

Grant agreement no. 675451

CompBioMed

Research and Innovation Action

H2020-EINFRA-2015-1

Topic: Centres of Excellence for Computing Applications

D2.2 Report on Deployment of Deep Track Tools and Services to Improve Efficiency of Research and Facilitating Access to CoE Capabilities

Work Package: 4

Due date of deliverable: Month 27

Actual submission date: December 23, 2018

Start date of project: October 01, 2016 Duration: 36 months

Lead beneficiary for this deliverable: *EPCC, UEDIN*

Contributors: Bull/Atos, UCL, UvA, UOXF, SARA

Project co-funded by the European Commission within the H2020 Programme (2014-2020)		
Dissemination Level		
PU	Public	YES
CO	Confidential, only for members of the consortium (including the Commission Services)	
CI	Classified, as referred to in Commission Decision 2001/844/EC	

Disclaimer

This document's contents are not intended to replace consultation of any applicable legal sources or the necessary advice of a legal expert, where appropriate. All information in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose. The user, therefore, uses the information at its sole risk and liability. For the avoidance of all doubts, the European Commission has no liability in respect of this document, which is merely representing the authors' view.

PU

Page 1

Version 0.3

"This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Grant Agreement No 675451"

Table of Contents

1	Version Log	4
2	Contributors	4
3	Acronyms and Definitions	5
4	Executive Summary	7
5	Introduction.....	7
6	Access to CompBioMed HPC systems.....	8
6.1	The CompBioMed HPC systems	8
6.2	How to get Access	8
7	Tools and Services to Improve Efficiency	9
7.1	Debugging tools.....	10
	Valgrind: Memory debugging	10
	Padb - parallel application debugger	10
	Cray ATP	10
	DDT Debugger	11
	TotalView	11
	GDB (GNU Debugger)	11
7.2	Profiling tools	11
	Scalasca.....	11
	Paraver	12
	Intel Parallel Studio XE and Vtune	12
	gperftools: profile and call-graph	12
	CrayPat	12
	Arm MAP	12
7.3	Workflow management tools.....	13
	Taverna	13
	MMSF and Muscle	13
	Radical-Cybertools.....	13
	Stopos	14
8	Collaboration with other CoEs.....	14
	POP	14
	ComPat	14
	VECMA.....	15
9	Progress on Improving the Efficiency of CompBioMed Applications	16
	Alya	16
	HemeLB	16
	HemoCell	17
	Palabos	17
	BAC	17
9.2	Scalasca Experience: POP collaboration	18
9.3	HemoCell at Bull Atos.....	19
9.4	Improving the BAC and HemeLB: ComPat Collaboration	21
10	General Guidance to Improve the Efficiency of Simulations.	21
10.1	Exascale machines.....	21
10.2	How to Exploit Current and Future HPC Machines Efficiently.....	22

11	Conclusions and Future Work.....	23
12	Appendix A: A Rough Guide to Preparing Software for Exascale	25
12.1	Software preparations.....	25
12.2	Improve serial code	26
12.3	Introduce vector processing.....	26
12.4	Improve MPI code	27
12.5	Improve MPI parallelism	28
12.6	Introduce OpenMP for threads on cores and OpenACC for GPUs.	28
12.7	Improve OpenMP parallelism.....	29
12.8	General programming tips	29
12.9	Code Longevity	29
13	Appendix B: Bull/Atos report on Task2.6 Work Done	31
13.1	Bull.....	31
13.2	Background	31
13.3	Objectives of this deliverable	32
13.4	Work performed in this deliverable	32
13.5	Organisation of this report.....	33
	Setup (hardware configuration and Simulation parameters).....	33
	Scalability Tests	35
	Profiling	38
	MPI time analysis.....	38
	Hotspot functions/loops.....	39
13.6	Optimisation Tracks.....	41
	Attempt to utilise full potential of x86 instruction set	41
13.7	Loop vectorisation.....	42
13.8	Reproducibility	46
13.9	Single Node Optimisation.....	47
13.10	Possible alternatives to Palabos.....	47
	OpenLB	48
	WaLBerla	48
13.11	Conclusion	48
	Meetings.....	49
14	Appendix C: Report of UCL's exascaling BAC and HemeLB.....	50

1 Version Log

Version	Date	Released by	Nature of Change
V0.1	16/10/2018	Gavin J. Pringle	Draft of template
V0.2	5/11/2018	Gavin J. Pringle	Included input from UCL, introduced temporary Section to help encourage input, responded to questions.
V0.3	19/11/2018	Gavin J. Pringle	Major update with all input from co-authors folded in.
V1.0	7/12/2018	Gavin J. Pringle	Updated text in light of Reviewers' comments
V1.1	21/12/2018	Gavin J. Pringle	Updated text in light of final Reviewers' comments.

2 Contributors

Name	Institution	Role
Gavin J. Pringle	UEDIN	Principal Author
Victor Azizi	UvA	Co-author
Francesc Florencio	UOXF	Co-author
Marco Verdicchio	SARA	Co-author
Robin Richardson	UCL	Co-author
Alexander Patronis	UCL	Co-author
Franck Chevalier	ACE	Co-author
Paul Karlshofer	BULL	Co-author
Okba-Nafie Hamitou	BULL	Co-author
Jonas Latt	UNIGE	Reviewer
Jazmin Aguado	BSC	Reviewer
Emily Lumley	UCL	Reviewer
Peter Coveney	UCL	Reviewer

3 Acronyms and Definitions

Acronyms	Definitions
AMD	Advanced Micro Devices
ATP	Abnormal Termination Processing
BAC	Binding Affinity Calculator
BSC	Barcelona Supercomputing Centre
CFD	Computational Fluid Dynamics
CoE	Centre of Excellence
ComPat	Computing Patterns for High Performance Multiscale Computing
CPU	Central Processing Unit
CSCS	Swiss National Supercomputing Centre (Centro Svizzero di Calcolo Scientifico)
CT-scan	Computerised Tomography Scan
CUDA	Compute Unified Device Architecture
DEM	Discrete Element Modelling
ETP4HPC	European Technology Platform for High Performance Computing
FLOPS	FLoating point OPerations per Second
FPGA	Field Programmable Gate Arrays
GASPI	Global Address Space Programming Interface
GDB	GNU Debugger
GNU	GNU's not unix
GPAS	Partitioned Global Address Space
GPGPU	General Purpose Graphics Processing Unit
GPU	Graphics Processing Unit
GSISsh	Grid Security Infrastructure Secure Shell
GUI	Graphical User Interface
HDF5	Hierarchical Data Format
HPC	High Performance Computing
HTC	High-Throughput Computing
I/O	Input/Output
LBM	Lattice-Boltzmann Method
MMSF	Multiscale Modelling and Simulation Framework

MPI	Message Passing Interface
OmpSs	OpenMP + StarSs extensions
OpenACC	For Open Accelerators
OpenCL	Open Computing Language
OpenMP	Open Multi-processing
POP	The Performance Optimisation and Productivity Centre of Excellence in Computing Applications
Qt	Q-Thread
Scalasca	SCALable performance analysis of large SCale Applications
SMP	Shared Memory Parallelism
SSH	Secure Shell
StarSs	Star (for wildcard '*') Super scalar
STL	Stereolithography
UPC	Unified Parallel C
VECMA	Verified Exascale Computing for Multiscale Applications
VVUQ	Validation, Verification and Uncertainty Quantification
WP	Work Package

4 Executive Summary

CompBioMed end users are able to gain direct access to one or more of its HPC ecosystems of Tier0, Tier1 and Tier2 platforms. CompBioMed offers expert HPC assistance, along with powerful tools for debugging, profiling and workflow managers, to ensure simulations run efficiently on up to at least tens of thousands of cores on today's HPC systems. Moreover, application authors also exploit these tools to prepare for exascale machines. This report details these tools, and presents progress to date on four of the largest codes, namely Alya, HemelB, HemoCell and BAC, achieved by CompBioMed in collaboration with co-existing CoEs and H2020 programmes. Lastly, for added value, detailed technical guidance is presented on how to improve applications in general to efficiently exploit future exascale systems.

5 Introduction

End users are looking to run their research simulations efficiently on CompBioMed HPC resources, and this document reports access mechanisms to these resources and the tools available to help run their codes efficiently, both on current and future resources, including exascale. In particular, the document describes progress in and impact of deployment of tools and services in support of complex workflows, including multiscale models, on available HPC environments. This report does not cover the work that was done to port the applications to the CompBioMed platforms, but does describe work done using these tools to prepare CompBioMed solutions/applications for future exascale systems.

This document reports on some of the results of activities conducted in WP2 Task 2.6:

Task 2.6: Develop Plans for and Implement the Upscaling of CompBioMed Production Applications for Future HPC Platforms, Including those Heading Toward the Exascale (M12-M30) [Deep Track]

Leader UEDIN (6 PM); Partners: UCL (6 PM), UvA (6), UOXF (6), SARA (6), BULL (6)

Use co-design principles in partnership between application code developers, HPC centres and hardware vendors to plan optimal development of future releases of the exemplar codes in this project so as to ensure that they will run effectively on forthcoming architectures on the path to exascale, as well as on GPGPU and novel architectures such as Intel Xeon Phi co-processors. These applications will not only consist of monolithic codes running across large numbers of cores within the production partition of a computer, but in many cases are componentised workflows, including multiscale applications, that will need to be optimally mapped onto these architectures. Performance prediction tools will be applied for some of the complex componentised applications to forecast performance on emerging systems and the impact on future biomedical applications.

The structure of the remainder of this deliverable is as follows. The following Section, Section 6, describes the CompBioMed HPC ecosystem and how end users can access HPC resources. Following this, we then describe the Tools and Services, including Deep Track tools, that have been deployed on these CompBioMed HPC resources, specifically for debugging, profiling and workflow management. The next Section details how CompBioMed collaborates with other Centres of Excellence and H2020 programmes, namely POP, ComPat, and VECMA. Following, in Section 9, we describe the progress on scaling our largest codes, namely Alya, HemeLB, HemoCell and BAC, in preparation for Exascale. Section 9 also contains three overviews: the first on our experience with the profiler Scalasca and our collaboration with POP; the second describes the work done by Bull under Task 2.6 to improve the performance of HemoCell; and the third presents an overview of a collaboration with ComPat to improve HemeLB and BAC. The full reports of the latter two are presented in the Appendices A and B, respectively. Finally, we present WP4's guidance on how to prepare applications to scale on thousands of cores on both the HPC systems of today and of the exascale systems of the future. Appendix A also includes an Exascale Crib Sheet for programmers.

6 Access to CompBioMed HPC systems

6.1 The CompBioMed HPC systems

Some of CompBioMed's Core Partners, namely BSC, University of Edinburgh, and SURFsara provide access to their HPC systems, either through participation in the Centre of Excellence, or via PRACE Tier0 access. These systems are ARCHER and Cirrus, at EPCC (University of Edinburgh), UK; MareNostrum, at BSC, Spain; along with Cartesius and Lisa at SURFsara, the Netherlands. Further, some of CompBioMed Associate Partners also offer HPC access, specifically SuperMUC and SuperMUC-NG (anticipated), at the Leibniz Supercomputing Centre, Germany; and Prometheus, at Cryfnet, Poland. Additionally, CompBioMed has access to HPC systems outwith of its Core and Associate Partners offerings, such as Piz Daint, at CSCS, Switzerland; and three massive platforms in the US, namely BlueWaters at the National Center for Supercomputing Applications, Titan at Oak Ridge Leadership Computing Facility, and Theta at the Argonne Leadership Computing Facility. Peter Coveney is currently anticipating major access to Summit, number one on the current Top500 with a peak performance of 200 petaflops, by early in 2019; and there are hopes to get access to Aurora also at the Argonne Leadership Computing Facility.

6.2 How to get Access

End users from both Core and Associate Partners may apply for access to many of these systems via the email address allocations@compbiomed.eu, where they should apply with a brief description of what HPC platform they wish to access, what simulations they are interested in performing, what software they aim to employ, and how many core hours and disk space they require. To gain access to Tier0 resources, we encourage end users to apply to PRACE. UCL also command access to a vast suit of

supercomputers throughout Europe and the US, including SuperMUC(-NG), Blue Waters, Titan and Summit, and are able to allocate time to collaborators on these systems.

Once permission has been granted, end users are then put in contact with the associated HPC centre, who will guide them through the process of direct access to the HPC platform, using either SSH, which requires a username/password combination, or GSISsh, which requires Certificates. Both access methods grant the end users access via the Unix command line interface.

Either the target simulation codes are already ported and tuned to the existing target HPC architecture, or the end user, working closely with the HPC centre's experts, can install the simulation codes themselves. However, the efficiency of a tuned simulation code depends on the target simulation, e.g. a small simulation will not scale to 1000s of cores; thus, HPC centres offer a range of tools and services to ensure the target simulations run efficiently.

The following tools and services describe how a particular simulation can be tuned to the target HPC platform. It is important to note that, once ported, these same tools and services can be used to prepare the codes for future exascale HPC systems using that current HPC platform. In conjuncture with these tools, we are preparing for exascale systems via the co-design process, wherein vendors work with end users to ensure future supercomputers are constructed to ensure key applications will run efficiently. CompBioMed is actively involved in co-design, and this is addressed in Section 10.2.

7 Tools and Services to Improve Efficiency

This Section describes the tools and services, including deep track tools, which are deployed on the CompBioMed HPC resources at BSC, EPCC and SURFsara.

The tools and services for improving a simulation's efficiency can be grouped into the following types: debugging tools, profiling tools, and workflow management tools. Debugging tools are powerful accessories which are essential when a simulation crashes and/or produces the incorrect expected result. Profiling tools produce reports on past simulations to provide insight on where the simulation is inefficient, such as highlighting bottlenecks due to load imbalance, or I/O, or MPI communications, etc. Workflow management tools enable multi-component simulations, where the input to one simulation is formed from the output of another simulation; or multi-scale simulations, where multiple simulations of different time/length scales, are tightly coupled together; or enable the end user to run ensemble runs, where the same code is run simultaneously running slightly different simulations, often referred to as HTC.

Outwith of CompBioMed, our end users may access the POP CoE's resources to increase the scalability of their simulations, to ensure efficiency and prepare for exascale. For instance, UCL have worked with POP to scale HemeLB up to 250,000+

cores. At such a core count, it is found that standard profiling and debugging tools, and parallelisation software such as MPI-2, fail due to lack of functionality, the massive scales involved, etc. To this end, Peter Coveney is working with the MPI Forum (in particular with Tony Skjellum at University of Tennessee) to get MPI-4 released in 2020, through development work on a use case of HemeLB. This work will ensure that 64-bit communications will function efficiently at very large core counts.

Inside CompBioMed, the HPC Core Partners, namely BSC, EPCC, and SURFsara, have enabled the most common debugging tools, profiling tools, and Workflow manager tools which enables CompBioMed end users to improve their simulation's efficiency and prepare for exascale. The following section describes the tools currently enabled at the three HPC sites.

It is interesting to note that some of these tools, whilst ensuring efficient use of the target HPC system, may not be suitable for exascale machines, given they currently fail at larger core counts, e.g. Parmetis has been seen to fail on 50,000 cores. To this end, CompBioMed are actively testing other tools, such as ALL from the CoE E-CAM, which is currently showing real potential.

7.1 Debugging tools

Valgrind: Memory debugging

Valgrind is an instrumentation framework for building dynamic analysis tools. There exist Valgrind tools that can automatically detect many memory management bugs and threading bugs, and can profile programs in good detail.

Deployed at BSC, EPCC and SURFsara.

Padb - parallel application debugger

Padb is a Job Inspection Tool for examining and debugging parallel programs. Primarily, it simplifies the process of gathering stack traces on compute clusters; however, it also supports a wide range of other functions. It is an open source, non-interactive, command line, which can be used within scripts, intended for use by both programmers and system administrators.

Deployed at SURFsara.

Cray ATP

Cray ATP (Abnormal Termination Processing) is a tool that monitors your application and, in the event of an abnormal termination, it will collate the failure information from all the running processes into files for analysis.

Deployed at EPCC

DDT Debugger

DDT is a tool produced by Allinea Software, now part of Arm of Warwick, UK, and is employed for debugging scalar, multi-threaded and large-scale parallel applications, which is widely used for debugging MPI and threaded (pthreads or OpenMP).

Deployed at BSC and EPCC.

TotalView

TotalView is a comprehensive debugging tool for parallel applications. It is easy-to-use, supports multiple platforms, compilers, and programming languages, including the MPI, OpenMP, OpenACC and CUDA parallel programming paradigms.

Deployed at EPCC and SURFsara.

GDB (GNU Debugger)

The standard GNU debugger: GDB. This debugger currently only supports the command line interface.

