MuCoSim: Analysis of a Compressible Lattice Boltzmann Solver (CLBM); Focusing Energy and Power Consumption

Table of Contents

Note: This PDF file was generated by using pandoc. To see the .md version of this file please visit git@gitlab.cs.fau.de:yh54ojyn/mucosim.git

1. Project Description

1.1 General Info

- Name : **Sudesh Rathnayake**
- Marticulation Number : **22849910**
- Email : sudesh.rathnayake@fau.de

1.2 Application Info

Code: CLBM
URL:git@gitlab.cs.fau.de:yh54ojyn/mucosim.git

The Compressible Lattice Boltzmann (CLBM) solver is aimed at simulating supersonic flows to observe the strong shock formations in compressible media.

This solver was developed for a collaborative master's thesis project between the departments of system simulation at Friedrich-Alexander-Universität Erlangen-Nürnberg and Fraunhofer Ernst-Mach-Institut.
Most of the time, the Lattice Boltzmann Method (LBM) adopts a structured grid in practice. A structured grid is represented by a regular grid where points on the grid are updated together, and it has high spatial locality [1].

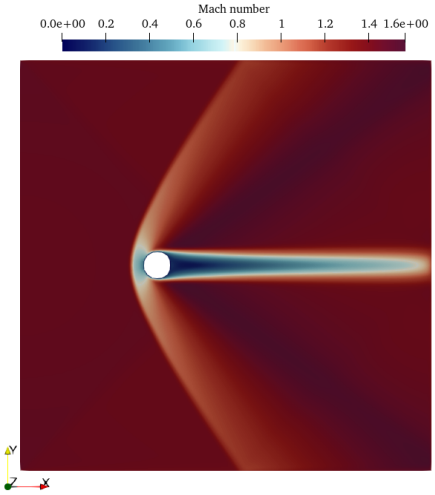


Figure 1: Simulation of compressible flow over a 2D cylinder using CLBM

In conventional LBM, a single distribution function is utilized in one lattice node to represent mass and momentum conservation when approximating incompressible Navier-Stokes Equations. However, the Double Distribution Function (DDF) approach, also known as the two-population method, is a popular choice for thermal Lattice Boltzmann simulations, and later, for compressible Lattice Boltzmann simulations[3]. For instance, considering 2D, there are two set of discrete distribution functions with nine elements for each in single D2Q9 standard stencil. Further, the CLBM incorporates the Entropic Lattice Boltzmann Method (ELBM) where several approximations are necessary compared to the standard Incompressible LBM. Therefore, it is worth investigating the performance of the newly developed CLBM application.

1.3. Testsystem

- Host/Clustername: Fritz
- Cluster Info URL: https://hpc.fau.de/systems-services/documentation-instructions/clusters/fritz-cluster/
- CPU type: 2x Intel Xeon Platinum 8360Y @ 2.4 GHz
- Number of cores per node: 72
- Memory capacity: 256 GB

1.4. Software Environment

- Compiler: GNU c++ compiler (g++)
- OpenMP
- Operating System: Ubuntu 20.04.3 LTS
- Addition libraries:
  - GSL-GNU Scientific Library
  - LIKWID 5.3.0

1.5. How to build software

All the required libraries, directories and relevant compiler directives are defined in the CMakeLists.txt file. After logging in to Fritz the CLBM solver can be built using the following commands. Furthermore, to observe the proper linking of libraries and compiler directives, `make VERBOSE=1` can be used.

```
$ module load cmake
$ module load likwid
$ module load gsl
$ git clone git@gitlab.cs.fau.de:yh54ojyn/mucosim.git
$ cd mucosim
$ cd <2D/3D case>
$ cd build
$ cmake ..
$ make
```

In CMakeLists.txt file, the following compiler optimization flags have been set for the g++ compiler.

- `-O3`: Highest optimization level
- `-march=native`: Optimize code for a given architecture
- `-mavx`: Enables support for AVX (Advanced Vector Extensions)
- `-ftree-vectorize`: Enable loop vectorization

1.6. Test case description

A 2D shock tube simulation setup is considered for the present performance analysis. The solver output for a given input parameters is validated before doing the performance analysis to check the correctness of the solver. Furthermore, there are no obstacles present inside the shock tube. All the input parameters are defined in the `input_parameters_ST_2.txt` file. For the analysis, the following input properties are used for a 3000x3000, 2D computational domain. As mentioned in 1.2 the standard D2Q9 lattice stencil is utilized.

```
input_file              input_parameters_ST_2.txt
vtk_out_name_base       ST_2_
part_out_name_base      ST_2_
vtk_out_freq            100
part_out_freq;          1000
```

```
domain_size_x           2998
domain_size_y           2998
domain_size_z           1
helper_cells            2
total_time              300
fluid_flag              0
Prandtl_number          0.71
sphr                    1.4
u_x                     0.0
u_y                     0.0
u_z                     0.0
kinematic_viscosity     0.015
x_shift_velocity        0.0
rho_l                   1.0
rho_r                   0.125
T_l                     0.15
T_r                     0.12
```

Note: The following simulation domain size is obtained after performing a single-core performance analysis of a Fritz node. According to analysis, it is found that the domain size should be approximately more than 500x500 to avoid high OpenMP overhead. Furthermore, it is advised that the run time should be at least more than 10 minutes to obtain accurate power and energy measurements using `likwid-perfctr`. Therefore, the simulation domain size is selected as 3000x3000 lattice nodes along the X and Y axes.

1.7 How to run software

After building the application as mentioned in section 1.5, following steps can be used to run the CLBM application.

```
$ module load likwid
$ module load gsl
$ cd mucosim
$ cd 2D_SHOCK_TUBE
$ cd build
$ export OMP_NUM_THREADS= X ./CLBM
```

However, when obtaining all the performance measurements, job scripts are submitted according to the Slurm batch system. Information regarding the basic usage of the Slurm batch system can be found at https://hpc.fau.de/systems-services/documentation-instructions/batch-processing. The exemplary usage of batch scripts used in the present analysis can be listed as follows.

```
#!/bin/bash -l
#SBATCH --job-name=CLBM
#SBATCH --partition=singlenode
#SBATCH --nodes=1
```

```
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --time=24:00:00
#SBATCH --export=NONE
#SBATCH --constraint=hwperf
#SBATCH --output=/home/%j_%x.out
unset SLURM_EXPORT_ENV
module load gsl
module load likwid
# This pins openMP threads to physical cores on the socket
export OMP_PLACES=cores
# This places threads as close as possible to one another
export OMP_PROC_BIND=true

export OMP_NUM_THREADS=72
srun --cpu-freq=2400000-2400000:performance likwid-perfctr -C S0:0-35@S1:0-35
  -g MEM_DP -m ./CLBM
```

2. Task1: Scaling runs

Usually, the performance matric for the LBM simulations is taken as the Mega Lattice Updates per seconds (MLUP/s). The `MLUP/s` is calculated using the output of the CLBM application.

```
3000    3000    runtime_in_seconds    MLUP/s
```

The obtained performance in `MLUP/s` versus the number of cores is shown in the following figure 2. According to the presented strong scaling results, the performance saturates in 1st NUMA domain, and then it scales linearly.

The figure 3 represent the simulation time for 300 lattice time steps at 2.4 `GHz`. In the next sections, several measurements are obtained by using `likwid-perfctr` to draw some conclusions about the whole application and selected hotspots of the applications.

