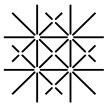


University of Basel

Investigation of Domain Decomposition and Scheduling in HPC Applications

Master project

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
High Performance Parallel And Distributed Computing
https://hpc.dmi.unibas.ch

Advisor: Prof. Dr. Florina M. Ciorba
Supervisor: Jonas Henrique Müller Korndörfer

Nderim Shatri
nderim.shatri@stud.unibas.ch
HS16-062-234

31.07.2022

Table of Contents

1 Introduction

In High Performance Computing, HPC, different parallel programming paradigms are being used in applications to work on a specific computation in parallel. Thus, enhancing the performance of the application. The applications in domains, such as Molecular Dynamics, Smoothed Particle Hydrodynamics, and many more, were released over the years with various usage of programming paradigms. The popular parallel programming paradigms tackle different aspects of parallelization. Those most common used among the community are OpenMP, MPI, OpenCL, OpenAcc, and CUDA. With this project, we aim to set an overview of open source HPC applications. We tried to capture the relevant information and characteristics of an application, consisting of the Domain, Programming Language, LoC, parallel programming paradigms, and releases. Furthermore, we analyzed the code to see how the data decomposition and scheduling on process level has been implemented. As the investigation is based on process level, a focus on the paradigms is the MPI library, namely the Message-Passing-Interface. MPI assures communication over each CPU (or core), so each CPU runs an independent program. The difficulty and the most challenging part is to deep dive into the community's code and search for their domain decomposition and scheduling on process level techniques, as there are infinitely many ways to formulate a code, discrepancy of Lines of Code, and various programming languages. Furthermore, among the applications there might be legacy code or another application which solves the same problem faster. To counter these difficulties a systematic approach consisting of automatized and manual processes is needed and provided. Within this project we created a data set providing our results to understand the common implementation practices within the community to be able to propose novel and new methods to enhance applications' performance.

2
Methods

Achieving the investigation of domain decomposition and scheduling in HPC applications, we were given 180 applications. Among these 180 applications, we successfully managed to analyze 71 of them during this project. The investigated applications have been randomly chosen and vary in benchmark suites, domains used, and used technologies. Briefly, the project has fulfilled two goals. First, gathering available information about the applications and secondly, investigating the domain decomposition and scheduling in HPC applications. Therefore, manual work which cannot be automatized had to be done.

2.1 Investigation of Open Source HPC applications

One of the fundamental question is: what is necessary in providing information that might help the community for the future use of our results? Naturally, it comes practical in building a data set which is visual and can be modified and adopted in the future. For this purpose, we build a table view using "google-sheets", as it is available, modifiable, and exportable in various formats. The data which is analyzed consist of 26 columns with different characteristics.

2.1.1 Benchmark suite, Name, Link, Domain and description

The columns Benchmark suite, Name, Link and Description provide information about the application analyzed. The Benchmark suite column shows the user the type of suite the application belongs to. It has been showed as the first column, as it offers a broad selection of scientific application to solve different domains. The name of the application and the link define what the application's name is and where it can be retrieved from the application. As for the link, mostly the repository of the application is being used. The reason is that updates of the application, bug fixes, and different branches can be explored. The domain was constructed to show the area of research of the application. Additionally, for a better understanding about the application, we added the description of the application. All those information have been gathered manually from the different links and sources provided by the publishers.

Suite(M)	Application Name(M)	Links(M)	Domain(M)	Description(M)

Figure 2.1: First part of the table containing the retrieved information

2.1.2 Lines of Code and Programming Langua

The Lines of Code is an indicator of how complex an application has been developed. This is also a measurement to represent the data graphically within this project. Thus, we introduced in our data set: C Lines, C++-Lines, Fortran Lines, C/C++ Header lines. Furthermore, we wanted to show the programming languages that the code has been written on and detect if there were multiple languages were used. Last but not least, we wanted to show the application's line distribution, consisting of each amount for the different programming languages used.

Programming Language (M)	Fortran Lines(A)	C Lines(A)	C++ Lines(A)	C_CPP_H_Lines(A)

Figure 2.2: Second part of the table containing the retrieved information

2.1.3 OpenMP, MPI, OpenACC, CUDA, OpenCL

Columns consisting the common parallel programming paradigms, such as OpenMP, MPI, OpenACC, CUDA, and OpenCL are necessary to determine the characteristics of an application. To make it human readable, we assured that the possible values are "yes" or "no" if one application shares, or does not share the paradigms.

MPI(M)	OpenACC(A/M)	CUDA(A/M)	OpenCL(A/M)

Figure 2.3: Third part of the table containing the retrieved information

2.1.4 Data Distribution and Location of Data Distribution

As one of the focus of this work is to identify the data decomposition on process level, we present those information as Data Distribution consisting of the type and Location of Data Distribution consisting of the locality in the code. The types of the Data Distribution can be:

- distributed: applications which have distributed data decomposition on process level
- replicated: applications which have replicated data decomposition on process level
- centralized: applications which have centralized data decomposition on process level
- unknown: applications where we could not found indices of one of the above types

2.1.5 Process Level Scheduling and Location of the Process Level Scheduling

Another aspect of this work has been to find what kind of and where process level scheduling occurs. Therefore, we captured our analysis inside the columns *ProcessLevelScheduling*

and $Location of the Process level Scheduling$. For simplicity, we distinguished only two types of the process level scheduling:

- dynamic: applications with dynamic process level scheduling
- static: applications with static process level scheduling
- unknown: applications where we were not able to find indices of one of the types above

Data Decomposition(M)	Location Data Decomposition(M)	Process level (M) Scheduling	Location Scheduling(M)

Figure 2.4: Fourth part of the table containing the retrieved information

2.1.6 Release Date, Version, new Version, and new Release date

Among the applications in the portfolio, there have been applications which have already newer versions. The release date column and version is meant to show the release date of the analyzed application. As another aspect, we updated the applications of the portfolio and marked if there are further releases and newer versions. Furtheremore, all the applications have been updated inside the portfolio where an update was found.

2.1.7 Storage

Within this column, we tried to get the application's storage that it takes. The storage was measured in Megabyte and consists of the application's whole directory.

2.1.8 Notes

The column Notes consists of remarks that have resulted during the analysis. Those notes may be, for instance, on difficulties to find the application's domain decomposition and scheduling or information about a frequent update of an application.

2.1.9 A/M, A, M

Within the headers of the column there are letters describing whether information was retrieved automatically by a script, denoted as A, manually by research and investigating, denoted as M, or whether it consisted manual and automatic investigation, denoted as A/M.

Storage (MB) (M)	Version(M)	Release Date (M) (github release dat	Newer Versions(M	New Release Date (M) (github)	Notes(M)

Figure 2.5: Remaining part of the table containing the retrieved information

2.2 Approach

To achieve the goal of the project, we used a systematic approach. The approach could be described by the following steps. The applications have been stored on the HPC-Cluster of the University of Basel. The applications have been chosen randomly.

1. Retrieve the application
2. Grep for the initialization of MPI with MPI_Init
3. Use of Visual Studio Code[60] **IDE**, Integrated Development Environment
4. Analyze Decomposition and Scheduling at process level
5. Collect the information about an application

2.2.1 Retrieve the application

As all the applications lie on the HPC cluster, we decided to firstly retrieve them on our local machines. Having applications on a local machine comes handy when we try to analyze the code, as the use of IDEs such as Visual Studio Code[60] is possible.

2.2.2 Grep for MPI_Init

Our main goal is to analyze the domain decomposition and scheduling at process level. Therefore, to grep for MPI_Init in each application is one of the first steps. By using the command:

```
grep -n "MPI_Init" "directory-name" # C or C++ application
grep -n "MPI_INIT" "driectory-name" # Fortran application
```

With this step, we assured that MPI was used in the application and therefore, a kind of process level scheduling and domain decomposition might have been used.

2.2.3 Use of IDE

An IDE is very useful when analyzing code. The searches of methods and code classes have various shortcuts. Therefore, the code can be investigated fast and efficiently by using short cuts. In our case, we used visual studio code, as extension of programming languages can be quickly installed. The user interface is fast forward and searches are handled quickly.

2.2.4 Domain Decomposition and Process Level Scheduling

Applications vary a lot in the code structure, programming languages and the way of code. The crucial step here, is to analyze the occurrence of the MPI library. Starting with the occurence of MPI_Init, we know the initialization point of the MPI execution environment. In most cases, immediately after the initialization of the MPI is the MPI_Comm_rank and MPI_Comm_size. MPI_Comm_rank takes as argument, the MPI_COMM_WORLD which contains all the MPI processes and the rank which determines the rank of the MPI process. In Figure 2.6, we can see the used variable for MPI_Comm_rank which is defined as

myrankid. By following the rank, we might end up at function which uses the rank and where we might encounter data decomposition and or scheduling at process level.

```
int main( int argc, char *argv[] )
{

int myrankid, nProcesses;
MPI_Init( &argc, &argv );
MPI_Comm_rank( MPI_COMM_WORLD, &myrankid );
MPI_Comm_size( MPI_COMM_WORLD, &nProcesses );
```

Figure 2.6: Search for MPI_Init

```
void exponential(int *flops, int len, int nProcesses, int myrankid)
{

double j;
int i;
double a=0;
int tilesize = len/nProcesses;

for(i = myrankid * tilesize; i < myrankid * tilesize + tilesize; i++) {
        // printf("My rank: %d, iter start: %d, iter end: %d\n", myrankid, myrankid * tilesize, myrankid * tilesize + tilesize);
        int iterations = flops[i]/resize;
        //COMPUTATION
        for( j = 0; j < iterations; j++){
            a += j*(iterations/2);
            //a += (j*a)/iterations;
            if (a < 0)
            {printf("impossible \n");}
        }
        //COMPUTATION

  }
}
```

Figure 2.7: Identifying Decomposition and Scheduling at process level

In Figure 2.7, we can see that the function *exponential*() contains *myrankid* and *nProcesses* as an input. In the following lines we detect that *tilesize* will always be a constant value. This is a strong indicator for static scheduling. In the following lines of code, we see that the for loop is statically divided and the computations are statically done in blocks ranging from $myrankid * tilesize$ up to $myrankid * tilesize + tilesize$. This manifests our belief for static scheduling. Furthermore, we can see that *tilesize* has a value which does not change. Therefore, the data decomposition is replicated.

2.2.5 Extract other data

As mentioned, there are multiple aspects of characteristics for an HPC application. Most information retrieved has been done manually. The Columns LoC, Programming Language, C Lines, C++ Lines, Fortran Lines, C_C++_Header_lines, OpenMP, CUDA, OpenACC, and OpenCL have been retrieved automatically with a script. The script was provided, therefore, execution of the script led to the information.

2.2.5.1 Retrieve the storage in MB

Getting the storage information, we decided to do this by the following Linux command:

```
du -r -sh [Application Name]
```

This command ensures that we get the information about the storage in Megabyte. We took Megabyte as a measurement, as the applications have varied by a huge size. While some applications had less then $1MB$ other had a size of almost $1GB$.

2.2.6 Challenges in Code Analysis

Investigating the code strongly depends on the length of code. In principle, we experienced that the longer the code the more complex an application was built up. Therefore, searching and investigating for data decomposition and process level scheduling resulted in a very high time effort. Understanding code of other developers depends on the comments made. In general, written code without comments is difficult to understand. Furthermore, there is no general naming convention on how to name variables. For instance, the rank of an MPI process can be called randomly. It matters if a rank is defined as *myrankid*, *rank*, or *id*. The latter comes not handy when searching deep into the code. Applications analyzed have different domains in which they are solving a scientific problem. Those domains ranges in different areas, such as MD, SPH, or Deep Learning. All those use computations on their domains which are computed and approximated by formulas. So, the specificity of those computations can lead to issues in understanding what the application is solving. Unfortunately, understanding code is subjective and the ability to understand HPC applications only grows with experience in this field.

3

Analyzed open source HPC Applications

3.1 No benchmark suite

1. **Adios**[31] (https://github.com/ornladios/ADIOS/releases): ADIOS is developed as part of the United States Department of Energy's Exascale Computing Project. It is a framework for scientific data I/O to publish and subscribe to data when and where required.

2. **ExaMiniMD**[58] (https://github.com/ECP-copa/ExaMiniMD/releases/tag/1.0): ExaMiniMD is a proxy application and research vehicle for particle codes, in particular Molecular Dynamics (MD). Compared to previous MD proxy apps (MiniMD, COMD), its design is significantly more modular in order to allow independent investigation of different aspects.

3. **MACSio**[33] (https://github.com/LLNL/MACSio): MACSio is being developed to fill a long existing void in co-design proxy applications that allow for I/O performance testing and evaluation of tradeoffs in data models, I/O library interfaces and parallel I/O paradigms for multi-physics, HPC applications.

4. **mcb**[27] (https://computing.llnl.gov/projects/co-design/mcb): The Monte Carlo Benchmark (MCB) is intended for use in exploring the computational performance of Monte Carlo algorithms on parallel architectures.

5. **OpenMD**[39] (https://github.com/OpenMD/OpenMD): OpenMD is an open source molecular dynamics engine which is capable of efficiently simulating liquids, proteins, nanoparticles, interfaces, and other complex systems using atom types with orientational degrees of freedom (e.g. "sticky" atoms, point dipoles, and coarse-grained assemblies).

6. **SAMRAI**[45] (https://github.com/LLNL/SAMRAI): SAMRAI (Structured Adaptive Mesh Refinement Application Infrastructure) is an object-oriented C++ software library that enables exploration of numerical, algorithmic, parallel computing, and software issues associated with applying structured adaptive mesh refinement (SAMR) technology in large-scale parallel application development. SAMRAI provides software

tools for developing SAMR applications that involve coupled physics models, sophisticated numerical solution methods, and which require high-performance parallel computing hardware. SAMRAI enables integration of SAMR technology into existing codes and simplifies the exploration of SAMR methods in new application domains.

7. **Siesta**[35] (https://gitlab.com/siesta-project/siesta/-/releases): SIESTA is both a method and its computer program implementation, to perform efficient electronic structure calculations and ab initio molecular dynamics simulations of molecules and solids. SIESTA's efficiency stems from the use of strictly localized basis sets and from the implementation of linear-scaling algorithms which can be applied to suitable systems.

8. **SimpleMOC**[36] (https://github.com/ANL-CESAR/SimpleMOC/tree/v4): The purpose of this mini-app is to demonstrate the performance characterterics and viability of the Method of Characteristics (MOC) for 3D neutron transport calculations in the context of full scale light water reactor simulation.

9. **souffle**[42] (https://github.com/souffle-lang/souffle/releases): Souffle is a logic programming language inspired by Datalog. It overcomes some of the limitations in classical Datalog. For example, programmers are not restricted to finite domains, and the usage of functors (intrinsic, user-defined, records/constructors, etc.) is permitted. Soufflé has a component model so that large logic projects can be expressed.

10. **sphynx**[43] (https://astro.physik.unibas.ch/en/people/ruben-cabezon/sphynx/): SPHYNX is an SPH hydrocode with its focus on Astrophysical applications. SPHYNX includes state-of-the-art methods that allow it to address subsonic hydrodynamical instabilities and strong shocks, which are ubiquitous in astrophysical scenarios. SPHYNX, is of Newtonian type and grounded on the Euler-Lagrange formulation of the smoothed-particle hydrodynamics technique.

11. **splatt**[37] (https://github.com/ShadenSmith/splatt): SPLATT is a library and C API for sparse tensor factorization. SPLATT supports shared-memory parallelism with OpenMP and distributed-memory parallelism with MPI.

12. **sw4lite-RAJA**[44] (https://github.com/geodynamics/sw4lite/tree/RAJA-v1.0): sw4lite is a bare bone version of SW4 (https://github.com/geodynamics/sw4) intended for testing performance optimizations in a few important numerical kernels of SW4.

13. **thornado_mini**[38] (https://github.com/ECP-Astro/thornado_mini): Thornado_mini solves the equation of radiative transfer in the multi-group two-moment approximation. The Discontinuous Galekin (DG) method is used for spatial discretization, and an implicit-explicit (IMEX) method is used to integrate the moment equations in time. The hyperbolic (streaming) part is treated explicitly, while the collision term is treated implicitly.

14. **Trillinos**[46] (https://github.com/trilinos/Trilinos): The Trilinos Project is an effort to develop algorithms and enabling technologies within an object-oriented software

framework for the solution of large-scale, complex multi-physics engineering and scientific problems. A unique design feature of Trilinos is its focus on packages.

15. **tycho2**[50] (https://github.com/lanl/tycho2): A mini-app for neutral-particle, discrete-ordinates (SN), transport on parallel-decomposed meshes of tetrahedra.

16. **vlasiator**[51] (https://github.com/fmihpc/vlasiator/releases/tag/v5.1): In Vlasiator, ions are represented as velocity distribution functions, while electrons are magneto-hydrodynamic fluid, enabling a self-consistent global plasma simulation that can describe multi-temperature plasmas to resolve non-MHD processes that currently cannot be self-consistently described by the existing global space weather simulations. The novelty is that by modelling ions as velocity distribution functions the outcome will be numerically noiseless.

17. **vmd**[48] (https://www.ks.uiuc.edu/Research/vmd/): VMD is a molecular visualization program for displaying, animating, and analyzing large biomolecular systems using 3-D graphics and built-in scripting.

18. **WRF**[47] (https://github.com/wrf-model/WRF/releases/tag/v4.2.1): WRF is a state-of-the-art atmospheric modeling system designed for both meteorological research and numerical weather prediction. It offers a host of options for atmospheric processes and can run on a variety of computing platforms.

19. **yambo**[49] (https://github.com/yambo-code/yambo): YAMBO implements Many-Body Perturbation Theory (MBPT) methods (such as GW and BSE) and Time-Dependent Density Functional Theory (TDDFT), which allows for accurate prediction of fundamental properties as band gaps of semiconductors, band alignments, defect quasi-particle energies, optics and out-of-equilibrium properties of materials.

20. **arbor-0.3**[54] (https://github.com/arbor-sim/arbor): Arbor is a high-performance library for computational neuroscience simulations with multi-compartment, morphologically-detailed cells, from single cell models to very large networks. Arbor is written from the ground up with many-cpu and gpu architectures in mind, to help neuroscientists effectively use contemporary and future HPC systems to meet their simulation needs.

21. **Caffe-MPI**[32] (https://github.com/Caffe-MPI/Caffe-MPI.github.io): The Caffe-MPI is designed for high density GPU clusters; The new version supports InfiniBand (IB) high speed network connection and shared storage system that can be equipped by distributed file system, like NFS and GlusterFS. The training dataset is read in parallel for each MPI process. The hierarchical communication mechanisms were developed to minimize the bandwidth requirements between computing nodes.

22. **CFDEMcoupling**[29] (https://github.com/CFDEMproject/CFDEMcoupling-PUBLIC): CFDEM® coupling provides an open source parallel coupled CFD-DEM framework combining the strengths of LIGGGHTS® DEM code and the Open Source CFD package OpenFOAM®(*). The CFDEM®coupling toolbox allows to expand standard CFD solvers of OpenFOAM®(*) to include a coupling to the DEM code LIGGGHTS®.

23. **Elemental**[30] (https://github.com/elemental/Elemental/releases/tag/v0.87.7): Elemental is a modern C++ library for distributed-memory dense and sparse-direct linear algebra, conic optimization, and lattice reduction. The library was initially released in Elemental: A new framework for distributed memory dense linear algebra and absorbed, then greatly expanded upon, the functionality from the sparse-direct solver Clique, which was originally released during a project on Parallel Sweeping Preconditioners.

24. **Gadget**[25] (https://wwwmpa.mpa-garching.mpg.de/gadget/): GADGET-4 is a massively parallel code for N-body/hydrodynamical cosmological simulations. It is a flexible code that can be applied to a variety of different types of simulations, offering a number of sophisticated simulation algorithms.

25. **hemelb**[55] (https://github.com/hemelb-codes/hemelb): HemeLB uses the lattice Boltzmann method to simulate fluid flow in complex geometries, such as a blood vessel network.

26. **horovod**[56] (https://github.com/horovod/horovod/releases): Horovod is a distributed deep learning training framework for TensorFlow, Keras, PyTorch, and Apache MXNet. The goal of Horovod is to make distributed deep learning fast and easy to use.

27. **meshkit**[40] (https://bitbucket.org/fathomteam/meshkit.git/src): MeshKit is an open-source library of mesh generation functionality. MeshKit has general mesh manipulation and generation functions such as Copy, Move, Rotate and Extrude mesh. In addition, new quad mesh and embedded boundary Cartesian mesh algorithm (EBMesh) are developed to be used. Interfaces to several public-domain tetrahedral meshing algorithms (Gmsh, netgen) are also offered.

28. **metag_partitioning**[28] (https://github.com/ParBLiSS/metag_partitioning): Parallel metagenomic assembler designed to handle very large datasets. Program identifies the disconnected subgraphs in the de Bruijn graph, partitions the input dataset and runs a popular assember Velvet independently on the partitions. This software is a high performance version of the khmer library for assembly.

29. **MITgcm**[52] (https://github.com/MITgcm/MITgcm): it can be used to study both atmospheric and oceanic phenomena; one hydrodynamical kernel is used to drive forward both atmospheric and oceanic models it has a non-hydrostatic capability and so can be used to study both small-scale and large scale processes.