Deployed at BSC, EPCC and SURFsara

7.2 Profiling tools

Scalasca

Scalasca is a powerful and popular profiling tool, and stands for SCALable performance analysis of large SScale Applications. Scalasca is an open-source toolset that can be used to analyse the performance behaviour of parallel applications and to identify opportunities for optimisation.

It is available with an open-source license from www.scalasca.org, and offers flexible runtime summarisation/profiling and event tracing. Its primary focus is on MPI, OpenMP and hybrid codes, namely which employ both MPI and OpenMP. Applications must be prepared in advance via “instrumentation”: MPI usage is instrumented simply by linking the application to the appropriate library, whilst OpenMP usage is instrumented by recompiling from source using Scalasca's own modified version of the target compiler.

Furthermore, it was recently extended to support a variety of other threading paradigms, such as Pthreads, Qt, and CUDA/OpenCL/OpenACC, and it supports most HPC systems, including IBM Blue Gene, K computer, Cray, and also heterogeneous systems like SURFsara's JURECA “Cluster+Booster” (Xeon + Xeon Phi).

It is CompBioMed's experience that Scalasca is the best profiling tool when working on hundreds of thousands of cores, where other tools have been seen to fail at a few tens of thousands of cores.

Deployed at BSC, EPCC and SURFsara.

Paraver

Paraver, a performance analyser based on traces with flexibility to explore the collected data. The associated Dimemas simulator can be used to predict the application's behaviour under different scenarios. Performance analytics modules extract insight from the raw performance data.

Deployed at BSC.

Intel Parallel Studio XE and Vtune

This software development product developed by Intel facilitates native code development on Windows, macOS and Linux in C++/C and Fortran for parallel computing. In addition to Intel compilers and specialised libraries it provides also debuggers, memory analysers, and profiling tools, via Vtune.

Deployed at EPCC and SURFsara.

gperftools: profile and call-graph

With gperftools it is possible to profile a program and create a call-graph. Profiling can be done on several levels, including performance studies of individual lines of code. Another advantage is that the binary can be used as-is. The tools work in principle with any binary created by either Gnu or Intel compilers.

Deployed at SURFsara.

CrayPat

CrayPat can perform two types of performance analysis: sampling experiments and tracing experiments. A sampling experiment probes the code at a predefined interval and produces a report based on these statistics. A tracing experiment explicitly monitors the code performance within named routines. Typically, the overhead associated with a tracing experiment is higher than that associated with a sampling experiment but provides much more detailed information.

Deployed at EPCC.

Arm MAP

Arm MAP, is an application profiler produced by Allinea Software now part of Arm of Warwick, UK, for profiling the performance of C, C++ and Fortran 90 software, parallelised using MPI and/or OpenMP.

Deployed at BSC and EPCC.

7.3 Workflow management tools

Taverna

In Taverna, workflows are represented in terms of direct acyclic graphs. Each node contains a part of the workflow in terms of calls to external software. These building blocks can be run independently, provided that all the required inputs are available at runtime. The graph edges represent message passing operations between the workflow building blocks.

Taverna is provided in two versions aimed at prototyping and production phases, respectively. Taverna Workbench is a desktop client including a GUI to ease the creation and editing of workflows through a drag and drop interface.

Deployed at BSC, EPCC and SURFsara.

MMSF and Muscle

Multiscale Modelling and Simulation Framework (MMSF) and MUSCLE: Multiscale Modelling and Simulation Framework (MMSF) for designing, programming, implementing and executing multiscale applications. The MMSF offers many benefits: a clear methodology, software and algorithm reuse, the possibility to couple new and legacy codes, heterogeneous distributed computing, and access to unprecedented computing resources. The framework is open-source and has been co-developed by CompBioMed Core Partners. It allows a more fine-grained and performance oriented coupling of simulation kernels than Taverna.

Deployed at BSC, EPCC and SURFsara.

Radical-Cybertools

This workflow tool seeks to address the limitations imposed by HPC queuing systems to create efficient large-scale hybrid applications. An example of such an application is the Binding Affinity Calculator (BAC), a decision support tool which uses molecular level computer simulation to reliably predict the binding affinities (free energies) of molecules with target proteins, and therefore identify those most likely to bind to the protein. BAC has been built to integrate and automate the multi-step process of model building, simulation, and data analysis for molecular level drug-receptor interactions. It constitutes a sophisticated computational pipeline built from selected software tools and services, and which relies on access to a range of computational resources.

BAC depends on the ability to perform hundreds of separate parallel simulations on a high performance computing platform, each of which can require 50-200 cores depending on the system. The BAC workflow automates much of the complexity of running and marshalling these simulations, as well as collecting and analysing data. This requires a workflow management tool that integrates closely with the queuing system on an HPC resource, in order to efficiently allocate replicas between available compute nodes, and also manage the execution and data staging between different steps of the simulation protocol. For this purpose, BAC uses the Pilot Job Manager, part

of RADICAL-Cybertools, a suite of abstractions-based and standards-driven tools that provide a common, consistent, and scalable approach to high-performance and distributed computing.

Deployed at EPCC.

Stopos

The Stopos software, developed by SURFsara and available on both their HPC systems, gives the opportunity to define and get lines, which can be used as parameters, in an orderly way. Stopos takes care that these lines are given out one after each other. It can be used to submit many jobs each with about the same content, but with different parameters for the program to run. Examples are parameter scans and Monte-Carlo simulations. Using Stopos, it is quite easy to build jobs that use the nodes of the computing system in an optimal fashion. Moreover, it is possible to correct for failing jobs (time limit, system problems etc).

Deployed at SURFsara.

8 Collaboration with other CoEs

Several CompBioMed applications undergo almost continuous performance monitoring and improvement, such as Gromacs, BAC, HemoCell, Alya, and Palabos. These improvements are not only done by CompBioMed staff, but also in collaboration with other EU programmes.

POP

The Performance Optimisation and Productivity Centre of Excellence in Computing Applications provides performance optimisation and productivity services for academic and industrial codes in all domains.

Highly scalable codes can effectively use thousands of compute nodes (a compute node contains many cores); however, equally scalable performance tools are needed to assist with additional tuning by quantifying parallelisation inefficiencies and identifying optimisation opportunities. POP assessments help characterise parallel application performance and scalability by identifying inefficiencies that constitute the most productive opportunities for optimisation.

CompBioMed and POP have collaborated on improving the performance of HemeLB on the BlueWaters Cray XE in the US (POP-AR-102: HemeLB on Cray XE for CompBioMed). HemeLB is a lattice-Boltzmann code for simulation of hæmodynamics in complex geometries.

ComPat

ComPat (<http://www.compat-project.eu>) is a science driven project on Computing Patterns for High Performance Multiscale Computing. The urgent need to push the

science forward, and stay world leading in simulation driven science and engineering is its major motivation.

Multiscale phenomena are ubiquitous, and they are the key to understanding the complexity of our world. Despite the significant progress achieved through computer simulations over the last decades, researchers are still limited in their capability to accurately and reliably simulate hierarchies of interacting multiscale physical processes that span a wide range of time and length scales, thus quickly reaching the limits of contemporary high performance computing at the tera and petascale. Exascale supercomputers promise to lift this limitation, and within ComPat, multiscale computing algorithms have been developed that are capable of producing high-fidelity scientific results and are scalable to exascale computing systems.

The ComPat approach was based on generic multiscale computing patterns that allow implementation of customised algorithms to optimise load balancing, data handling, fault tolerance and energy consumption under generic exascale application scenarios.

ComPat's ambition is to establish new standards for multiscale computing at exascale, and provision a robust and reliable software technology stack that empowers multiscale modellers to transform computer simulations into predictive science.

Several of CompBioMed's Core Partners are also ComPat members and are actively involved in the methodologies required to tightly couple different biomedical simulations of differing time/length scales.

VECMA

VECMA, or Verified Exascale Computing for Multiscale Applications, is a FET HPC project, where the purpose of the VECMA project (<https://www.vecma.eu>) is to enable a diverse set of multiscale, multiphysics applications — from fusion and advanced materials through climate and migration, to drug discovery and the sharp end of clinical decision making in personalised medicine — to run on current multi-petascale computers and emerging exascale environments with high fidelity such that their output is “actionable”. That is, the calculations and simulations are certifiable as validated (V), verified (V) and equipped with uncertainty quantification (UQ) by tight error bars such that they may be relied upon for making important decisions in all the domains of concern. The central deliverable will be an open source toolkit for multiscale VVUQ based on generic multiscale VV and UQ primitives, to be released in stages over the lifetime of this project, fully tested and evaluated in emerging exascale environments, actively promoted over the lifetime of this project, and made widely available in European HPC centres.

CompBioMed will follow the VECMA project closely, and will employ the VVUQ toolkit where applicable.

9 Progress on Improving the Efficiency of CompBioMed Applications

Several CompBioMed applications undergo almost continuous performance monitoring and improvement, such as Alya, HemeLB, HemoCell, Palabos, and BAC.

Alya

Perform Cardiac Computational Mechanics simulations, from tissue to organ level. FEM-based electro-mechanical coupling solver, specifically designed for the efficient use of supercomputing resources. The contact is mariano.vazquez@bsc.es.

Provider: BSC
Current users: 40 internal and 40 external users
Access mode: Direct
URL: <https://www.bsc.es/research-and-development/software-and-apps/software-list/alya>
Use scenario: Non-clinical research; Clinical research; Clinical decision support; Design & optimisation for medical devices; In silico clinical trial.
HPC Systems: MareNostrum, ARCHER, Cartesius
HPC motivation: Solve unreducible model; Multiscale model; Strongly coupled multiphysics model.

HemeLB

This code simulates the blood flow through a stent (or other flow diverting device) inserted in a patient's brain. The aim is to discover how different stent designs (surface patterns) affect the stress the blood applies to the blood vessel, in particular in the region of the aneurysm being treated. The pipeline also allows the motion of magnetically steered particles, for example coated with drugs, to be simulated and estimates made as to where they might statistically end up. More technically, the pipeline takes as input an STL file of the surface geometry of the patient, generally obtained via segmentation of DICOM images from a CT-scan. Also required is the (peak) velocity-time profile of fluid flow at each of the inlets to the simulated region. If inserting a stent, the start and end points of the stent in the vessel must be specified, as well as an image file containing a black and white representation of the surface pattern (black signifying 'solid'). The HemeLB setup tool voxelises the geometry bounded by the input STL at the given resolution, and HemeLB (lattice-Boltzmann CFD solver) then simulates the fluid flow within that geometry, using the given velocity-time profiles for each inlet. Once complete, the simulation output is analysed using the hemeXtract utility, which can produce images of cross-sectional flow, or 3D shots of wall shear stress distribution in the geometry using ParaView visualisation software. The contact is robin.richardson@ucl.ac.uk.

Provider: UCL
Current users: 40 users (mostly academia)
Access mode: Direct
URL: <https://github.com/UCL/hemelb>

Use scenario: Open Source software used primarily in academia. Clinical research; Clinical decision support; In silico clinical trial.
HPC Systems: EPCC ARCHER, LRZ SuperMUC, PSNC Prometheus
HPC motivation: Solve unreducible model.

HemoCell

High-performance library to simulate the transport properties of dense cellular suspensions, such as blood. It contains validated material model for red blood cells and additional support for further cell types (white blood cells, platelets). The blood plasma is represented as a continuous fluid simulated with an open-source LBM solver. The cells are represented as DEM membranes coupled to the plasma flow through a tested in-house immersed-boundary implementation. HemoCell is computationally capable of handling a large domain size with high number of cells ($> 10^4$ - 10^6 cells). The contact is g.zavodszky@uva.nl.

Provider: UvA
Current users: NTU, BME
Access mode: Source
URL: <http://www.hemocell.eu>
Use scenario: Clinical research, Clinical decision support, In silico clinical trial.
HPC Systems: Cartesius, Lisa, SuperMUC
HPC motivation: Solve unreducible model; Multiscale model; Strongly coupled model.

Palabos

Palabos is a software library for the computation of complex flows with the lattice Boltzmann method. It is highly parallel and versatile, and used in many areas of industry. For computational biomedicine, it offers front-end applications: A simulation program for the effect of deployed stents in an artery, and a simulation program for efficiency improvement of cement injection in vertebroplasty.

Provider: UniGe
Current users: 200 known users from academia and industry
Access mode: Direct
URL: <https://www.palabos.org>
Use scenario: Clinical research; Clinical decision support; In silico clinical trial.
HPC Systems: UniGe Baobab.
HPC motivation: Solve unreducible model.

BAC

Workflow tool that runs and analyses simulations designed to assess how well drugs bind to their target proteins and the impact of changes to those proteins. A collection of scripts which wrap around common molecular dynamics codes to facilitate free energy calculations. Use of ensemble simulations to produce robust, accurate and

precise free energy computations from both alchemical and end-point analysis methodologies. The contact is dave.wright@ucl.ac.uk.

Provider: UCL
Current users: UCL, GSK
Access mode: Service
URL: No current website - DNA Nexus app available only to UCL/GSK at present
Use scenario: Non-clinical research; Drug discovery; Design & optimisation.
HPC Systems: DNAnexus
HPC motivation: Solve unreducible model; performs uncertainty quantification.

The remainder of this Section describes progress achieved on these applications on both CompBioMed Core Partner HPC systems, and on HPC systems outwith CompBioMed.

9.2 Scalasca Experience: POP collaboration

Scalasca is a performance toolset boasting a scalable design and effectiveness that allows it to be used for the analysis of parallel application execution behaviour on massively parallel architectures with many thousands of processors. It supports measurement and analysis of highly-scalable HPC applications, such as HemeLB, which run on hundreds of thousands of ranks; Scalasca has been shown to scale (with HemeLB) from 3,744 to 239,615 MPI ranks (288 to 18,432 XE compute nodes of Blue Waters), with minimal computational overhead. CompBioMed developers have worked closely with the Performance Optimisation and Productivity (POP) Centre of Excellence based at the Jülich Supercomputing Centre, where Scalasca is developed, to optimise the initialisation phase of HemeLB.

Our basic analysis workflow when using Scalasca begins with instrumentation of the HemeLB binary. When running the instrumented code, we choose to generate a summary report with aggregate performance metrics for individual function call paths (but may also select to include event traces recording individual runtime events). Our use of Scalasca, when applied to HemeLB, has been limited to the production of summary reports, which provide hardware counter-based performance metrics and a general overview of performance behaviour. With these reports members of POP CoE prepare complementary performance audits (HemeLB has been profiled to 99,600 ranks on ARCHER, and to 239,615 ranks on Blue Waters, i.e. 81% of the available XE compute nodes). We have yet to perform finer-grained profiling using tracing, which instructs each process to generate a trace file containing records for its process-local events. Future work will involve the instrumentation of more classes/methods and tracing (providing time-line visualisation) for an in-depth analysis of HemeLB's simulation phase (which uses a sophisticated non-blocking communication pattern known as 'coalesced communication' that can only be effectively monitored using profiling tools).

Observations have led to significant reductions in HemeLB's memory utilisation. With metrics obtained from Scalasca, we were able to identify large data structures that were limiting HemeLB's application to larger problems. HemeLB is now capable of loading, decomposing, and simulating problems consisting of ~11 billion lattice sites (with approximately 50% peak memory usage on Blue Waters). Blue Waters has 22,640 XE nodes, or a possible 362,240 MPI ranks. Decomposing a problem of ~11 billion sites over the entire machine will result in approximately 30,000 sites/rank - the already excellent strong scaling of HemeLB to 256,000 ranks (demonstrated without Scalasca and using a problem dataset of approximately 5 billion sites) may theoretically be extended to the 362,240 ranks.

9.3 HemoCell at Bull Atos

As part of CompBioMed's Task 2.6, two of CompBioMed's Core Partners, namely Bull in France, and the University of Amsterdam, in The Netherlands, have been working on the HemoCell application. The goal is to port, profile, optimise and report on the performance results of a set of applications provided by the CompBioMed community. HemoCell has been profiled and tested on several compute nodes (Skylake, Haswell, Broadwell). They also have access to AMD and Intel Xeon Phi compute nodes.

In general, part of Bull's goal is to co-work with CompBioMed core partners on the selected HPC codes and report on the implementations of CompBioMed applications on emerging HPC architectures and porting to new and maturing architectures. The first step is to port the applications on Bull HPC platform, run scalability tests, then perform a profiling of the applications to target the most time-consuming function and loops for optimisations. Finally, the deliverable consists of this report and suggestions for improving the performances of the applications.