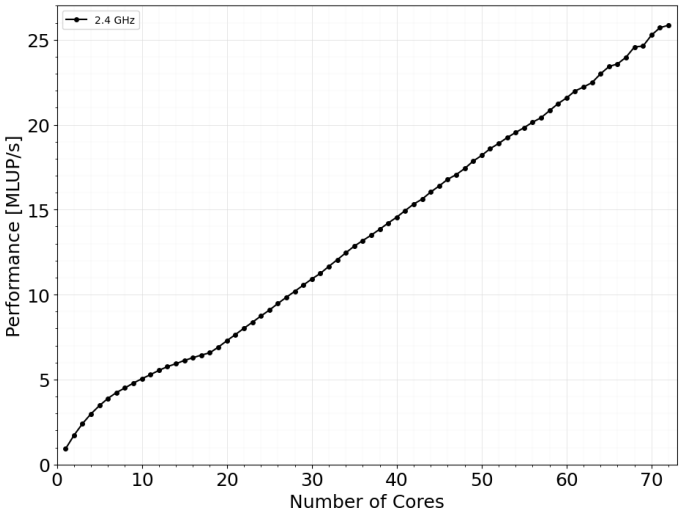


Figure 2: Number of cores versus performance in MLUP/s

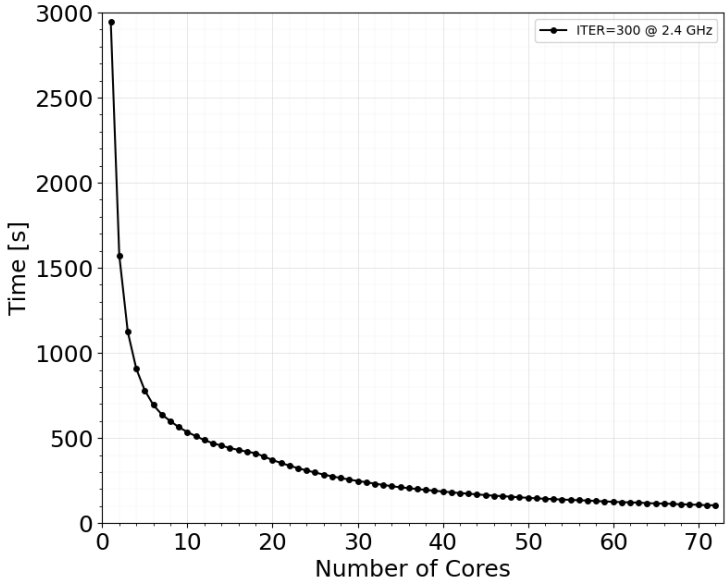


Figure 3: Number of cores versus execution time in seconds

3. Task2: Whole application measurements

The main aim of this study is to investigate the energy efficiency and power consumption of the present CLBM implementation. Therefore, the main LIKWID performance group used in this study is `MEM_DP`. An exemplary output from the `MEM_DP` group is shown in the following listing.

```
likwid-perfctr -C S0:0 -g MEM_DP -m ./CLBM
+-----------------------------------+------------+
|               Metric              | HWThread 0 |
+-----------------------------------+------------+
|        Runtime (RDTSC) [s]        |  2974.8850 |
|        Runtime unhalted [s]       |  2961.7851 |
|            Clock [MHz]            |  2394.3449 |
|                CPI                |     0.7580 |
|             Energy [J]            | 40677.5900 |
|             Power [W]             |    13.6737 |
|          Energy DRAM [J]          | 23610.0800 |
|           Power DRAM [W]          |     7.9365 |
|           DP [MFLOP/s]            |  1199.3221 |
|          AVX DP [MFLOP/s]         |    77.7390 |
|          Packed [MUOPS/s]         |    19.4347 |
|          Scalar [MUOPS/s]         |  1121.5832 |
|  Memory read bandwidth [MBytes/s] |  9752.9123 |
|  Memory read data volume [GBytes] | 29013.7925 |
| Memory write bandwidth [MBytes/s] |  1816.0125 |
| Memory write data volume [GBytes] |  5402.4282 |
|    Memory bandwidth [MBytes/s]    | 11568.9248 |
|    Memory data volume [GBytes]    | 34416.2207 |
| Operational intensity [FLOP/Byte] |     0.1037 |
|      Vectorization ratio [%]      |     1.7033 |
+-----------------------------------+------------+
```

Here, all the relevant measurements for the present study can be extracted such as,

- Memory bandwidth (MBytes/s)
- Energy (J)
- Power (W)
- Energy DRAM (J)
- Power DRAM (W)
- Double precision; DP performance (MFlop/s)
- Operational intensity (Flop/Byte)

However, there are some separate performance groups in `likwid-perfcter` to measure energy and memory bandwidth, such as `ENERGY`, `MEM` if relevant. Initially, the memory bandwidth, total energy, total power consumption, DRAM energy and DRAM power of the whole application are measured with varying numbers

of cores in one Fritz node. According to figures 4–8, the following observations can be made considering the whole application.

- Memory bandwidth also saturates in 1st NUMA domain. At saturation, it is approximately 75.5 GB/s. Further, according to the strong scaling results, the maximum attainable performance using a full node is ~25 `MLUP/s` and the maximum attainable memory bandwidth is close to 290 `GB/s`.
- Energy measurements have been obtained for several frequencies ranging from 1.8 `GHz` to 3.0 `GHz` with an increment of 0.2 `GHz`. It is because, to investigate the optimal frequency or frequency for the present implementation in terms of energy.
- In energy and power measurements, there is an unusual behavior in the single-core measurements. This behavior can be reproduced consistently for several data samples. However, the proper energy variation can be observed for 2.2 `GHz` consistently for several data samples. `likwid-powermeter` also produced the same variations.
- The energy values have a saturating optimum point at 1st NUMA domain before reaching a second optimum point at 1st socket.
- Sudden energy jumps when moving to the second socket before decreasing in a quite linear fashion in the second socket. Likewise, a sudden power jump can be observed for the power measurements too.
- Usually, DRAM energy contributes a considerable portion to the total energy consumption. Therefore, the contribution from the DRAM energy and power is also investigated.
- As in the total energy measurements, the unexpected data point at the beginning of the energy and power measurements cannot be observed for the DRAM energy and power measurements. However, the DRAM energy and power trends follow the same trend as the total energy and power measurements, except for a slight saturation in DRAM power measurements in 1st NUMA domain.
- 2.2 `GHz` shows the lowest energy curve for a wider domain across the node.
- The frequencies (`GHz`) 1.8, 2.0, 2.2, and 2.4 show the lowest CPU energy measurements while having approximately the same energy values in the first socket. For instance, at 18th core 2.4 `GHz` has the lowest total energy, and at 36th core 1.8 `GHz` shows the lowest total energy value.
- Surprisingly, the DRAM energy contribution to the total energy is in the range of 5%–11%.