30. **MLSL-IntelMLSL**[34] (https://github.com/intel/MLSL): Intel(R) Machine Learning Scaling Library (Intel(R) MLSL) is a library providing an efficient implementation of communication patterns used in deep learning.

31. **mxx**[57] (https://github.com/patflick/mxx): mxx is a C++/C++11 template library for MPI. The main goal of this library is to provide two things: First, simplified, efficient, and type-safe C++11 bindings to common MPI operations. Second, a collection of scalable, high-performance standard algorithms for parallel distributed memory architectures, such as sorting.

32. **Nek5000**[53] (https://github.com/Nek5000/nekRS/releases/tag/v21.1): High-order methods have the potential to overcome the current limitations of standard CFD solvers.It features state-of-the-art, scalable algorithms that are fast and efficient on platforms ranging from laptops to the world's fastest computers. Applications span a wide range of fields, including fluid flow, thermal convection, combustion and magnetohydrodynamics.

33. **phyml**[41] (https://github.com/stephaneguindon/phyml): PhyML is a software package that uses modern statistical approaches to analyse alignments of nucleotide or amino acid sequences in a phylogenetic framework. The main tool in this package builds phylogenies under the maximum likelihood criterion. It implements a large number of substitution models coupled to efficient options to search the space of phylogenetic tree topologies.

34. **PrincetonCBEMDMPI**[26] (https://github.com/PrincetonUniversity/PrincetonCBEMDMPI): CBEMD: Parallel Molecular Dynamics Under Various Thermodynamic Ensembles.

3.2 LLNL ASC Proxy Apps

1. **Lulesh**[24] (https://github.com/LLNL/LULESH): LULESH is a highly simplified application, hard-coded to only solve a simple Sedov blast problem with analytic answers – but represents the numerical algorithms, data motion, and programming style typical in scientific C or C++ based applications.

3.3 Mantevo

1. **miniAMR**[72] (https://github.com/Mantevo/miniAMR): miniAMR applies a stencil calculation on a unit cube computational domain, which is divided into blocks. The blocks all have the same number of cells in each direction and communicate ghost values with neighboring blocks.

2. **miniMD**[71] (https://github.com/Mantevo/miniMD): miniMD is a parallel molecular dynamics (MD) simulation package written in C++ and intended for use on parallel supercomputers and new architechtures for testing purposes. The software package is meant to be simple, lightweight, and easily adaptable to new hardware.

3. **miniFE**[69] (https://github.com/Mantevo/miniFE/releases): MiniFE is an proxy application for unstructured implicit finite element codes. It is a similar to HPCCG and pHPCCG but provides a much more complete vertical covering of the steps in this class of applications.

4. **miniSMAC**[70] (https://github.com/Mantevo/miniSMAC/releases): Solves the finite-differenced 2D incompressible Navier-Stokes equations with Spalart-Allmaras one-equation turbulence model on a structured body conforming grid. The grid is partitioned into subgrids load balanced for the number of MPI ranks requested by the user

5. **miniTri**[67] (https://github.com/Mantevo/miniTri): miniTri is a proxy for a class of triangle based data analytics (Mantevo). This simple code is a self-contained piece of C++ software that uses triangle enumeration with a calculation of specific vertex and edge properties.

6. **miniAero**[66] (https://github.com/Mantevo/miniAero/releases): MiniAero is a mini-application for the evaulation of programming models and hardware for next generation platforms. MiniAero is an explicit (using RK4) unstructured finite volume code that solves the compressible Navier-Stokes equations.

7. **miniXyce**[68] (https://github.com/Mantevo/miniXyce): At this time, miniXyce is a simple linear circuit simulator with a basic parser that performs transient analysis on any circuit with resistors (R), inductors (L), capacitors (C), and voltage/current sources. The parser incorporated into this version of miniXyce is a single pass parser, where the netlist is expected to be flat (no hierarchy via subcircuits is allowed). Simulating the system of DAEs generates a nonsymmetric linear problem, which is solved using un-preconditioned GMRES. The time integration method used in miniXyce is backward Euler with a constant time-step. The simulator outputs all the solution variables at each time step in a 'prn' file.

3.4 ExaGraph

1. **miniVite**[64] (https://github.com/Exa-Graph/miniVite): miniVite is a proxy app that implements a single phase of Louvain.

3.5 Fiber Mini App

1. **ntchemini**[23] (https://github.com/fiber-miniapp/ntchem-mini): NTChem is a high-performance software package for the molecular electronic structure calculation for general purpose on the K computer.

3.6 SPEC CPU 2017

1. **104.milc**[2] (http://www.spec.org/auto/mpi2007/Docs/104.milc.html): The MILC Code is a set of codes written in C developed by the MIMD Lattice Computation (MILC) collaboration for doing simulations of four dimensional SU(3) lattice gauge theory on MIMD parallel machines. The code is used for millions of node hours at DOE and NSF supercomputer centers.

2. **107.leslie3d**[3] (http://www.spec.org/auto/mpi2007/Docs/107.leslie3d.html): 107.leslie3d is derived from LESlie3d (Large-Eddy Simulations with Linear-Eddy Model in 3D), a research-level Computational Fluid Dynamics (CFD) code. It is the primary solver used to investigate a wide array of turbulence phenomena such as mixing, combustion, acoustics and general fluid mechanics.

3. **122.tachyon**[4] (http://www.spec.org/auto/mpi2007/Docs/122.tachyon.html): Tachyon is a parallel ray tracing application.

4. **126.lammps**[1] (http://www.spec.org/auto/mpi2007/Docs/126.lammps.html): LAMMPS is a classical molecular dynamics simulation code designed to run efficiently on parallel computers. It was developed at Sandia National Laboratories, a US Department of Energy facility, with funding from the DOE. It is an open-source code, distributed freely under the terms of the GNU Public License (GPL).

5. **129.tera_tf**[5] (http://www.spec.org/auto/mpi2007/Docs/129.tera_tf.html): 3D eulerian hydrodynamics application 2nd godunov-type scheme, 3rd order remapping requires only a Fortran 90 compiler, and an MPI (1.2) implementation uses mostly point-to-point messages, and some reductions use non-blocking messages.

6. **130.socorro**[6] (http://www.spec.org/auto/mpi2007/Docs/130.socorro.html): Socorro is a modular, object oriented code for performing self-consistent electronic-structure calculations utilizing the Kohn-Sham formulation of density-functional theory.

7. **132.zeusmp2**[9] (http://www.spec.org/auto/mpi2007/Docs/132.zeusmp2.html): ZEUS-MP is a computational fluid dynamics code developed at the Laboratory for Computational Astrophysics (NCSA, SDSC, University of Illinois at Urbana-Champaign, UC San Diego) for the simulation of astrophysical phenomena.

8. **137.lu**[10] (http://www.spec.org/auto/mpi2007/Docs/137.lu.html): The 137.lu code has a rich ancestry in benchmarking. Its immediate predecessor is the LU benchmark in NPB3.2-MPI, part of the NAS Parallel Benchmark suite. It is sometimes referred to as APPLU (a version of that was 173.applu in CPU2000) or NAS-LU. Solution of five coupled nonlinear PDE's, on a 3-dimensional logically structured grid, using an implicit pseudo-time marching scheme, based on two-factor approximate factorization of the sparse Jacobian matrix.

9. **142.dmilc**[7] (http://www.spec.org/auto/mpi2007/Docs/142.dmilc.html): The MILC Code is a set of codes written in C developed by the MIMD Lattice Computation (MILC) collaboration for doing simulations of four dimensional SU(3) lattice gauge theory on MIMD parallel machines.

10. **143.dleslie**[8] (http://www.spec.org/auto/mpi2007/Docs/143.dleslie.html): 143.dleslie is derived from LESlie3d (Large-Eddy Simulations with Linear-Eddy Model in 3D), a research-level Computational Fluid Dynamics (CFD) code.

11. **145.lGemsFDTD**[11] (http://www.spec.org/auto/mpi2007/Docs/145.lGemsFDTD.html): GemsFDTD solves the Maxwell equations in 3D in the time domain using the finite-difference time-domain (FDTD) method. GemsFDTD is a Computational Electromagnetic(CEM) application.

3.7 Chatterbug

1. **chatterbug**[59] (https://github.com/hpcgroup/chatterbug): A suite of communication-intensive proxy applications that mimic commonly found communication patterns in HPC codes. These codes can be used as synthetic codes for benchmarking, or for trace generation using OTF2.

3.8 CORAL-2

1. **Kripke**[62] (https://github.com/LLNL/Kripke): Kripke is a simple, scalable, 3D Sn deterministic particle transport code. Its primary purpose is to research how data layout, programming paradigms and architectures effect the implementation and performance of Sn transport. Kripke is also a Proxy-App since it is a proxy for the LLNL transport code ARDRA.

2. **Laghos**[63] (https://github.com/CEED/Laghos): Laghos (LAGrangian High-Order Solver) is a miniapp that solves the time-dependent Euler equations of compressible gas dynamics in a moving Lagrangian frame using unstructured high-order finite element spatial discretization and explicit high-order time-stepping.

3.9 Rodinia 3.1

1. **SRAD**[21] (https://github.com/JuliaParallel/rodinia/tree/master/openmp/srad_v1): SRAD (Speckle Reducing Anisotropic Diffusion) is a diffusion method for ultrasonic and radar imaging applications based on partial differential equations (PDEs). It is used to remove locally correlated noise, known as speckles, without destroying important image features. SRAD consists of several pieces of work: image extraction, continuous iterations over the image (preparation, reduction, statistics, computation 1 and computation 2) and image compression. The sequential dependency between all of these stages requires synchronization after each stage (because each stage operates on the entire image).

2. **Streamcluster**[22] (https://github.com/JuliaParallel/rodinia/tree/master/openmp/streamcluster): It assigns each point of a stream to its nearest center Medium-sized working sets of user-determined size.

3.10 UK Mini-App Consortium

1. **TeaLeaf**[61] (https://github.com/UK-MAC/TeaLeaf_ref): TeaLeaf is a mini-app that solves the linear heat conduction equation on a spatially decomposed regularly grid using a 5 point stencil with implicit solvers. TeaLeaf currently solves the equations in two dimensions, but three dimensional support is in beta.

3.11 SPEC CPU 2017

1. **500.perlbench_r**[12] (https://www.spec.org/cpu2017/Docs/benchmarks/500.perlbench_r.html): is a cut-down version of Perl v5.22.1, the popular scripting language. SPEC's version of Perl has had most of OS-specific features removed.

2. **508.namd_r**[13] (https://www.spec.org/cpu2017/Docs/benchmarks/508.namd_r.html): The 508.namd_r benchmark is derived from the data layout and inner loop of NAMD, a parallel program for the simulation of large biomolecular systems. Although NAMD scales to over 200,000 cores for very large systems, serial performance is equally important to the over 50,000 users who have downloaded the program over the past decade. Almost all of the runtime is spent calculating inter-atomic interactions in a small set of functions. This set was separated from the bulk of the code to form a compact benchmark for CPU2017.

3. **510.parest_r**[14] (https://www.spec.org/cpu2017/Docs/benchmarks/510.parest_r.html): solves a problem from biomedical imaging. Specifically, the underlying problem is the reconstruction of interior properties of a 3d body from multiple observations at its two-dimensional surface, in much the same way as multiple 2d X-ray images are combined to do 3d CT (computed tomography) scans. The difference to CT scans is that the method this program describes is infrared light that does not go through tissues in a straight line, but diffuses.

4. **511.povray_r**[15] (https://www.spec.org/cpu2017/Docs/benchmarks/511.povray_r.html): POV-Ray is a free and open source ray-tracing application. The CPU 2017 version is based on POV-Ray version 3.7.

5. **520.omnetpp_r**[16] (https://www.spec.org/cpu2017/Docs/benchmarks/520.omnetpp_r.html): The benchmark performs discrete event simulation of a large 10 gigabit Ethernet network. The simulation is based on the OMNeT++ discrete event simulation system ([1]www.omnetpp.org), a generic and open simulation framework. OMNeT++'s primary application area is the simulation of communication networks, but its generic and flexible architecture allows for its use in other areas such as the simulation of IT systems, queueing networks, hardware architectures or business processes as well.

6. **523.xalancbmk_r**[17] (https://www.spec.org/cpu2017/Docs/benchmarks/523.xalancbmk_r.html): XSLT processor for transforming XML documents into HTML, text, or other XML document types.

7. **531.deepsjeng_r**[18] (https://www.spec.org/cpu2017/Docs/benchmarks/531.deepsjeng_r.html): 531.deepsjeng_r is based on Deep Sjeng WC2008, the 2008 World Computer Speed-Chess Champion. Deep Sjeng is a rewrite of the older Sjeng-Free program, focused on obtaining the highest possible playing strength. (alpha-beta tree search & pattern recognition).

8. **541.leela_r**[19] (https://www.spec.org/cpu2017/Docs/benchmarks/541.leela_r.html): 541.leela_r is a Go playing engine featuring Monte Carlo based position estimation,

selective tree search based on Upper Confidence Bounds, and move valuation based on Elo ratings. (Monte Carlo simulation, game tree search & pattern recognition).

9. **548.exchange2_r**[20] (https://www.spec.org/cpu2017/Docs/benchmarks/548.exchange2_r.html): This program was written for development of non-trivial 9x9 sudoku puzzles. It has been used extensively in informal competitions, which run for days. Incidentally, it tests many Fortran 95 array handling features (including some intrinsic functions) for use with integer arrays. Unusually, it relies heavily on recursion (up to eight levels deep) but, in contrast to most Fortran programs, uses no floating-point arithmetic.

3.12 ExMatEx

1. **ASPA-master**[65] (https://github.com/exmatex/ASPA/tree/master/doc): The purpose of ASPA (Adaptive Sampling Proxy Application) is to enable the evaluation of a technique known as adaptive sampling on advanced computer architectures. Adaptive sampling is of interest in simulations involving multiple physical scales, wherein models of individual scales are combined using some form of scale bridging.

4

Data Processing

4.1 Data preparation

Gathering the information of all the applications, resulted to a broad table. The table needed to be pre-processed where cleaning and preparation for data input was done. The cleaning consisted of removing redundant data where no information was found. Furthermore, consistency was needed in every column. For instance, the column $Domain$ for applications that solve partial differential equations, the entries have been varying between PDE and $PartialDifferentialequations$. Thus, ensuring consistency by writing PDE for the applications' domain led to consistency. As there is not a large data set, the pre-processing was done manually.

4.2 Data-input

Having broad characteristics of applications, we needed to distinguish which characteristics we want to show. This resulted in removing columns which we cannot show, have not enough information about, or are not usable for the sake of our analysis. So, we decided to remove the columns: $Links$, $Description(M)$, $LocationDataDecomposition(M)$, $LocationScheduling(M)$, $Version(M)$, $NewerVersions(M)$, $NewReleaseDate(M)(github)$, and $Notes(M)$. This procedure removed eight columns for the analysis and hence, we created the new table $ProcessedControlTable$. In the $ProcessedControlTable$ we decided to rename columns because of simplicity for plotting the results. Furthermore, we added the column $ReleaseYear$ which indicates only the release year of the application. Due to this modification we have been able to show the applications' characteristics over year. Lastly, we removed applications which did not have the use of MPI, such that 60 applications remained.

5 Results

Investigation of domain decomposition and scheduling in HPC applictions included the retrieval of various characteristics of an application. Having a table consisting of 26 characteristics per application, yield to a broad range of information. While the focus lied mainly in domain decomposition and scheduling at process level, we still managed to display various results for the analyzed application.

5.1 Generic information

Providing generic information of the data, one can see in Figure 5.1 that most analyzed applications are moderate complex with a lines of code ranging between 10′000 - 100′000. This means that 28 applications lie between that range.
In figure 5.2 we can see that among the applications analyzed more than a half of them were using only MPI for performance enhancement. The second biggest part consisting of a quarter of the analyzed applications is the combination between OpenMP and MPI.
Not surprisingly, most of the applications have been using a distributed data decomposition, as shown in Figure 5.3. The fraction with *unknown* shows applications that we analyzed but could not be defined what type of decomposition has been used.
In Figure 5.4, we can observe that scheduling at process level is in most cases static. The reason might be that using static scheduling is simpler to code.