HemoCell is developed by the University of Amsterdam (UvA) and is a high-performance code which simulates the transport properties of dense cellular suspensions such as blood (Figure 1) below. The blood plasma is modelled by a continuous fluid simulated with an open-source Lattice Boltzmann Method (LBM) solver. On the other hand, the cells are modelled by discrete element method membranes. Both models are coupled using an immersed boundary implementation.

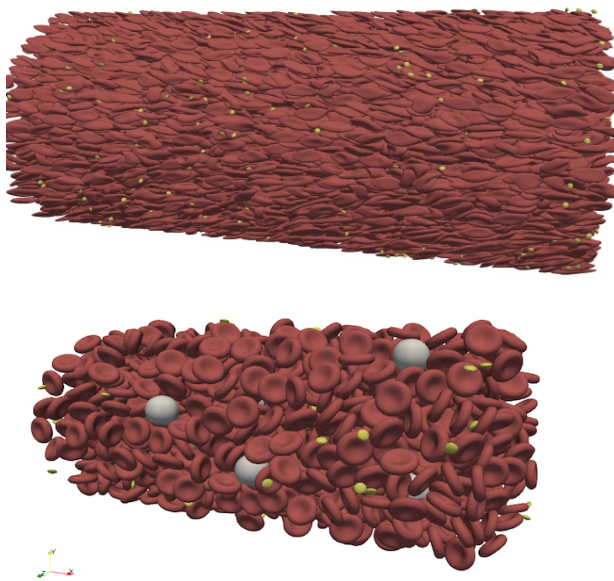


Figure 1 Dense flow (left) and white blood cell extension (right)

Starting from the portable implementation of HemoCell v1.4, the team members of Task 2.6 took the application and performed scaling test and optimisations on the Intel Skylake compute nodes.

The profiling of the application led to targeting the hotspots of the application, where the optimisations need to be directed.

In addition to the optimisation suggestions of the application, this report aims at determining the best practices for the implementation of numerical methods on emerging HPC architectures.

The main outcome resides in the fact that the HemoCell code consists in an MPI implementation of the Lattice Boltzmann Method of Palabos and intensive I/O through the HDF5 library. The HPC code suffers from a lack of a load balancing method which causes the application to perform poorly. Fortunately, the upcoming version of HemoCell shall include a load balancing technique (Parmetis library). It was noticed that the Palabos library could not fully benefit from the vectorisation possibilities of the compiler and the processor architecture despite many intrusive and non-intrusive optimisations.

As a conclusion, a better memory management or data structure can lead to a higher vectorisation. One may suggest an LBM implementation as well which may benefit from an OpenMP parallelisation. The latter shall take advantage from future multi-threaded architectures.

The full report can be found in the Appendix.

9.4 Improving the BAC and HemeLB: ComPat Collaboration

This Section outlines the performance studies performed by CompBioMed partners under the ComPat programme. The full ComPat report can be found in the Appendix.

The CompBioMed Core Partner UCL considered both HemeLB and the BAC with a view to scaling associated simulations on exascale platforms. Various performance metrics were considered, as may be most appropriate for the given application e.g. wall clock time, file sizes, scalability (strong and weak), and energy consumption (where available). In one weak-scaling application we also consider scalability of the middleware itself.

To further explore and assess the potential impact of extreme parallelism, we also carried out performance studies on two applications using even larger supercomputers. To simplify the assessment, we note that the multiscale applications can be abstracted as sections of replica computing steps, and large monolithic applications. For this reason, our predictions as pertain to exascale resource usage largely come from detailed studies of application performance for a Replica-based exemplar (the Binding Affinity Calculator, or BAC) and for an exemplar containing a large monolithic application (HemeLB).

Furthermore, a detailed mathematical model was developed to predict the time and length scales attainable by a lattice-Boltzmann solver (such as Palabos or HemeLB) in a fixed time on computers with e.g. 1 billion cores (exascale).

With respect to the exascale, a major conclusion of this work was that, even for those applications exhibiting excellent strong scaling characteristics, the trade-off between resolving time or physical length scales in the system will frequently render such simulations inefficient on enormous core counts when compared to the weak scaling (replica) case. We therefore expect that the actual impact of exascale resources on future science applications will be to encourage the use of uncertainty quantification (techniques that often require multiple runs) in a field where researchers too often only run large simulations once.

The full report can be found in Appendix B.

10 General Guidance to Improve the Efficiency of Simulations.

This section introduces an overview of how to improve the efficiency of simulations on generic HPC systems, both current and future exascale platforms. Moreover, this Section focuses on preparing your simulations for Exascale; however, this guidance holds for getting the best performance out of today's current HPC systems as well.

10.1 Exascale machines

HPC platforms have seen their performances increase over the past years and are reaching a plateau. At the core level, the frequency, i.e. processor speed, together with the cache, has increased from 1.1GHz to 3.2 GHz and has almost reached an upper

bound. At the same time, CPUs are now able to execute several instructions per cycle, introducing thus a first level of parallelism. At the node level, the memory bandwidth continued to grow hand in hand with the processor speed, to keep the processor busy. A race over the miniaturisation of processor is leading towards more populated chips but with several constraints: technology for miniaturisation and heat dissipation.

One other important key point is that for many high-performance applications, I/O performance is a blocking factor. I/O controls the amount of data that can be saved on disk and enables checkpoint files that can be written to prevent system failure which will become more crucial as HPC platforms continue growing in size and complexity. Therefore, I/O performances is already identified as key due to their significant impact on data intensive applications.

The first special-purpose exascale machines are likely to be for a small set of applications, where the platforms themselves are built via co-design principles. The issue of adopting co-design strategies is an extremely important one when it comes to the transition to exascale high-performance computing. It is based on developing partnerships with computer vendors and application scientists and engaging them in a highly collaborative and iterative design process well before a given system is available for commercial use. The process is built around identifying leading edge, high-impact scientific applications and providing concrete optimisation targets. CompBioMed actively promotes HemelB as a target application program, since it is one of very few applications that currently utilises multi-petaflop computing platforms to answer key scientific questions, and is actively being optimised for the emerging exascale. Specifically, it is hoped that HemelB is part of the co-design process for the future Aurora platform at the Argonne Leadership Computing Facility, in the US, where Aurora is likely to be a pure Intel machine with their own specially developed accelerator technology.

The first generic exascale machines are likely to be tightly coupled heterogeneous clusters of many-core/multi-core SMPs, and smaller clusters of accelerators, such as GPUs, FPGAs, etc. It is our understanding that the first exascale machine that CompBioMed partners will get access will be Aurora at Argonne Leadership Computing Facility. This is planned to be a pure Intel machine, with their own specially developed accelerator technology, built from co-design.

10.2 How to Exploit Current and Future HPC Machines Efficiently

Four key areas can be pointed out which must be invested to prepare software for exascale platforms, namely performance profiling, deployment of new programming models, development of new libraries and application co-design.

Profiling, using the tools available today on HPC platforms (see above), enables users to locate performance bottlenecks and produces efficiency metrics in terms of both FLOPS and power consumption.

Programming models are key to exploiting future exascale platforms. For instance, the most common programming paradigms, namely MPI and OpenMP, have had their standards extended to prepare for exascale. Moreover, fewer common paradigms are becoming more popular, such as Global Address Space (PGAS) Languages, e.g., UPC, Co-Array Fortran, and GASPI, will aid programmers when seeking to exploit future hardware. Given the heterogeneous nature of future exascale systems, which will likely include GPUs and other accelerators such as FPGAs, their associated programming models, e.g., CUDA, OpenACC, OpenMP-4.0 directives (planned), and Hybrid programming, e.g., MPI+X, and OmpSs, are all becoming increasingly important.

Employing efficient libraries can typically save the programmer from “reinventing the wheel”, given that the libraries are often tuned for the target platform, and yet the programmer’s code stack can remain portable.

Lastly, co-design is where HPC vendors and end users can design the future HPC systems together, thus avoiding HPC systems that grab the headlines with powerful performance figures but are impractical for massive scale scientific computation.

Application codes rarely perform and scale well when first parallelised: each doubling of scale typically exposes a new issue. Ensuring your application will scale on HPC systems - both today and on the exascale systems of the future - requires stepwise increasing of scale and validation of correctness, debugging, performance analysis and tuning. Then, the same process is repeated for each significant code extension or optimisation.

According to VECMA, there exist three routes to exascale. They are (1) parallelising single scale models: enabling the use of more cores and reducing execution time; (2) construct multi-scale models: enabling the use of even more cores, more coupling, and more simulations incorporated; and finally (3) adding VVUQ (see above): enabling even more cores, more robustness, reproducibility and reliability.

In the Appendix, we present A Rough Guide to Preparing Software for Exascale. This is essentially a Crib Sheet where programmers can consult when preparing their code for future exascale machines. Moreover, this Crib Sheet can also be of use when looking to make applications more efficient on not only future Exascale machines, but also the HPC systems available today.

11 Conclusions and Future Work

This deliverable has described the tools, such as debuggers, profilers and workflow managers, that have been deployed on CompBioMed’s HPC ecosystem that enable programmers to ensure their simulations make efficient use of the resources.

Further, we have presented how our work, in collaboration with other CoEs and H2020 programmes, have seen the improvement of four of CompBioMed’s largest applications, namely Alya, HemeLB, HemoCell, and BAC.

Lastly, and as added value, we presented some general guidance and a programmer's crib sheet, on how to prepare applications for future exascale systems.

12 Appendix A: A Rough Guide to Preparing Software for Exascale

This Section is to be used as a Crib Sheet for improving the efficiency of software.

The term *Exascale* is used to describe HPC hardware capable of at least one exaFLOPS, or 10^{18} Floating point Operation per Second. It is envisioned that such machines will have many multi-core processors, and that the available memory per core will be far inferior to those on current HPC platforms. This can be seen when attempting to port MPI codes to IBM Blue Gene machines, or the Intel Xeon Phi family, where the amount of memory per core is prohibitively small for many codes parallelised using MPI only. As such, the common practice of running one MPI task per physical core may no longer be possible for the majority of codes in the future.

The solution for getting codes ready for exascale platforms requires both software and hardware related strategies. The former, the subject of this note, is described below. The latter, beyond the scope of this note, is achieved via Co-Design, where hardware vendors and end users work together to ensure future platforms are not built to achieve exascale performance at the expense of usability.

Application codes rarely perform and scale well when first parallelised: each doubling of scale typically exposes a new issue. Ensuring the application will scale on HPC systems - both today and in on the exascale systems of the future - requires stepwise increasing of scale and validation of correctness, debugging, performance analysis and tuning. Then, repeat for each significant code extension/optimisation.

Through performance analysis, programmers can locate so-called “hot spots”, i.e. code which takes the most time, as this code should then be targeted for improvement.

12.1 Software preparations

Given the memory per core will most likely be substantially reduced when compared to today’s HPC platforms, the practice of assigning one MPI task per physical core will have to be substituted by using every 2nd or 4th core for each MPI task. This is known as under-populating nodes. At first glance, this appears to suggest that we cannot fully exploit the hardware, as we simply avoid using 50% or even 75% of the cores; however, these spare cores can be employed via mix-mode codes, where each MPI task runs threaded routines/loops to run on the remaining cores.

Essentially, authors must expose as many levels of parallelism as possible within their code. Coding this can involve ensemble runs to coupling multiscale codes, multiprocessing (with interprocess communication) to multithreading, vector processing to accelerator-specific commands. This process can slow the code down on present day platforms but will future-proof the code.

For instance, there are sets of serial algorithms, so-called Optimal Serial Algorithms, which are often difficult or simply impossible to parallelise, as these algorithms employ

data from the previous steps or even the current step to make improvements at the current step. Such dependencies can prevent concurrent execution of threads in the program, for instance.

The inelegant yet empowering solution is to replace the optimal serial algorithm with a sub-optimal serial algorithm which is, however, parallelisable. Whilst the serial performance may be worse, the parallel performance will soon outperform the serial version as the number of cores increases.

12.2 Improve serial code

Before considering how the code is parallelised, the first step is to consider the serial sections of the code.

- Remove excess memory use in serial code.
- When using C++, find good balance of OOP and functional programming, as an intensive use of OOP might introduce an unnecessary layer of complexity of the scientific code.
- Ensure proper use standard libraries

12.3 Introduce vector processing

- Use appropriate compiler options
- Write ordered loops or leave this to compilers?
- Innermost loop must have independent iterations
- Loop length is either larger of multiple of vector length
- It is possible to set this at compiler time but not "probe and populate"
- No function calls, except maths libraries
 - functions can be vectorised using OpenMP "declare simd" feature
- No complex control flow
- Determinable trip count (i.e. no while)
 - the trip count must be known before entering the function at runtime
- Data access should be vector aligned, i.e. start at vector boundaries, and preferably continuous
- <http://www.archer.ac.uk/training/course-material/2017/11/sgl-node-ox/L04-vectorisation.pdf>
- Be aware of the ISA (SSE, AVX ..)
 - it determines the vector length
 - may target vectorised FMA instructions
 - Do loop padding manually to get rid of peel/remainder loops
 - Concerning vectorisation, we check compiler output or asm code to see what was vectorised
 - Use inline hints for functions or routines to help out the compiler to inline
 - Remember that YOU know your application better than the compiler does.
 - It all depends on how the data is aligned in RAM

- Hints with pragmas might be useful, also
- Force data alignment with compiler instructions (usually done automatically by the compiler)

12.4 Improve MPI code

- MPI messages should be grouped to avoid multiple smaller messages
 - e.g. use derived data types to avoid double buffering
- Avoid any storage or computation of $O(n_{\text{ranks}})$
- Avoid all-to-all communication
 - e.g. if(rank==0) then do work over all other ranks
- Remove unnecessary MPI_BARRIERS
- Do not over schedule cores when using threaded maths libraries
 - typically control using OMP_NUM_THREADS even when libs do not use OpenMP
- Use nonblocking collective communications.
 - overlap computation and communications where possible
- Remove unnecessary communication synchronisation
 - use MPI_TEST rather than MPI_WAIT
 - avoid MPI_Probe
 - it most likely forces internal buffering to report the size of the pending message
 - Avoid ordered halo swapping,
 - e.g. don't delay y-direction sends until x-direction receives have completed.
 - however, huge network bursts are also not ideal
 - sometimes, ordered sends allow ordered receives.
 - and ordered sends might allow to take advantage of the network topology
 - e.g. one can completely load the network with x-direction halo swaps and so on
- Ensure load is balanced
 - Avoided the receive-before-send scenario
 - one-sided communications can alleviate this
- Be aware that not all MPI libraries are equal
 - e.g., there are many ways to implement collective communications.
- Respect the fact that the MPI standard prohibits concurrent read accesses on the same buffer (even though there is no race condition)
 - It may reduce the efficiency (or cause bugs)
- Tag the source
- Be aware: “blocking” has an alternative meaning in the MPI standard.
 - This can easily lead to serialisation of huge chunks of the program.
- Interleave/overlap communication with computation where possible

12.5 Improve MPI parallelism

- Give each MPI task multiple sub-domains
 - a subdomain is a distinct region of the computational domain and a result of the domain decomposition algorithm.
 - this allows light weight parallelism on a socket, keeps cache logically together, etc.
- Enable multiple tiles per task.
 - this might make tiles fit into cache but will spend time swapping boundary information with yourself
- Use MPI Communicators, to map the communication to the target HPC topology
 - Collective operations are possible on a subset of processes.
 - Explicit communicators are very useful to leverage MPI Shared Memory
- One-sided communication (or Remote Memory Access (RMA)), can be faster than the message passing model
 - May be beneficial when the load is hard to balance, since delays in the receiving process are not necessarily propagating to the sender

12.6 Introduce OpenMP for threads on cores and OpenACC for GPUs.

A code which uses both MPI and OpenMP, or a code that uses both MPI and OpenACC, is referred to as a mixed-mode code. This is done for two reasons: reduce memory footprint or/and speed up application.

Not all MPI codes benefit from becoming mixed-mode codes. The benefits are as follows.

- Hybrid applications have a reduced memory footprint (the shared memory model allows threads to avoid halo regions or ghost cells)
- Eases load balance issue (usually the complexity of (adaptive) load balance grows with the number of subdomains)
- Load balancing in threads is much easier
 - thread-pool model, or
 - tasks
- For applications which are MPI-bound due to load imbalance (long barriers in MPI_Wait or MPI_Receive/Send), it might be advisable to reduce the number of processes and increase the threads, while using OpenMP's built in load balancing features

Whilst the drawbacks are as follows:

- In case of MPI_THREAD_MULTIPLE, the application might lose portability
 - forked threads are allowed to call any MPI routines
- Shared memory applications have their own problems
 - e.g. false sharing, where a cache line is voided repeatedly
 - this is naturally avoided by MPI processes
- NUMA effects, e.g. where the data is placed in memory
 - this can be resolved by careful task mapping.