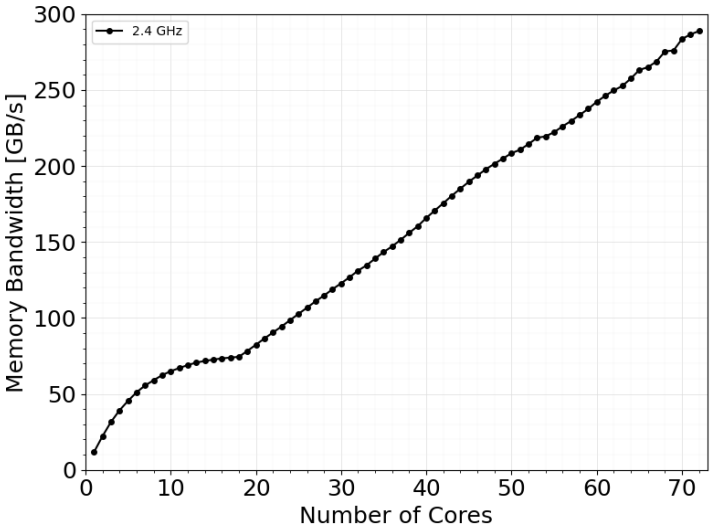


Figure 4: Number of cores versus memory bandwidth in GB/s

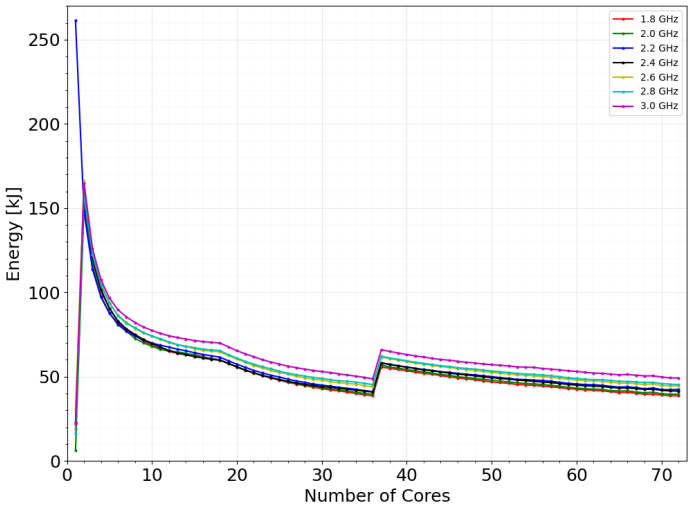


Figure 5: Energy measurements in one node

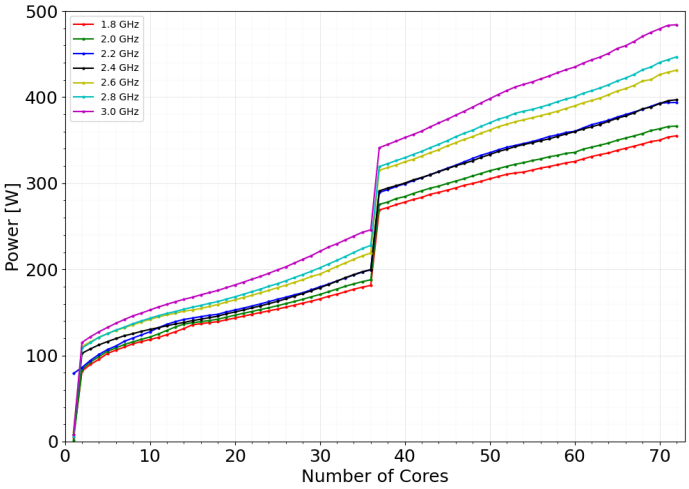


Figure 6: Power measurements in one node

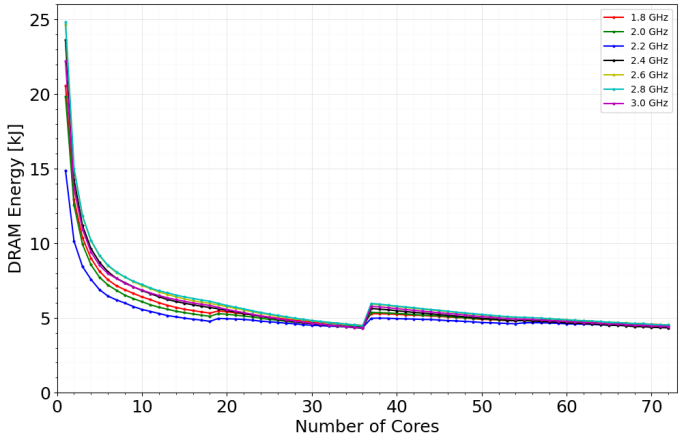


Figure 7: DRAM Energy measurements in one node

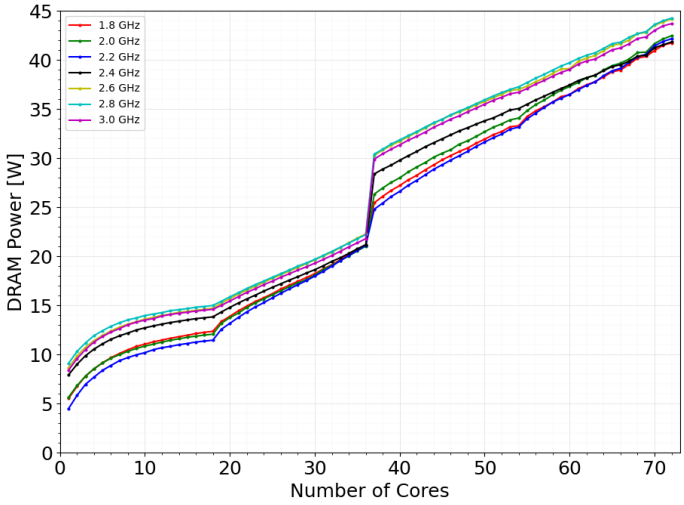


Figure 8: DRAM Power measurements in one node

3.1 Energy Delay Product (EDP)

Energy-to-solution and the energy-delay product (EDP) constitute the fundamental metrics for measuring energy efficiency [2], [5]. EDP for a given application can be obtained by using `Z`-plots. Typically, `Z`-plots represents the energy to solution versus a suitable performance metric. EDP is the gradient of a line passing through the `Z`-plot [4]. As mentioned in some of the literature, energy efficiency increases with decreasing EDP [2], [5].
Therefore, to observe the EDP, a `Z`-plot is constructed as shown in figure 9, for the above-mentioned frequency range. The `MLUP/s` is taken as the performance metric. Furthermore, energy and performance measurements at the first socket are considered when constructing the `Z`-plot. It is because, according to previous observations, the lowest energy point for the whole application is observed at the first socket in the Fritz node.
Figure 9 shows the acquired EDP values at the first NUMA domain and at the first socket. Here, the EDP values are related to the 2.4 `GHz`. According to the observations, it is evident that with the increasing core count, the EDP values decrease for the CLBM application. Further, this behaviour is quite common in most of the modern architectures [2]. However, considering figure 10, it is evident that the frequencies 1.8–2.4 `GHz` approximately lie on the same EDP gradient. Therefore, the CLBM application shows the highest energy efficiency at first socket in Fritz node and optimal frequencies in terms of energy efficiency can be 1.8–2.4 `GHz`.

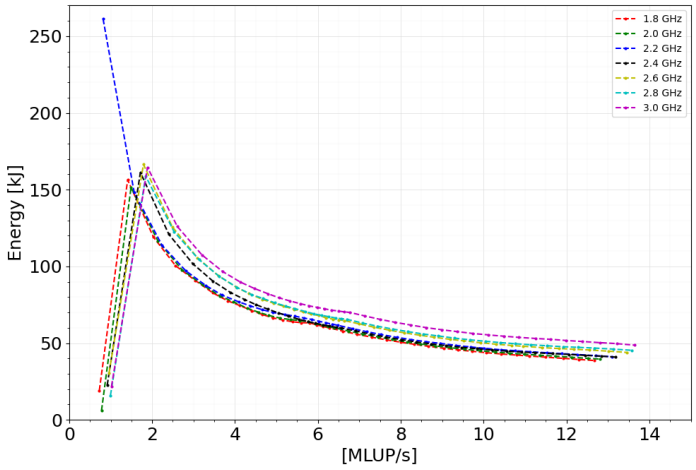


Figure 9: Z-plot considering one socket in a Fritz node.

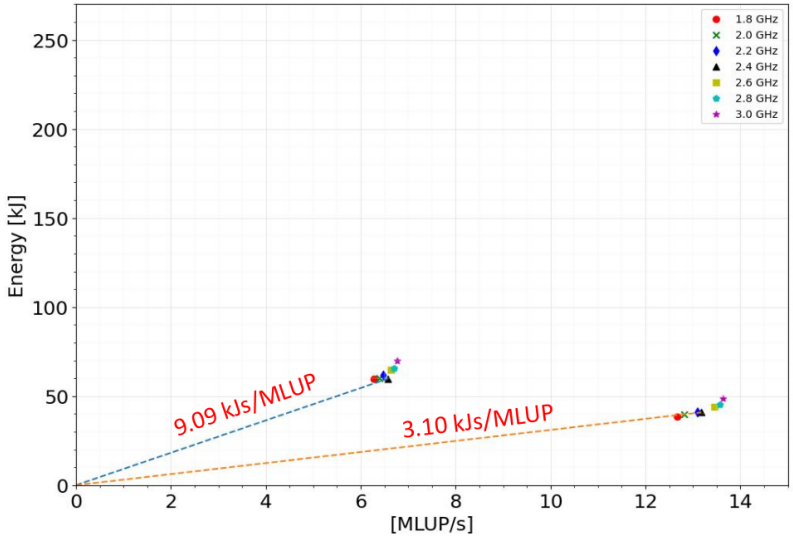


Figure 10: EDP related to 18 and 36 cores at 2.4 GHz

4. Task3: Runtime profile

To obtain the runtime profile of the application, the `VTune` profiler is utilized. After getting into the build directory, the following steps can be used to get a runtime profile for the application.

```
$ module load vtune
$ vtune -collect hotspots -result-dir hotspots_CLBM ./CLBM
```

Thereafter, the results files inside the `hotspots_CLBM` directory can be opened using `vtune-gui`. The following table lists the hotspots provided by the summary tab in `vtune-gui`.

Functions	Module	CPU Time	% of CPU Time
stream._omp_fn.18	CLBM	769.654s	9.9%
collision._omp_fn.12	CLBM	685.792s	8.8%
resetDDFshifts._omp_fn.14	CLBM	631.249s	8.1%
swap._omp_fn.14	CLBM	591.541s	7.6%
calcQuasiEqG._omp_fn.10	CLBM	576.830s	7.4%
[others]	N/A*	4508.774s	58.1%

* N/A applied to non-summable metrics.