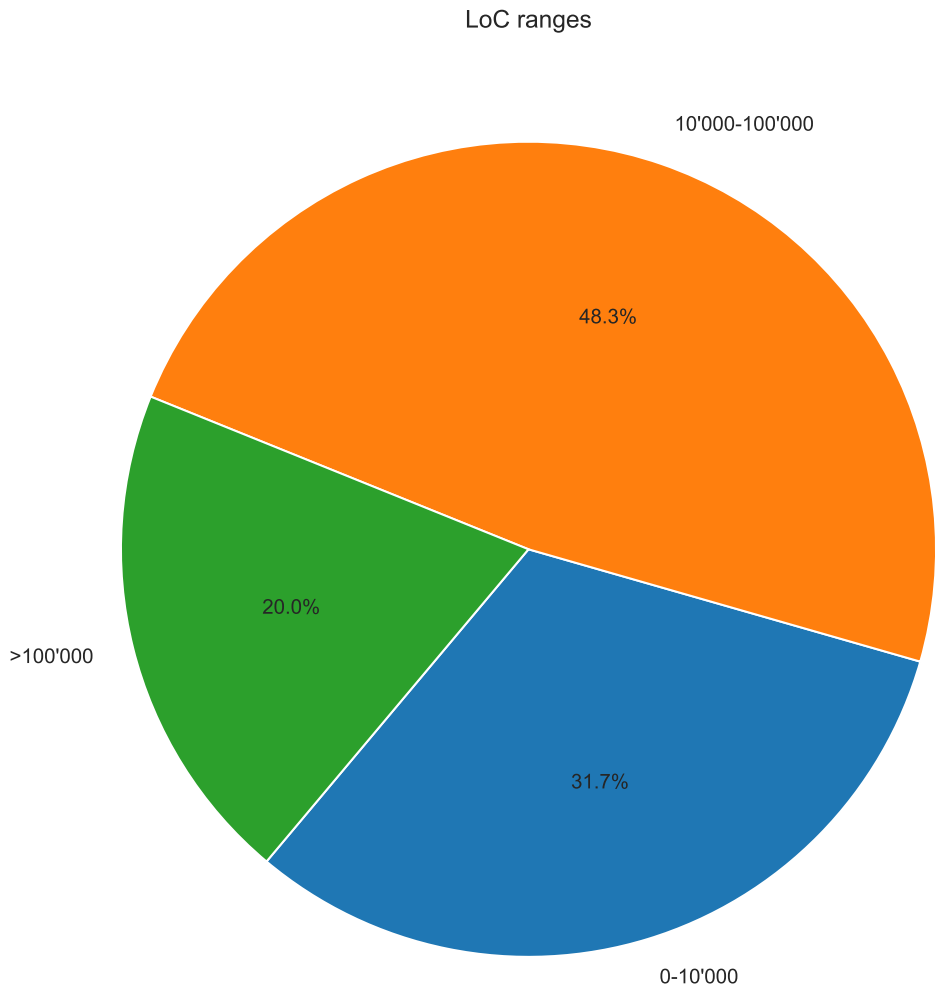


Figure 5.1: LoC distributed among the applications

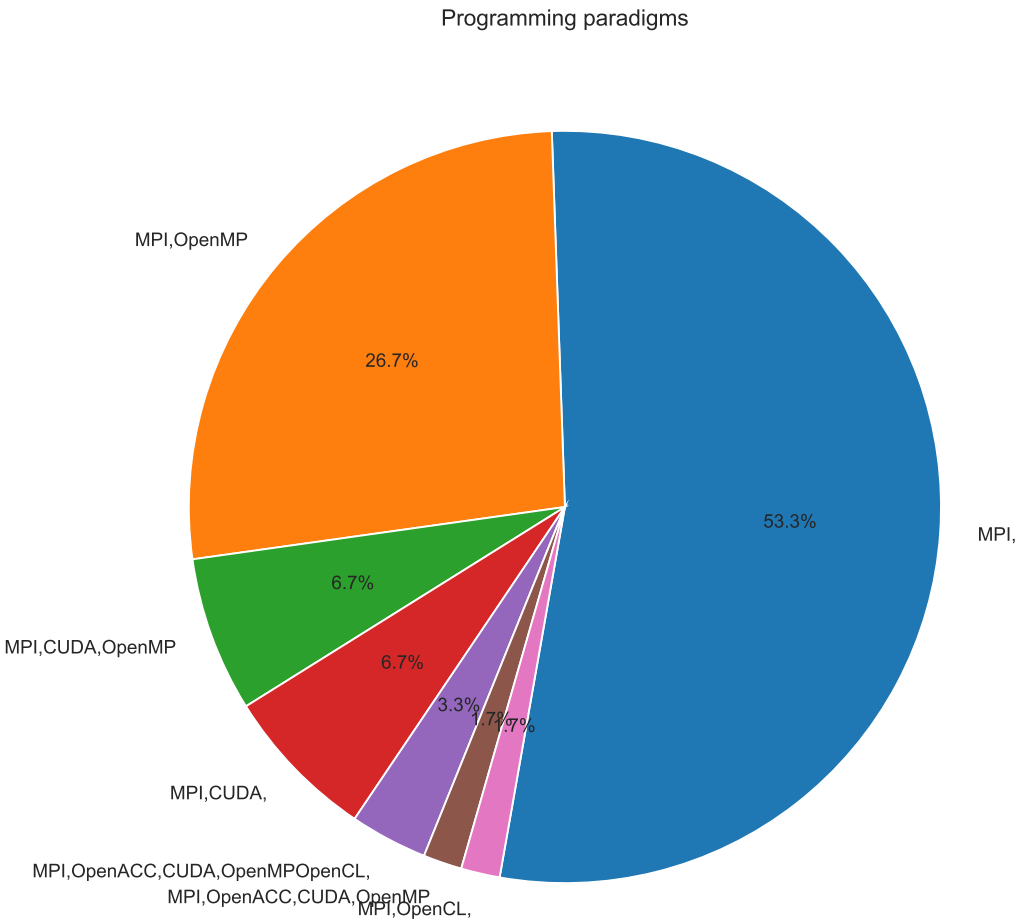


Figure 5.2: Used parallel programming paradigms

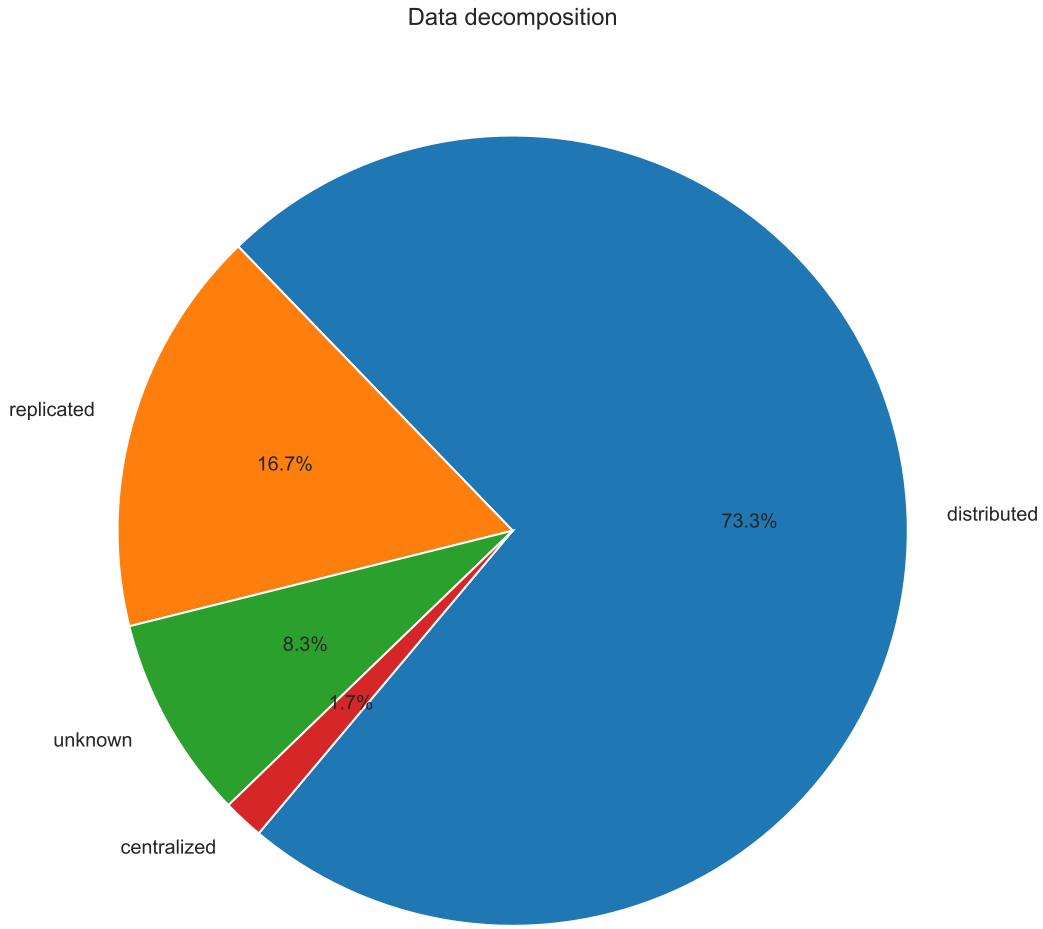


Figure 5.3: Data Decomposition at process level

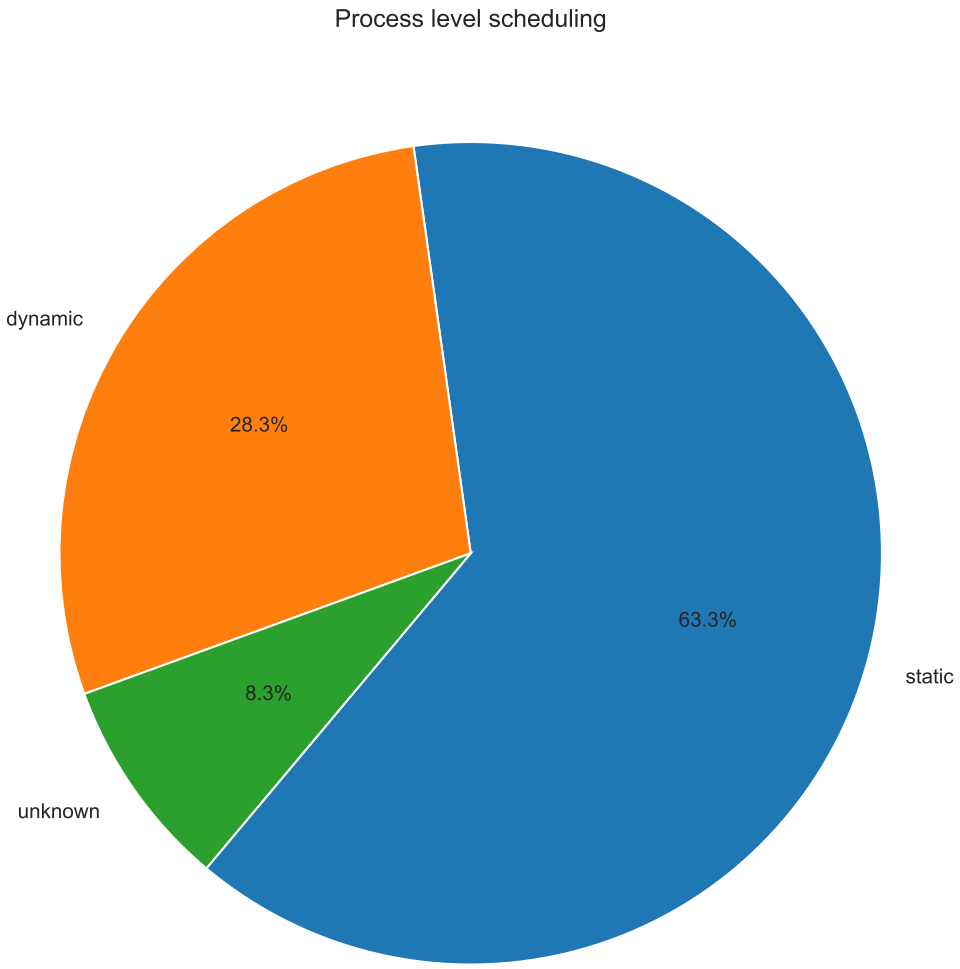


Figure 5.4: Process level scheduling

5.2 Information about the Length of Code and Storage

Generally, we can see in Figure 5.5 that with growing storage, more lines of code are written. As the storage of an application varies depends on other files, such as .pdfs, we use for the following results the line of code, denoted as LoC. In the following in figure 5.6, we can see

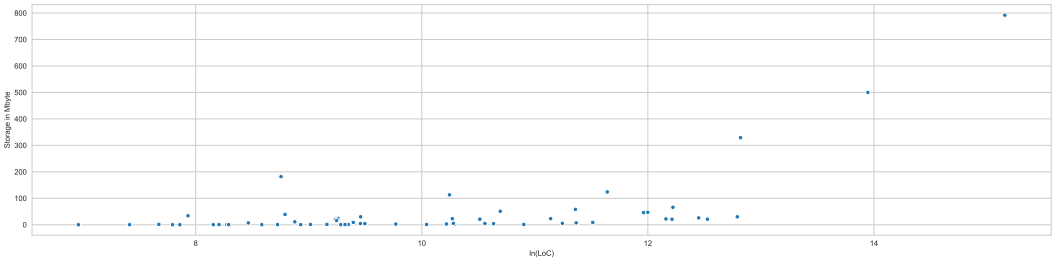


Figure 5.5: Storage in relation with the natural logarithm of lines of code

the distribution of the different programming languages used per application. Noticeably, there are applications which consist of a lot of header lines used in header files, C-files, and C++-files. This might indicate also that there are are applications which might have imported common libraries used within the community.

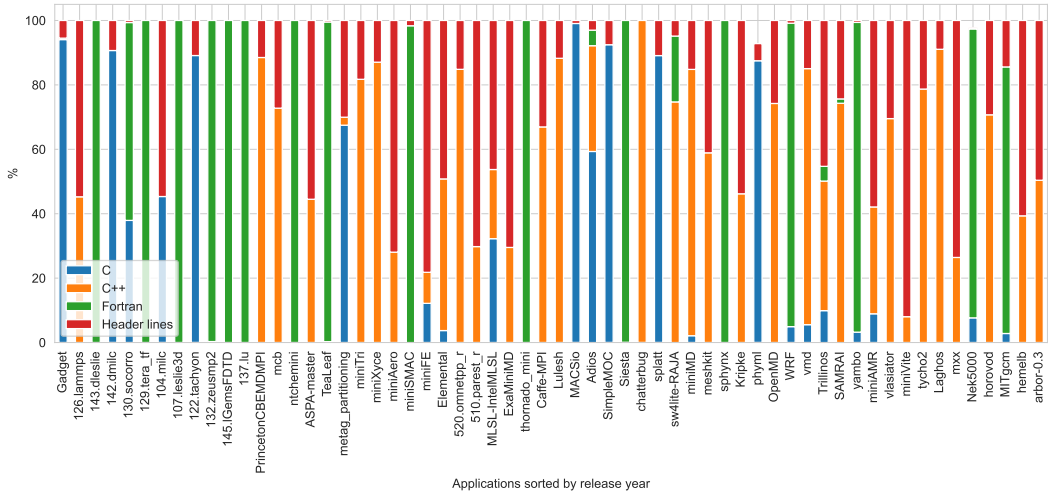


Figure 5.6: Code relation between the applications

5.3 Parallel programming paradigm, decomposition and scheduling at process level

In figure 5.7, we display the resulting table. This provides us an overview among the used paradigms, decomposition and scheduling at process level per application. We distinguished the used parallel programming paradigms, the decomposition types, and the scheduling types. By looking at the figure, we have chosen to show only applications that use MPI. Interestingly, there are only two applications that are using all considered parallel programming paradigms. Those applications are *vmd*[48] and *vlasiator*[51]. By providing this plot,

we think that the community may introduce another parallel programming paradigm to enhance the application's performance.

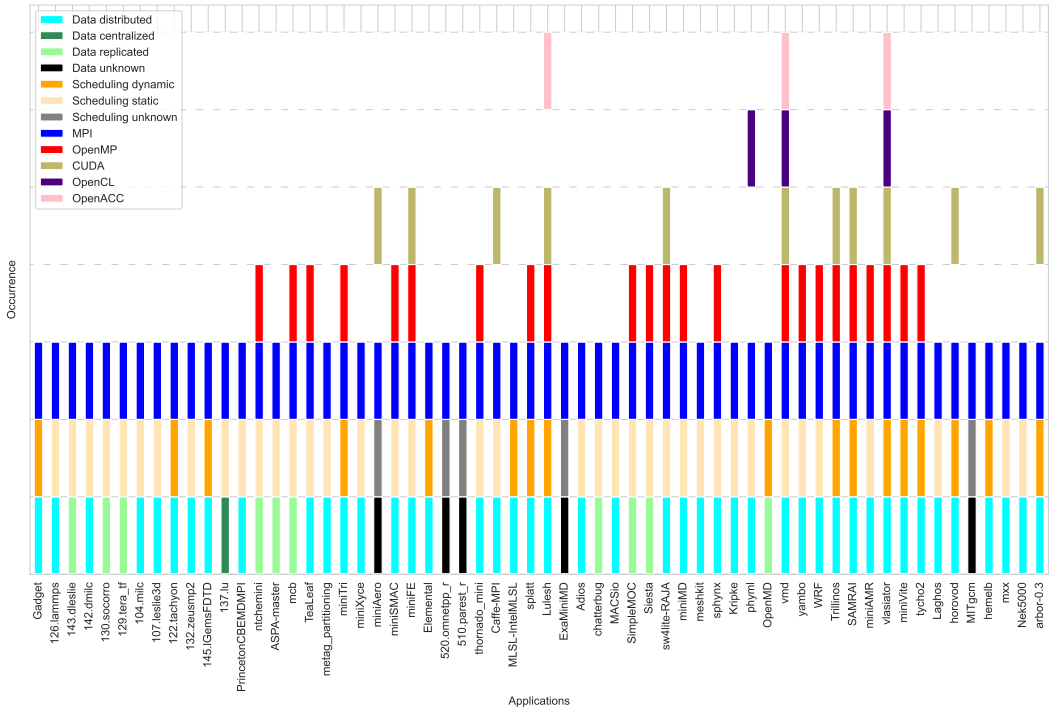
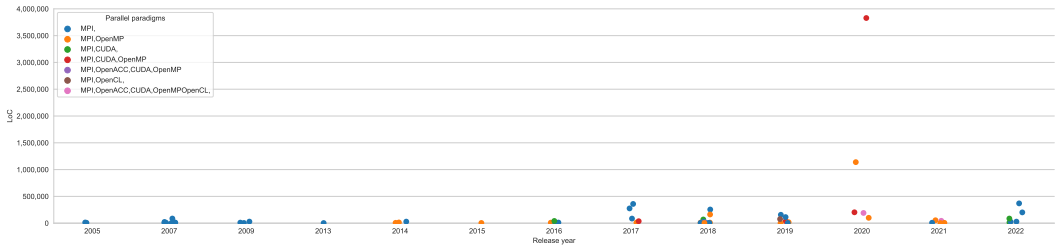


Figure 5.7: Occurrences of paradigm, decomposition and scheduling at process level

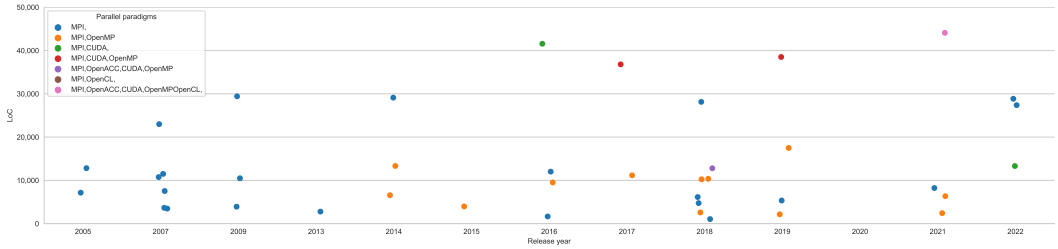
5.4 Paradigms and Programming Language over years

The analyzed applications were all released between 2005 and 2021. Thus, we have been interested in: How complex did the community implement the application within this period with its used paradigms, respectively the code?. In Fig. 5.8, we can see the parallel programming paradigms used over years in relation with the LoC. Within Fig. 5.8a, we see that over the years the trend was to develop applications with a higher LoC. Followed by a zoom in Fig. 5.8b, we look at applications with LoC <50′000 to investigate the used paradigms. In Fig. 5.8c, we look at application lying having a LoC between 50′000 and 400′000. Interestingly, applications with their solely usage of MPI, seem to be present within all ranges of LoC.

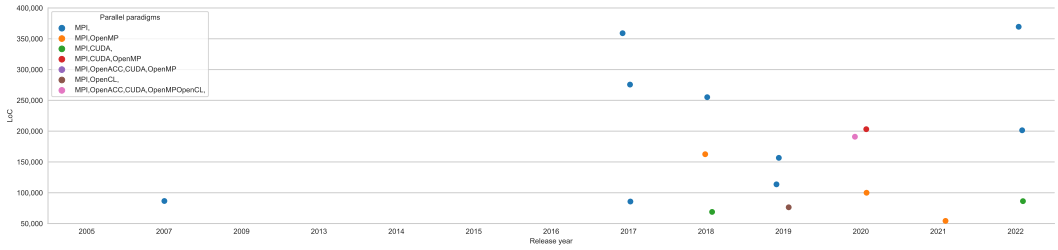
In Fig. 5.9, we can see the applications' programming languages over the years with the LoC. As in Fig. 5.9a illustrated, we can see that within applications <50′000, there seem to be more applications, using C++ and C. In Fig. 5.9c, applications ranging between 50′000 and 400′000 LoC, tend to be released from 2017.



(a) Parallel programming paradigms over years with complete LoC

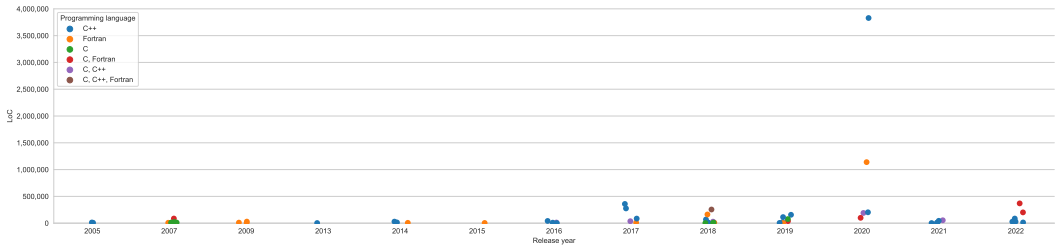


(b) Parallel programming paradigms over years with LoC: 0 - 50'000

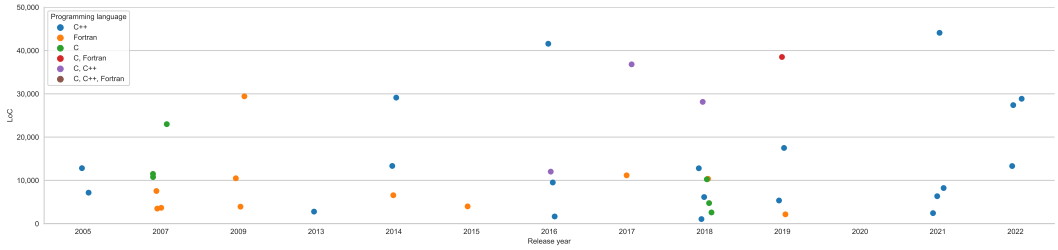


(c) Parallel programming paradigms over years with LoC: 50'000 - 400'000

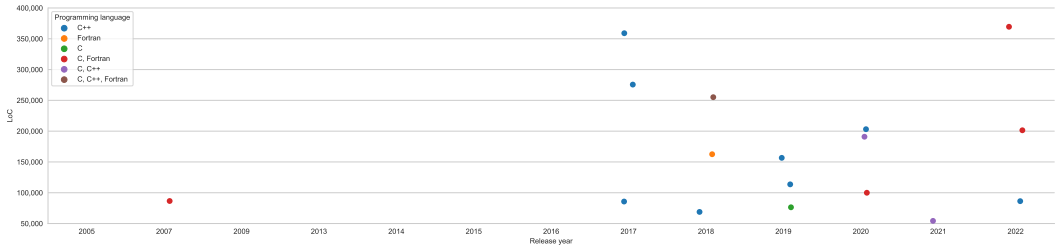
Figure 5.8: Used paradigms over time in relation with the LoC



(a) Programming language over years with complete LoC



(b) Programming language over years with LoC: 0 - 50'000



(c) Programming language over years with LoC: 50'000 - 400'000

Figure 5.9: Programming language over time in relation with the LoC

5.5 Domains in Programming Languages

We wanted to analyze the domains inside applications with their usage over programming language. In figure 5.10, we can see that we have analyzed various domains among the applications. For instance, we see that MD has been mostly represented within the analyzed applications. Furthermore, MD applications have been implemented in almost every combination of the code languages. Followed by the domains SPH and CFD which make the second most used domains within the data set.