Maybe also better to use OpenMP 4.5 target directives than OpenACC.

12.7 Improve OpenMP parallelism

An excellent Best Practice Guide for writing mixed-mode programs, i.e. MPI+OpenMP, can be found via the Intertwine project pages: <https://www.intertwine-project.eu/mpi-plus-openmp-threads-resource-pack>

- Investigate OpenMP tasks
- Try different schedules and/or tasks
- Avoid over-scheduling threads when calling threaded maths libraries.
- Minimise sequential code
- Replicating computation rarely works
- Ensure load balance over threads.
 - Use different loop schedules or tasks
 - May includes balancing communication in one thread with calculations in the rest
- Avoid MPI data types as packing data is done on one thread: better to pack data in parallel using threads, as MPI should not need to double pack when data is contiguous
- Take care with NUMA effects by considering mapping, i.e. task placement
 - e.g. run at least one MPI process per NUMA node
- Take care with process and thread binding: threads should run on the same socket as their parent MPI process.
- Minimise the number of OpenMP barriers
- Use OpenMP directives to force SIMD operations
 - OpenMP allows explicit vectorisation of functions called from vectorised loops

12.8 General programming tips

- Be aware of the Load-Hit-Store problem (it does exist on multiple levels)
 - prevents caching by the compiler and causes pipeline stalls.
 - e.g., appears in MPI-IO (sometimes referred to as Read-Modify-Write effect)
- IO can dominate
 - consider MPI-IO or, better still, HDF5

12.9 Code Longevity

Whilst this section includes good practice for software engineers in general, the following points are key when preparing for exascale systems, primarily because large popular codes outlive the programmers who, in turn, typically outlive the HPC platforms for which the software was written.

- follow a strict coding style guide
- use readable variables

- use internal documentation
- keep routines to less than one page
- copyright statements for every module/subroutine/function
 - key for IP monitoring

13 Appendix B: Bull/Atos report on Task2.6 Work Done

13.1 Bull

Bull is part of the Atos group. As Europe's only computer manufacturer through its Bull brand, Atos operates in the ultra-high processing power market, to liberate its customers' ambitions. The CEPP (Centre for Excellence in Parallel Programming), a branch of Bull, aims at hosting collaboration and Partnership. Most of the experts in this centre have a PhD in a scientific domain; the others a PhD in Computer Science. This choice is driven by the wish to create a bridge between scientists and performance experts.

The key advantage is that, scientists of the same domain will immediately and deeply understand each other. The quality of the exchanges will always be higher than the one we may achieve with pure computing experts.

Obviously, the core of expertise of the whole team is optimisation, porting, and knowledge about the platform and their evolution. Based on this core expertise, a lot of high level services can be derived.

13.2 Background

Computational methods, based on human biology, are now reaching maturity in the biomedical domain, rendering predictive models of health and disease increasingly relevant to clinical practice by providing a personalised aspect to treatment. Computer based modelling and simulation are well established in the physical sciences and engineering, where the use of high performance computing (HPC) is now routine.

CompBioMed is a European Commission H2020 funded Centre of Excellence (CoE) focused on the use and development of computational methods for biomedical applications. Users within academia, industry and clinical environments are working to train more people in the use of these tools and methods in this domain.

The objectives of CompBioMed are met by a cyclic collaboration of individual work package (WP) teams: WP1 focuses on the project management, WP2 consists in research activities, WP3 and WP4 relates to networking activities and finally WP5 and WP6 handle service activities.

In this report, the work performed by WP2 on *“developing plans for and implementing the upscaling of CompBioMed production application for future HPC platforms, including those heading toward the Exascale (Task 2.6)”*, are presented. The goal is to port, profile, optimise and report on the performance results of a set of applications provided by the CompBioMed community.

List of core partners of CompBioMed: University college London (UCL), University of Amsterdam (UVA), University of Edinburgh, SURFsara, Barcelona Supercomputing Centre (BSC), University of Oxford, University of Geneva, University of Sheffield, CBK Sci Con Ltd, Universtat Pompeu Fabra, LifeTec Group, Acellera, Evotec, Bull, Janssen.

List of associate partners of CompBioMed: Avicenna Alliance, Birmingham City University, Briel University London, GSK, LRZ, Rutgers, DNAnexus, London Science Museum, University of Leeds, VHP Institute, Zayed University, HITS, Hartree Centre, University of Southampton, KINDI, University Catolica de Murcia, Aix Marseille University, e-Cardiology, Oxford NHR, Alcesflight, Cyfronet, Electric Ant Lab BV, Norton Straw, ITMO University, Pozlab, Qatar Rototoc Surgery Centre, Microsoft, Dassault Systemes, Lightox, InSilicoTrials, Diamond Light Source, EnsembleMD, ANSYS, Medtronic, Université Libre de Bruxelles, PIE Medical Imaging, Astra Zeneca.

13.3 Objectives of this deliverable

The aim of this document is to report on the activities performed on WP2 Task 2.6. The goal is to co-work with CompBioMed core partners on the selected HPC codes and report on the implementations of CompBioMed applications on emerging HPC architectures and porting to new and maturing architectures. The first step is to port the applications on Bull HPC platform, run scalability tests, then perform a profiling of the applications to target the most time-consuming function and loops for optimisations. Finally, the deliverable consists of this report and suggestions for improving the performances of the applications.

13.4 Work performed in this deliverable

The work performed in this deliverable consisted of Task 2.6 in WP2 of the CompBioMed description of action. The latter includes the porting, profiling and optimisation of the selected HPC applications on emerging HPC platforms.

Among the high-performance computing codes which lie at the heart of the CompBioMed CoE, the HemoCell code was chosen by Bull's team to be investigated.

HemoCell is developed by the University of Amsterdam (UVA) and is a high-performance code which simulates the transport properties of dense cellular suspensions such as blood (Figure 2Error! Reference source not found.). The blood plasma is modelled by a continuous fluid simulated with an open-source Lattice Boltzmann Method (LBM) solver. On the other hand, the cells are modelled by discrete element method membranes. Both models are coupled using an immersed boundary implementation.

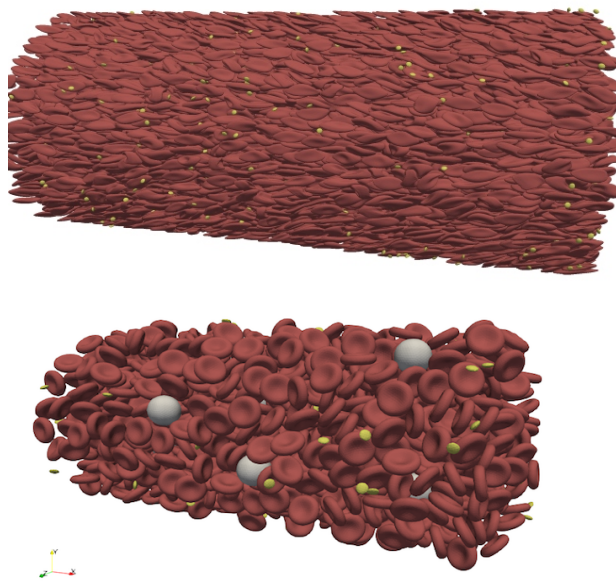


Figure 2 Dense flow (left) and white blood cell extension (right)

Starting from the portable implementation of HemoCell v1.4, the team members of Task 2.6 took the application and performed scaling test and optimisations on the Intel Skylake compute nodes.

The profiling of the application led to targeting the hotspots of the application, where the optimisations need to be directed.

In addition to the optimisation suggestions of the application, this report aims at determining the best practices for the implementation of numerical methods on emerging HPC architectures.

13.5 Organisation of this report

In the following chapters, the main results of the work effort are summarised. A brief description the HPC platform which hosted the computations, is carried out together with the dependencies of the application HemoCell. The scalability, profiling results are then presented. The following chapter is dedicated to the optimisation effort. The last chapter consists of suggestions of alternatives to Palabos, the LBM library which lies at the heart of the HemoCell code.

Setup (hardware configuration and Simulation parameters)

The HemoCell code (V1.4) was retrieved from the HemoCell webpage (www.hemocell.eu) and was setup from source. With respect to the compiling and running requirements, the following dependencies versions in Table 1 were used.

Dependency	Version
PU	Page 33
	Version 0.3

Intel C/C++ Compiler	18.0.5
Intel MPI	2018 Update 1
Cmake	3.10.0
HDF5	1.10.2
Palabos	2.0
GNU Patch	2.7.1

Table 1 HemoCell dependencies

The HemoCell code was installed on Bull's HPC medium scale platform. It is composed of several compute nodes of type Intel Skylake (16, 18 and 20 cores per socket and 2 sockets per node) with EDR interconnect. Lustre and GPFS parallel filesystems are available (Table 2).

Processor type (number)	Intel Skylake (Xeon Gold 6148)	Intel Skylake (Xeon Gold 6130)
Number of cores per socket	20	16
Number of threads per socket	40	32
Number of sockets per node	2	2
Base frequency	2.4 GHz	2.1 GHz
Cache	28 160 KB	22 528 KB
Bus speed	192 GB DDR4 2667 MT/s DR	192 GB DDR4 2667 MT/s DR
Thermal Design Power (TDP)	105 W	125 W
Instruction Set Extensions	Intel SSE4.2, AVX, AVX2, AVX512	Intel SSE4.2, AVX, AVX2, AVX512
Number of AVX-512 FMA units	2	2

Table 2 Main characteristics of the Intel processor used for Task 2.6

The HemoCell code provides a configuration file which sets up the parameters for the numerical simulation. One may find the description of the shape of the cells,

```
<ibm>
<shape> 1 </shape> <!-- shape: Sphere:[0], RBC from sphere:[1], Cell(defined):[2], RBC from file [3] RBC from Octahedron [4] Sphere from Octahedron [5] -->
<radius> 3.91e-6 </radius> <!-- Radius of the particle in [m] (dx) [3.3e-6, 3.91e-6, XX and 4.284 for shapes [0,1,2,3] respectively -->
<stepMaterialEvery> 20 </stepMaterialEvery> <!-- Update particle material model after this many fluid time steps. -->
<stepParticleEvery> 5 </stepParticleEvery> <!-- Update particles position after this many fluid time steps. -->
</ibm>
```

And more details about the domain geometry and the number of cells of reference direction.

```
<domain>
<geometry> tube.stl </geometry>
```

```
<fluidEnvelope> 2 </fluidEnvelope>
<rhoP> 1025 </rhoP> <!--Density of the surrounding fluid, Physical units [kg/m^3]-->
<nuP> 1.1e-6 </nuP> <!-- Dynamic viscosity of blood plasma, physical units [m^2/s]-->
<dx> 5e-7 </dx> <!--Physical length of 1 Lattice Unit -->
<dt> 1e-7 </dt> <!-- Time step for the LBM system. A negative value will set Tau=1 and calc. the
corresponding time-step. -->
<refDir> 1 </refDir> <!-- Used for resloution setting and Re calculation as well -->
<refDirN> 64 </refDirN> <!-- Number of numerical cell in the reference direction -->
<blockSize> -1 </blockSize>
<kBT>4.100531391e-21</kBT> <!-- in SI, m2 kg s-2 (or J) for T=300 -->
<Re> 10 </Re> <!--Reynolds number-->
<particleEnvelope> 25 </particleEnvelope>
<kRep> 2e-22 </kRep> <!-- Repulsion Constant -->
<RepCutoff> 0.7 </RepCutoff> <!-- RepulsionCutoff -->
</domain>
```

Scalability Tests

The scalability abilities of the HemoCell code are investigated in this chapter. First, the strong scalability is presented. The experiments were performed on Intel Skylake nodes (Intel Xeon 6130, see Table 2).

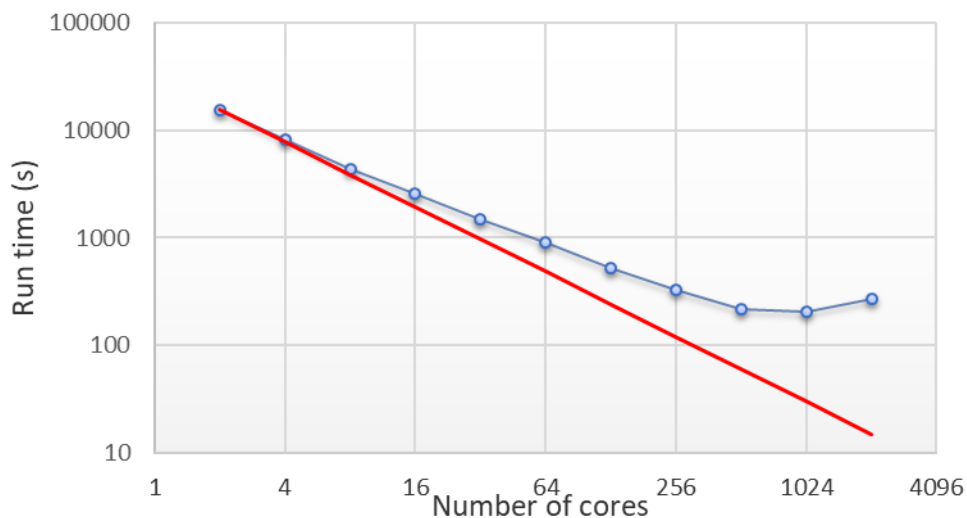


Figure 3 HemoCell Strong scaling. Red line denotes the ideal scaling curve.

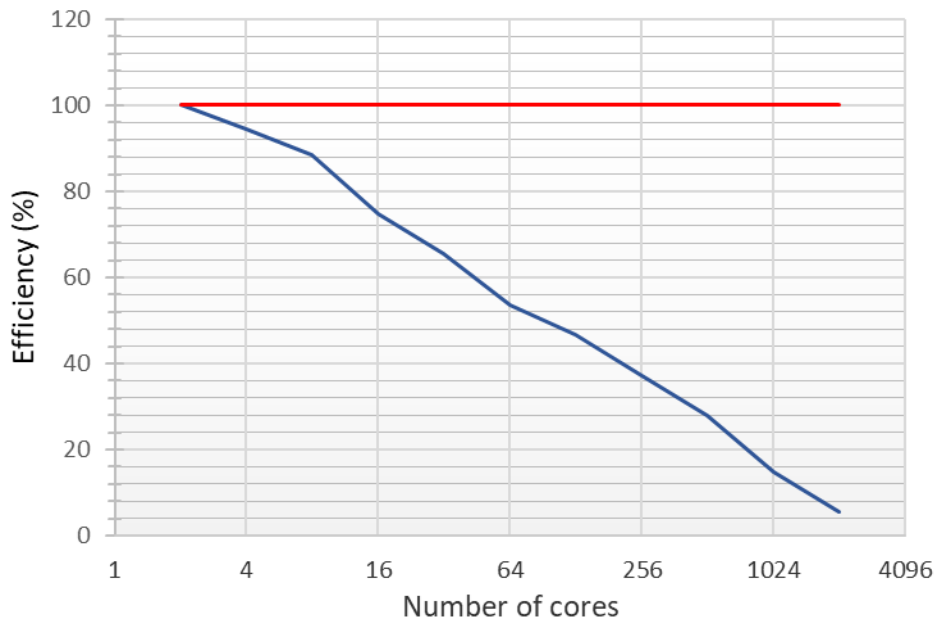


Figure 4 HemoCell strong scaling efficiency. The horizontal red line denotes the ideal cluster efficiency.

The efficiency drops below 70% within one node which can be improved. This strongly suggests that the ratio compute per core is decreasing rapidly and the synchronisation time takes an important part in the execution time. In the following, experiments on the weak scalability are performed in the same settings.

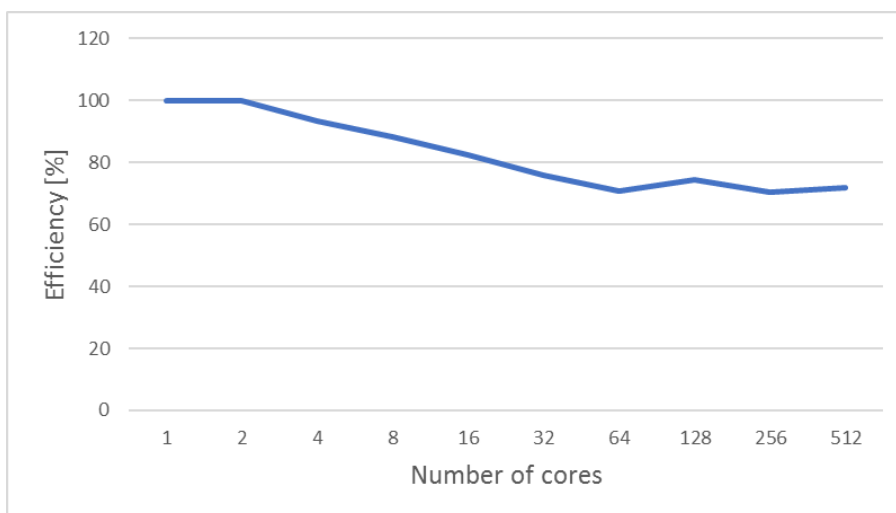


Figure 5 HemoCell weak scaling efficiency

Figure 5 shows the overall weak scaling performance of the test case which was provided on demand by UVA. The test cases were generated by doubling the size of the computational domain and filling it with blood cells such that the density of blood cells remains constant. The efficiency is settling around 70% for 512 cores.