`Vtune` provides an extensive set of tools to analyse the code, which can be used to develop better implementation. For instance, during phase 1, the CLBM implementation showed bad strong scaling results. To obtain the better results as above, `Vtune` flame graph representations, detailed function call stack details helped to understand the underlying issues during phase 1.
For instance, during phase 1, the top hotspot was the `expf4` function call, which helps to calculate an exponential function using the `math.h` library inside the `newtonRaphson` subroutine. Therefore, several changes have been done to optimize the `newtonRaphson.h` implementation to reduce the number of `expf4` function calls. The results obtained for the phase 1 can be seen in the presentation 1 and 2 in the Appendix section.
According to the runtime profile analysis, the `stream`, `collision` and `newtonRaphson` subroutines have been selected for further investigations. According to the literature related to compressible DDF-LBM implementations, considerable attention was given to root-finding scheme considering the performance of respective DDF-LBM implementations[3]. Therefore, in this study, the `newtonRaphson` subroutine is selected even though it is not in the top hotspots.
Note: Apart, from the code optimization in `newtonRaphson.h` several investigations have been carried out in terms of OpenMP scheduling, as well as OpenMP parallelization of several kernels which were not parallelized during phase 1.

5. Task4: Instrument kernels with MarkerAPI

After getting the runtime profile, the LIKWID MarkerAPI calls are placed around the hotspots to obtain the relevant measurements[7]. The MarkerAPI calls are placed as shown in the following code snippet.

```
#include <likwid-marker.h>

// initialize LIKWID MarkerAPI in a serial region.
LIKWID_MARKER_INIT;
// register a region name tag(=stream) to the Marker API
#pragma omp parallel
{
    LIKWID_MARKER_REGISTER("stream");

}

Data: Read computational grid and fluid properties;
Initialization: density, velocity, temperature and set temperature dependent weights;
Modification of c_i according to the shifted lattice velocity U ;
Calculate: f_eq ;
Estimate Lagrangian multipliers <-- Newton-Raphson ;
Calculate: g_eq and Initialize g, f ;
6 for (int i = 0; i < N umber of time steps; + + i)
{
    Calculate density, velocity, temperature and set temperature dependent weights;
    Calculate time dependent relaxation times;
    Write .vtk files for post-processing;
    Calculate f_eq ;
    Estimate Lagrangian multipliers <-- Newton-Raphson;
    Calculate g_eq ;
    Calculate pressure tensor;
    Calculate quasi-equilibrium distribution;
    Applying Knudsen number dependent stabilization;
    Collide;
    Reconstruction of f, g at grid nodes;
    Set boundary conditions;
    #pragma omp parallel
    {
    LIKWID_MARKER_START("stream");
    }
    Stream;
    #pragma omp parallel
    {
    LIKWID_MARKER_STOP("stream");
    }
```

```
    Swap f_new and f_old ;
}
// finalize the LIKWID MarkerAPI in a serial region.
LIKWID_MARKER_CLOSE;
```

As shown in the above code snippet, measurements for all the three kernels have been obtained by placing LIKWID MarkerAPI calls respectively. When using `likwid-perfctr`, the `-m` CLI switch activates the MarkerAPI[7].

6. Task5: Measurements of the selected hot spots

By using MarkerAPI calls, relevant measurements for the present study have been obtained following the same procedure as in Task 2. Here, the strong scaling results, memory bandwidth, energy consumption and power dissipation are measured. The DRAM energy and DRAM power measurements are not considered due to the less contribution towards the total energy contribution as observed in Task 2. Energy measurements have been obtained for three different frequencies, and remarks considering the hotspots will be discussed in Task 6.