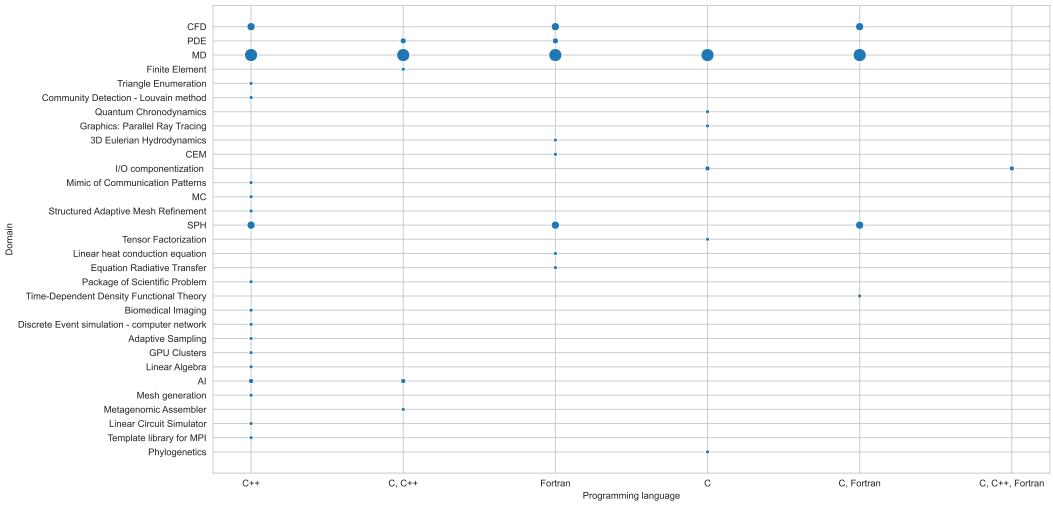


Figure 5.10: Domain representation with their used programming language

5.6 Trends over years in Paradigms and Programming language

In the following, we showed the trend over years of all analyzed applications. Furthermore, we illustrated their used parallel programming paradigms, respectively their used programming language. As shown in figure 5.11, the releases of 2021, the analyzed applications have been majorly using the paradigms of MPI and the combination between MPI and CUDA. Whereas the applications released between 2005 and 2009 primarily used MPI.

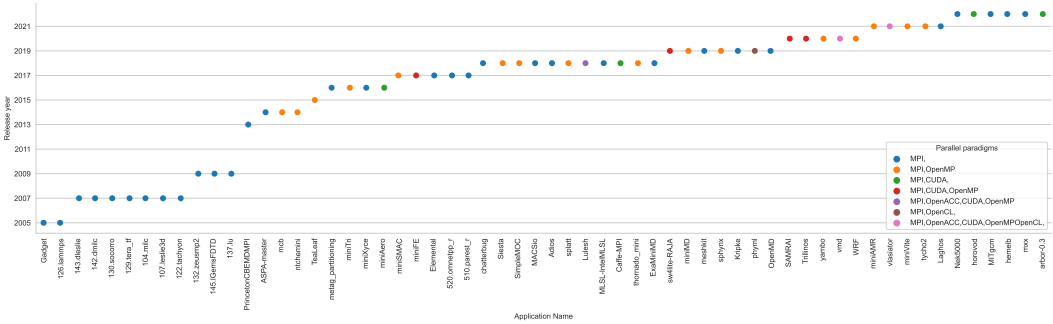


Figure 5.11: Occurrences of paradigm, decomposition and scheduling at process level

In figure 5.12, we see that the oldest programming language among the group of C, C++, and Fortran is still used in the newer releases. Generally, the programming language C++

seem to be mostly represented among the applications.

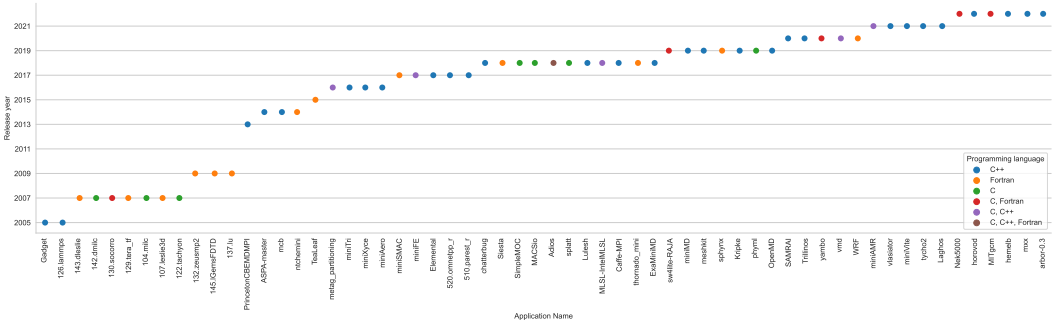


Figure 5.12: Occurrences of paradigm, decomposition and scheduling at process level

6
Conclusion

Investigation of domain decomposition and scheduling in HPC applications consisted of analytical work with a broad area inside HPC applications. First, open source applications have been gathered. Second, information about the domain decomposition and scheduling have been manually retrieved from each application. Third, results have been analyzed and followed by looking at various aspects within applications. Fortunately, number based information, such as LoC, Storage, or OpenMP usages can be retrieved automatically with the provided script. The difficulty lied in retrieving places where scheduling and domain decomposition have occurred, as the time-effort of analysis depend on the application itself. Nevertheless, we managed to show results, covering a broad part of the data. Most of the analyzed applications used MPI as their solely programming paradigm. Furthermore, nearly 75% have used a distributed domain decomposition. Last but not least, most of the applications have been scheduled statically. Another point of view is the dependence of lines of code within the applications. Furthermore, we displayed all the analyzed applications with their release dates and their used parallel programming paradigms, domain decomposition and scheduling at process level. At the end, we believe that we fulfilled the aim to set an overview, displaying a wide area of HPC applications describing their usage of the common way of code within the community.

Bibliography

[1] SPEC MPI 2007. 126.lammps, 2005. URL http://www.spec.org/auto/mpi2007/Docs/126.lammps.html. Release date; 1/17/2005.

[2] SPEC MPI 2007. 104.milc, 2007. URL http://www.spec.org/auto/mpi2007/Docs/104.milc.html. Release date; 3/16/2007.

[3] SPEC MPI 2007. 107.leslie3d, 2007. URL http://www.spec.org/auto/mpi2007/Docs/107.leslie3d.html. Release date; 4/11/2007.

[4] SPEC MPI 2007. 122.tachyon, 2007. URL http://www.spec.org/auto/mpi2007/Docs/122.tachyon.html. Release date; 2/2/2007.

[5] SPEC MPI 2007. 129.tera, 2007. URL http://www.spec.org/auto/mpi2007/Docs/129.tera_tf.html. Release date; 2/5/2007.

[6] SPEC MPI 2007. 130.socorro, 2007. URL http://www.spec.org/auto/mpi2007/Docs/130.socorro.html. Release date; 2/6/2007.

[7] SPEC MPI 2007. 142.dmilc, 2007. URL http://www.spec.org/auto/mpi2007/Docs/142.dmilc.html. Release date; 3/16/2007.

[8] SPEC MPI 2007. 143.dleslie, 2007. URL http://www.spec.org/auto/mpi2007/Docs/143.dleslie.html. Release date; 4/11/2007.

[9] SPEC MPI 2007. 132.zeusmp2, 2009. URL http://www.spec.org/auto/mpi2007/Docs/132.zeusmp2.html. Release date; 9/3/2009.

[10] SPEC MPI 2007. 137.lu, 2009. URL http://www.spec.org/auto/mpi2007/Docs/137.lu.html. Release date; 9/3/2009.

[11] SPEC MPI 2007. 145.lgemsfdtd, 2009. URL http://www.spec.org/auto/mpi2007/Docs/145.lGemsFDTD.html. Release date; 4/20/2009.

[12] SPEC CPU 2017. 500.perlbench_r, 2017. URL https://www.spec.org/cpu2017/Docs/benchmarks/500.perlbench_r.html. Release date; 1/1/2017.

[13] SPEC CPU 2017. 508.namd_r, 2017. URL https://www.spec.org/cpu2017/Docs/benchmarks/508.namd_r.html. Release date; 1/1/2017.

[14] SPEC CPU 2017. 510.parest_r, 2017. URL https://www.spec.org/cpu2017/Docs/benchmarks/510.parest_r.html. Release date; 1/1/2017.

[15] SPEC CPU 2017. 511.povray_r, 2017. URL https://www.spec.org/cpu2017/Docs/benchmarks/511.povray_r.html. Release date; 1/1/2017.

[16] SPEC CPU 2017. 520.omnetpp_r, 2017. URL https://www.spec.org/cpu2017/Docs/benchmarks/520.omnetpp_r.html. Release date; 1/1/2017.

[17] SPEC CPU 2017. 523.xalancbmk_r, 2017. URL https://www.spec.org/cpu2017/Docs/benchmarks/523.xalancbmk_r.html. Release date; 1/1/2017.

[18] SPEC CPU 2017. 531.deepsjeng_r, 2017. URL https://www.spec.org/cpu2017/Docs/benchmarks/531.deepsjeng_r.html. Release date; 1/1/2017.

[19] SPEC CPU 2017. 541.leela_r, 2017. URL https://www.spec.org/cpu2017/Docs/benchmarks/541.leela_r.html. Release date; 1/1/2017.

[20] SPEC CPU 2017. 548.exchange2_r, 2017. URL https://www.spec.org/cpu2017/Docs/benchmarks/548.exchange2_r.html. Release date; 1/1/2017.

[21] Rodinia 3.1. Srad, 2015. URL https://github.com/JuliaParallel/rodinia/tree/master/openmp/srad_v1. Release date; 11/10/2015.

[22] Rodinia 3.1. Streamcluster, 2016. URL https://github.com/JuliaParallel/rodinia/tree/master/openmp/streamcluster. Release date; 9/19/2016.

[23] Fiber Mini App. ntchemini, 2014. URL https://github.com/fiber-miniapp/ntchem-mini. Release date; 10/31/2014.

[24] LLNL ASC Proxy Apps. Lulesh, 2018. URL https://github.com/LLNL/LULESH. Release date; 7/18/2018.

[25] No benchmark suite. Gadget, 2005. URL https://wwwmpa.mpa-garching.mpg.de/gadget/. Release date; 1/5/2005.

[26] No benchmark suite. Princetoncbemdmpi, 2013. URL https://github.com/PrincetonUniversity/PrincetonCBEMDMPI. Release date; 1/17/2013.

[27] No benchmark suite. mcb, 2014. URL https://computing.llnl.gov/projects/co-design/mcb. Release date; 1/6/2014.

[28] No benchmark suite. metag_partitioning, 2016. URL https://github.com/ParBLiSS/metag_partitioning. Release date; 9/29/2016.

[29] No benchmark suite. Cfdemcoupling, 2017. URL https://github.com/CFDEMproject/CFDEMcoupling-PUBLIC. Release date; 12/4/2017.

[30] No benchmark suite. Elemental, 2017. URL https://github.com/elemental/Elemental/releases/tag/v0.87.7. Release date; 2/7/2017.

[31] No benchmark suite. Adios, 2018. URL https://github.com/ornladios/ADIOS/releases. Release date; 4/18/2018.

[32] No benchmark suite. Caffe-mpi, 2018. URL https://github.com/Caffe-MPI/Caffe-MPI.github.io. Release date; 2/5/2018.

[33] No benchmark suite. Macsio, 2018. URL https://github.com/LLNL/MACSio. Release date; 10/2/2018.

[34] No benchmark suite. Mlsl-intelmlsl, 2018. URL https://github.com/intel/MLSL. Release date; 10/1/2018.

[35] No benchmark suite. Siesta, 2018. URL https://gitlab.com/siesta-project/siesta/-/releases. Release date; 7/19/2018.

[36] No benchmark suite. Simplemoc, 2018. URL https://github.com/ANL-CESAR/SimpleMOC/tree/v4. Release date; 7/31/2018.

[37] No benchmark suite. splatt, 2018. URL https://github.com/ShadenSmith/splatt. Release date; 9/5/2018.

[38] No benchmark suite. thornado_mini, 2018. URL https://github.com/ECP-Astro/thornado_mini. Release date; 9/9/2018.

[39] No benchmark suite. Openmd, 2019. URL https://github.com/OpenMD/OpenMD. Release date; 8/31/2019.

[40] No benchmark suite. meshkit, 2019. URL https://bitbucket.org/fathomteam/meshkit.git/src. Release date; 12/9/2019.

[41] No benchmark suite. phyml, 2019. URL https://github.com/stephaneguindon/phyml. Release date; 3/21/2019.

[42] No benchmark suite. souffle, 2019. URL https://github.com/souffle-lang/souffle/releases. Release date; 8/19/2019.

[43] No benchmark suite. sphynx, 2019. URL https://astro.physik.unibas.ch/en/people/ruben-cabezon/sphynx/. Release date; 7/3/2019.

[44] No benchmark suite. sw4lite-raja, 2019. URL https://github.com/geodynamics/sw4lite/tree/RAJA-v1.0. Release date; 3/15/2019.

[45] No benchmark suite. Samrai, 2020. URL https://github.com/LLNL/SAMRAI. Release date; 9/30/2020.

[46] No benchmark suite. Trillinos, 2020. URL https://github.com/trilinos/Trilinos. Release date; 8/6/2020.

[47] No benchmark suite. Wrf, 2020. URL https://github.com/wrf-model/WRF/releases/tag/v4.2.1. Release date; 7/22/2020.

[48] No benchmark suite. vmd, 2020. URL https://www.ks.uiuc.edu/Research/vmd/. Release date; 10/13/2020.

[49] No benchmark suite. yambo, 2020. URL https://github.com/yambo-code/yambo. Release date; 7/20/2020.

[50] No benchmark suite. tycho2, 2021. URL https://github.com/lanl/tycho2. Release date; 6/1/2021.

[51] No benchmark suite. vlasiator, 2021. URL https://github.com/fmihpc/vlasiator/releases/tag/v5.1. Release date; 4/26/2021.

[52] No benchmark suite. Mitgcm, 2022. URL https://github.com/MITgcm/MITgcm. Release date; 4/27/2022.

[53] No benchmark suite. Nek5000, 2022. URL https://github.com/Nek5000/nekRS/releases/tag/v21.1. Release date; 05/13/2022.

[54] No benchmark suite. arbor-0.3, 2022. URL https://github.com/arbor-sim/arbor. Release date; 01/26/2022.

[55] No benchmark suite. hemelb, 2022. URL https://github.com/hemelb-codes/hemelb. Release date; 6/20/2022.

[56] No benchmark suite. horovod, 2022. URL https://github.com/horovod/horovod/releases. Release date; 4/21/2022.

[57] No benchmark suite. mxx, 2022. URL https://github.com/patflick/mxx. Release date; 4/13/2022.

[58] No benchmark suite "Co-design center for Particle Applications (CoPA)". Examinimd, 2018. URL https://github.com/ECP-copa/ExaMiniMD/releases/tag/1.0. Release date; 3/14/2018.

[59] Chatterbug. chatterbug, 2018. URL https://github.com/hpcgroup/chatterbug. Release date; 9/18/2018.

[60] VS Code. Visual studio code, 2022. URL https://code.visualstudio.com/. IDE in which code was analyzed.

[61] UK Mini-App Consortium. Tealeaf, 2015. URL https://github.com/UK-MAC/TeaLeaf_ref. Release date; 6/2/2015.

[62] CORAL-2. Kripke, 2019. URL https://github.com/LLNL/Kripke. Release date; 6/14/2019.

[63] CORAL-2. Laghos, 2021. URL https://github.com/CEED/Laghos. Release date; 4/10/2021.

[64] ExaGraph. minivite, 2021. URL https://github.com/Exa-Graph/miniVite. Release date; 4/10/2021.

[65] ExMatEx. Aspa-master, 2014. URL https://github.com/exmatex/ASPA/tree/master/doc. Release date; 1/23/2014.

[66] Mantevo. miniaero, 2016. URL https://github.com/Mantevo/miniAero/releases. Release date; 7/6/2016.

[67] Mantevo. minitri, 2016. URL https://github.com/Mantevo/miniTri. Release date; 7/6/2016.

[68] Mantevo. minixyce, 2016. URL https://github.com/Mantevo/miniXyce. Release date; 7/6/2016.

[69] Mantevo. minife, 2017. URL https://github.com/Mantevo/miniFE/releases. Release date; 11/22/2017.

[70] Mantevo. minismac, 2017. URL https://github.com/Mantevo/miniSMAC/releases. Release date; 10/24/2017.

[71] Mantevo. minimd, 2019. URL https://github.com/Mantevo/miniMD. Release date; 2/28/19.

[72] Mantevo. miniamr, 2021. URL https://github.com/Mantevo/miniAMR. Release date; 11/23/2021.

[73] Nderim Shatri. Controltable, 2022. URL https://bitbucket.org/unibasdmihpc/nderim-master-project/src/master/tables/ControlTable.csv. Controltable contains all the data.

[74] Nderim Shatri. Controltable, 2022. URL https://bitbucket.org/unibasdmihpc/nderim-master-project/src/master/tables/ProcessedControlTable.csv. Processed Controltable which is after processing the original ControlTable.

A

Appendix

A.1 ControlTable[73]