To further understand where the scaling is dampened, the weak scaling of each function in the main computational loop is investigated in Figure 6.

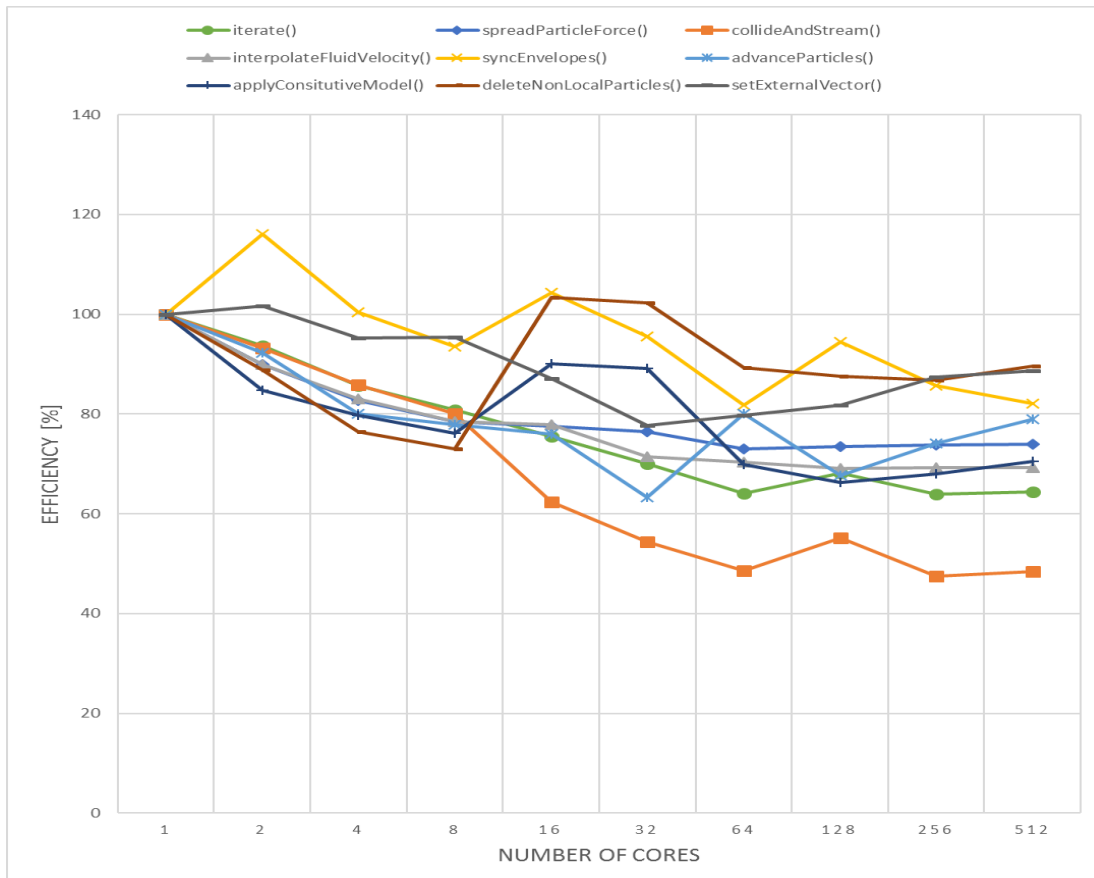


Figure 6 HemoCell weak scaling efficiency of inner functions

The LBM solver's efficiency is dropping rapidly. To improve the weak scaling behaviour of the code, the *collideAndStream* function is either not used correctly (e.g. with badly balanced load) or needs to be replaced by a more efficient, possibly single-instruction-multiple-data (SIMD) accelerated function.

Remark: The fact that a saturation of the efficiency at around 70% is observed, might suggest that we should not evaluate the scaling performance against the runtime with 1 node. In other words, the code might not have been designed to be run at very low core counts, thus we should maybe move the base to the right (e.g. 16 cores = 100% efficiency)

The weak scaling results are quite satisfying compared to the strong scaling results although weak scaling tends to work better as more workload tends to keep the processors busy.

Profiling

MPI time analysis

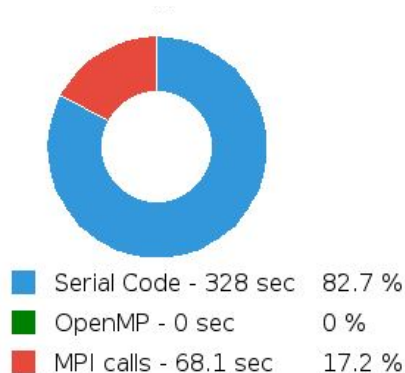


Figure 7 HemoCell application ratio – Intel profiling tool ITAC.

Figure 7 represents the ratio of all MPI calls to the rest of the code in the application. This information is collected by Intel Trace Analyzer and Collector (ITAC) formerly known as Vampirtrace. This tool is developed for analysing MPI communication which includes the tracing and analysing of MPI functions calls and messages being transferred.

Figure 7 shows that the total MPI time is quite large. More importantly, Figure 8 shows that half of this time is spent in MPI wait and MPI barrier which indicates synchronisation problems and wait at barriers for I/O calls.

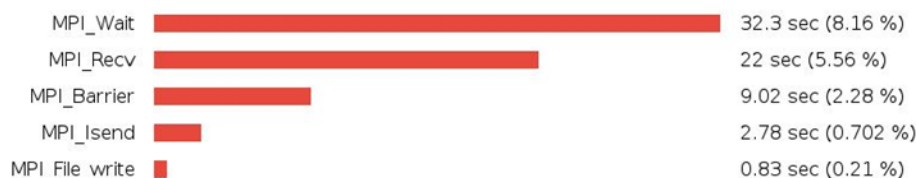


Figure 8 HemoCell top MPI functions

The list of the most active MPI functions from all MPI calls in the application are presented above in Figure 8.

Performance Issue	Duration (%)	Duration
Late Sender	11.56%	45.7379 s
Wait at Barrier	2.41%	9.52123 s
Late Receiver	0.39%	1.54951 s
Late Broadcast	0.00%	17.934e-3 s
Early Reduce	0.00%	2.762e-3 s

Figure 9 HemoCell late sender performance issue

Figure 9 shows the late sender problem due to imbalance in the MPI workload. Unlike the late receiver problem which can be mitigated with non-blocking communications,

the late sender issue can be mitigated by putting some effort in a better workload between MPI processes.

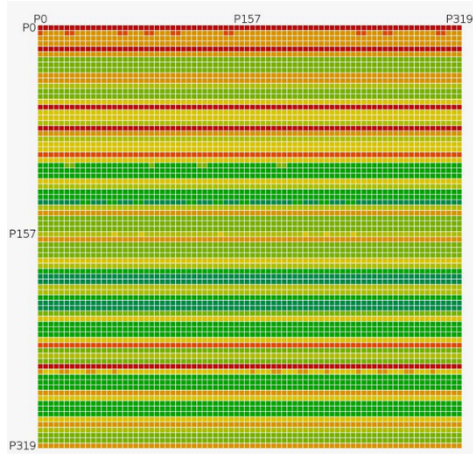


Figure 10 HemoCell global MPI communication time

Figure 10 shows the rank-to-rank communication matrix. This pattern underlines the unbalanced workload and specifically the *all-to-all* communication which becomes limiting when the number of cores is high enough.

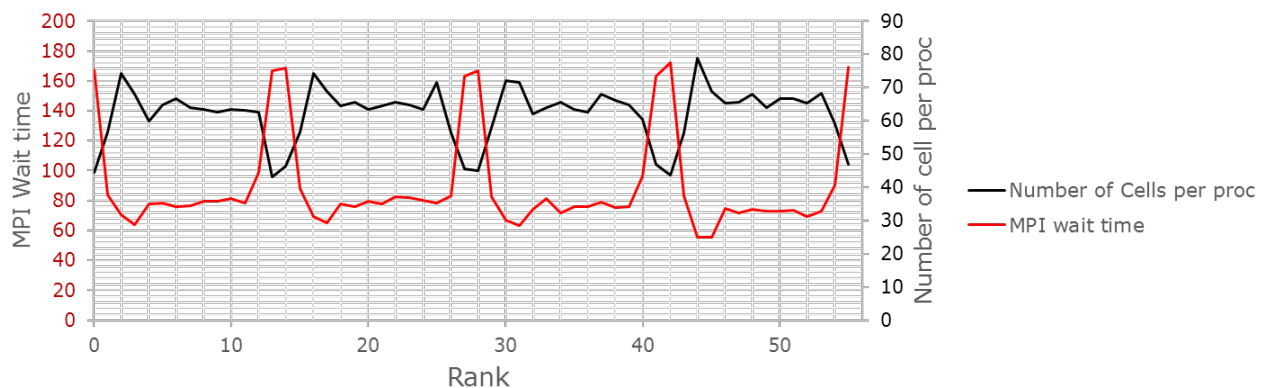


Figure 11 HemoCell MPI wait time vs workload at a fixed time step of the simulation

Figure 11 indicates the amount of MPI wait time versus the number of cells per rank at a fixed timestep which impacts the workload. MPI wait time is the highest for ranks with smallest number of cells e.g. workload.

Hotspot functions/loops

A global analysis of the HemoCell application shows that it divides mainly in calls to the LBM solver called Palabos and I/O using the HDF5 library for storing intermediate snapshots at regular timesteps and checkpointing (see Figure 12).

■ Palabos ■ Hemocell ■ I/O (hdf5)

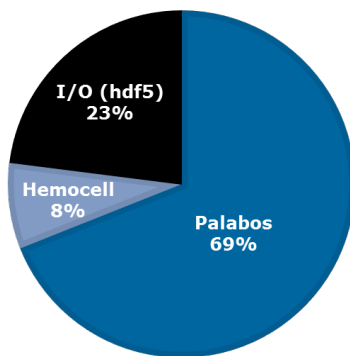


Figure 12 HemoCell profile chart

At the compilation level, the flags “-O3 -xHost -qopt-zmm-usage=high” enables optimisations for code speed quite aggressively. The compiler performs some basic loop optimisations (transformations such as fusion, Block-Unroll-and-Jam and collapsing IF statements), inlining of intrinsic, intra-file interprocedural optimisation and most common compiler optimisation technologies. The “-xHost -qopt-zmm-usage=high” allows to target the highest ISA, e.g. AVX-512, which enables vectorisation when possible.



Figure 13 HemoCell Memory stalls and floating-Point Instruction usage – Intel Application Performance Snapshot

Figure 13 shows that, despite the high capacity of the instruction set of the processor, the compiler could not retrieve a good vectorisation of the code leading to an almost fully scalar floating-point instruction. For instance, the loop at line 95 in the Palabos solver, *interpolationCoefficientsPhi2()* which is the a hotspot loop (see Figure 14), the compiler could not vectorise the loop due to multiple IF statements with an exit (Figure 15). Similarly, the loop at 543 in the *blockLattice3D.hh*, *blockwisebulkCollideAndStream()*, the compiler is unable as well to vectorise due to the impossibility to compute in advance the loop count (Figure 16).

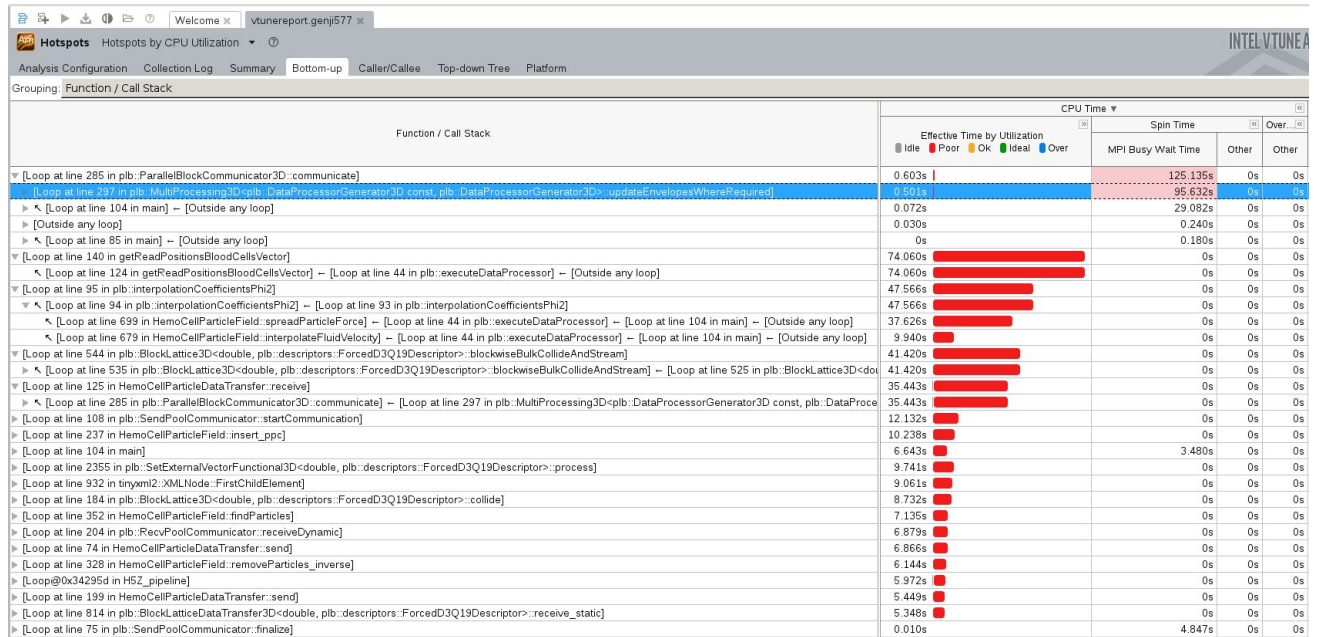


Figure 14 HemoCell BottomUp hotspot loops – Intel VTune

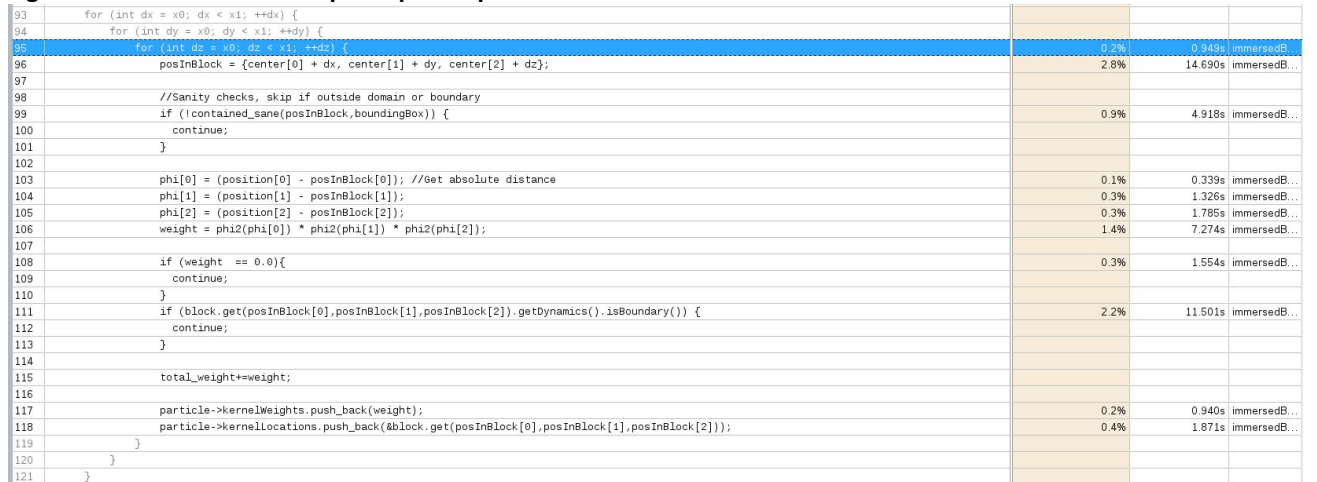


Figure 15 Palabos solver, interpolationCoefficientsPhi loop – Intel VTune



Figure 16 Palabos solver, blockwiseBulkCollideAndStream loop -- Intel VTune

13.6 Optimisation Tracks

Attempt to utilise full potential of x86 instruction set

Early in the hotspot analysis, the compiler was only capable of vectorizing small chunks of the code. About 99% of the executable and with it the computationally extensive regions were composed out of serial instructions. As the computationally most

expensive region, the focus is set on the *blockwiseBulkCollideAndStream()* function in *blockLattice3D.hh*.