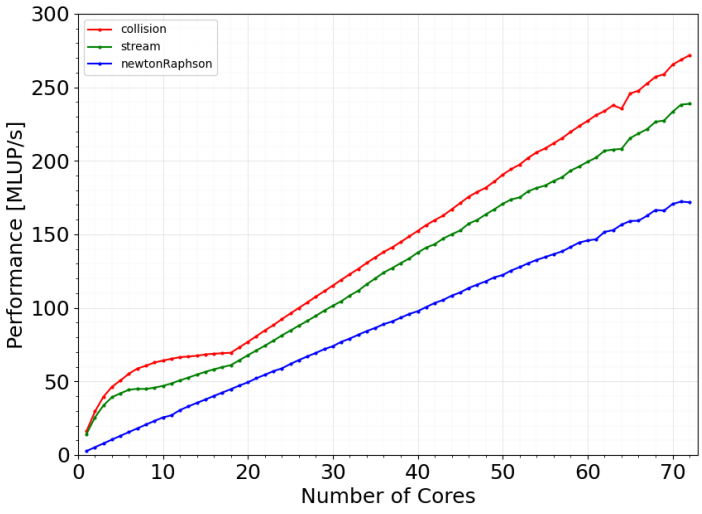


Figure 11: Performance versus number of core count for selected hotspots

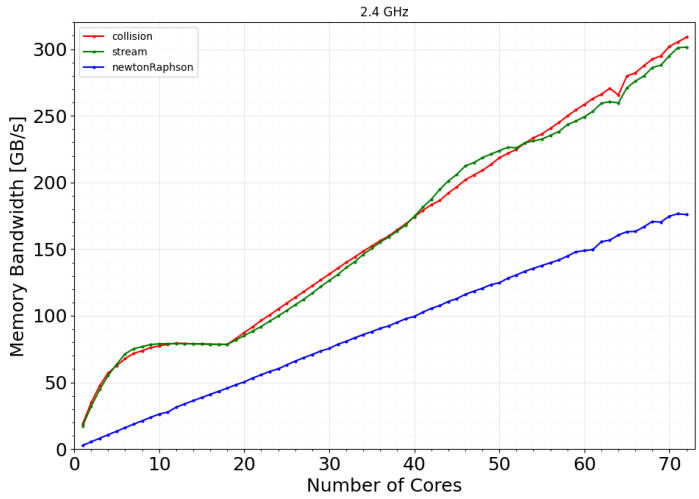


Figure 12: Memory bandwidth versus number of core count for selected hotspots

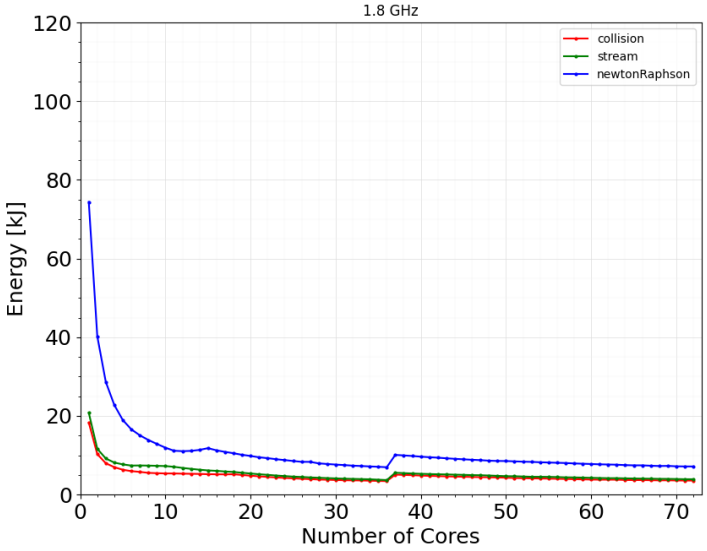


Figure 13: Energy versus number of core count for selected hotspots at 1.8 GHz

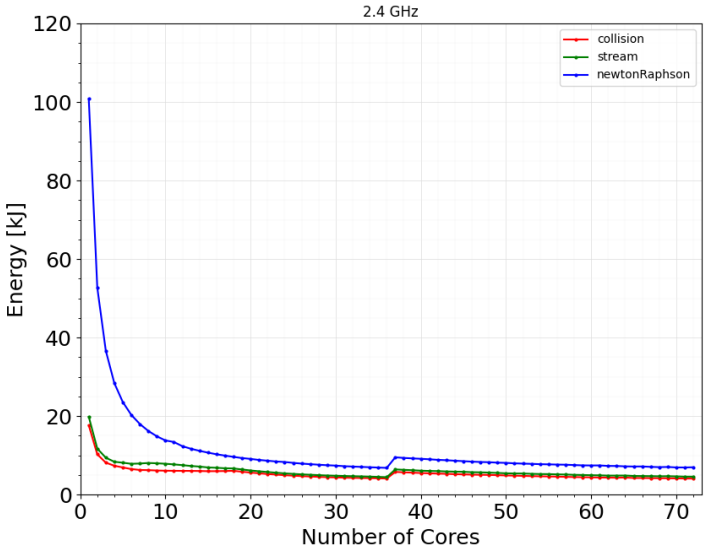


Figure 14: Energy versus number of core count for selected hotspots at 2.4 GHz

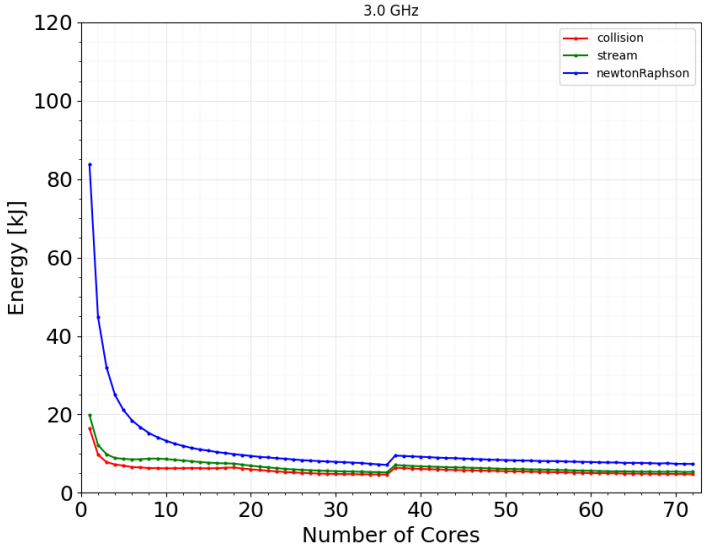


Figure 15: Energy versus number of core count for selected hotspots at 3.0 GHz

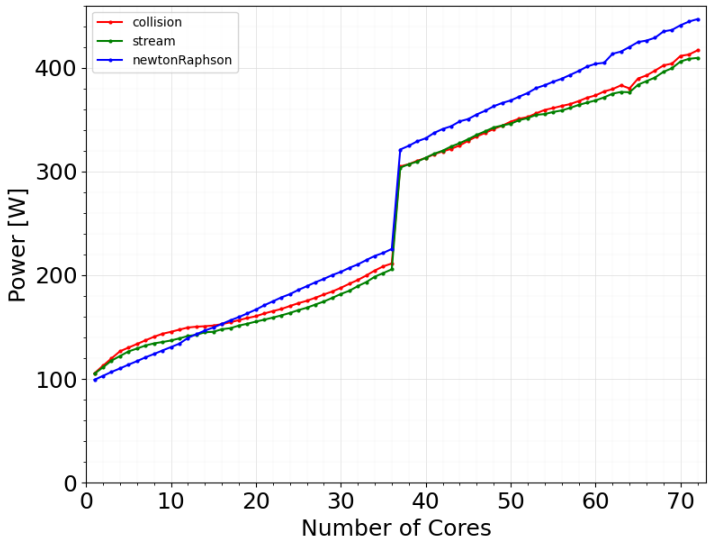


Figure 16: Power versus number of core count for selected hotspots at 2.4 GHz

7. Task6: Discussion of hot spot measurements

The following remarks can be drawn from the results obtained in Task 5.

- According to the strong scaling results as shown in figure 11, `collision` and `streaming` kernels show a saturation in 1st NUMA domain and thereafter scales linearly across the node. Furthermore, `newtonRaphson` kernel does not show any saturation across the node. However, `newtonRaphson` kernel has the lower `MLUP/s` compared to both the other top hotspots.
- As shown in figure 12, the memory bandwidth for the top 2 hotspots saturates at 1st NUMA domain, while `newtonRaphson` does not seem to have any saturation. Usually, the strong scaling trends and the memory bandwidth scaling runs have the similar characteristic. However, according to the results there is a slight discrepancy between strong scaling trends and the memory bandwidth for `streaming`.
- One of the reasons for both the above points may be the usage of `MLUP/s` as the performance metric. Because, it is obvious that the total runtime is much greater than the hotspot runtime for 2.7 billion lattice updates, and therefore `MLUP/s` has higher values for hotspots. Maybe the `Flop/s` is a realistic metric for this comparison. However, we cannot use this, because then we cannot put stream kernel for the comparison, since it does not have any floating-point operations.
- All three selected hotspots contribute 9% - 16% to the main energy mea-

surements.

- `newtonRaphson` has the highest contribution while showing a scalable performance within the node. Furthermore, all the hotspots seem to follow the same energy trend as the energy rend of the whole application, as discussed in the Task 2.
- Both the highest hotspot ranks seem to have similar variation in terms of power dissipation.

To further investigate the capabilities of the present CLBM implementation, an empirical roof-line model is constructed [6]. Here, the DP, double precision `Flop/s` is taken as the performance metric. Further, the position of the whole application along with selected hotspots is placed in the roof-line diagram. However, it is worth noting that the streaming kernel does not have any `Flop/s`. Therefore, it is not included here. In the roof-line diagram, the Maximum horizontal roof is related to the AVX `Flop/s` and maximum data throughput is related to the `load_avx`. For all the measurements, the functionalities of `likwid-bench` is utilized.

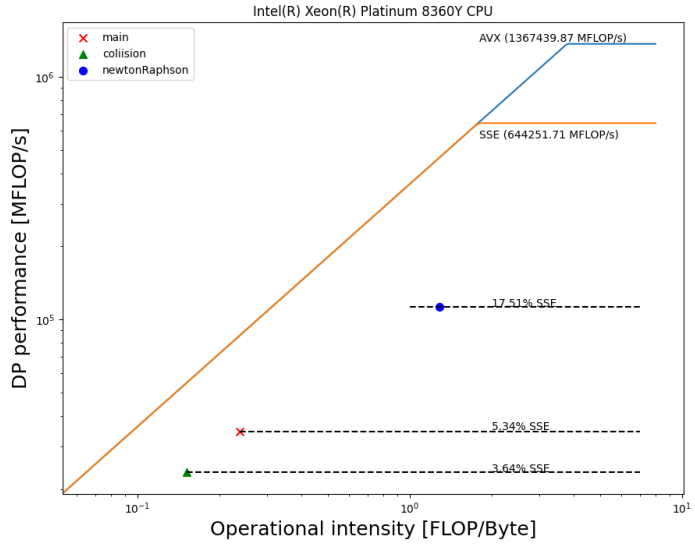


Figure 17: Roof-line diagram

According to the roofline diagram, it is evident that the most realistic maximum horizontal roof for the CLBM application would be the line related to the `SSE Flop/s`. The whole application shows approximately 5.34% percent performance of the maximum horizontal `SSE` roof. Further, the whole application and the selected hotspots are close to memory bound, which is the typical scenario of the most LBM implementations.

Moreover, the usage of AVX is limited for the present CLBM implementation, even though the required compiler optimizations have incorporated. May be one of the reasons for this is the usage of the GNU g++ compiler. However, the attempts to use the intel compiler were not successful due to the dependency of GNU scientific library in the implementation.
To conclude, the studied shock tube case with CLBM shows scalable performance, and it shows one of the main advantages of using LBM schemes, which is the natural adaptability to parallel processes computing [8]. Further, the Solver is close to memory bound. According to the roofline analysis, the application has arbitrary horizontal roofs. Therefore, maybe there are more opportunity to increase bandwidth utilization, for example by using SIMD instructions in the newtonRaphson routine.
One of the primary focuses of this study is to understand the behaviour of energy and power dissipation of the application. According to the analysis, the optimal energy can be observed at the first socket in Fritz node considering 1.8 GHz, 2.0 GHz, 2.2 GHz and 2.4 GHz and EDP results verified it. However, DRAM energy contribution to the total energy is around 5%-11%, and it has comparatively less impact on the total energy consumption. Further, the root finding scheme has less significance in terms of performance, even though the root finding algorithm was the highlight in previous literature regarding, this LBM regime. According to the authors' knowledge this is first kind of extensive benchmarking study related to compressible DDF-LBM.

Reference

1. **The Landscape of Parallel Computing Research A View from Berkeley, 2006 Electrical Engineering and Computer** Sciences, University of California at Berkeley, Electrical Engineering and Computer Sciences, University of California at Berkeley.
2. Ayesha Afzal, **The Cost of Computation: Metrics and Models for Modern Multicore-based Systems in Scientific Computing**, Master thesis, July 2015, DOI: 10.13140/RG.2.2.35954.25283
3. Jonas Latt, Christophe Coreixas, Joël Beny, and Andrea Parmigiani. **Efficient supersonic flow simulations using lattice boltzmann methods based on numerical equilibria**. Philosoph ical Transactions of the Royal Society A:Mathematical, Physical and Engineering Sciences,378(2175):20190559, jun 2020. 5.
4. https://blogs.fau.de/hager/archives/tag/energy
5. Ayesha Afzal,Georg Hager,Gerhard Wellein,**SPEChpc 2021 Benchmarks on Ice Lake and Sapphire Rapids Infiniband Clusters: A Performance and Energy Case Study, SC-W '23: Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis**,November 2023, Pages 245–1254,https://doi.org/10.1145/3624062.3624197

6. https://github.com/RRZE-HPC/likwid/wiki/Tutorial:-Empirical-Roofline-Model
7. https://github.com/RRZE-HPC/likwid/wiki/TutorialMarkerC
8. Timm Krüger, Halim Kusumaatmaja, Alexandr Kuzmin, Orest Shardt, Goncalo Silva, and Erlend Magnus Viggen. **The Lattice Boltzmann Method**. Springer International Publishing, 2017.