Suite(M)	Application Name (M)	Links(M)	Domain(M)	Description(M)	LoC(A)	Programming Language (M)	Fortran Lines(A)	C Lines(A)	C++ Lines(A)	C_CPP_H_Lines(A)	OpenMP(A/M)	MPI(M)	OpenACC(A/M)	CUDA(A/M)	OpenCL(A/M)	Data Decomposition (M)	Location Data Decomposition(M)	Process level (M) Scheduling	Location Scheduling(M)	Storage (MB) (M)	Version(M)	Release Date (M) (github release date	Newer Versions(M	New Release Date (M) (github)	Notes(M)
LLNL ASC Proxy Apps	Lulesh	https://github.com/LLNL/LULESH	CFD	LULESH is a highly simplified application, hard-coded to only solve a simple Sedov blast problem with analytic answers – but represents the numerical algorithms, data motion, and programming style typical in scientific C or C++ based applications	12798	C++	0	0	11296	1502	yes	yes	yes	yes	no	distributed	lulesh-init.cc [189 - 238]	dynamic	lulesh-init.cc [377-484]	4.4	2.0.2	7/18/2018	no	-	
Mantevo	miniAMR	https://github.com/Mantevo/miniAMR	PDE	miniAMR applies a stencil calculation on a unit cube computational domain, which is divided into blocks. The blocks all have the same number of cells in each direction and communicate ghost values with neighboring blocks.	54312	C, C++	0	4850	18017	31445	yes	yes	no	no	no	distributed	init.c	static	init.c	1.1	1.7	11/23/2021	no	-	
Mantevo	miniMD	https://github.com/Mantevo/miniMD	MD	miniMD is a parallel molecular dynamics (MD) simulation package written in C++ and intended for use on parallel supercomputers and new architechtures for testing purposes. The software package is meant to be simple, lightweight, and easily adapted to new hardware.	17496	C++	0	371	14468	2657	yes	yes	no	no	no	distributed	setup.cc [278-282]	static	setup.cc [359 -392]	2.6	1.2	2/28/19	no	-	
Mantevo	miniFE	https://github.com/Mantevo/miniFE/releases	Finite Element	MiniFE is an proxy application for unstructured implicit finite element codes. It is similar to HPCCG and pHPCCG but provides a much more complete vertical covering of the steps in this class of applications.	36812	C, C++	0	4479	3549	28784	yes	yes	no	yes	no	distributed	BoxPartitionIB.c : box_partition	static	BoxPartionIB.c	21	2.2	11/22/2017	no	-	
Mantevo	miniSMAC	https://github.com/Mantevo/miniSMAC/releases	PDE	Solves the finite-differenced 2D incompressible Navier-Stokes equations with Spalart-Allmaras one-equation turbulence model on a structured body conforming grid. The grid is partitioned into subgrids load balanced for the number of MPI ranks requested by the user	11167	Fortran	10984	0	0	183	yes	yes	no	no	no	distributed	hint in smac2d.f [536-546]	static	hint in smac2d.f [536-546]	0.812	1	10/24/2017	no	-	
Mantevo	miniTri	https://github.com/Mantevo/miniTri	Triangle Enumeration	miniTri is a proxy for a class of triangle based data analytics (Mantevo). This simple code is a self-contained piece of C++ software that uses triangle enumeration with a calculation of specific vertex and edge properties.	9517	C++	0	0	7779	1738	yes	yes	no	no	no	distributed	CSRMatrix.cc[1053-1096]	dynamic	CSRMatrix.cpp [1105-1203]	1.2	2.0.0	7/6/2016	no	-	
ExaGraph	miniVite	https://github.com/Exa-Graph/miniVite	Community Detection - Louvain method	miniVite is a proxy app that implements a single phase of Louvain	2429	C++	0	0	195	2234	yes	yes	no	no	no	distributed	graph.hpp [367-377]	dynamic	graph.hpp [from 980 onwards?]	0.148	1.2	4/10/2021	no	-	
Fiber Mini App	ntchemini	https://github.com/fiber-miniapp/ntchem-mini	MD	NTChem is a high-performance software package for the molecular electronic structure calculation for general purpose on the K computer	6571	Fortran	6571	0	0	0	yes	yes	no	no	no	replicated	util_initmpi	static	util_initmpi.f90 [96-124]	39	1.2	10/31/2014	no	-	No hints of other data decomposition
SPEC MPI 2007	104.milc	http://www.spec.org/auto/mpi2007/Docs/104.milc.html	Quantum Chronodynamics	The MILC Code is a set of codes written in C developed by the MIMD Lattice Computation (MILC) collaboration for doing simulations of four dimensional SU(3) lattice gauge theory on MIMD parallel machines. The code is used for millions of node hours at DOE and NSF supercomputer centers.	22989	C	0	10424	0	12565	no	yes	no	no	no	distributed	setup.c [146-253]	static	control.c [59-110]	0.948	-	3/16/2007	no	-	
SPEC MPI 2007	107.leslie3d	http://www.spec.org/auto/mpi2007/Docs/107.leslie3d.html	CFD	107.leslie3d is derived from LESlie3d (Large-Eddy Simulations with Linear-Eddy Model in 3D), a research-level Computational Fluid Dynamics (CFD) code. It is the primary solver used to investigate a wide array of turbulence phenomena such as mixing, combustion, acoustics and general fluid mechanics.	7545	Fortran	7545	0	0	0	no	yes	no	no	no	distributed	main.f [17-55]	static	parallel.f[202-235]	0.536	-	4/11/2007	no	-	
SPEC MPI 2007	122.tachyon	http://www.spec.org/auto/mpi2007/Docs/122.tachyon.html	Graphics: Parallel Ray Tracing	Tachyon is a parallel ray tracing application.	10763	C	0	9592	0	1171	no	yes	no	no	no	distributed	parallel.c	dynamic	parallel.c	0.387	0.97	2/2/2007	no	-	
SPEC MPI 2007	126.lammps	http://www.spec.org/auto/mpi2007/Docs/126.lammps.html	MD	LAMMPS is a classical molecular dynamics simulation code designed to run efficiently on parallel computers. It was developed at Sandia National Laboratories, a US Department of Energy facility, with funding from the DOE. It is an open-source code, distributed freely under the terms of the GNU Public License (GPL)	7162	C++	0	0	3241	3921	no	yes	no	no	no	distributed	read_data.cpp different methods using distribution ex. ReadData:impropers() [580-618]	static	universe.cpp [55-87]	11	-	1/17/2005	no	-	
SPEC MPI 2007	129.tera_tf	http://www.spec.org/auto/mpi2007/Docs/129.tera_tf.html	3D Eulerian Hydrodynamics	3D eulerian hydrodynamics application 2nd godunov-type scheme, 3rd order remapping requires only a Fortran 90 compiler, and an MPI (1.2) implementation uses mostly point-to-point messages, and some reductions uses non-blocking messages	3665	Fortran	3665	0	0	0	no	yes	no	no	no	replicated	TF_GENDATA.f90 [2...]	static	TF_INIT.f90 [INITIALIZE_RUN]	0.588	-	2/5/2007	no	-	
SPEC MPI 2007	130.socorro	http://www.spec.org/auto/mpi2007/Docs/130.socorro.html	MD	Socorro is a modular, object oriented code for performing self-consistent electronic-structure calculations utilizing the Kohn-Sham formulation of density-functional theory.	86593	C, Fortran	53199	32842	0	552	no	yes	no	no	no	replicated	mpi_mod.f90[380-480]	static	mpi_mod.f90[225-273]	5.5	1.0	2/6/2007	no	-	
SPEC MPI 2007	132.zeusmp2	http://www.spec.org/auto/mpi2007/Docs/132.zeusmp2.html	CFD	ZEUS-MP is a computational fluid dynamics code developed at the Laboratory for Computational Astrophysics (NCSA, SDSC, University of Illinois at Urbana-Champaign, UC San Diego) for the simulation of astrophysical phenomena.	29414	Fortran	29323	91	0	0	no	yes	no	no	no	distributed	configure.F	static	configure.F	2.1	-	9/3/2009	no	-	
SPEC MPI 2007	137.lu	http://www.spec.org/auto/mpi2007/Docs/137.lu.html	PDE	The 137.lu code has a rich ancestry in benchmarking. Its immediate predecessor is the LU benchmark in NPB3.2-MPI, part of the NAS Parallel Benchmark suite. It is sometimes referred to as APPLU (a version of that was 173.applu in CPU2000) or NAS-LU. Solution of five coupled nonlinear PDE's, on a 3-dimensional logically structured grid, using an implicit pseudo-time marching scheme, based on two-factor approximate factorization of the sparse Jacobian matrix.	3924	Fortran	3924	0	0	0	no	yes	no	no	no	centralized	read_input.F (Hint in comments)	static	subdomain.F	0.532	-	9/3/2009	no	-	
SPEC MPI 2007	142.dmilc	http://www.spec.org/auto/mpi2007/Docs/142.dmilc.html	MD	The MILC Code is a set of codes written in C developed by the MIMD Lattice Computation (MILC) collaboration for doing simulations of four dimensional SU(3) lattice gauge theory on MIMD parallel machines.	11498	C	0	10426	0	1072	no	yes	no	no	no	distributed	io_lat4.c	static	com_mpi.c [962-983]	0.944	-	3/16/2007	no	-	
SPEC MPI 2007	143.dleslie	http://www.spec.org/auto/mpi2007/Docs/143.dleslie.html	CFD	143.dleslie is derived from LESlie3d (Large-Eddy Simulations with Linear-Eddy Model in 3D), a research-level Computational Fluid Dynamics (CFD) code.	3483	Fortran	3483	0	0	0	no	yes	no	no	no	replicated	main.f [36-78]	static	parallel.f:GRIDMAP() [168-253]	0.32	2.0	4/11/2007	no	-	
SPEC MPI 2007	145.lGemsFDTD	http://www.spec.org/auto/mpi2007/Docs/145.lGemsFDTD.html	CEM	GemsFDTD solves the Maxwell equations in 3D in the time domain using the finite-difference time-domain (FDTD) method. GemsFDTD is a Computational Electromagnetic (CEM) application	10487	Fortran	10487	0	0	0	no	yes	no	no	no	distributed	Loadbalance.f90	dynamic	Loadbalance.f90	24	-	4/20/2009	no	-	No hints of other data decomposition
NO	Adios	https://github.com/ornladios/ADIOS/releases	I/O componentization	ADIOS is developed as part of the United States Department of Energy's Exascale Computing Project. It is a framework for scientific data I/O to publish and subscribe to data when and where required.	255137	C, C++, Fortran	12319	151373	83788	7657	no	yes	no	no	no	distributed	read_dimes.c[1233-1296]	static	adios_nssi_config.c [218-276]	26	1.3.1	4/18/2018	no	-	
Chatterbug	chatterbug	https://github.com/hpcgroup/chatterbug	Mimic of Communication Patterns	A suite of communication-intensive proxy applications that mimic commonly found communication patterns in HPC codes. These codes can be used as synthetic codes for benchmarking, or for trace generation using OTF2.	1056	C++	0	0	1056	0	no	yes	no	no	no	replicated	unstr-mesh.c	static	unstr-mesh.c[86-94]	0.156	1.0	9/18/2018	no	-	No hints of other data decomposition
NO "Co-design center for Particle Applications (CoPA)"	ExaMiniMD	https://github.com/ECP-copa/ExaMiniMD/releases/tag/1.0	MD	ExaMiniMD is a proxy application and research vehicle for particle codes, in particular Molecular Dynamics (MD). Compared to previous MD proxy apps (MiniMD, COMD), its design is significantly more modular in order to allow independent investigation of different aspects	6149	C++	0	0	1816	4333	no	yes	no	no	no	unknown	-	unknown	-	0.556	1.0	3/14/2018	no	-	Usage of Kokkos
CORAL-2	Kripke	https://github.com/LLNL/Kripke	MD	Kripke is a simple, scalable, 3D Sn deterministic particle transport code. Its primary purpose is to research how data layout, programming paradigms and architectures effect the implementation and performance of Sn transport. Kripke is also a Proxy-App since it is a proxy for the LLNL transport code ARDRA.	5350	C++	0	0	2471	2879	no	yes	no	no	no	distributed	PartionSpace.cpp[75-123]	static	PartitionSpace.cpp	0.42	1.2.4	6/14/2019	no	-	
NO	MACSio	https://github.com/LLNL/MACSio	I/O componentization	MACSio is being developed to fill a long existing void in co-design proxy applications that allow for I/O performance testing and evaluation of tradeoffs in data models, I/O library interfaces and parallel I/O paradigms for multi-physics, HPC applications.	4749	C	0	4705	0	44	no	yes	no	no	no	distributed	make_mesh.c[329-390]	static	make_mesh.c[43-58]	6.7	1.1	10/2/2018	no	-	
NO	mcb	https://computing.llnl.gov/projects/co-design/r	MC	The Monte Carlo Benchmark (MCB) is intended for use in exploring the computational performance of Monte Carlo algorithms on parallel architectures.	13334	C++	0	0	9704	3630	yes	yes	no	no	no	replicated	see Notes	static	MCBenchmark.cc [689-695]	1.5	1	1/6/2014	no	-	No hints of other data decomposition
Mantevo	miniAero	https://github.com/Mantevo/miniAero/releases	CFD	MiniAero is a mini-application for the evaulation of programming models and hardware for next generation platforms. MiniAero is an explicit (using RK4) unstructured finite volume code that solves the compressible Navier-Stokes equations.	41570	C++	0	0	11666	29904	no	yes	no	yes	no	unknown	-	unknown	-	4.2	1.0.0	7/6/2016	no	-	Usage of Kokkos
NO	OpenMD	https://github.com/OpenMD/OpenMD	MD	OpenMD is an open source molecular dynamics engine which is capable of efficiently simulating liquids, proteins, nanoparticles, interfaces, and other complex systems using atom types with orientational degrees of freedom (e.g. "sticky" atoms, point dipoles, and coarse-grained assemblies).	113635	C++	0	0	84362	29273	no	yes	no	no	no	replicated	see Notes	dynamic	Simcreator.cpp[530-669]	124	2.6	8/31/2019	no	-	No hints of other data decomposition
NO	SAMRAI	https://github.com/LLNL/SAMRAI	Structured Adaptive Mesh Refinement	SAMRAI (Structured Adaptive Mesh Refinement Application Infrastructure) is an object-oriented C++ software library that enables exploration of numerical, algorithmic, parallel computing, and software issues associated with applying structured adaptive mesh refinement (SAMR) technology in large-scale parallel application development. SAMRAI provides software tools for developing SAMR applications that involve coupled physics models, sophisticated numerical solution methods, and which require high-performance parallel computing hardware. SAMRAI enables integration of SAMR technology into existing codes and simplifies the exploration of SAMR methods in new application domains.	203168	C++	2741	0	150953	49474	yes	yes	no	yes	no	distributed	TreeLoadBalancer.c	dynamic	TreeLoadBalancer.c	66	4.01	9/30/2020	4.1.1	3/5/2022	

Suite(M)	Application Name (M)	Links(M)	Domain(M)	Description(M)	LoC(A)	Programming Language (M)	Fortran Lines(A)	C Lines(A)	C++ Lines(A)	C_CPP_H_Lines(A)	OpenMP(A/M)	MPI(M)	OpenACC(A/M)	CUDA(A/M)	OpenCL(A/M)	Data Decomposition (M)	Location Data Decomposition(M)	Process level (M) Scheduling	Location Scheduling(M)	Storage (MB) (M)	Version(M)	Release Date (M) (github release date	Newer Versions(M	New Release Date (M) (github)	Notes(M)
NO	Siesta	https://gitlab.com/siesta-project/siesta/-/releases	MD	SIESTA is both a method and its computer program implementation, to perform efficient electronic structure calculations and ab initio molecular dynamics simulations of molecules and solids. SIESTA's efficiency stems from the use of strictly localized basis sets and from the implementation of linear-scaling algorithms which can be applied to suitable systems.	162544	Fortran	162418	126	0	0	yes	yes	no	no	no	replicated	domain_decom.F[293-349]	static	sys.f [66-89]	47	4.0.2	7/19/2018	4.1.5	1/27/2021	
NO	SimpleMOC	https://github.com/ANL-CESAR/SimpleMOC/tree/v4	MD	The purpose of this mini-app is to demonstrate the performance characterterics and viability of the Method of Characteristics (MOC) for 3D neutron transport calculations in the context of full scale light water reactor simulation.	2591	C	0	2396	0	195	yes	yes	no	no	no	replicated	main.c	static	init.c[162-224]	0.164	v4	7/31/2018	no	-	
NO	souffle	https://github.com/souffle-lang/souffle/releases	Logic programming language	Souffle is a logic programming language inspired by Datalog. It overcomes some of the limitations in classical Datalog. For example, programmers are not restricted to finite domains, and the usage of functors (intrinsic, user-defined, records/constructors, etc.) is permitted. Soufflé has a component model so that large logic projects can be expressed.	58128	C++	0	0	25847	32281	yes	no	no	no	no	-	-	-	-	61	2.1	8/19/2019	2.2	1/17/2022	No MPI
NO	sphynx	https://astro.physik.unibas.ch/en/people/ruben-cabezon/sphynx/	SPH	SPHYNX is an SPH hydrocode with its focus on Astrophysical applications. SPHYNX includes state-of-the-art methods that allow it to address subsonic hydrodynamical instabilities and strong shocks, which are ubiquitous in astrophysical scenarios. SPHYNX, is of Newtonian type and grounded on the Euler-Lagrange formulation of the smoothed-particle hydrodynamics technique	2152	Fortran	2152	0	0	0	yes	yes	no	no	no	distributed	sphynx_hybrid.f90[40-60]	static	buildtreemod_grav.f90	1.5	1.5.3	7/3/2019	no	-	
NO	splatt	https://github.com/ShadenSmith/splatt	Tensor Factorization	SPLATT is a library and C API for sparse tensor factorization. SPLATT supports shared-memory parallelism with OpenMP and distributed-memory parallelism with MPI	10252	C	0	9135	0	1117	yes	yes	no	no	no	distributed	mpi_io.c[852-896]	dynamic	matrix.c[490-415]	20	1.1.0	9/5/2018	no	-	
Rodinia 3.1	SRAD	https://github.com/JuliaParallel/rodinia/tree/master/openmp/srad_v1	PDE	SRAD (Speckle Reducing Anisotropic Diffusion) is a diffusion method for ultrasonic and radar imaging applications based on partial differential equations (PDEs). It is used to remove locally correlated noise, known as speckles, without destroying important image features. SRAD consists of several pieces of work: image extraction, continuous iterations over the image (preparation, reduction, statistics, computation 1 and computation 2) and image compression. The sequential dependency between all of these stages requires synchronization after each stage (because each stage operates on the entire image).	322	C	0	308	0	14	yes	no	no	yes	yes	-	-	-	-	0.084	1.1	11/10/2015	yes	-	No MPI
Rodinia 3.1	Streamcluster	https://github.com/JuliaParallel/rodinia/tree/master/openmp/streamcluster	Heterogeneous Computing	It assigns each point of a stream to its nearest center Medium-sized working sets of user-determined size	1945	C++	0	0	1945	0	yes	no	no	yes	yes	-	-	-	-	0.08		9/19/2016		-	This is a modified application of PARSEC Benchmark Suite(https://parsec.cs.princeton.edu/parsec3-doc.htm). It is mentioned here: http://www.cs.virginia.edu/rodinia/doku.php?id=streamcluster
NO	sw4lite-RAJA	https://github.com/geodynamics/sw4lite/tree/RAJA-v1.0	SPH	sw4lite is a bare bone version of SW4 (https://github.com/geodynamics/sw4) intended for testing performance optimizations in a few important numerical kernels of SW4.	38510	C, Fortran	7874	0	28774	1862	yes	yes	no	yes	no	distributed	EW.c[140-218]	static	EW.c[264-473]	4.9	1.0	3/15/2019	no	-	
UK Mini-App Consortium	TeaLeaf	https://github.com/UK-MAC/TeaLeaf_ref	Linear heat conduction equation	TeaLeaf is a mini-app that solves the linear heat conduction equation on a spatially decomposed regularly grid using a 5 point stencil with implicit solvers. TeaLeaf currently solves the equations in two dimensions, but three dimensional support is in beta.	3989	Fortran	3959	11	0	19	yes	yes	no	no	no	distributed	tea.f90[195-235]	static	tea.f90[121-193]	0.344	"Matrix Powers"	6/2/2015	no	-	
NO	thornado_mini	https://github.com/ECP-Astro/thornado_mini	Equation Radiative Transfer	Thornado_mini solves the equation of radiative transfer in the multi-group two-moment approximation. The Discontinuous Galekin (DG) method is used for spatial discretization, and an implicit-explicit (IMEX) method is used to integrate the moment equations in time. The hyperbolic (streaming) part is treated explicitly, while the collision term is treated implicitly.	10368	Fortran	10368	0	0	0	yes	yes	no	no	no	distributed	Meshmodule.f90[29-64]	static	PolynomialBasisModule_Lagrange[215-241]	16	master	9/9/2018	no	-	
NO	Trillinos	https://github.com/trilinos/Trilinos	Package of Scientific Problem	The Trilinos Project is an effort to develop algorithms and enabling technologies within an object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific problems. A unique design feature of Trilinos is its focus on packages.	3828846	C++	175022	379380	1540325	1734119	yes	yes	no	yes	no	distributed	different application examples	dynamic	different localities	792	13.0.0	8/6/2020	13.2.0	10/29/2021	Collection of different applications
NO	tycho2	https://github.com/lanl/tycho2	MD	A mini-app for neutral-particle, discrete-ordinates (SN), transport on parallel-decomposed meshes of tetrahedra.	6341	C++	0	0	4991	1350	yes	yes	no	no	no	distributed	Tychomesh.cc	dynamic	different localities	182	0.2	6/1/2021	no	-	
NO	vlasiator	https://github.com/fmihpc/vlasiator/releases/tag/v5.1	MD	In Vlasiator, ions are represented as velocity distribution functions, while electrons are magnetohydrodynamic fluid, enabling a self-consistent global plasma simulation that can describe multi-temperature plasmas to resolve non-MHD processes that currently cannot be self-consistently described by the existing global space weather simulations. The novelty is that by modelling ions as velocity distribution functions the outcome will be numerically noiseless.	44095	C++	0	0	30662	13433	yes	yes	yes	yes	yes	distributed	grid.cc	dynamic	vlasiator.cpp	51	5.1	4/26/2021	no	-	
NO	vmd	https://www.ks.uiuc.edu/Research/vmd/	MD	VMD is a molecular visualization program for displaying, animating, and analyzing large biomolecular systems using 3-D graphics and built-in scripting.	190943	C, C++	0	10593	151812	28538	yes	yes	yes	yes	yes	distributed	Quicksurf.c	static	Quicksurf.c[1372-1395]	22	1.9.4	10/13/2020	no	-	
NO	WRF	https://github.com/wrf-model/WRF/releases/tag/v4.2.1	MD	WRF is a state-of-the-art atmospheric modeling system designed for both meteorological research and numerical weather prediction. It offers a host of options for atmospheric processes and can run on a variety of computing platforms.	1139122	Fortran	1073300	56279	0	9543	yes	yes	no	no	no	distributed	module_dm.f[1804-1905]	static	module_dm.f[1908-1928]	500	4.2.1	7/22/2020	yes	01/11/2022	
NO	yambo	https://github.com/yambo-code/yambo	Time-Dependent Density Functional Theory	YAMBO implements Many-Body Perturbation Theory (MBPT) methods (such as GW and BSE) and Time-Dependent Density Functional Theory (TDDFT), which allows for accurate prediction of fundamental properties as band gaps of semiconductors, band alignments, defect quasi-particle energies, optics and out-of-equilibrium properties of materials.	99987	C, Fortran	96164	3228	0	595	yes	yes	no	no	no	distributed	only hints	static	only hints	8.7	4.5	7/20/2020	yes	04/18/2021	Based on hints in comment section of the code
SPEC CPU 2017	500.perlbench_r	https://www.spec.org/cpu2017/Docs/benchmarks/500.perlbench_r.html	Perl interpreter	is a cut-down version of Perl v5.22.1, the popular scripting language. SPEC's version of Perl has had most of OS-specific features removed.	291132	C	0	148054	0	143078	no	no	no	no	no	-	-	-	-	106		1/1/2017	-	-	NO MPI
SPEC CPU 2017	508.namd_r	https://www.spec.org/cpu2017/Docs/benchmarks/508.namd_r.html	MD	The 508.namd_r benchmark is derived from the data layout and inner loop of NAMD, a parallel program for the simulation of large biomolecular systems. Although NAMD scales to over 200,000 cores for very large systems, serial performance is equally important to the over 50,000 users who have downloaded the program over the past decade. Almost all of the runtime is spent calculating inter-atomic interactions in a small set of functions. This set was separated from the bulk of the code to form a compact benchmark for CPU2017.	6396	C++	0	0	2174	4222	no	no	no	no	no	-	-	-	-	8.5	2017	1/1/2017	-	-	MPI as a top layer, no specific scheduling or data distribution technique
SPEC CPU 2017	510.parest_r	https://www.spec.org/cpu2017/Docs/benchmarks/510.parest_r.html	Biomedical Imaging	solves a problem from biomedical imaging. Specifically, the underlying problem is the reconstruction of interior properties of a 3d body from multiple observations at its two-dimensional surface, in much the same way as multiple 2d X-ray images are combined to do 3d CT (computed tomography) scans. The difference to CT scans is that the method this program describes is infrared light that does not go through tissues in a straight line, but diffuses.	359012	C++	0	0	106968	252044	no	yes	no	no	no	unknown	?	unknown	?	30	2017	1/1/2017	-	-	MPI as a top layer, no specific scheduling or data distribution technique
SPEC CPU 2017	511.povray_r	https://www.spec.org/cpu2017/Docs/benchmarks/511.povray_r.html	Ray tracing	POV-Ray is a free and open source ray-tracing application. The CPU 2017 version is based on POV-Ray version 3.7.	76598	C++	0	1019	72798	2781	no	no	no	no	no	-	-	-	-	24	2017	1/1/2017	-	-	NO MPI
SPEC CPU 2017	520.omnetpp_r	https://www.spec.org/cpu2017/Docs/benchmarks/520.omnetpp_r.html	Discrete Event simulation - computer network	The benchmark performs discrete event simulation of a large 10 gigabit Ethernet network. The simulation is based on the OMNeT++ discrete event simulation system ([1] www.omnetpp.org), a generic and open simulation framework. OMNeT++'s primary application area is the simulation of communication networks, but its generic and flexible architecture allows for its use in other areas such as the simulation of IT systems, queueing networks, hardware architectures or business processes as well.	85732	C++	0	0	72727	13005	no	yes	no	no	no	unknown	?	unknown	?	58	2017	1/1/2017	-	-	MPI as a top layer, no specific scheduling or data distribution technique
SPEC CPU 2017	523.xalancbmk_r	https://www.spec.org/cpu2017/Docs/benchmarks/523.xalancbmk_r.html	XML to HTML conversion via XSLT	XSLT processor for transforming XML documents into HTML, text, or other XML document types	294614	C++	0	3454	197453	93707	no	no	no	no	no	-	-	-	-	215	2017	1/1/2017	-	-	NO MPI
SPEC CPU 2017	531.deepsjeng_r	https://www.spec.org/cpu2017/Docs/benchmarks/531.deepsjeng_r.html	AI	531.deepsjeng_r is based on Deep Sjeng WC2008, the 2008 World Computer Speed-Chess Champion. Deep Sjeng is a rewrite of the older Sjeng-Free program, focused on obtaining the highest possible playing strength. (alpha-beta tree search & pattern recognition)	7284	C++	0	0	6659	625	no	no	no	no	no	-	-	-	-	0.66	2017	1/1/2017	-	-	NO MPI
SPEC CPU 2017	541.leela_r	https://www.spec.org/cpu2017/Docs/benchmarks/541.leela_r.html	AI	541.leela_r is a Go playing engine featuring Monte Carlo based position estimation, selective tree search based on Upper Confidence Bounds, and move valuation based on Elo ratings. (Monte Carlo simulation, game tree search & pattern recognition)	30524	C++	0	0	4655	25869	no	no	no	no	no	-	-	-	-	4.9	2017	1/1/2017	-	-	NO MPI