It is responsible of advancing the fluid according to the Lattice Boltzmann Method. The rather complex nature of the 6 nested for loops made it necessary to traverse the call stack down to where the actual computations are performed. In the following, the subroutines *addNaiveForce()* and *addGuoForce()*, which are implemented in *externalForceTemplateD.h* are investigated.

Within these functions, the force with respect to each neighbouring cell is computed. The computations differ slightly, depending for instance on the distance of the centre points of the respective cells. To enable vectorisation or fused multiply-add (FMA) operations, we need to unify these computations and even more importantly, the data on the random-access memory (RAM) must be aligned. In principle, we archived this by introducing an additional buffer which stored precomputed results which had to be computed individually and once this was done, vector operations were performed on the buffer.

```
static T A[27] = {...} __attribute__((aligned(64)));
static T B[27] = {...} __attribute__((aligned(64)));
for(unsigned int i = 0; i < 9; ++i){
#ifdef _USE_BLAS
    C[i] = blas_ddot(3, &(A[i*3]), 1, &(B[i*3]), 1);
#else
    C[i] = A[i*3] * B[i*3] + A[i*3 + 1] * B[i*3 + 1] + A[i*3+2] * B[i*3+2] ;
#endif
}
//some more computations to get to the final result for array f[]
```

Throughout multiple test, the structure was altered to obtain more performance. For instance, instead of statically allocated buffers, a memory pool was used. Also, as shown in the sample code, BLAS level one routines were put in place.

However, this technique did not lead to the desired decrease in execution time. Unfortunately, the way the 6 nested loops operate, results in rather short loops. Thus, the vector length is very limited. The benefits of using SIMD instructions were at best about even with the effort to prepare and align the data at each iteration. Additionally, the BLAS routines couldn't show their full potential, due to the short vector length. Eventually, the idea to vectorise computations within a loop iteration had to be abandoned.

However, one could argue vectorisation should be feasible across loop iterations, thus inlining all function calls and vectorizing the loop itself.

13.7 Loop vectorisation

To archive vectorisation, the function which were called within the loop were declared SIMD.

```
#pragma omp declare simd [list]
```

Next, vectorisation is enforced on the loop, while disrespecting hits from the compiler whether it is efficient or not. The data access pattern can be found in the overloaded operator in

```
$(Palabos_src_dir)/core/implicitGrid3D.h .
```

It depends on the 3 inner loop variables and is linearised as such:

```
[z + nZ*(y + nY*x)].
```

Where x, y and z are the loop iteration counters of the three inner loops, respectively. nZ and nY are the array bounds in their respective dimensions.

Note, that the operator (), which is decaled and defined in implicitGrid3D.h is shadowed by the definition given in

```
$(Palabos_src_dir)/atomicBlock/blockLattice3D.hh.
```

The access pattern, however, remains unchanged.

Compiler messages show, that the force calculation in the collision step was successfully inlined into the loop.

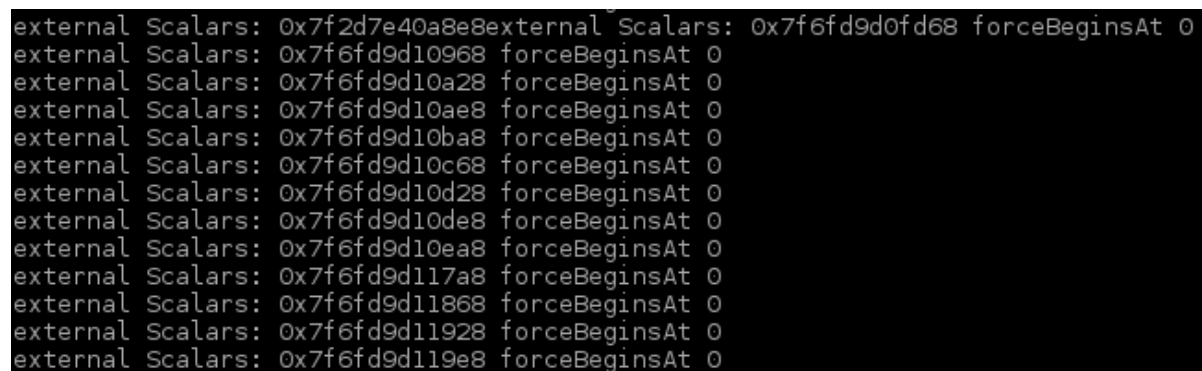
Unfortunately, the way the data is structured in RAM is not allowing for an efficient vectorisation. Each cell has a size of 192 Bytes and they are consecutively placed in storage. Thus, from one loop iteration to the next, 192 Bytes must be traversed. This is too much to loaded efficiently into vector registers. During the compilation the compiler is giving the following output:

```
remark #15415: vectorization support: non-unit strided load was generated for the variable <grid-
>parent->rawData->data[innerZ+?(innerY+?*innerX)][4]>, stride is 24 [
/home_nfs_robin_ib/bkarlshoeferp/hemocell_perf_test/hemocell_vec/hemocell-
1.4/palabos/src/core/array.h(66,16) ]
```

The stride being 24 is counted in elements of double precision. Consequently, there is:

```
24 * sizeof(double) = 192
```

During runtime, this jump is observed in Figure 17.



```
external Scalars: 0x7f2d7e40a8e8external Scalars: 0x7f6fd9d0fd68 forceBeginsAt 0
external Scalars: 0x7f6fd9d10968 forceBeginsAt 0
external Scalars: 0x7f6fd9d10a28 forceBeginsAt 0
external Scalars: 0x7f6fd9d10ae8 forceBeginsAt 0
external Scalars: 0x7f6fd9d10ba8 forceBeginsAt 0
external Scalars: 0x7f6fd9d10c68 forceBeginsAt 0
external Scalars: 0x7f6fd9d10d28 forceBeginsAt 0
external Scalars: 0x7f6fd9d10de8 forceBeginsAt 0
external Scalars: 0x7f6fd9d10ea8 forceBeginsAt 0
external Scalars: 0x7f6fd9d117a8 forceBeginsAt 0
external Scalars: 0x7f6fd9d11868 forceBeginsAt 0
external Scalars: 0x7f6fd9d11928 forceBeginsAt 0
external Scalars: 0x7f6fd9d119e8 forceBeginsAt 0
```

Figure 17 Starting address for adjacent cells. The storage location is given in HEX, and the displacement is indeed 192 Bytes (e.g. the first two lines yield $0x7f6fd9d10968 - 0x7f6fd9d10a28 = 0xC0 = 192$).

Thus, two corresponding force entries, e.g. Cell[0]->force[0] and Cell[1]->force[0], are 192 Bytes apart (Figure 18). That this is too much for AVX512. Vectorisation which reaches across loops in the innermost loop nest is basically ruled out.

Essentially, there is an Array of Structs (AoS) where the Array is “rawData” and the Struct is “Cell<...>”. Thus, the stride becomes large and consequently vectorisation unbeneficial.

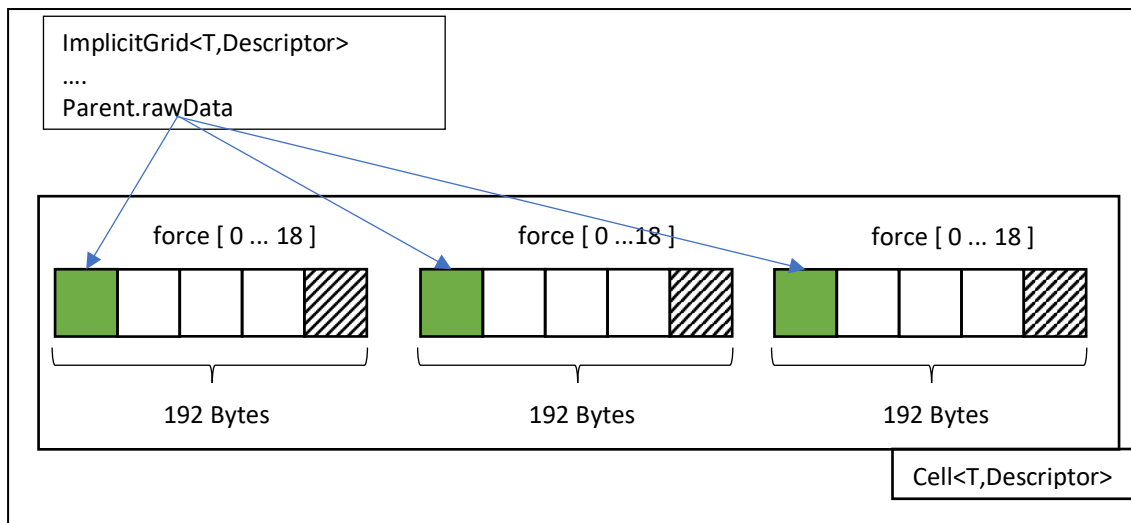


Figure 18 Orange highlight: section, which is loaded into AVX register.

A Struct of Arrays (SoA) would be better:

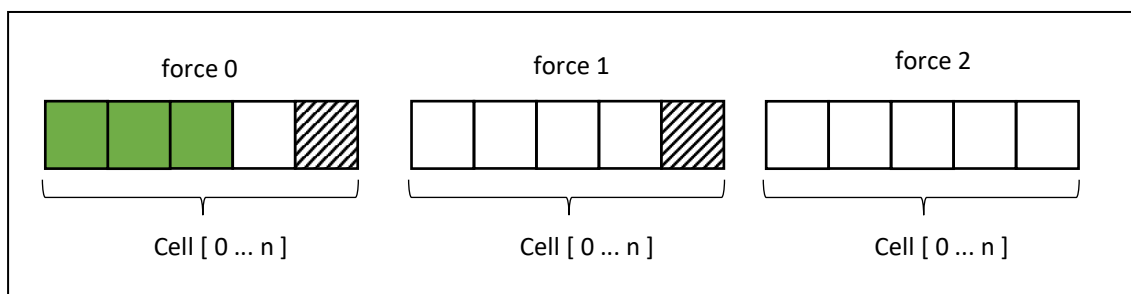


Figure 19 Struct of Arrays, better memory localisation for AVX register

The orange-highlighted section in Figure 19 represents the memory which is loaded into the vector registers at one point in the computation. Obviously, a SoA is superior to an AoS.

Restructuring the data layout would be expensive. The Layout of a cell is described in: `$(Palabos_src_dir)/latticeBoltzmann/nearestNeighborLattices3D`.

The description is rather low level, essentially stating the offset from the beginning of the object to the requested variable.

```
struct Force3dDescriptor {
    static const int numScalars = 3;
    static const int numSpecies = 1;
    static const int forceBeginsAt = 0;
    static const int sizeOfForce = 3;
};
```

In practice, this looks like the following (externalForceTemplates3D.h):

```
Array<T,Descriptor<T>::d> externalScalars = ... ;
```

`T* force = (T*)externalScalars + forceBeginsAt;`

Modifying the actual address, where it is pointed to, makes it hard to track in which parts of the program might break if the code is modified from AoS to SoA access. Additionally, other functions might benefit from the fact that data is clustered by cell, not by orientation in space.

So far, we haven't even talked about data dependencies. Clearly, they pose an issue, as can be seen in the Figure 20.

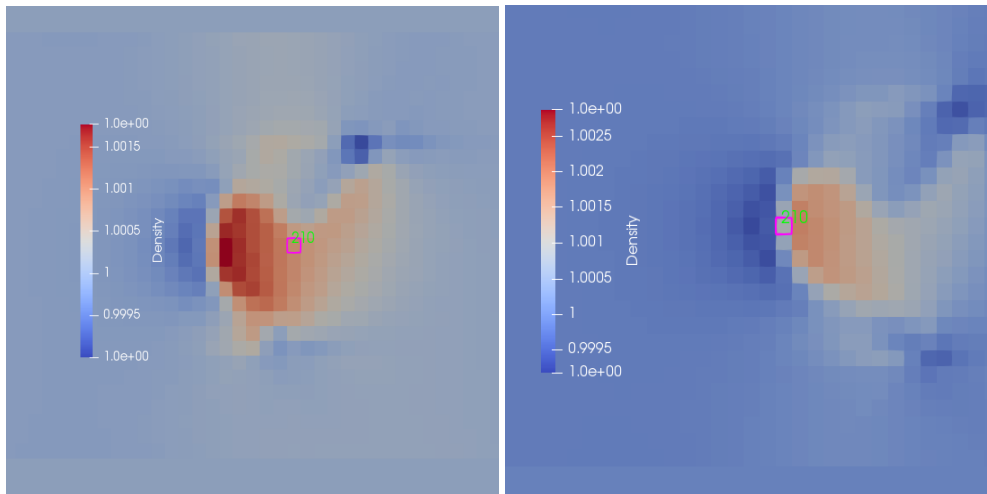


Figure 20: Location of cell with ID 210 in the density field at the same point in time in 2 identical runs. The value of the density is different.

Also, slight differences can be observed while investigating the force vector field in Figure 21 and Figure 22.

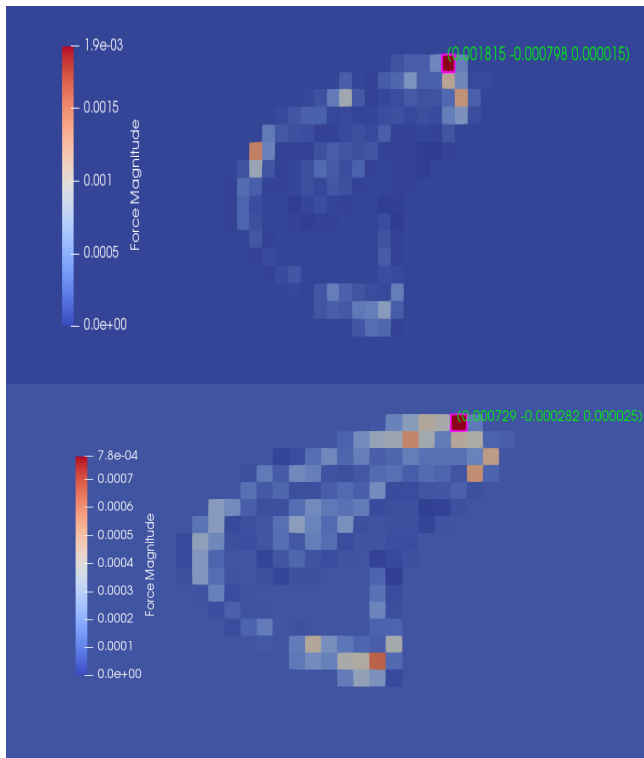


Figure 21 Difference in the force field becomes evident between two identical runs

However this might not be inherited by the forced vectorisation, as it is discussed in the next section.

13.8 Reproducibility

Another subject is reproducibility of results from simulations. Runs with identical input data do not lead to identical output files. The force vector field as well as the scalar field of density do vary potentially up to 10% as shown in Figure 22.

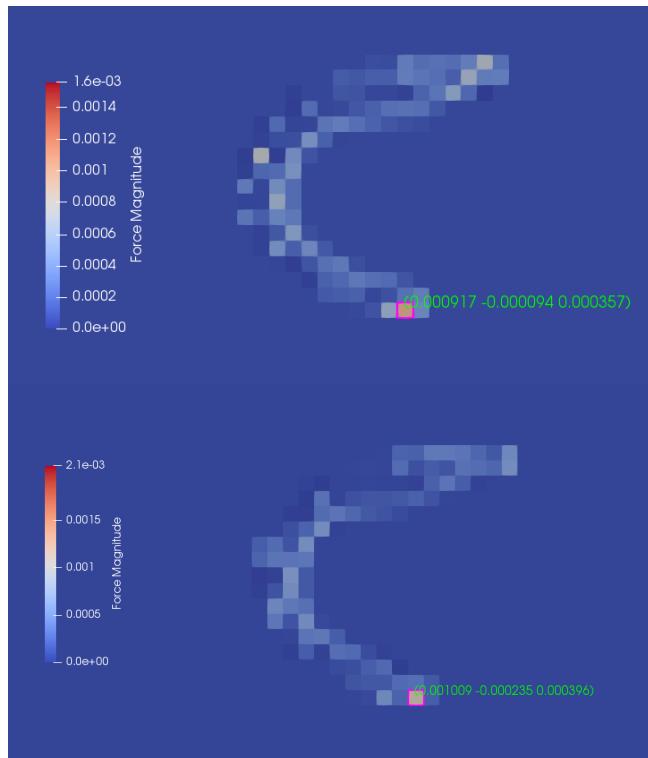


Figure 22 Difference in the force field becomes evident between two identical runs, once the values are displayed (left: $[9.17e-4, -9.4e-5, 3.57e-4]$ and right: $[1.0e-3, -2.35e-4, 3.96e-4]$).