Appendix

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

COMPRESSIBLE LATTICE BOLTZMANN SOLVER CPU BENCHMARK

MuCoSim WS 2023/24

PHASE: 1

Sudesh Rathnayake

Introduction

- Performance analysis of a OpenMP parallelize compressible lattice Boltzmann solver.
- Dwarfs → "Capture a pattern of computation and communication common to a class of important applications"[1]

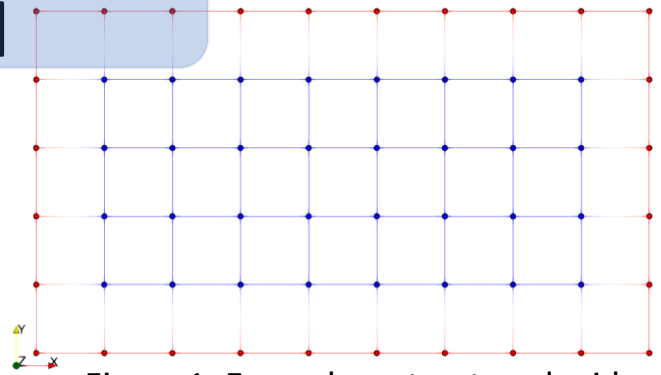


Figure 1: Exemplary structured grid

- Seven Dwarfs
  1. Dense Linear Algebra.
  2. Sparse Linear Algebra.
  3. Spectral Methods.
  4. N-Body Methods.
  5. Structured Grids. → "Represented by a regular grid; points on grid are conceptually updated together. It has high spatial locality"'[1]
  6. Unstructured Grids.
  7. Monte Carlo.
- Lattice Boltzmann Method(LBM) → Commonly, LBM adopt a structured grid in practice.

Lattice Boltzmann Method: Overview

- Two main approaches to simulate transport equations
  - 1.Continuum
  - 2.Discrete
- The numerical scheme for the Boltzmann equation ,
  - ➢ Quite simple, both to implement and to parallelize. (hyperbolic equation with force free formulation)
- Easy to apply.
  - ➢ For complex domains,
  - ➢ Easy to treat multi-phase and multi-component flows.
- Naturally adapted to parallel processes computing. [3]
- No need to solve Laplace equation at each time step as in FVM.

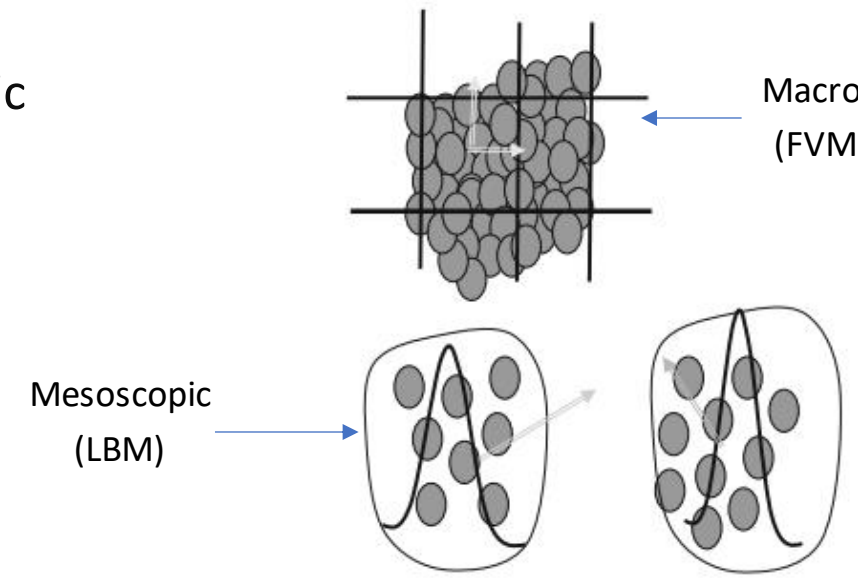


Figure02:Different scales in simulation.[2]

- The “Collide and Stream” algorithms.

$$f_i^*(x,t) = f_i(x,t) + \Omega_i(x,t) \qquad f_i(x + c_i\Delta t, t + \Delta t) = f_i^*(x,t)$$

- Collision operator: Ω_i ← f_i^{eq} ; Needs to be estimated

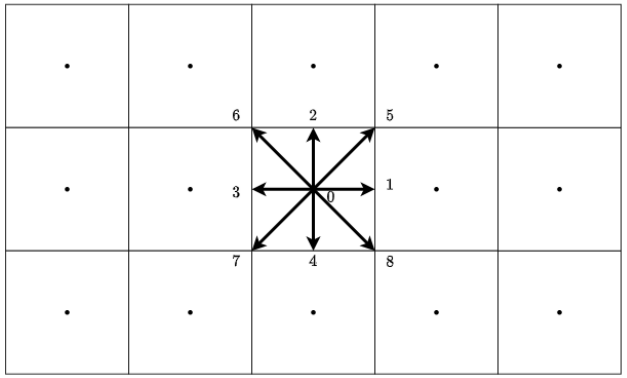


Figure 03:Lattice with D2Q9 stencil

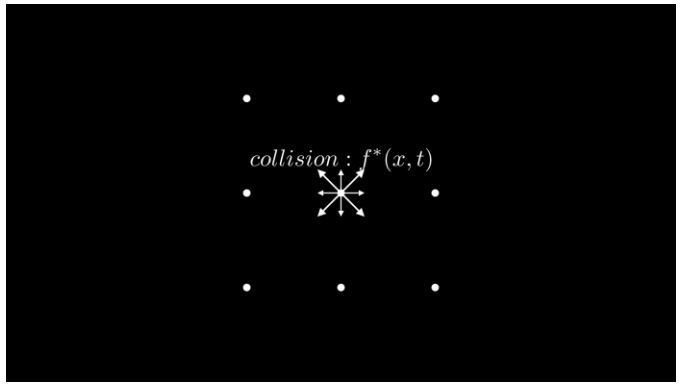


Figure 04: Collide and Stream

- Beauty, yet a weakness, of the LBM lies in its explicitness and uniform grid.
- Widely used for simulating incompressible fluid flows

- Compressible LBM → f_i^{eq} using discreate entropy function by formulating a minimization problem.

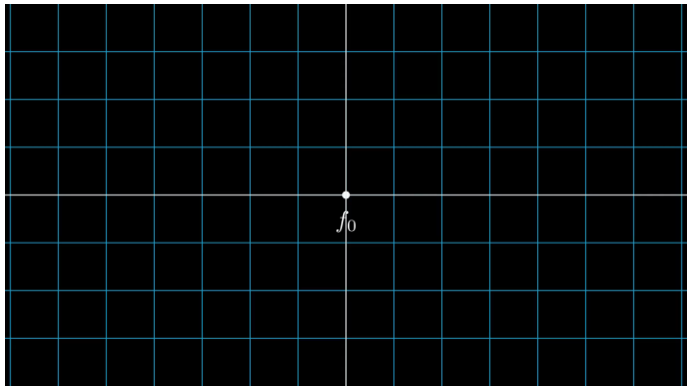


Figure 05: f and g populations in a lattice node

- 2 populations to integrate energy equation apart from the mass, and momentum equations.
- The present solver based on Entropic Lattice Boltzmann Method (ELBM)

CLBM: Algorithm

Algorithm : Compressible LBM algorithm for strong compressible fluid flows.