Suite(M)	Application Name (M)	Links(M)	Domain(M)	Description(M)	LoC(A)	Programming Language (M)	Fortran Lines(A)	C Lines(A)	C++ Lines(A)	C_CPP_H_Lines(A)	OpenMP(A/M)	MPI(M)	OpenACC(A/M)	CUDA(A/M)	OpenCL(A/M)	Data Decomposition (M)	Location Data Decomposition(M)	Process level (M) Scheduling	Location Scheduling(M)	Storage (MB) (M)	Version(M)	Release Date (M) (github release date	Newer Versions(M	New Release Date (M) (github)	Notes(M)
SPEC CPU 2017	548.exchange2_r	https://www.spec.org/cpu2017/Docs/benchmarks/548.exchange2_r.html	Games: Sudoku Puzzle Generator	This program was written for development of non-trivial 9x9 sudoku puzzles. It has been used extensively in informal competitions, which run for days. Incidentally, it tests many Fortran 95 array handling features (including some intrinsic functions) for use with integer arrays. Unusually, it relies heavily on recursion (up to eight levels deep) but, in contrast to most Fortran programs, uses no floating-point arithmetic.	1478	Fortran	1478	0	0	0	no	no	no	no	no	-	-	-	-	0.24	2017	1/1/2017	-	-	NO MPI
NO	arbor-0.3	https://github.com/arbor-sim/arbor	MD	Arbor is a high-performance library for computational neuroscience simulations with multi-compartment, morphologically-detailed cells, from single cell models to very large networks. Arbor is written from the ground up with many-cpu and gpu architectures in mind, to help neuroscientists effectively use contemporary and future HPC systems to meet their simulation needs.	86336	C++	0	0	43506	42830	no	yes	no	yes	no	distributed	mpi_context.cpp	dynamic	mpi_context.cpp	7.2	v0.6	01/26/2022	no	-	
ExMatEx	ASPA-master	https://github.com/exmatex/ASPA/tree/master/doc	Adaptive Sampling	The purpose of ASPA (Adaptive Sampling Proxy Application) is to enable the evaluation of a technique known as adaptive sampling on advanced computer architectures. Adaptive sampling is of interest in simulations involving multiple physical scales, wherein models of individual scales are combined using some form of scale bridging.	29119	C++	0	0	12956	16163	no	yes	no	no	no	replicated	MPI.cc	static	MPI.cc	5.5	master	1/23/2014	no	-	
NO	Caffe-MPI	https://github.com/Caffe-MPI/Caffe-MPI.github.io	GPU Clusters	The Caffe-MPI is designed for high density GPU clusters; The new version supports InfiniBand (IB) high speed network connection and shared storage system that can be equipped by distributed file system, like NFS and GlusterFS. The training dataset is read in parallel for each MPI process. The hierarchical communication mechanisms were developed to minimize the bandwidth requirements between computing nodes.	68871	C++	0	0	46110	22761	no	yes	no	yes	no	distributed	data_reader.cpp[230-268]	static	parallel.cpp[145193]	23	master	2/5/2018	no	-	
NO	CFDEMcoupling	https://github.com/CFDEMproject/CFDEMcoupling-PUBLIC	CFD	CFDEM® coupling provides an open source parallel coupled CFD-DEM framework combining the strengths of LIGGGHTS® DEM code and the Open Source CFD package OpenFOAM®(*). The CFDEM®coupling toolbox allows to expand standard CFD solvers of OpenFOAM®(*) to include a coupling to the DEM code LIGGGHTS®.	18394	C, C++	0	0	12981	5413	no	NO	no	no	no	-	-	-	-	24	master	12/4/2017	no	-	No Mpi
NO	Elemental	https://github.com/elemental/Elemental/releases/tag/v0.87.7	Linear Algebra	Elemental is a modern C++ library for distributed-memory dense and sparse-direct linear algebra, conic optimization, and lattice reduction. The library was initially released in Elemental: A new framework for distributed memory dense linear algebra and absorbed, then greatly expanded upon, the functionality from the sparse-direct solver Clique, which was originally released during a project on Parallel Sweeping Preconditioners.	275609	C++	0	10260	129772	135577	no	yes	no	no	no	distributed	DistMap.cpp[26-35]	dynamic	DistMap.cpp[65-136]	21	0.87.7	2/7/2017	no	-	
NO	Gadget	https://wwwmpa.mpa-garching.mpg.de/gadget/	SPH	GADGET-4 is a massively parallel code for N-body/hydrodynamical cosmological simulations. It is a flexible code that can be applied to a variety of different types of simulations, offering a number of sophisticated simulation algorithms.	12823	C++	57	12063	0	703	no	yes	no	no	no	distributed	begrun.c	dynamic	gravtree.c	30	2	1/5/2005	4	10/01/2020	https://wwwmpa.mpa-garching.mpg.de/gadget4/
NO	hemelb	https://github.com/hemelb-codes/hemelb	SPH	HemeLB uses the lattice Boltzmann method to simulate fluid flow in complex geometries, such as a blood vessel network.	28856	C++	0	0	11353	17503	no	yes	no	no	no	distributed	LatticeData.cc	dynamic	ColloidController.cc	23	main	6/20/2022	no	-	No release versions, still ongoing work
NO	horovod	https://github.com/horovod/horovod/releases	AI	Horovod is a distributed deep learning training framework for TensorFlow, Keras, PyTorch, and Apache MXNet. The goal of Horovod is to make distributed deep learning fast and easy to use.	13313	C++	0	0	9414	3899	no	yes	no	yes	no	distributed	adasum_mpi.cc	dynamic	adasum_mpi.cc	4.3	v0.24.3	4/21/2022	no	-	frequent hot fixes and updates, might be possible for newer versions
CORAL-2	Laghos	https://github.com/CEED/Laghos	SPH	Laghos (LAGrangian High-Order Solver) is a miniapp that solves the time-dependent Euler equations of compressible gas dynamics in a moving Lagrangian frame using unstructured high-order finite element spatial discretization and explicit high-order time-stepping.	8230	C++	0	0	7494	736	no	yes	no	no	no	distributed	laghos_solver.cpp	static	laghos_solver.cpp	0.928	3.1	4/10/2021	no	-	
NO	meshkit	https://bitbucket.org/fathomteam/meshkit.git/src	Mesh generation	MeshKit is an open-source library of mesh generation functionality. MeshKit has general mesh manipulation and generation functions such as Copy, Move, Rotate and Extrude mesh. In addition, new quad mesh and embedded boundary Cartesian mesh algorithm (EBMesh) are developed to be used. Interfaces to several public-domain tetrahedral meshing algorithms (Gmsh, netgen) are also offered.	156659	C++	0	0	92285	64374	no	yes	no	no	no	distributed	different meshes	static	different meshes	46	master-201912	12/9/2019	no	-	
NO	metag_partitioning	https://github.com/ParBLiSS/metag_partitioning	Metagenomic Assembler	Parallel metagenomic assembler designed to handle very large datasets. Program identifies the disconnected subgraphs in the de Bruijn graph, partitions the input dataset and runs a popular assember Velvet independently on the partitions. This software is a high performance version of the khmer library for assembly.	12021	C, C++	0	8113	298	3610	no	yes	no	no	no	distributed	mesh.cpp	static	graph.cpp	9	master-201609	9/29/2016	no	-	
Mantevo	miniXyce	https://github.com/Mantevo/miniXyce	Linear Circuit Simulator	At this time, miniXyce is a simple linear circuit simulator with a basic parser that performs transient analysis on any circuit with resistors (R), inductors (L), capacitors (C), and voltage/current sources. The parser incorporated into this version of miniXyce is a single pass parser, where the netlist is expected to be flat (no hierarchy via subcircuits is allowed). Simulating the system of DAEs generates a nonsymmetric linear problem, which is solved using un-preconditioned GMRES. The time integration method used in miniXyce is backward Euler with a constant time-step. The simulator outputs all the solution variables at each time step in a 'prn' file.	1661	C++	0	0	1446	215	no	yes	no	no	no	distributed	mx_linear_ckt_simulator.cpp	static	mX_linear_ckt_simulator.cpp	0.388	v1.0.0	7/6/2016	no	-	
NO	MITgcm	https://github.com/MITgcm/MITgcm	SPH	it can be used to study both atmospheric and oceanic phenomena; one hydrodynamical kernel is used to drive forward both atmospheric and oceanic models it has a non-hydrostatic capability and so can be used to study both small-scale and large scale processes	369484	C, Fortran	305819	10386	0	53279	no	yes	no	no	no	unknown	-	unknown	-	329	cp68i	4/27/2022	no	-	
NO	MLSL-IntelMLSL	https://github.com/intel/MLSL	AI	Intel(R) Machine Learning Scaling Library (Intel(R) MLSL) is a library providing an efficient implementation of communication patterns used in deep learning.	28139	C, C++	0	9075	6043	13021	no	yes	no	no	no	distributed	mlsl.cpp[765-776]	dynamic	mlsl.cpp[648-682]	113	v2018.2	10/1/2018	v2018.3	4/12/2019	
NO	mxx	https://github.com/patflick/mxx	Template library for MPI	mxx is a C++/C++11 template library for MPI. The main goal of this library is to provide two things: 1. Simplified, efficient, and type-safe C++11 bindings to common MPI operations. 2. A collection of scalable, high-performance standard algorithms for parallel distributed memory architectures, such as sorting.	27391	C++	0	0	7238	20153	no	yes	no	no	no	distributed	benchmark_a2a.cpp[46-72]	static	benchmark_a2a.cpp [46-72]	2.8	master-202204	4/13/2022	no	-	
NO	Nek5000	https://github.com/Nek5000/nekRS/releases/tag/v21.1	CFD	High-order methods have the potential to overcome the current limitations of standard CFD solvers.It features state-of-the-art, scalable algorithms that are fast and efficient on platforms ranging from laptops to the world's fastest computers. Applications span a wide range of fields, including fluid flow, thermal convection, combustion and magnetohydrodynamics.	201433	C, Fortran	180695	15442	0	720	no	yes	no	no	no	distributed	gs.c[3257-3311]	static	vtk_interface.c[194-219]	20.8	v22	05/13/2022	no	-	
NO	phyml	https://github.com/stephaneguindon/phyml	Phylogenetics	PhyML is a software package that uses modern statistical approaches to analyse alignments of nucleotide or amino acid sequences in a phylogenetic framework. The main tool in this package builds phylogenies under the maximum likelihood criterion. It implements a large number of substitution models coupled to efficient options to search the space of phylogenetic tree topologies.	76383	C	0	66832	0	4062	no	yes	no	no	yes	distributed	m4.c	static	utilities.c[201-393]	5.6	v3.3.20190321	3/21/2019	v3.3.20220408	04/08/2022	
NO	PrincetonCBEMDMPI	https://github.com/PrincetonUniversity/PrincetonCBEMDMPI	MD	CBEMD: Parallel Molecular Dynamics Under Various Thermodynamic Ensembles	2785	C++	0	0	2465	320	no	yes	no	no	no	distributed	atom.cpp	static	atom.cpp	34	v20130117	1/17/2013	no	-	

A.2 Processed ControlTable[74]

Suite	Application Nam	Domain	LoC	Programming	FortranLines	CLines	CppLines	C_CPP_H_Lin	OpenMP	MPI	OpenACC	CUDA	OpenCL	DataDistributi on	ProcessLevel	Storage	ReleaseDate	ReleaseYear
LLNL ASC Proxy Apps	Lulesh	CFD	12798	C++	0	0	11296	1502	yes	yes	yes	yes	no	distributed	dynamic	4.4	07-18-2018	2018
Mantevo	miniAMR	PDE	54312	C, C++	0	4850	18017	31445	yes	yes	no	no	no	distributed	static	1.1	11-23-2021	2021
Mantevo	miniMD	MD	17496	C++	0	371	14468	2657	yes	yes	no	no	no	distributed	static	2.6	02-28-2019	2019
Mantevo	miniFE	Finite Element	36812	C, C++	0	4479	3549	28784	yes	yes	no	yes	no	distributed	static	21	11-22-2017	2017
Mantevo	miniSMAC	PDE	11167	Fortran	10984	0	0	183	yes	yes	no	no	no	distributed	static	0.812	10-24-2017	2017
Mantevo	miniTri	Triangle Enumeration	9517	C++	0	0	7779	1738	yes	yes	no	no	no	distributed	dynamic	1.2	07-06-2016	2016
ExaGraph	miniVite	Community Detection - Louvain method	2429	C++	0	0	195	2234	yes	yes	no	no	no	distributed	dynamic	0.148	04-10-2021	2021
Fiber Mini App	ntchemini	MD	6571	Fortran	6571	0	0	0	yes	yes	no	no	no	replicated	static	39	10-31-2014	2014
SPEC MPI 2007	104.milc	Quantum Chronodynamic s	22989	C	0	10424	0	12565	no	yes	no	no	no	distributed	static	0.948	03-16-2007	2007
SPEC MPI 2007	107.leslie3d	CFD	7545	Fortran	7545	0	0	0	no	yes	no	no	no	distributed	static	0.536	04-11-2007	2007
SPEC MPI 2007	122.tachyon	Graphics: Parallel Ray Tracing	10763	C	0	9592	0	1171	no	yes	no	no	no	distributed	dynamic	0.387	02-02-2007	2007
SPEC MPI 2007	126.lammps	MD	7162	C++	0	0	3241	3921	no	yes	no	no	no	distributed	static	11	01-17-2005	2005
SPEC MPI 2007	129.tera_tf	3D Eulerian Hydrodynamics	3665	Fortran	3665	0	0	0	no	yes	no	no	no	replicated	static	0.588	02-05-2007	2007
SPEC MPI 2007	130.socorro	MD	86593	C, Fortran	53199	32842	0	552	no	yes	no	no	no	replicated	static	5.5	02-06-2007	2007
SPEC MPI 2007	132.zeusmp2	CFD	29414	Fortran	29323	91	0	0	no	yes	no	no	no	distributed	static	2.1	09-03-2009	2009
SPEC MPI 2007	137.lu	PDE	3924	Fortran	3924	0	0	0	no	yes	no	no	no	centralized	static	0.532	09-03-2009	2009
SPEC MPI 2007	142.dmilc	MD	11498	C	0	10426	0	1072	no	yes	no	no	no	distributed	static	0.944	03-16-2007	2007
SPEC MPI 2007	143.dleslie	CFD	3483	Fortran	3483	0	0	0	no	yes	no	no	no	replicated	static	0.32	04-11-2007	2007
SPEC MPI 2007	145.lGemsFDTD	CEM	10487	Fortran	10487	0	0	0	no	yes	no	no	no	distributed	dynamic	24	04-20-2009	2009
NO	Adios	I/O componentizati on	255137	C, C++, Fortran	12319	151373	83788	7657	no	yes	no	no	no	distributed	static	26	04-18-2018	2018
Chatterbug	chatterbug	Mimic of Communication Patterns	1056	C++	0	0	1056	0	no	yes	no	no	no	replicated	static	0.156	09-18-2018	2018
NO "Co-design center for Particle Applications (CoPA)"	ExaMiniMD	MD	6149	C++	0	0	1816	4333	no	yes	no	no	no	unknown	unknown	0.556	03-14-2018	2018
CORAL-2	Kripke	MD	5350	C++	0	0	2471	2879	no	yes	no	no	no	distributed	static	0.42	06-14-2019	2019
NO	MACSio	I/O componentizati on	4749	C	0	4705	0	44	no	yes	no	no	no	distributed	static	6.7	10-02-2018	2018
NO	mcb	MC	13334	C++	0	0	9704	3630	yes	yes	no	no	no	replicated	static	1.5	01-06-2014	2014
Mantevo	miniAero	CFD	41570	C++	0	0	11666	29904	no	yes	no	yes	no	unknown	unknown	4.2	07-06-2016	2016
NO	OpenMD	MD	113635	C++	0	0	84362	29273	no	yes	no	no	no	replicated	dynamic	124	08-31-2019	2019
NO	SAMRAI	Structured Adaptive Mesh Refinement	203168	C++	2741	0	150953	49474	yes	yes	no	yes	no	distributed	dynamic	66	09-30-2020	2020
NO	Siesta	MD	162544	Fortran	162418	126	0	0	yes	yes	no	no	no	replicated	static	47	07-19-2018	2018
NO	SimpleMOC	MD	2591	C	0	2396	0	195	yes	yes	no	no	no	replicated	static	0.164	07-31-2018	2018
NO	sphynx	SPH	2152	Fortran	2152	0	0	0	yes	yes	no	no	no	distributed	static	1.5	07-03-2019	2019
NO	splatt	Tensor Factorization	10252	C	0	9135	0	1117	yes	yes	no	no	no	distributed	dynamic	20	09-05-2018	2018
NO	sw4lite-RAJA	SPH	38510	C, Fortran	7874	0	28774	1862	yes	yes	no	yes	no	distributed	static	4.9	03-15-2019	2019
UK Mini-App Consortium	TeaLeaf	Linear heat conduction equation	3989	Fortran	3959	11	0	19	yes	yes	no	no	no	distributed	static	0.344	06-02-2015	2015
NO	thornado_mini	Equation Radiative Transfer	10368	Fortran	10368	0	0	0	yes	yes	no	no	no	distributed	static	16	09-09-2018	2018
NO	Trilinos	Package of Scientific Problem	3828846	C++	175022	379380	1540325	1734119	yes	yes	no	yes	no	distributed	dynamic	792	08-06-2020	2020
NO	tycho2	MD	6341	C++	0	0	4991	1350	yes	yes	no	no	no	distributed	dynamic	182	06-01-2021	2021
NO	vlasiator	MD	44095	C++	0	0	30662	13433	yes	yes	yes	yes	yes	distributed	dynamic	51	04-26-2021	2021
NO	vmd	MD	190943	C, C++	0	10593	151812	28538	yes	yes	yes	yes	yes	distributed	static	22	10-13-2020	2020