13.9 Single Node Optimisation

Upon realising that the code is parallelised with MPI only, the idea of implementing shared memory parallelism within Palabos was considered using OpenMP. The 6-times nested loop, which is mentioned above, was targeted, again. However, the granularity of a single task was too small within the 3 inner nests of the loop. The benefits of calculating the loop in parallel were outweighed by the overhead to fork the threads pool. Going to the outer levels of the loop, loop and data dependencies prevented to go further.

Eventually, without refactoring the code of Palabos, light weight parallelism is not beneficial.

13.10 Possible alternatives to Palabos

As a major issue for the poor scalability of HemoCell, we had to investigate the Palabos library. Palabos manages the advancement in time via the Lattice Boltzmann Method and does the initial domain decomposition and voxelisation.

As described above, the rather poor scaling performance of the HemoCell code on a compute cluster is due to the uneven spread of the workload amongst the MPI tasks (processes). So far, the load balancing appears to be static. Thus, it is not flexible over

time and does not react or depend on the actual amount of blood cells in one multiblock.

We recognise the fact that HemoCell will soon incorporate ParMetis to do load balancing. However, it might be easier and more efficient to outsource the task to the LBM solver itself. Unfortunately, Palabos does not provide this feature.

In addition to the lack of a load balancing routine, Palabos is relying mostly on MPMD programming (MPI) to do the parallelism. The lack of the shared memory programming model (e.g. OpenMP) lays even more weight on very accurate load balancing since imbalances can usually be handled more easily in one continuous block of memory.

Lastly, during many compilations of Palabos, we did detect only a very small percentage of SIMD capable code. This makes us believe, the Palabos might not be the optimal choice, especially since exist other, highly optimised LBM solvers. In the following, we would like to suggest a very brief overview of possible alternatives to Palabos.

OpenLB

OpenLB is, as the name suggests, open source and hybrid (MPI + OpenMP). The code has implemented its own load balancing routine. Moving from Palabos might be facilitated by the fact that it supports “.stl” geometries.

WaLBerla

Also, a hybrid code (OpenMP + MPI), which is written in C++. In contrast to Palabos it does support multiple LBM collision/streaming models. It has a strong remark on GPGPU usage, which suggests, that the code uses SIMD instructions. The domain decomposition is performed in parallel and the code is available (open source). It might be worth a look since it already ran on SuperMUC with 10^{18} cells.

13.11 Conclusion

In this report, a detailed profiling and scalability runs of the HemoCell application have been presented in addition to a set of optimisation tracks and their corresponding results.

The main outcome resides in the fact that the HemoCell code consists in an MPI implementation of the Lattice Boltzmann Method of Palabos and intensive I/O through the HDF5 library. The HPC code suffers from a lack of a load balancing method which causes the application to perform poorly. Fortunately, the upcoming version of HemoCell shall include a load balancing technique (Parmetis library). It was noticed that the Palabos library could not fully benefit from the vectorisation possibilities of the compiler and the processor architecture despite many intrusive and non-intrusive optimisations.

As a conclusion, a better memory management or data structure can lead to a higher vectorisation. One may suggest an LBM implementation as well which may benefit from an OpenMP parallelisation. The latter shall take advantage from future multi-threaded architectures.

Meetings

Date	Topic	People
13 Sep 2018	HemoCell: run and optimisation repots	V. Azizi, E. Raffin, O. Hamitou, P. Karlshofer
26 Sep 2018	Large case study	V. Azizi, O. Hamitou, P. Karlshofer
8 Nov 2018	Discussion on final output	V. Azizi, O. Hamitou, P. Karlshofer, E. Raffin

14 Appendix C: Report of UCL's exascaling BAC and HemeLB.

This Appendix is a copy of relevant Sections from the ComPat deliverable “D3.3: Report on performance measurements and prediction of HPMC Application”. This is a public document; however, at the time of writing, the report is currently unavailable. Thus, for the sake of completeness, the relevant sections are included here, namely Section 2.1 and the relevant part of the general Conclusions. Finally, references have been inlined for clarity.

Largest scale performance tests

In this section, we detail the performance studies carried out on ComPat applications on the largest resources. These were typically tested on resources larger than that available under the Experimental Execution Environment. A weak scaling and a strong scaling application are considered, and some exascale predictions are formulated.

Binding Affinity Calculator (replica-based representative)

For multiscale applications following the replica computing pattern, it makes more sense to think in terms not of a single simulation, but rather of simulation campaigns. It is the orchestration of these campaigns that is key here, and therefore the choice and performance of the middleware is highly important, particularly as regards the efficient use of putative exascale resources.

In the first year of this project, the BAC was on the fast track and, with the giant, full-SuperMUC run, we demonstrated the feasibility of such enormous RC runs. In the second year, BAC was the application with which we developed and built the RC pattern, and we demonstrated that with RC we can aid in running BAC on non-trivial distributed environments. Finally, this year we want to demonstrate how BAC behaves on a single resource, with systematic studies of weak scaling. By adding up these three parts, we get the full picture whereby all ingredients are ready to go into production, using RC, the pilot jobs, and so on.

Our selected multiscale application demonstrating replica computing is the High Throughput Binding Affinity Calculator (HTBAC), which builds upon the RADICAL Cybertools (a middleware component of the COMPAT stack), as the framework solution to support the coordination of the required scale of computations, allowing the exploitation of thousands of cores at a time.

To determine the performance of HTBAC, particularly as regards the extension to extreme parallelism, a number of performance studies were carried out. The main resource used was NCSA Blue Waters, with additional runs on LRZ SuperMUC and ORNL Titan.

Scalability and resource usage

We explored the performance of HTBAC on NCSA Blue Waters with two different protocols:

1. ESMACS (Enhanced sampling of molecular dynamics with approximation of continuum solvent), consisting of 25 replicas, i.e. 25 pipelines
2. TIES (Thermodynamic integration with enhanced sampling) consisting of 13 lambda windows and 5 replicas, i.e. 65 pipelines

Both protocols run for a total of 6 ns simulation durations. ESMACS produces 3.5 GB/system (24 MB/ns) while TIES produces 10 GB/system (24 MB/ns). Each simulation step in TIES and ESMACS requires 32 cores. Protocols run approximately 10-12 hours, depending on the physical system and the number of timesteps provided by the user.

When considering an application following the replica computing (RC) pattern, the most pertinent performance property is that of weak scaling. This is also the most scientifically relevant property, as it demonstrates the ability of HTBAC to solve large number of drug candidates in essentially the same amount of time (as the resources increase).

To this end, in our first study we investigated the weak scaling behaviour when screening sixteen drug candidates concurrently using thousands of multi-stage pipelines on more than 32,000 cores on NCSA Blue Waters (we observed similar scaling on other platforms such as ORNL Titan for different protocols).

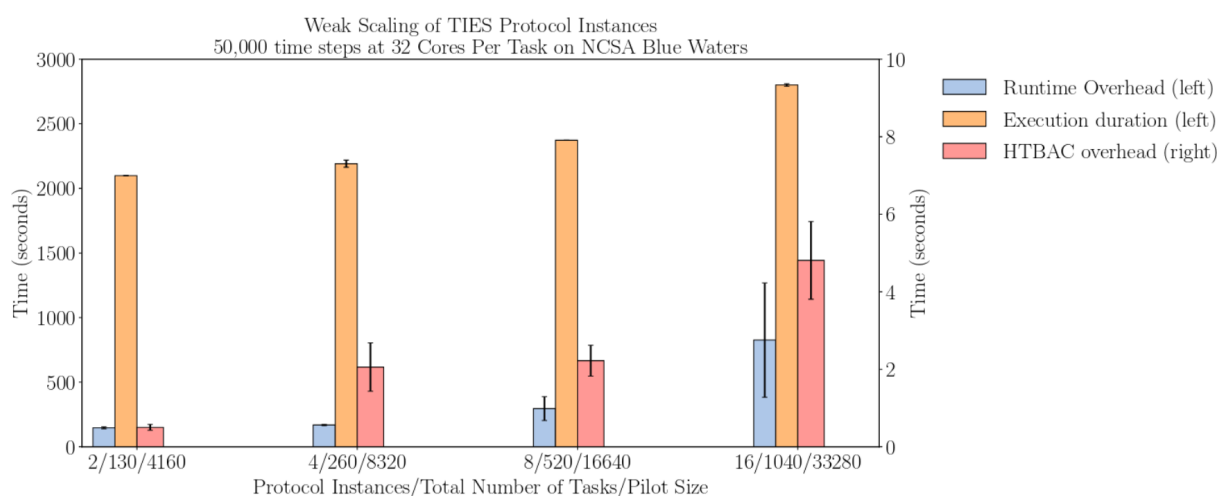


Fig. 1: Weak scaling properties of HTBAC. We investigate the weak scaling of HTBAC as the ratio of the number of protocol instances to resources is kept constant. Overheads of HTBAC framework (right), and RCT overhead (left) and total execution time TTX (left) for experimental configurations investigating the weak scaling of TIES. We ran two trials for each protocol instance configuration. Error bars in TTX in 2 and 8-protocol runs are insignificant.

A detailed representation of the weak scaling performance of HTBAC for the TIES protocol is presented in Fig. 1, demonstrating almost perfect scaling to hundreds of concurrent multi-stage pipelines.

In our second set of studies [“Concurrent and Adaptive Extreme Scale Binding Free Energy Calculations” Dakka et al., 2018 (<https://arxiv.org/abs/1801.01174>)] we carried out a number of experiments on Blue Waters using both the ESMACS and TIES protocols. We present here the results of the weak scaling experiments:

ID	Type of Experiment	Physical System(s)	Protocol(s)	No. Protocol(s)	Total Cores
1	Weak scaling	BRD4	ESMACS	(2, 4, 8, 16)	1600, 3200, 6400
2	Weak scaling	BRD4	TIES	(2, 4, 8)	4160, 8320, 16640
3	Weak scaling	BRD4	ESMACS + TIES	(2, 4, 8)	5280, 10560, 21120

In Fig. 2 we show (a) the weak scaling of HTBAC with the TIES protocol, (b) with the ESMACS protocol, and (c) with instances of both TIES and ESMACS protocols.

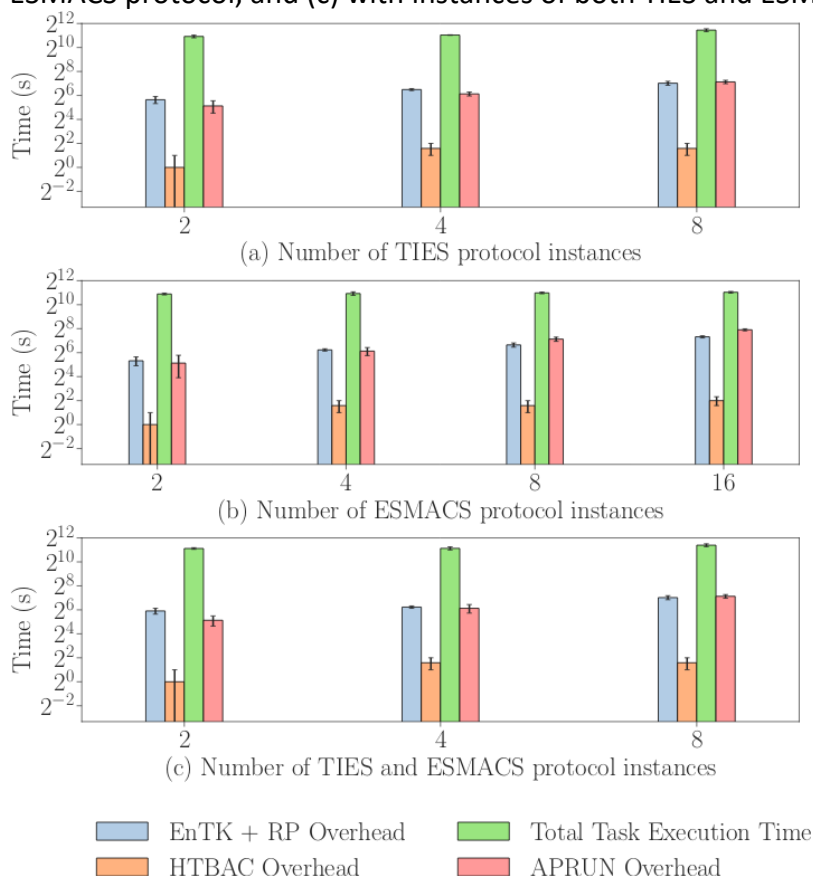


Fig. 2. Weak scaling of HTBAC. The ratio number of protocol instances to resources is constant. Task Execution Time with and HTBAC, EnTK+RP, aprun overheads with (a) TIES (Experiment 1), (b) ESMACS (Experiment 2), and (c) TIES and ESMACS (Experiment 3).

For all weak scaling experiments (1–3) we used physical systems from the BRD4-GSK library (16 ligands made available for this work by GlaxoSmithKline) with the same number of atoms and similar chemical properties. The uniformity of these physical systems ensures a consistent workload with insignificant variability when characterizing their performance under different conditions.

In all weak scaling experiments (Fig. 1 and 2) we observed minimal variation in the task duration as the number of protocol instances increases. We conclude that HTBAC

shows near-ideal weak scaling behaviour under the conditions tested. The overhead for the TIES results includes the adaptive sampling algorithms. The HTBAC overhead depends mostly on the number of protocol instances that need to be generated for an application. This overhead shows a super linear increase as we grow the number of protocol instances, but the duration of the overhead is negligible when compared to Total Task Execution Time.

This detailed performance data supplements and reinforces our earlier experiences of the excellent weak scaling of the BAC on large supercomputing platforms such as LRZ SuperMUC in 2016, in which both phases (a total of 250,000 cores) were used simultaneously for 37 hours, testing 50 candidate drugs and generating around 5 terabytes of data¹.

Node failure rate

The probability of node failures is likely to increase as supercomputers are constructed with ever larger numbers of nodes, and might therefore become significant on some exascale platforms. However, on the resources used for our performance measurements, we observed typically very few node failures, even when under high stress. During a campaign of 64 proteins, 25 replicas each, and 2-4 nodes per replica (executed on Blue Waters), only 2 node failures occurred (even though this campaign was executed twice). It should be noted that these two campaigns were executed shortly after Blue Waters came back online after a shutdown period, so the system may have been in a more stable state than after a long period of continuous usage. Nevertheless, there is little evidence on present systems (even when using hundreds of thousands of cores) that node failures will significantly impact the scalability of HTBAC in the short to medium term, although unforeseen issues might well arise on the e.g. billions of cores a full exascale machine may contain. This remains an active research topic, and in principle we understand, in the context of RC, how to deal with potential node failure in an automatic way. However, given this experiment we have not yet implemented automatic detection and recovery of node failures into the RC pattern. We intend, as larger machines come available, to continue running such huge campaigns to understand the actual node failures, and when needed, to realise fault tolerance and recovery mechanisms into the RC pattern.

Conclusions and prediction for Exascale

Extrapolating from the promising weak scaling performance analysis presented above, we might expect good scaling of replica based applications at even greater node counts. Our studies have not yet shown any limitations that might preclude efficient use of exascale services. Differences in architecture and hardware may, naturally, affect this, and as the COMPAT stack matures, we will obtain more performance data to further clarify the viability of such applications on exascale machines.

As we demonstrated in deliverable D2.2 and D3.2, replica computing can also very well be executed in a distributed mode, running replicas across a range of supercomputers,

¹ http://www.gauss-centre.eu/SharedDocs/Pressemitteilungen/GAUSS-CENTRE/EN/2016-03_SuperMUC_Pers_Med.html?nn=1290050

with the multiscale computing patterns algorithms and software facilitating the detailed deployment. We continue to explore these capabilities, but this will require a production ready distributed supercomputing environment, such as the ComPat EEE. The European Supercomputing landscape would, in our opinion, very much benefit from such an environment, e.g. operated under the governance of PRACE. We have demonstrated that our middleware (QCG) is production-ready, and that our RC pattern is capable of exploiting such distributed HPC resources in a very efficient way. In combination with the weak scaling performance as reported in this deliverable, this would even allow us to reach the Exascale on a RC application by aggregating the power of sub-exascale machines. To conclude, ComPat has demonstrated that this is a viable option.