Data: Read computational grid and fluid properties;
Initialization: $\rho, \mathbf{u}, T$ and set temperature dependent weights;
Modification of c_i according to the shifted lattice velocity U;
Calculate: f^{eq};
Estimate Lagrangian multipliers ← Newton-Raphson ;
Calculate: g^{eq} and Initialize g, f;
for $(int\ i = 0; i < Number\ of\ time\ steps; ++i)$
{
    Calculate $\rho, \mathbf{u}, T$ and set temperature dependent weights;
    Calculate time dependent relaxation times;
    Write .vtk files for post-processing;
    Calculate f^{eq};
    Estimate Lagrangian multipliers ← Newton-Raphson;
    Calculate g^{eq};
    Calculate pressure tensor;
    Calculate quasi-equilibrium distribution g^*;
    Applying Knudsen number dependent stabilization;
    Collide;
    Reconstruction of f, g at grid nodes;
    Set boundary conditions;
    Stream;
    Swap f^{new} and f^{old};
}

Figure 07: CLBM algorithm

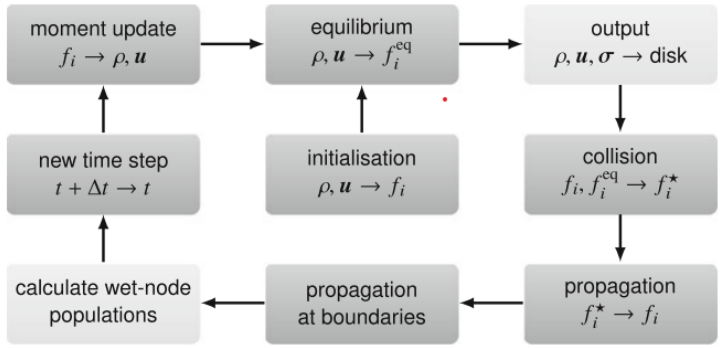


Figure 06: Standard LBM algorithm.[3]

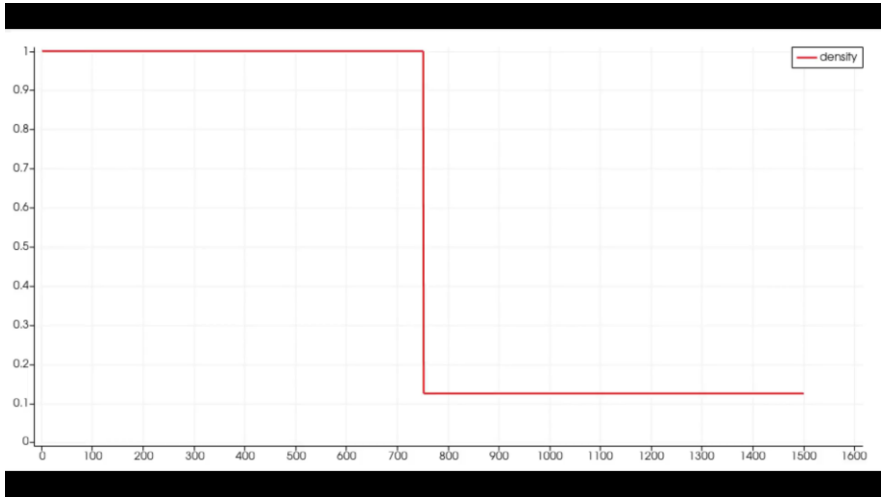


Figure 08: Results for density variation of air inside a 2D shock tube using CLBM solver

- Solver is based on C++ and OpenMP.
- Build using CMake.
- Input parameter file defines the simulation properties.
- For initial benchmarking and profiling, a 2D shock tube simulation is considered.

git@gitlab.cs.fau.de:yh54ojyn/mucosim.git

CLBM runtime profile

- A runtime profile in “Fritz”.
- Initially use gprof for the profiling.
- Vtune is preferred; much more insights.

CPU name: Intel Xeon Platinum 8360Y CPU @ 2.40GHz
CPU type : Intel Icelake SP processor
Number of lattice nodes: 2000 x 2000= 4000000 Lattice nodes.
Compiler: g++
Compiler Directives: -O3 –pg –march=native –mavx –ftree-vectorize

Required modules:
gsl (GNU scientific library)
Cmake
vtune

time	seconds	self seconds	calls	self (ns/call)	self(ns/call)	name
49.99	735.5	735.5				frame_dummy
20.57	1038.1	302.6				j33(node&)
4.83	1109.16	71.06				calc_parameters_from_PDF(grid&, double const&, int)
4.7	1178.29	69.13	55652400	1.24	1.24	j23(node&)
4.03	1237.63	59.35	55666385	1.07	1.07	j13(node&)
2.56	1275.27	37.64	27848884	1.35	1.35	j11(node&)
2.44	1311.15	35.87	27812635	1.29	1.29	func1(node&, double)
2.36	1345.86	34.71	27845283	1.25	1.25	j22(node&)
2.34	1380.24	34.38	27845516	1.23	1.23	func3(node const&, double)
2.05	1410.41	30.18	27852987	1.08	1.08	func2(node&, double)
2.04	1440.47	30.06	27855020	1.08	1.08	j12(node&)
1.97	1469.45	28.98	27833950	1.04	1.04	j21(node&)
0.13	1471.36	1.91				grid::grid(int, int)
0.02	1471.6	0.24				j32(node&)
0.01	1471.68	0.08				createFlags(grid&)
0.01	1471.76	0.08				swap(grid&)
0	1471.76	0	1	0	0	_GLOBAL__sub_I__Z7readingRNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEER4grid

Figure09: gprof hotspot results.

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time	% of CPU Time
expf64	libm.so.6	1687.306s	23.2%
stream._omp_fn.15	CLBM	679.169s	9.3%
collision._omp_fn.9	CLBM	610.130s	8.4%
calcQuasiEqG._omp_fn.8	CLBM	505.590s	7.0%
resetDDFshitS._omp_fn.11	CLBM	475.925s	6.5%
[Others]	N/A*	3311.410s	45.6%

*N/A is applied to non-summable metrics.

Figure10: Vtune hotspot results.

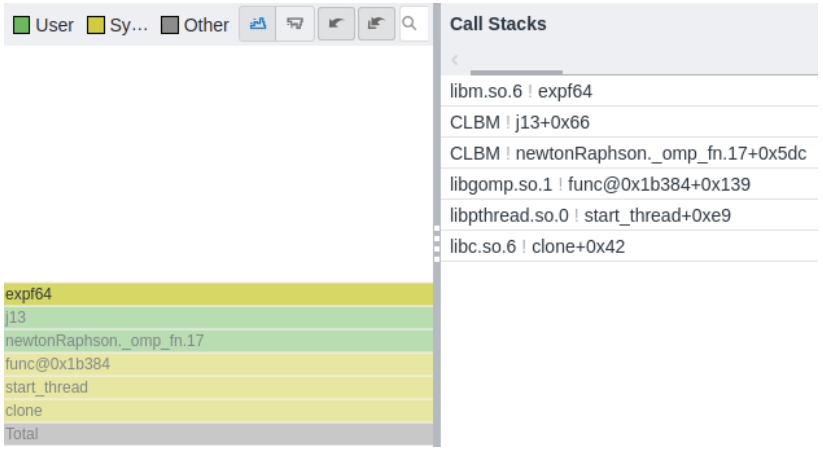
CLBM runtime profile

- High OpenMP overhead is observed for small data set.
- The newtonRaphson function is the most critical one in CLBM function.
- On top of that the calculation of exponential function using math.h library is critical.

Grouping: Function / Call Stack

Function / Call Stack	CPU Time ▼	Module	Function (I
expf64	1687.306s	libm.so.6	expf64
↖ j23 ← newtonRaphson._omp_fn.17	261.509s	CLBM	j23(node&)
↖ j13 ← newtonRaphson._omp_fn.17	255.705s	CLBM	j13(node&)
↖ j33 ← newtonRaphson._omp_fn.17	131.396s	CLBM	j33(node&)
↖ func2 ← newtonRaphson._omp_fn.17	128.771s	CLBM	func2(node&, double)
↖ j22 ← newtonRaphson._omp_fn.17	128.538s	CLBM	j22(node&)
↖ j21 ← newtonRaphson._omp_fn.17	127.912s	CLBM	j21(node&)
↖ j11 ← newtonRaphson._omp_fn.17	127.344s	CLBM	j11(node&)
↖ func3 ← newtonRaphson._omp_fn.17	126.925s	CLBM	func3(node const&, double)
↖ j12 ← newtonRaphson._omp_fn.17	126.888s	CLBM	j12(node&)
↖ func1 ← newtonRaphson._omp_fn.17	121.991s	CLBM	func1(node&, double)
calcGeq._omp_fn.5	95.709s	CLBM	calcGeq._omp_fn.5
[No call stack information]	54.618s		
stream._omp_fn.15	679.169s	CLBM	stream._omp_fn.15
collision._omp_fn.9	610.130s	CLBM	collision._omp_fn.9
calcQuasiEqG._omp_fn.8	505.590s	CLBM	calcQuasiEqG._omp_fn.8
resetDDFshitS._omp_fn.11	475.925s	CLBM	resetDDFshitS._omp_fn.11
pTensor._omp_fn.6	425.356s	CLBM	pTensor._omp_fn.6
swap._omp_fn.16	422.428s	CLBM	swap._omp_fn.16
pEqTensor._omp_fn.7	410.548s	CLBM	pEqTensor._omp_fn.7
calcGeq._omp_fn.5	323.667s	CLBM	calcGeq._omp_fn.5
calcFeq._omp_fn.4	256.802s	CLBM	calcFeq._omp_fn.4
func@0x770e4	248.515s	libm.so.6	func@0x770e4
func@0x1dfd4	227.869s	libgomp.so.1	func@0x1dfd4
setWeights._omp_fn.3	214.170s	CLBM	setWeights._omp_fn.3

Figure10: Vtune hotspot results.