NO	WRF	MD	1139122	Fortran	1073300	56279	0	9543	yes	yes	no	no	no	distributed	static	500	07-22-2020	2020
NO	yambo	Time-Dependent Density Functional Theory	99987	C, Fortran	96164	3228	0	595	yes	yes	no	no	no	distributed	static	8.7	07-20-2020	2020
SPEC CPU 2017	510.parest_r	Biomedical Imaging	359012	C++	0	0	106968	252044	no	yes	no	no	no	unknown	unknown	30	01-01-2017	2017
SPEC CPU 2017	520.omnetpp_r	Discrete Event simulation - computer network	85732	C++	0	0	72727	13005	no	yes	no	no	no	unknown	unknown	58	01-01-2017	2017
NO	arbor-0.3	MD	86336	C++	0	0	43506	42830	no	yes	no	yes	no	distributed	dynamic	7.2	01-26-2022	2022
ExMatEx	ASPA-master	Adaptive Samplir	29119	C++	0	0	12956	16163	no	yes	no	no	no	replicated	static	5.5	01-23-2014	2014
NO	Caffe-MPI	GPU Clusters	68871	C++	0	0	46110	22761	no	yes	no	yes	no	distributed	static	23	02-05-2018	2018
NO	Elemental	Linear Algebra	275609	C++	0	10260	129772	135577	no	yes	no	no	no	distributed	dynamic	21	02-07-2017	2017
NO	Gadget	SPH	12823	C++	57	12063	0	703	no	yes	no	no	no	distributed	dynamic	30	01-05-2005	2005
NO	hemelb	SPH	28856	C++	0	0	11353	17503	no	yes	no	no	no	distributed	dynamic	23	06-20-2022	2022
NO	horovod	AI	13313	C++	0	0	9414	3899	no	yes	no	yes	no	distributed	dynamic	4.3	04-21-2022	2022
CORAL-2	Laghos	SPH	8230	C++	0	0	7494	736	no	yes	no	no	no	distributed	static	0.928	04-10-2021	2021
NO	meshkit	Mesh generation	156659	C++	0	0	92285	64374	no	yes	no	no	no	distributed	static	46	12-09-2019	2019
NO	metag_partitioning	Metagenomic Assembler	12021	C, C++	0	8113	298	3610	no	yes	no	no	no	distributed	static	9	09-29-2016	2016
Mantevo	miniXyce	Linear Circuit Simulator	1661	C++	0	0	1446	215	no	yes	no	no	no	distributed	static	0.388	07-06-2016	2016
NO	MITgcm	SPH	369484	C, Fortran	305819	10386	0	53279	no	yes	no	no	no	unknown	unknown	329	04-27-2022	2022
NO	MLSL-IntelMLSL	AI	28139	C, C++	0	9075	6043	13021	no	yes	no	no	no	distributed	dynamic	113	10-01-2018	2018
NO	mxx	Template library for MPI	27391	C++	0	0	7238	20153	no	yes	no	no	no	distributed	static	2.8	04-13-2022	2022
NO	Nek5000	CFD	201433	C, Fortran	180695	15442	0	720	no	yes	no	no	no	distributed	static	20.8	05-13-2022	2022
NO	phyml	Phylogenetics	76383	C	0	66832	0	4062	no	yes	no	no	yes	distributed	static	5.6	03-21-2019	2019
NO	PrincetonCBEMDMI	MD	2785	C++	0	0	2465	320	no	yes	no	no	no	distributed	static	34	01-17-2013	2013

