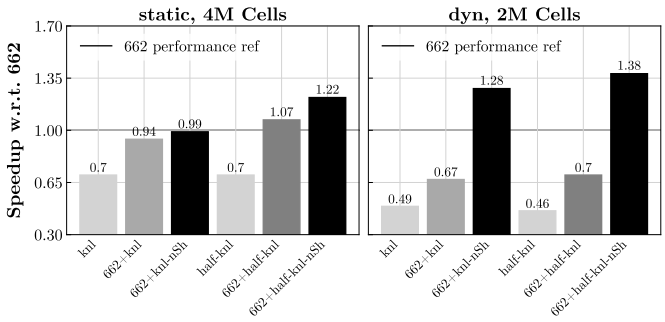

Fig. 6. Speedup in combining a 662 node and a KNL by using nSharma

more load is assigned to 662 node, reducing its idle time and increasing resource utilization. Performance gains between 22% to 38% are observed (*-*nSh* bars). The gain is more substantial with dynamic workloads where the potential for load imbalances is larger: heterogeneous resources plus execution time locally varying number of cells. nSharma works at its best under these more challenging conditions, effectively rebalancing the workload and efficiently exploiting the available resources.

5 Conclusions and Future work

This paper proposes and assesses the integration of heterogeneity aware DLB techniques on CFD simulations running on distributed memory heterogeneous parallel clusters. Such simulations most often imply dynamic workloads due to execution time mesh refinement. Combined with the hardware heterogeneity such dynamics cause a two-fold load imbalance, which impacts severely on system utilization, and consequently on execution time, if not appropriately catered for. The proposed approach has been implemented as a software package, designated nSharma, which fully integrates with the latest version of OpenFOAM.

Substantial performance gains are demonstrated for both static and dynamic workloads. These gains are shown to be caused by reduced busy times RSD among ranks, i.e. computing resources are kept busy with useful work due to a more effective workload distribution. Strong and weak scalability results further support this conclusion, with nSharma enabled executions exhibiting significantly larger efficiencies for a range of experimental conditions. Performance gains increase with problem size, which is a very desirable feature since the potential to load imbalances under dynamic loads grows with the number of cells.

Experimental results show that performance gains associated with nSharma are affected by increasing the number of ranks for larger node counts. This is due to inherent increase of load migration costs associated with a growing number of balancing episodes. Future work will necessarily imply addressing this issue, to allow for increased number of parallel resources by further mitigating load migration overheads. Additionally, nSharma will be validated against a more extensive set of case studies and heterogeneous devices; upon successful

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_33

validation it will be made publicly available in order to foster its adoption by the large community of OpenFOAM users.

Acknowledgements

The authors would like to thank the financial funding by FEDER through the COMPETE 2020 Program, the National Funds through FCT under the projects UID/CTM/50025/2013. The first author was partially funded by the PT-FLAD Chair on Smart Cities & Smart Governance and also by the School of Engineering, University of Minho within project *Performance Portability on Scalable Heterogeneous Computing Systems*. The authors also wish to thank Kyle Mooney for making available his code supporting migration of dynamically refined meshes, as well as acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing HPC resources.

References

1. ANSYS: ANSYS CFX Users' Guide (2017)
2. ANSYS: ANSYS Fluent User's Guide (2017)
3. Barker, K., et al.: A Load Balancing Framework for Adaptive and Asynchronous Applications. IEEE Trans. Parallel Distrib. Syst. (2004)
4. Basermann, A., et al.: Dynamic load-balancing of finite element applications with the DRAMA library. Appl. Math. Model. (2000)
5. Brodtkorb, A.R., et al.: State-of-the-art in Heterogeneous Computing. Sci. Program. (2010)
6. CD-adapco: STAR-CCM+ (2017)
7. Chevalier, C., Pellegrini, F.: PT-Scotch: A tool for efficient parallel graph ordering. Parallel Comput. (2008)
8. Clarke, D., Lastovetsky, A., Rychkov, V.: Dynamic Load Balancing of Parallel Computational Iterative Routines on Highly Heterogeneous Hpc Platforms. Parallel Process. Lett. (2011)
9. Clarke, D., et al.: FuPerMod: a software tool for the optimization of data-parallel applications on heterogeneous platforms. J. Supercomput. (2014)
10. Da Costa, G., et al.: Exascale Machines Require New Programming Paradigms and Runtimes. Supercomput. Front. Innov. (2015)
11. Devine, K., et al.: Design of dynamic load-balancing tools for parallel applications. Proc. 14th Int. Conf. Supercomput. - ICS '00 (2000)
12. Faik, J., et al.: A model for resource-aware load balancing on heterogeneous clusters. Compute (2005)
13. Martínez, J.A., Garzón, E.M., Plaza, A., García, I.: Automatic tuning of iterative computation on heterogeneous multiprocessors with ADITHE. J. Supercomput. (2011)
14. OpenFOAM Foundation: OpenFOAM Users' Guide. Tech. rep. (2018)
15. Schloegel, K., Karypis, G., Kumar, V.: Multilevel Diffusion Schemes for Repartitioning of Adaptive Meshes. J. Parallel Distrib. Comput. (1997)
16. Top500: TOP500 Supercomputer Site (2017)
17. Zhong, Z.: Optimization of Data-Parallel Scientific Applications on Highly Heterogeneous Modern HPC Platforms. Ph.D. thesis (2014)

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_33