HemeLB (monolithic representative)

The Experimental Execution Environment did not have sufficiently large resources for determining large scale monolithic runs. As we have argued in Deliverables D2.1 and D3.1, and demonstrated in D2.2 and D3.2, the primary models in Extreme Scaling patterns are large scale monolithic codes. We have already demonstrated how the multiscale computing patterns algorithms and software can efficiently deploy Extreme Scaling applications on the EEE. The next step is to study in detail if and how the primary models themselves can scale to the largest HPC machines currently available to us.

We therefore used ARCHER (up to 96k cores) and Blue Waters (up to 300k cores) for our largest runs. The ARCHER supercomputer in Edinburgh, UK is a Cray XC30, with dual 12-core Intel Xeon E5-2697v2 (Ivy Bridge) 2.7 GHz processors joined by two QPI links, connected via a proprietary Cray Aries interconnect in a dragonfly topology. The Blue Waters supercomputer in Illinois is a Cray XE6/XK7 system consisting of more than 22,500 XE6 compute nodes (each containing two AMD Interlagos processors, with 8 floating point cores each).

Scalability and resource usage

Unlike the Replica Computing case explored earlier in this report, the most scientifically relevant scaling for such a monolithic application was determined to be strong scaling. While weak scaling would allow (physically) larger systems to be simulated in the same time (on more cores), the characteristic time scales of processes of interest typically scale as a power (greater than or equal to 1) of the system size, and thus aiming for constant wall clock time would not yield scientifically useful results.

Instead, we focussed on how a system of fixed size might be simulated faster through the use of more cores (on the same supercomputer). Our test system was the circle of Willis, an important vascular system located at the centre of the brain (and a region in which many aneurysms form). Such a system can already be simulated using (coarser) finite element methods, but we use it here as a useful geometry for benchmarking. On EPCC ARCHER, we benchmarked with a 15 micrometre resolution geometry (777 million fluid sites), and on NCSA Blue Waters we used a 7 micrometre resolution geometry (5.5 billion fluid sites). Note that in both cases the geometries are highly

sparse ($\ll 1\%$ fluid fraction), posing challenges for load decomposition as compared with a dense geometry.

The results of the performance measurements on ARCHER are shown in Figs. 3 and 4 [Patronis, A., Richardson, R. A., Schmieschek, S., Wylie, B. J., Nash, R. W., & Coveney, P. V. (2018). Modelling Patient-Specific Magnetic Drug Targeting within the Intracranial Vasculature. *Frontiers in physiology*, 9, 331.]. The profiling of the code was carried out using the parallel performance tool, Scalasca (<http://scalasca.org/>).

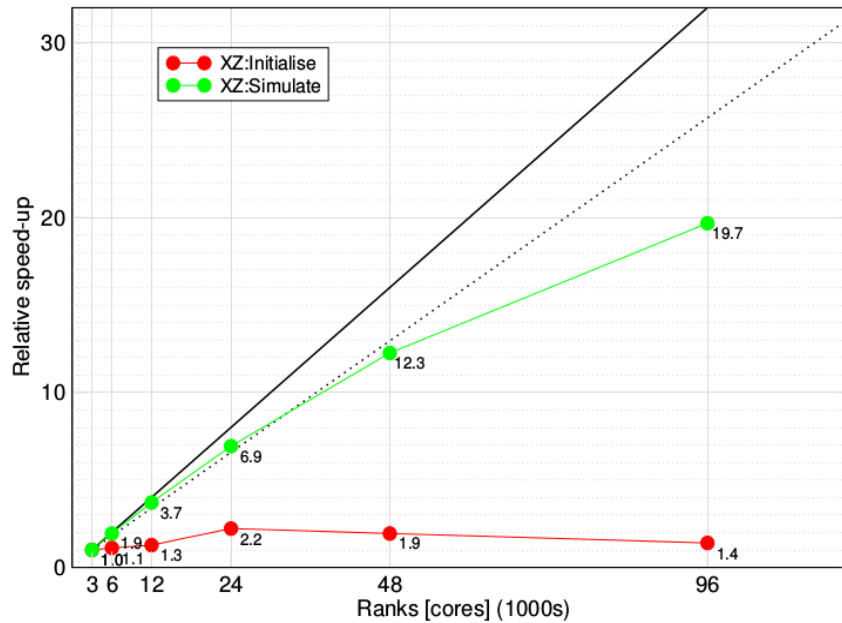


Fig. 3: Strong scaling of HemeLB up to 96k cores on EPCC ARCHER, showing both initialisation and simulate phases.

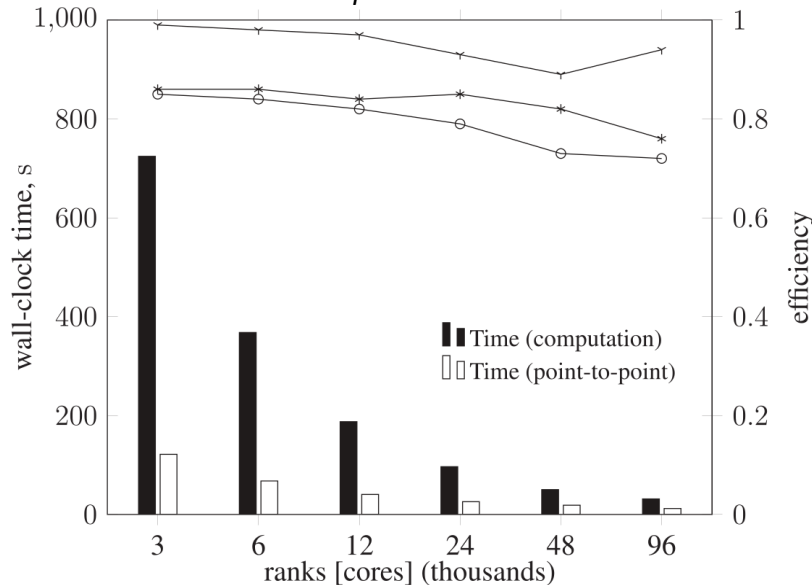


Fig. 4: Wall clock time and efficiency metric for strong scaling of HemeLB on EPCC ARCHER, up to 96k cores.

In Fig. 3 we see the speed-up of HemeLB from 3000 cores to 96000 cores, while Fig. 4 shows the corresponding measured wall-clock time, and measure of parallel efficiency.

There is a negligible amount of MPI collective communication, and the amount of non-blocking point-to-point communication for data exchange decreases in proportion to computation time. Therefore, communication efficiency remains above 0.89. Load balance, however, starts at 0.86 and progressively deteriorates to 0.76, such that the overall parallel efficiency degrades to 0.72 once at 96,000 cores.

In our second study, we performed benchmarking of HemeLB on NCSA Blue Waters up to 300000 cores, using a higher resolution system (with approximately double the number of fluid sites). At such a high number of cores and low fluid site count per core (approximately 5000 sites per core) it was more challenging to avoid overheads from the use of a profiling tool such as Scalasca, so we focussed only on wall clock time per run. The resultant performance data is given in the following table:

# cores	# nodes	wall clock time (simulate phase) [s]
16000	1000	3490.868
32000	2000	1799.434
64000	4000	0942.717
128000	8000	0494.497
256000	16000	0376.673
300000	32000	0557.471

Table 1: Data for strong scaling study on Blue Waters plotted in Fig. 5.

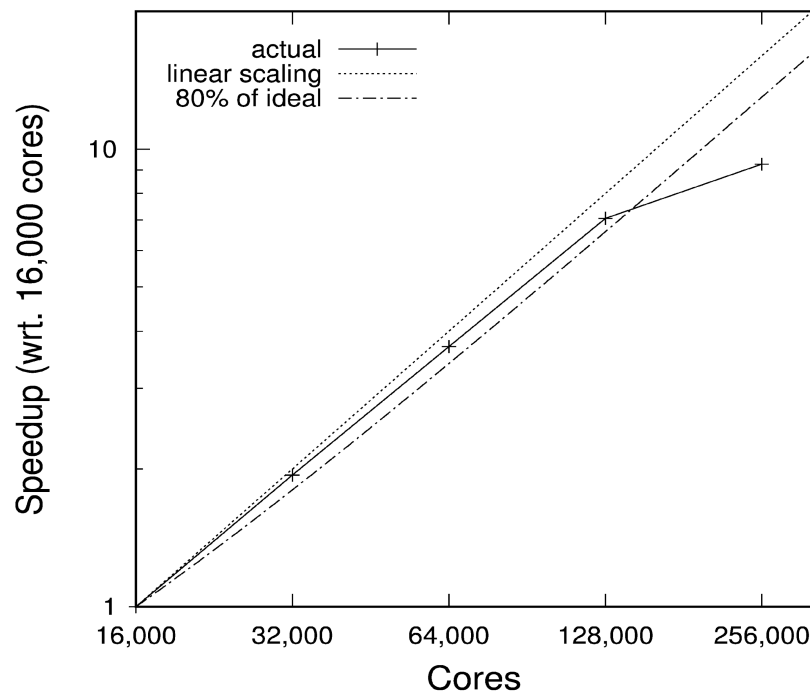


Fig. 5: Strong scaling of memory-optimized HemeLB on Blue Waters, up to 256k cores, for a 5.5 billion fluid site circle of Willis geometry.

In Fig. 5, we see the results of the strong scaling on of HemeLB on Blue Waters, shown here up to 256k cores. The performance degradation thereafter is attributed to significant load imbalances (due to the difficulty of minimising the communication surface in such a complex, sparse geometry) and the very low computational load per core (5000 sites on average).

It was unfortunately not possible to obtain energy usage information from ARCHER or Blue Waters (they do not make this information available to users).

Node failure rate

Node failure rate on BW was low, even at 300k cores, although on exascale machines this is expected to be a significant issue - monolithic applications will be particularly vulnerable to this. Similarly, on ARCHER we found a negligible node failure rate under normal conditions - however: when running multiple OOM jobs over the whole system, subsequent jobs appeared to fail on the released nodes.

Conclusions and prediction for Exascale

Our monolithic test application used in the above performance analysis shows very good strong scaling for the given system sizes. However, due to the locality of interactions in the lattice-Boltzmann formulation (and hence locality of information transfer) the challenges of efficiently exploiting extreme parallelism will likely lie not so much in the simulation phase - a larger or higher resolution input file may always be used - but rather in the creation and initialisation of such enormous input files, and the *physical* time scales one may reach (given that processor speed increases little). The focus here on strong scaling is precisely due to this practical need for parallelism to increase physical time evolution in the system (rather than merely allow physically larger or higher resolution systems) but a more intelligent performance model must

take into account the trade-off between the spatial and temporal scales, and the combinations allowed at different core counts.

To this end, such a performance model has been developed for lattice-Boltzmann simulations (soon to be published by [A. G. Hoekstra, B. Chopard, D. Coster, S. Portegies Zwart, P. V. Coveney, " Multiscale Computing for Science and Engineering in the Era of Exascale Performance", Phil Trans R Soc A, In Press (2018), DOI: 10.1098/rsta.2018.0144]) the results of which are shown in Fig. 6.

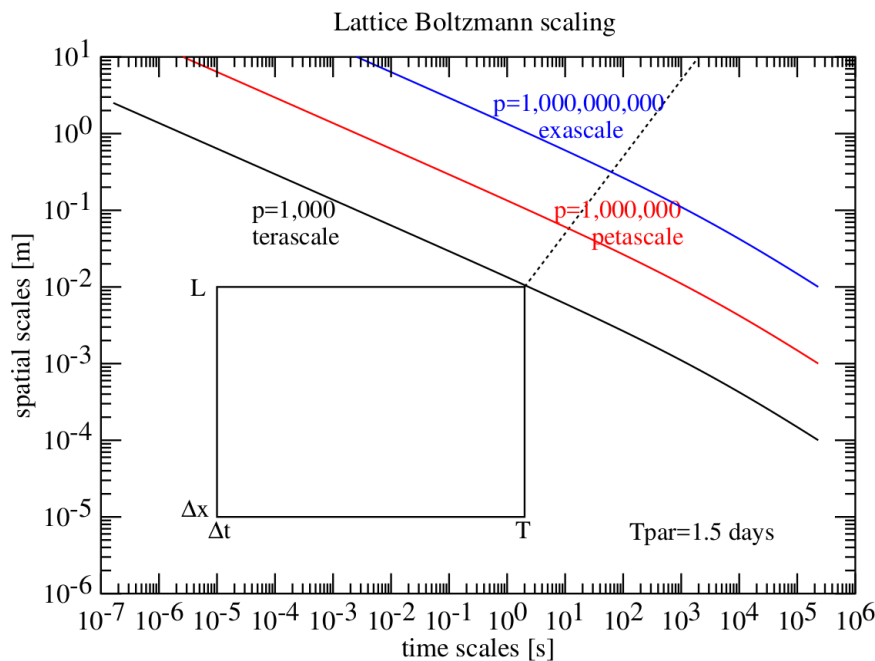


Fig. 6: Reachable spatial and temporal scales for a lattice-Boltzmann simulation at 10 micrometre resolution, given a fixed 1.5 days of calculation time, for cores ranging from 1000 (terascale) up to 1 billion (exascale).

In Fig. 6, we see the performance prediction using typical lattice-Boltzmann model parameters (in this case for Palabos, but equally applicable to HemeLB), showing the achievable time and spatial scales (at 10um resolution) achievable on 1k, 1M and 1G cores (the latter representing exascale).

The above has so far considered only the simulate phase of the lattice-Boltzmann application. However, as system sizes increase, the initialisation time (during which the load decomposition of the system occurs, and ranks read the relevant parts of the geometry into their memory) will also increase. The initialisation phase in the Blue Waters benchmark simulations varied (approximately) from 15 to 40 minutes, with more cores corresponding to longer initialisation.

Conclusion

The performance work on the Binding Affinity Calculator provides a good indication of how Replica Computing (RC) pattern applications are likely to scale under extreme parallelism.

This includes, to a large extent, the Heterogeneous Multiscale Computing (HMC) pattern, for which the greatest computational cost generally lies in the execution of replicas. However, as we have elaborated in deliverable D2.2, the dynamic nature of HMC, depending on the quality of the surrogate model to capture accurately enough the microscale dynamics, makes this less obvious than for the pure RC patterns. It might well be that exascale performance is required in phase 1 of HMC (the initial training of the surrogate) after which HMC applications could resort to reduced resources. This remains a topic of research. In the materials HMC application (UCL), for example, the similarity between microsimulations is determined in parallel by the primary model (which is mostly a Finite Element Solver), but the costs are dwarfed by those of running the many submodels (replicas). In the less common case of an HMC pattern with a very expensive macroscale (primary) model, the performance on exascale will likely more closely follow that of the Extreme Scaling pattern. It is our opinion that both RC and HMC type of applications are viable candidates for exascale computing, as we have demonstrated on several occasions in ComPat. However, we have only been able to ‘scratch the surface’, and more research and demonstrators are required to substantiate these conclusions.

In the case of applications following the Extreme Scaling (ES) pattern, the performance of the primary model will be of most interest at levels of extreme parallelism. The strong scaling performance studies of HemeLB were carried out to test this aspect. Prediction work carried out by Chopard *et al.* [4] indicated (for a general lattice-Boltzmann application) the expected attainability of physical and temporal scales with putative exascale systems (1 billion cores) of order 100 s for 10 cm (or 10 s and 100cm) after 1.5 days of wall clock time. Depending on the problem size, it may be more efficient to run at lower resolutions, while using several replicas, in order to derive uncertainties from the simulations. Again, we believe that ComPat has demonstrated that ES can scale to the exascale, if the primary model is capable of extreme scaling *and* if the auxiliary models that may serialize the execution, are deployed in an efficient way, maybe even by staging two independent ES runs. This was discussed and demonstrated in deliverable D2.2 and D3.2.

Impact of exascale resources on future scientific applications

The major conclusion of this work is the excellent performance of replica based calculations to extreme parallelism supercomputing. While this is directly applicable to RC or HMC applications in phase 1 of the performance cycle, the efficient use of ES applications may need more careful treatment (depending on the physical and temporal scales of the process of interest). One way will likely be to respond to the growing desire for uncertainty quantification (UQ) in computational science, leveraging the efficiency of replica computing on exascale systems by running several ES applications in the same allocation. This will have the benefit of rendering the simulation output “actionable”, in the sense that UQ will give users more faith in the applicability of the results and thus greater ability to make decisions thereon. Additionally, Replica Computing is typically more resistant to node failures.

We therefore expect that the actual impact of exascale resources on multiscale computing is likely to be to encourage the *inter alia* use of replica based computing patterns (RC or HMC), and quantifying uncertainties in larger simulations (such as the primary model of the ES pattern) which is not feasible with present day petascale facilities.