B
Appendix

B.1 Script to get automatic information

```
#!/usr/bin/env python

import sys
import os
import re
import argparse
import subprocess

# Missing OpenMP pragmas - to be decided
# pragma omp single
# pragma omp barrier
# !$omp barrier
# omp ordered
# omp linear
#
#



# Global variables

appname = "t"
printcsv = 0
# loops
openMPLoops = 0
openMPLoopsC = 0
openMPLoopsF = 0

# SIMD
```

```
openMPSimd = 0
openMPSimdLoop = 0

# schedule
openMPSchedule_Implicit = 0
openMPSchedule_NOT_Implicit = 0
openMPScheduleStatic = 0
openMPScheduleGuided = 0
openMPScheduleDynamic = 0
openMPScheduleAuto = 0
openMPScheduleRuntime = 0
openMPScheduleStaticChunk = 0
openMPScheduleGuidedChunk = 0
openMPScheduleDynamicChunk = 0
openMPScheduleAutoChunk = 0

# tasks
openMPTasks = 0
openMPTaskwait = 0
openmMPTaskFinal = 0
openmMPTaskMergeable = 0
openMPTaskPriority = 0
openMPTaskDepend = 0
openMPTaskyield = 0
openMPTaskgroup = 0
openMPTaskloop = 0

# target
openMPTarget = 0
openMPReqRevOfload = 0
openMPDevicetype = 0
openMPMap = 0
openMPDefaultMap = 0

# other
openMPTeams = 0
openMPNoWait = 0
openMPCollapse = 0
openMPSingle = 0
openMPSections = 0
openMPMaster = 0
openMPProc_bind = 0
openMPThreadprivate = 0
```

```
openMPCritical = 0
openMPDeclare = 0

openACCPragmas = 0
CUDASymbols = 0
OpenCLSymbols = 0

c_lines = 0
cpp_lines = 0
c_cpp_header_lines = 0
fortran_lines = 0
total_lines_of_code = 0

# Matches OpenACC in C
openacc_c_re = re.compile(r"(?P<openacc>^[\s]*(\#pragma)[\s]+(acc)
    )")

# Matches CUDA __global__ kernel
cuda_global_kernel_re = re.compile(r"(?P<cuda_kernel>(__global__)
    [\s]+)")
# Matches CUDA __device__ kernel
cuda_device_kernel_re = re.compile(r"(?P<cuda_kernel>(__device__)
    [\s]+)")

# Matches OpenCL __global
opencl_global_re = re.compile(r"(?P<opencl_global>(__global)[\s]+)
    ")
# Matches OpenCL __kernel
opencl_kernel_re = re.compile(r"(?P<cuda_kernel>(__kernel)[\s]+)")

# Matches OpenACC in Fortran
openacc_fortran_re = re.compile(r"(?P<openacc>^[\s]*(\!\$(ACC|acc)
    ))")

#################################
### Regexp for loops OpenMP ###
#################################

# C and C++ openmp_c_re = re.compile(r"(?P<openmp>^[\s]*(\#pragma)
    [\s]+(omp))") openmp_c_re = re.compile(r"(
# ?P<openmp>^[\s]*(\#pragma)[\s]+(omp)[.]*[\s]*\w+[\s]+for)|(\#
    pragma[\s]+omp[\s]+for)", re.IGNORECASE)
```

```
openmp_c_re = re.compile(r"(?P<openmp>\#(.+)?\bpragma\b.+\bomp\b
    .+\bfor\b)", re.IGNORECASE)

# FORTRAN
# openmp_fortran_re = re.compile(r"(?P<openmp>^[\s]*(\!\$(OMP|omp)
    ))")
# openmp_fortran_re = re.compile(r"(?P<openmp>^[\s]*(\!\$OMP[.]*[\
    s]*\w+[\s]+DO)|(\!\$OMP[\s]+DO))", re.IGNORECASE)
openmp_fortran_re = re.compile(r"(?P<openmp>\!\$(.+)?\bomp\b.+\bdo
    \b)", re.IGNORECASE)

################################################
### Regexp for tasks and taskwait OpenMP ###
################################################

# regext all tasks
openmp_tasks_re = re.compile(r"(?P<openmp>\!\$(.+)?\bomp\b.+\btask
    \b|\#(.+)?\bpragma\b.+\bomp\b.+\btask\b)",
                                re.IGNORECASE)

# regex end task fortran
openmp_endTask_re = re.compile(r"(?P<openmp>\!\$(.+)?\bomp\b.+\
    bend\b.+\btask\b)", re.IGNORECASE)

# regex taskwait
openmp_taskwait_re = re.compile(r"(?P<openmp>\!\$(.+)?\bomp\b.+\
    btaskwait\b|\#(.+)?\bpragma\b.+\bomp\b.+\btaskwait\b)",
                                    re.IGNORECASE)

# regex task final
openmp_taskfinal_re = re.compile(
    r"(?P<openmp>\!\$(.+)?\bomp\b.+\btask\b.+\bfinal\b|\#(.+)?\
        bpragma\b.+\bomp\b.+\btask\b.+\bfinal\b)", re.IGNORECASE)

# regex task mergeable
openmp_taskmergeable_re = re.compile(
    r"(?P<openmp>\!\$(.+)?\bomp\b.+\btask\b.+\bmergeable\b|\#(.+)
        ?\bpragma\b.+\bomp\b.+\btask\b.+\bmergeable\b)",
    re.IGNORECASE)

# regex end task priority
openmp_taskpriority_re = re.compile(
```

```
    r"(?P<openmp>\!\$(.+)?\bomp\b.+\btask\b.+\bpriority\b|\#(.+)?\
        bpragma\b.+\bomp\b.+\btask\b.+\bpriority\b)",
    re.IGNORECASE)

# regex task depend
openmp_taskdepend_re = re.compile(
    r"(?P<openmp>\!\$(.+)?\bomp\b.+\btask\b.+\bdepend\b|\#(.+)?\
        bpragma\b.+\bomp\b.+\btask\b.+\bdepend\b)",
    re.IGNORECASE)

# regex taskyield
openmp_taskyield_re = re.compile(
    r"(?P<openmp>\!\$(.+)?\bomp\b.+\btaskyield\b|\#(.+)?\bpragma\b
        .+\bomp\b.+\btaskyield\b)", re.IGNORECASE)

# regex taskgroup and taskloop
openmp_tasksgroup_re = re.compile(
    r"(?P<openmp>\!\$(.+)?\bomp\b.+\btaskgroup\b|\#(.+)?\bpragma\b
        .+\bomp\b.+\btaskgroup\b)", re.IGNORECASE)
openmp_tasksloop_re = re.compile(r"(?P<openmp>\!\$(.+)?\bomp\b.+\
    btaskloop\b|\#(.+)?\bpragma\b.+\bomp\b.+\btaskloop\b)",
                                 re.IGNORECASE)
# regex END taskgroup and taskloop
openmp_endTaskgroup_re = re.compile(r"(?P<openmp>\!\$(.+)?\bomp\b
    .+\bend\b.+\btaskgroup\b)", re.IGNORECASE)
openmp_endTaskloop_re = re.compile(r"(?P<openmp>\!\$(.+)?\bomp\b
    .+\bend\b.+\btaskloop\b)", re.IGNORECASE)

##########################################
### Regexp for target - device OpenMP ###
##########################################

openmp_target_re = re.compile(r"(?P<openmp>\!\$(.+)?\bomp\b.+\
    btarget\b|\#(.+)?\bpragma\b.+\bomp\b.+\btarget\b)",
                              re.IGNORECASE)

openmp_endTarget_re = re.compile(r"(?P<openmp>\!\$(.+)?\bomp\b.+\
    bend\b.+\btarget\b)", re.IGNORECASE)

openmp_recrevofload_re = re.compile(
    r"(?P<openmp>\!\$(.+)?\bomp\b.+\brequires\b.+\breverse_offload
        \b|\#(.+)?\bpragma\b.+\bomp\b.+\brequires\b."
    r"+\breverse_offload\b)",re.IGNORECASE)
```

```
openmp_devicetype_re = re.compile(
    r"(?P<openmp>\!\$(.+)?\bomp\b.+\bdevice_type\b|\#(.+)?\bpragma
        \b.+\bomp\b.+\bdevice_type\b)", re.IGNORECASE)

openmp_map_re = re.compile(r"(?P<openmp>\!\$(.+)?\bomp\b.+\bmap\b
    |\#(.+)?\bpragma\b.+\bomp\b.+\bmap\b)", re.IGNORECASE)
openmp_defaultmap_re = re.compile(
    r"(?P<openmp>\!\$(.+)?\bomp\b.+\bdefaultmap\b|\#(.+)?\bpragma\
        b.+\bomp\b.+\bdefaultmap\b)", re.IGNORECASE)

#####################################
### Regexp for declare OpenMP ###
#####################################

openmp_declare_re = re.compile(r"(?P<openmp>\!\$(.+)?\bomp\b.+\
    bdeclare\b|\#(.+)?\bpragma\b.+\bomp\b.+\bdeclare\b)",
                                    re.IGNORECASE)

#####################################
### Regexp for schedule OpenMP ###
#####################################
openmp_schedule_static_re = re.compile(r'(?P<openmp>schedule\((\s
    +)?static(\s+)?\))', re.IGNORECASE)
openmp_schedule_guided_re = re.compile(r'(?P<openmp>schedule\((\s
    +)?guided(\s+)?\))', re.IGNORECASE)
openmp_schedule_dynamic_re = re.compile(r'(?P<openmp>schedule\((\s
    +)?dynamic(\s+)?\))', re.IGNORECASE)
openmp_schedule_auto_re = re.compile(r'(?P<openmp>schedule\((\s+)?
    auto(\s+)?\))', re.IGNORECASE)
openmp_schedule_runtime_re = re.compile(r'(?P<openmp>schedule\((\s
    +)?runtime(\s+)?\))', re.IGNORECASE)
openmp_schedule_static_chunk_re = re.compile(r'(?P<openmp>schedule
    \((\s+)?static,(\s+)?\w+(\s+)?\))', re.IGNORECASE)
openmp_schedule_guided_chunk_re = re.compile(r'(?P<openmp>schedule
    \((\s+)?guided,(\s+)?\w+(\s+)?\))', re.IGNORECASE)
openmp_schedule_dynamic_chunk_re = re.compile(r'(?P<openmp>
    schedule\((\s+)?dynamic,(\s+)?\w+(\s+)?\))', re.IGNORECASE)
openmp_schedule_auto_chunk_re = re.compile(r'(?P<openmp>schedule
    \((\s+)?auto,(\s+)?\w+(\s+)?\))', re.IGNORECASE)

#############################
### Regexp OpenMP teams ###
```

```
##############################

openmp_teams_re = re.compile(r"(?P<openmp>\!\$(.+)?\bomp\b.+\
    bteams\b|\#(.+)?\bpragma\b.+\bomp\b.+\bteams\b)",
                                re.IGNORECASE)

##############################
### Regexp OpenMP nowait ###
##############################

openmp_nowait_re = re.compile(r"(?P<openmp>\!\$(.+)?\bomp\b.+\
    bnowait\b|\#(.+)?\bpragma\b.+\bomp\b.+\bnowait\b)",
                                re.IGNORECASE)

##################################
### Regexp OpenMP collapse ###
##################################

openmp_collapse_re = re.compile(r"(?P<openmp>\!\$(.+)?\bomp\b.+\
    bcollapse\b|\#(.+)?\bpragma\b.+\bomp\b.+\bcollapse\b)",
                                re.IGNORECASE)

# ############################
# ### Regexp OpenMP simd ###
# ############################

openmp_simd_re = re.compile(r"(?P<openmp>\!\$(.+)?\bomp\b.+\bsimd\
    b|\#(.+)?\bpragma\b.+\bomp\b.+\bsimd\b)",
                                re.IGNORECASE)

openmp_for_simd_re = re.compile(
    r"(?P<openmp>\!\$(.+)?\bomp\b.+\bdo\b.+\bsimd\b|\#(.+)?\
        bpragma\b.+\bomp\b.+\bfor\b.+\bsimd\b)", re.IGNORECASE)

##############################
### Regexp OpenMP single ###
##############################

openmp_single_re = re.compile(r"(?P<openmp>\!\$(.+)?\bomp\b.+\
    bsingle\b|\#(.+)?\bpragma\b.+\bomp\b.+\bsingle\b)",
                                re.IGNORECASE)

##############################
```

```
### Regexp OpenMP sections ###
#################################

openmp_sections_re = re.compile(r"(?P<openmp>\!\$(.+)?\bomp\b.+\
    bsection\b|\#(.+)?\bpragma\b.+\bomp\b.+\bsection\b)",
                                    re.IGNORECASE)

##############################
### Regexp OpenMP master ###
##############################

openmp_master_re = re.compile(r"(?P<openmp>\!\$(.+)?\bomp\b.+\
    bmaster\b|\#(.+)?\bpragma\b.+\bomp\b.+\bmaster\b)",
                                   re.IGNORECASE)

################################
### Regexp OpenMP proc_bind ###
################################

openmp_proc_bind_re = re.compile(
    r"(?P<openmp>\!\$(.+)?\bomp\b.+\bproc_bind\b|\#(.+)?\bpragma\b
        .+\bomp\b.+\bproc_bind\b)", re.IGNORECASE)

###################################
### Regexp OpenMP threadprived ###
###################################

openmp_threadprivate_re = re.compile(
    r"(?P<openmp>\!\$(.+)?\bomp\b.+\bthreadprivate\b|\#(.+)?\
        bpragma\b.+\bomp\b.+\bthreadprivate\b)", re.IGNORECASE)

################################
### Regexp OpenMP critical ###
################################

openmp_critical_re = re.compile(r"(?P<openmp>\!\$(.+)?\bomp\b.+\
    bcritical\b|\#(.+)?\bpragma\b.+\bomp\b.+\bcritical\b)",
                                    re.IGNORECASE)

########################
### Regexp for Cloc ###
########################
```

```
cloc_c_re = re.compile(r"(^C[\s]+[0-9]+[\s]+[0-9]+[\s]+[0-9]+[\s
    ]+[0-9]+)")
cloc_cpp_re = re.compile(r"(^C\+\+[\s]+[0-9]+[\s]+[0-9]+[\s
    ]+[0-9]+[\s]+[0-9]+)")
cloc_c_cpp_header_re = re.compile(r"(^(C\/C\+\+ Header)[\s
    ]+[0-9]+[\s]+[0-9]+[\s]+[0-9]+[\s]+[0-9]+)")
cloc_fortran_re = re.compile(r"(^(Fortran [0-9]+)[\s]+[0-9]+[\s
    ]+[0-9]+[\s]+[0-9]+[\s]+[0-9]+)")

############################
#### Control variables ####
############################

outputFileName = None
inputPath = "./"
verbose = False
# Max size of an MPI Call in terms of lines
maxBufferSize = 10


def checkOpenMP(line):
    # loops
    global openMPLoopsF
    global openMPLoopsC
    global openMPLoops
    # Simd
    global openMPSimd
    global openMPSimdLoop
    # schedule
    global openMPScheduleStatic
    global openMPScheduleGuided
    global openMPScheduleDynamic
    global openMPScheduleAuto
    global openMPScheduleRuntime
    global openMPScheduleStaticChunk
    global openMPScheduleGuidedChunk
    global openMPScheduleDynamicChunk
    global openMPScheduleAutoChunk
    # tasks
    global openMPTasks
    global openMPTaskwait
    global openmMPTaskMergeable
    global openmMPTaskFinal
```

```
    global openMPTaskPriority
    global openMPTaskDepend
    global openMPTaskyield
    global openMPTaskgroup
    global openMPTaskloop
    # target - devices
    global openMPTarget
    global openMPDevicetype
    global openMPReqRevOfload
    global openMPMap
    global openMPDefaultMap
    # others
    global openMPTeams
    global openMPNoWait
    global openMPCollapse
    global openMPSingle
    global openMPSections
    global openMPMaster
    global openMPProc_bind
    global openMPThreadprivate
    global openMPCritical
    global openMPDeclare

    # Matching OpenMP Loops
    # C C++
    loopsC = openmp_c_re.search(line)
    # FORTRAN
    loopsF = openmp_fortran_re.search(line)

    # SIMD
    simdConstruct = openmp_simd_re.search(line)
    simdClause = openmp_for_simd_re.search(line)

    # Matching OpenMP Tasks
    endTasksConstruct = openmp_endTask_re.search(line)
    tasksConstruct = openmp_tasks_re.search(line)
    taskwaitConstruct = openmp_taskwait_re.search(line)
    taskfinalClause = openmp_taskfinal_re.search(line)
    taskmergeableClause = openmp_taskmergeable_re.search(line)
    taskpriorityClause = openmp_taskpriority_re.search(line)
    taskdependClause = openmp_taskdepend_re.search(line)
    taskyieldClause = openmp_taskyield_re.search(line)
```

```
    taskGroupConstruct = openmp_tasksgroup_re.search(line)
    taskLoopConstruct = openmp_tasksloop_re.search(line)

    endTaskGroupConstruct = openmp_endTaskgroup_re.search(line)
    endTaskLoopConstruct = openmp_endTaskloop_re.search(line)

    # Matching OpenMP Target - Devices

    targetConstruct = openmp_target_re.search(line)
    endTargetConstruct = openmp_endTarget_re.search(line)

    recrevofloadDirectiveClause = openmp_recrevofload_re.search(
        line)
    devicetypeClause = openmp_devicetype_re.search(line)

    mapClause = openmp_master_re.search(line)
    defaultmapClause = openmp_defaultmap_re.search(line)

    # Matching OpenMP SCHEDULE clauses
    scstatic = openmp_schedule_static_re.search(line)
    scguided = openmp_schedule_guided_re.search(line)
    scdynamic = openmp_schedule_dynamic_re.search(line)
    scauto = openmp_schedule_auto_re.search(line)
    scruntime = openmp_schedule_runtime_re.search(line)
    scstaticChunk = openmp_schedule_static_chunk_re.search(line)
    scguidedChunk = openmp_schedule_guided_chunk_re.search(line)
    scdynamicChunk = openmp_schedule_dynamic_chunk_re.search(line)
    scautoChunk = openmp_schedule_auto_chunk_re.search(line)

    # Matching OpenMP teams clauses
    teamClause = openmp_teams_re.search(line)

    # Matching OpenMP nowait
    nowaitClause = openmp_nowait_re.search(line)

    # Matching OpenMP collapse
    collapseClause = openmp_collapse_re.search(line)

    # Matching OpenMP single
    singleClause = openmp_single_re.search(line)

    # Matching OpenMP sections
    sectionClause = openmp_sections_re.search(line)
```

```
# Matching OpenMP master
masterClause = openmp_master_re.search(line)

# Matching OpenMP proc_bind
proc_bindClause = openmp_proc_bind_re.search(line)

# Matching OpenMP threadprivate
threadprivateClause = openmp_threadprivate_re.search(line)

# Matching OpenMP critical
criticalClause = openmp_critical_re.search(line)

# Matching OpenMP declare
declareDirective = openmp_declare_re.search(line)

# Count loops
if loopsC != None:
    openMPLoopsC = openMPLoopsC + 1

if loopsF != None:
    openMPLoopsF = openMPLoopsF + 1

# SIMD
if simdConstruct != None:
    openMPSimd = openMPSimd + 1
if simdClause != None:
    openMPSimdLoop = openMPSimdLoop + 1

# Count tasks
if tasksConstruct != None:
    openMPTasks = openMPTasks + 1

if endTasksConstruct != None:
    openMPTasks = openMPTasks - 1

# Count tasks clauses

if taskwaitConstruct != None:
    openMPTaskwait = openMPTaskwait + 1

if taskfinalClause != None:
    openmMPTaskFinal = openmMPTaskFinal + 1
```

```
    if taskmergeableClause != None:
        openmMPTaskMergeable = openmMPTaskMergeable + 1

    if taskpriorityClause != None:
        openMPTaskPriority = openMPTaskPriority + 1

    if taskdependClause != None:
        openMPTaskDepend = openMPTaskDepend + 1

    if taskyieldClause != None:
        openMPTaskyield = openMPTaskyield + 1

    # Count taskgroup and taskloop

    if taskGroupConstruct != None:
        openMPTaskgroup = openMPTaskgroup + 1

    if endTaskGroupConstruct != None:
        openMPTaskgroup = openMPTaskgroup - 1

    if taskLoopConstruct != None:
        openMPTaskloop = openMPTaskloop + 1

    if endTaskLoopConstruct != None:
        openMPTaskloop = openMPTaskloop - 1

    # Count target - devices

    if targetConstruct != None:
        openMPTarget = openMPTarget + 1

    if endTargetConstruct != None:
        openMPTarget = openMPTarget - 1

    if recrevofloadDirectiveClause != None:
        openMPReqRevOfload = openMPReqRevOfload + 1

    if devicetypeClause != None:
        openMPDevicetype = openMPDevicetype + 1

    if mapClause != None:
        openMPMap = openMPMap + 1
```

```
    if defaultmapClause != None:
        openMPDefaultMap = openMPDefaultMap + 1

    # Count schedule
    if scstatic != None:
        openMPScheduleStatic = openMPScheduleStatic + 1
    if scguided != None:
        openMPScheduleGuided = openMPScheduleGuided + 1
    if scdynamic != None:
        openMPScheduleDynamic = openMPScheduleDynamic + 1
    if scauto != None:
        openMPScheduleAuto = openMPScheduleAuto + 1
    if scruntime != None:
        openMPScheduleRuntime = openMPScheduleRuntime + 1
    if scstaticChunk != None:
        openMPScheduleStaticChunk = openMPScheduleStaticChunk + 1
    if scguidedChunk != None:
        openMPScheduleGuidedChunk = openMPScheduleGuidedChunk + 1
    if scdynamicChunk != None:
        openMPScheduleDynamicChunk = openMPScheduleDynamicChunk +
            1
    if scautoChunk != None:
        openMPScheduleAutoChunk = openMPScheduleAutoChunk + 1

    # Count teams
    if teamClause != None:
        openMPTeams = openMPTeams + 1

    # Count nowait
    if nowaitClause != None:
        openMPNoWait = openMPNoWait + 1

    # Count collapse
    if collapseClause != None:
        openMPCollapse = openMPCollapse + 1

    # Count single
    if singleClause != None:
        openMPSingle = openMPSingle + 1

    # Count sections
    if sectionClause != None:
```

```
        openMPSections = openMPSections + 1

    # Count master
    if masterClause != None:
        openMPMaster = openMPMaster + 1

    # Count proc_bind
    if proc_bindClause != None:
        openMPProc_bind = openMPProc_bind + 1

    # Count threadprivate
    if threadprivateClause != None:
        openMPThreadprivate = openMPThreadprivate + 1

    # Count critical
    if criticalClause != None:
        openMPCritical = openMPCritical + 1

    # Count declare
    if declareDirective != None:
        openMPDeclare = openMPDeclare + 1

# Main


def main():
    parseArgs()
    # input = sys.argv[1]
    input = inputPath
    printOut(["Analyzing", input, "..."])

    # Check if input exist
    if os.path.exists(input):

        # Traver tree if input is a directory
        if os.path.isdir(input):

            # This loop traverses the entire directory tree
            rootDir = input
            for dirName, subdirList, fileList in os.walk(rootDir):
                for fname in fileList:
                    # Full path of the file
                    filePath = dirName + "/" + fname
```

```
                    printOut([filePath])
                    analyzeFile(filePath)

        else: ## this is a file
            # print input
            analyzeFile(input)
    else:
        print("Error:", input, "does not exist")
        exit()

    # cloc statistics
    getClocStatistics(input)

    # openMPLoopsF is divided by 2 since in FORTRAN it always
        appear twice for every loop e.g. DO ... END DO
    global openMPLoops, openMPSchedule_Implicit,
        openMPSchedule_NOT_Implicit
    openMPLoops = (openMPLoopsF / 2) + openMPLoopsC

    # calculate how many loops use the schedule clause.
    openMPSchedule_NOT_Implicit = openMPScheduleStatic +
        openMPScheduleGuided + openMPScheduleDynamic + \
                                openMPScheduleRuntime +
                                    openMPScheduleStaticChunk +
                                    openMPScheduleGuidedChunk +
                                    \
                                openMPScheduleDynamicChunk +
                                    openMPScheduleAuto +
                                    openMPScheduleAutoChunk
    # calculate how many loops do not use the schedule clause
    openMPSchedule_Implicit = openMPLoops -
        openMPSchedule_NOT_Implicit

    # Print results
    global printcsv
    if printcsv == 0:
        printResults()
    else:
        printResultsCSV()


# Helper Functions
```

```
def parseArgs():
    global inputPath, outputFileName, verbose, appname, printcsv

    parser = argparse.ArgumentParser()
    parser.add_argument("path", type=str,
                        help="file or directory to analyze")
    parser.add_argument("-a", "--app", help="name of application",
        type=str)
    parser.add_argument("-c", "--csv", help="print in CSV", type=
       str)
    parser.add_argument("-o", "--output", help="name of output
       file", type=str)
    parser.add_argument("-v", "--verbose", help="print what the
       script does", action="store_true")
    args = parser.parse_args()

    inputPath = args.path

    if args.output:
        outputFileName = args.output
    if args.verbose:
        verbose = True
    if args.app:
        appname = args.app
    if args.csv:
        printcsv = args.csv


# Wrapper of print function
# out: a list of strings, e.g., ["hello", "world"]
def printOut(outList):
    global verbose
    if verbose:
        print(" ".join(outList))


# This is the main function to analyze a file
def analyzeFile(filePath):
    fd = open(filePath, 'r')
    fileLines = fd.readlines()
    for i in range(len(fileLines)):
        # print("test"+fileLines[i])
        # Check for languages
```

```
            checkOpenMP(fileLines[i])
            checkOpenACC(fileLines[i])
            checkCUDA(fileLines[i])
            checkOpenCL(fileLines[i])
        fd.close()


def printResultsCSV():
    global appname, outputFileName, openACCPragmas, CUDAkernels,
        c_lines, cpp_lines, c_cpp_header_lines, fortran_lines,\
         total_lines_of_code
    out = ""
    if outputFileName != None:
        if not os.path.exists(outputFileName):
            out = "app, \
OPENMP_loop, \
OPENMP_task, \
OPENMP_taskloop, \
OPENMP_taskgroup, \
OPENMP_taskwait, \
OPENMP_taskfinal, \
OPENMP_taskmergeable, \
OPENMP_taskpriority, \
OPENMP_taskdepend, \
OPENMP_taskyield, \
OPENMP_target, \
OPENMP_req_rev_offload, \
OPENMP_device_type, \
OPENMP_map, \
OPENMP_defaultmap, \
OPENMP_declare, \
OPENMP_scimplicit, \
OPENMP_scstatic, \
OPENMP_scguided, \
OPENMP_scdynamic, \
OPENMP_scauto, \
OPENMP_scruntime, \
OPENMP_scstatic_chunk, \
OPENMP_scguided_chunk, \
OPENMP_scdynamic_chunk, \
OPENMP_scauto_chunk, \
OPENMP_teams, \
OPENMP_nowait, \
```

```
OPENMP_collapse, \
OPENMP_single, \
OPENMP_sections, \
OPENMP_master, \
OPENMP_proc_bind, \
OPENMP_threadprivate, \
OPENMP_critical, \
OPENMP_simd, \
OPENMP_simd_loop, \
OPENACC, \
CUDA, \
OPENCL, \
C_LINES, \
CPP_LINES, \
C_CPP_H_LINES, \
FORTRAN_LINES, \
LINES_OF_CODE \n"

    printOut(["*** OpenMP Usage ***"])
    # We save output into a string

    # for k in MPI_CALLS_TABLE.keys():
    #     line = "  " + "\"" + k + "\"" + ": " + str(
        MPI_CALLS_TABLE[k]) + ",\n"
    #     out = out + line

    out = out + str(appname) + ', ' # App
    out = out + str(openMPLoops) + ', ' # All OMP loops construct
    out = out + str(openMPTasks) + ', ' # All OMP tasks construct
    out = out + str(openMPTaskloop) + ', ' # All OMP taskloops
        construct
    out = out + str(openMPTaskgroup) + ', ' # All OMP taskgroup
        construct
    out = out + str(openMPTaskwait) + ', ' # All OMP taskwait
        construct
    out = out + str(openmMPTaskFinal) + ', ' # All OMP taskfinal
        clause
    out = out + str(openmMPTaskMergeable) + ', ' # All OMP task
        mergeable clause
    out = out + str(openMPTaskPriority) + ', ' # All OMP task
        pritority
    out = out + str(openMPTaskDepend) + ', ' # All OMP task
        depend
```

```
out = out + str(openMPTaskyield) + ', ' # All OMP taskyield
out = out + str(openMPTarget) + ', ' # All OMP target
out = out + str(openMPReqRevOfload) + ', '
out = out + str(openMPDevicetype) + ', '
out = out + str(openMPMap) + ', '
out = out + str(openMPDefaultMap) + ', '
out = out + str(openMPDeclare) + ', '
out = out + str(openMPSchedule_Implicit) + ', ' # Number of
    loops without schedule clause
out = out + str(openMPScheduleStatic) + ', ' # Number of
    loops with static schedule clause
out = out + str(openMPScheduleGuided) + ', ' # Number of
    loops with guided schedule clause
out = out + str(openMPScheduleDynamic) + ', ' # Number of
    loops with dynamic schedule clause
out = out + str(openMPScheduleAuto) + ', ' # Number of loops
    with auto schedule clause
out = out + str(openMPScheduleRuntime) + ', ' # Number of
    loops with runtime schedule clause
out = out + str(
    openMPScheduleStaticChunk) + ', ' # Number of loops with
        static schedule clause with chunksize paramenter
out = out + str(
    openMPScheduleGuidedChunk) + ', ' # Number of loops with
        guided schedule clause with chunksize paramenter
out = out + str(
    openMPScheduleDynamicChunk) + ', ' # Number of loops with
         dynamic schedule clause with chunksize paramenter
out = out + str(
    openMPScheduleAutoChunk) + ', ' # Number of loops with
        auto schedule clause with chunksize paramenter
out = out + str(openMPTeams) + ', ' # All OMP teams
out = out + str(openMPNoWait) + ', ' # All OMP nowait
out = out + str(openMPCollapse) + ', ' # All OMP collapse
out = out + str(openMPSingle) + ', ' # All OMP single
out = out + str(openMPSections) + ', ' # All OMP sections
out = out + str(openMPMaster) + ', ' # All OMP master
out = out + str(openMPProc_bind) + ', ' # All OMP proc_bind
out = out + str(openMPThreadprivate) + ', ' # All OMP thread
    private
out = out + str(openMPCritical) + ', ' # All OMP critical
out = out + str(openMPSimd) + ', '
out = out + str(openMPSimdLoop) + ', '
```

```
    out = out + str(openACCPragmas) + ', '
    out = out + str(CUDASymbols) + ', '
    out = out + str(OpenCLSymbols) + ', '
    out = out + str(c_lines) + ', '
    out = out + str(cpp_lines) + ', '
    out = out + str(c_cpp_header_lines) + ', '
    out = out + str(fortran_lines) + ', '

    # For consistency, let's make LINES_OF_CODE the last one
    out = out + str(total_lines_of_code) + '\n'
    # out = out + "}"

    print(out)
    if outputFileName != None:
        saveResultsCSV(out)


def saveResultsCSV(out):
    fd = open(outputFileName, 'a')
    fd.write(out)
    fd.close()


# Print results to stdout
def printResults():
    global outputFileName, openACCPragmas, CUDAkernels, c_lines,
        cpp_lines, c_cpp_header_lines, \
        fortran_lines, total_lines_of_code

    printOut(["*** OpenMP Usage ***"])

    # We save output into a string
    out = "{\n"
    # for k in MPI_CALLS_TABLE.keys():
    #     line = "  " + "\"" + k + "\"" + ": " + str(
        MPI_CALLS_TABLE[k]) + ",\n"
    #     out = out + line

    out = out + '  "OPENMP_loop": ' + str(openMPLoops) + ',\n'  #
        All OMP loops construct
    out = out + '  "OPENMP_task": ' + str(openMPTasks) + ',\n'  #
        All OMP tasks construct
```

```
out = out + '  "OPENMP_taskloop": ' + str(openMPTaskloop) +
    ',\n'  # All OMP taskloops construct
out = out + '  "OPENMP_taskgroup": ' + str(openMPTaskgroup) +
    ',\n'  # All OMP taskgroup construct
out = out + '  "OPENMP_taskwait": ' + str(openMPTaskwait) +
    ',\n'  # All OMP taskwait construct
out = out + '  "OPENMP_taskfinal": ' + str(openmMPTaskFinal) +
     ',\n'  # All OMP taskfinal clause
out = out + '  "OPENMP_taskmergeable": ' + str(
    openmMPTaskMergeable) + ',\n'  # All OMP task mergeable
    clause
out = out + '  "OPENMP_taskpriority": ' + str(
    openMPTaskPriority) + ',\n'  # All OMP task pritority
out = out + '  "OPENMP_taskdepend": ' + str(openMPTaskDepend)
    + ',\n'  # All OMP task depend
out = out + '  "OPENMP_taskyield": ' + str(openMPTaskyield) +
    ',\n'  # All OMP taskyield
out = out + '  "OPENMP_target": ' + str(openMPTarget) + ',\n'
     # All OMP target
out = out + '  "OPENMP_req_rev_offload": ' + str(
    openMPReqRevOfload) + ',\n'
out = out + '  "OPENMP_device_type": ' + str(openMPDevicetype)
     + ',\n'
out = out + '  "OPENMP_map": ' + str(openMPMap) + ',\n'
out = out + '  "OPENMP_defaultmap": ' + str(openMPDefaultMap)
    + ',\n'
out = out + '  "OPENMP_declare": ' + str(openMPDeclare) + ',\n
    '
out = out + '  "OPENMP_scimplicit": ' + str(
     openMPSchedule_Implicit) + ',\n'  # Number of loops
         without schedule clause
out = out + '  "OPENMP_scstatic": ' + str(
     openMPScheduleStatic) + ',\n'  # Number of loops with
         static schedule clause
out = out + '  "OPENMP_scguided": ' + str(
     openMPScheduleGuided) + ',\n'  # Number of loops with
         guided schedule clause
out = out + '  "OPENMP_scdynamic": ' + str(
     openMPScheduleDynamic) + ',\n'  # Number of loops with
         dynamic schedule clause
out = out + '  "OPENMP_scauto": ' + str(openMPScheduleAuto) +
    ',\n'  # Number of loops with dynamic schedule clause
out = out + '  "OPENMP_scruntime": ' + str(
```

```
    openMPScheduleRuntime) + ',\n'  # Number of loops with
        runtime schedule clause
out = out + '  "OPENMP_scstatic_chunk": ' + str(
    openMPScheduleStaticChunk) + ',\n'  # Number of loops with
         static schedule clause with chunksize paramenter
out = out + '  "OPENMP_scguided_chunk": ' + str(
    openMPScheduleGuidedChunk) + ',\n'  # Number of loops with
         guided schedule clause with chunksize paramenter
out = out + '  "OPENMP_scdynamic_chunk": ' + str(
    openMPScheduleDynamicChunk) + ',\n'  # Number of loops
        with dynamic schedule clause with chunksize paramenter
out = out + '  "OPENMP_scauto_chunk": ' + str(
    openMPScheduleAutoChunk) + ',\n'  # Number of loops with
        dynamic schedule clause with chunksize paramenter
out = out + '  "OPENMP_teams": ' + str(openMPTeams) + ',\n'  #
    All OMP teams
out = out + '  "OPENMP_nowait": ' + str(openMPNoWait) + ',\n'
    # All OMP nowait
out = out + '  "OPENMP_collapse": ' + str(openMPCollapse) +
    ',\n'  # All OMP collapse
out = out + '  "OPENMP_single": ' + str(openMPSingle) + ',\n'
    # All OMP single
out = out + '  "OPENMP_sections": ' + str(openMPSections) +
    ',\n'  # All OMP sections
out = out + '  "OPENMP_master": ' + str(openMPMaster) + ',\n'
    # All OMP master
out = out + '  "OPENMP_proc_bind": ' + str(openMPProc_bind) +
    ',\n'  # All OMP proc_bind
out = out + '  "OPENMP_threadprivate": ' + str(
   openMPThreadprivate) + ',\n'  # All OMP thread private
out = out + '  "OPENMP_critical": ' + str(openMPCritical) +
    ',\n'  # All OMP critical
out = out + '  "OPENMP_simd": ' + str(openMPSimd) + ',\n'  #
   All OMP critical
out = out + '  "OPENMP_loop_simd": ' + str(openMPSimdLoop) +
    ',\n'  # All OMP critical
out = out + '  "OPENACC": ' + str(openACCPragmas) + ',\n'
out = out + '  "CUDA": ' + str(CUDASymbols) + ',\n'
out = out + '  "OPENCL": ' + str(OpenCLSymbols) + ',\n'

out = out + '  "C_LINES": ' + str(c_lines) + ',\n'
out = out + '  "CPP_LINES": ' + str(cpp_lines) + ',\n'
```

```
    out = out + '  "C_CPP_H_LINES": ' + str(c_cpp_header_lines) +
        ',\n'
    out = out + '  "FORTRAN_LINES": ' + str(fortran_lines) + ',\n'

    # For consistency, let's make LINES_OF_CODE the last one
    out = out + '  "LINES_OF_CODE": ' + str(total_lines_of_code) +
         '\n'
    out = out + "}"

    print(out)
    if outputFileName != None:
        saveResults(out)


# Save results into a file
def saveResults(out):
    fd = open(outputFileName, 'w')
    fd.write(out)
    fd.close()


# Language detection

def checkOpenACC(line):
    global openACCPragmas

    # Matching C OpenACC
    result1 = openacc_c_re.search(line)
    # Matching Fortran OpenACC
    result2 = openacc_fortran_re.search(line)

    if result1 != None or result2 != None:
        openACCPragmas = openACCPragmas + 1


def checkCUDA(line):
    global CUDASymbols

    result1 = cuda_global_kernel_re.search(line)
    result2 = cuda_device_kernel_re.search(line)

    if result1 != None or result2 != None:
        CUDASymbols = CUDASymbols + 1
```

```
def checkOpenCL(line):
    global OpenCLSymbols

    result1 = opencl_global_re.search(line)
    result2 = opencl_kernel_re.search(line)

    if result1 != None or result2 != None:
        OpenCLSymbols = OpenCLSymbols + 1


# Cloc calling and parsing

def getClocStatistics(input):
    global c_lines, cpp_lines, c_cpp_header_lines, fortran_lines,
        total_lines_of_code

    printOut(["Calling cloc..."])
    dir_path = os.path.dirname(os.path.realpath(__file__))
    cmd = [dir_path + "/cloc", input]
    cmdOutput = subprocess.check_output(cmd)

    for line in cmdOutput.split("\n"):
        test_c = cloc_c_re.search(line)
        test_cpp = cloc_cpp_re.search(line)
        test_c_cpp_header = cloc_c_cpp_header_re.search(line)
        test_fortran = cloc_fortran_re.search(line)

        if test_c != None:
            c_lines = int(line.split()[-1:][0])
        elif test_cpp != None:
            cpp_lines = int(line.split()[-1:][0])
        elif test_c_cpp_header != None:
            c_cpp_header_lines = int(line.split()[-1:][0])
        elif test_fortran != None:
            fortran_lines = fortran_lines + int(line.split()
                [-1:][0])
        elif "SUM:" in line:
            total_lines_of_code = int(line.split()[-1:][0])


main()
```