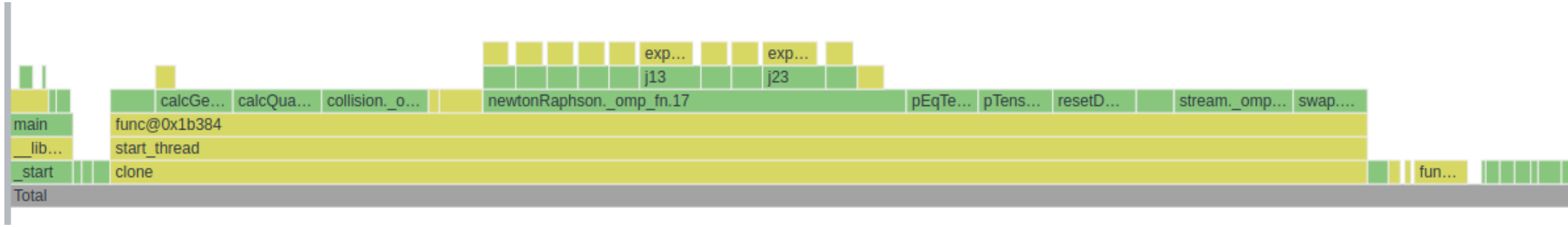


Figure09: Vtune hotspot flame graph.

Strong scaling

CPU name: Intel Xeon Platinum 8360Y CPU @ 2.40GHz
CPU type : Intel Icelake SP processor
Compiler: g++
Compiler Directives: -O3 –march=native –mavx –ftree-vectorize

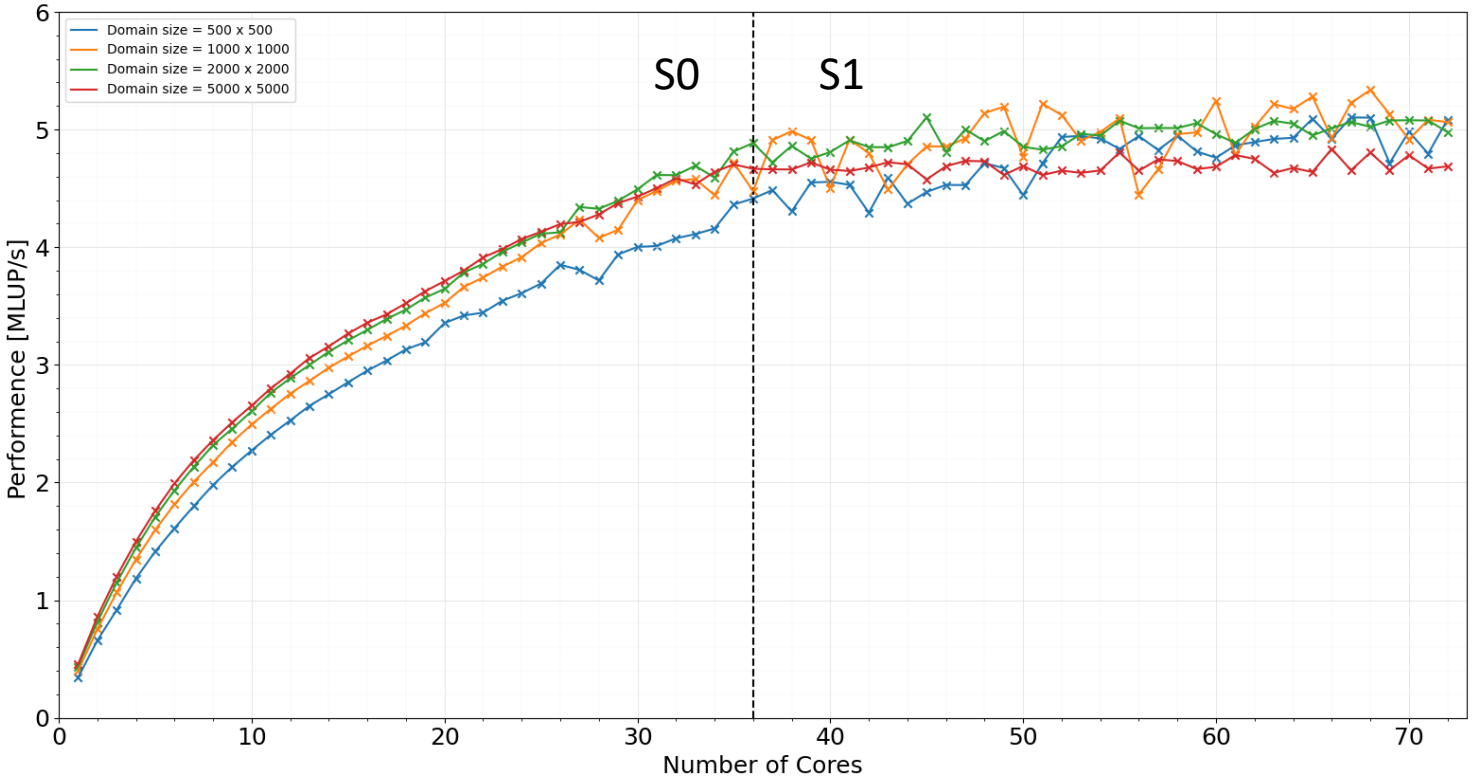


Figure10: Performance saturation in one node.

- 2000 x 2000 elements seems to be good starting point for measurements.

- 500 x 500 domain size does not have performance saturation.
- According to single core performance it fits to L3 cache.

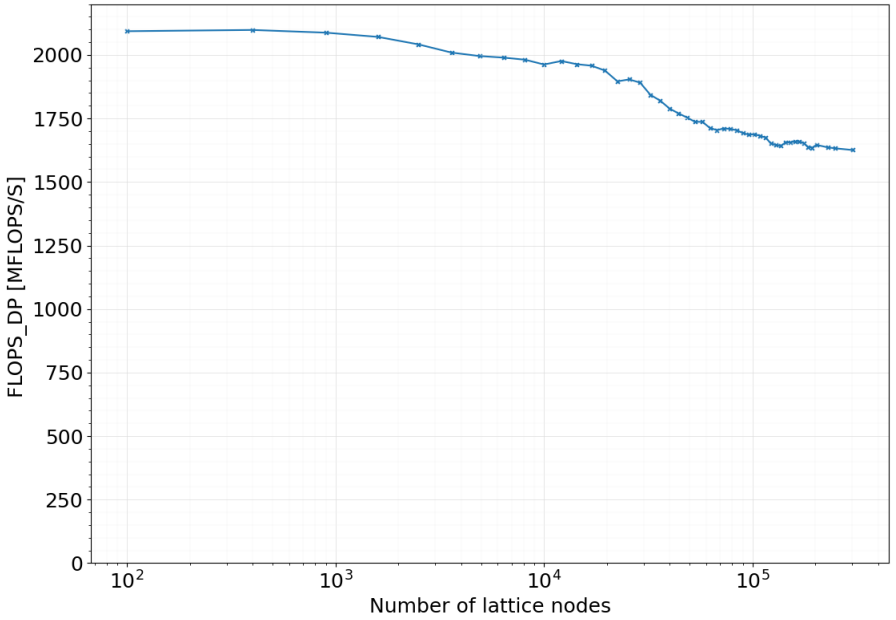


Figure 11: single core performance for fixed 2.4GHz.

Memory bandwidth

CPU name: Intel Xeon Platinum 8360Y CPU @ 2.40GHz
CPU type : Intel Icelake SP processor
Compiler: g++
Compiler Directives: -O3 –march=native –mavx –ftree-vectorize

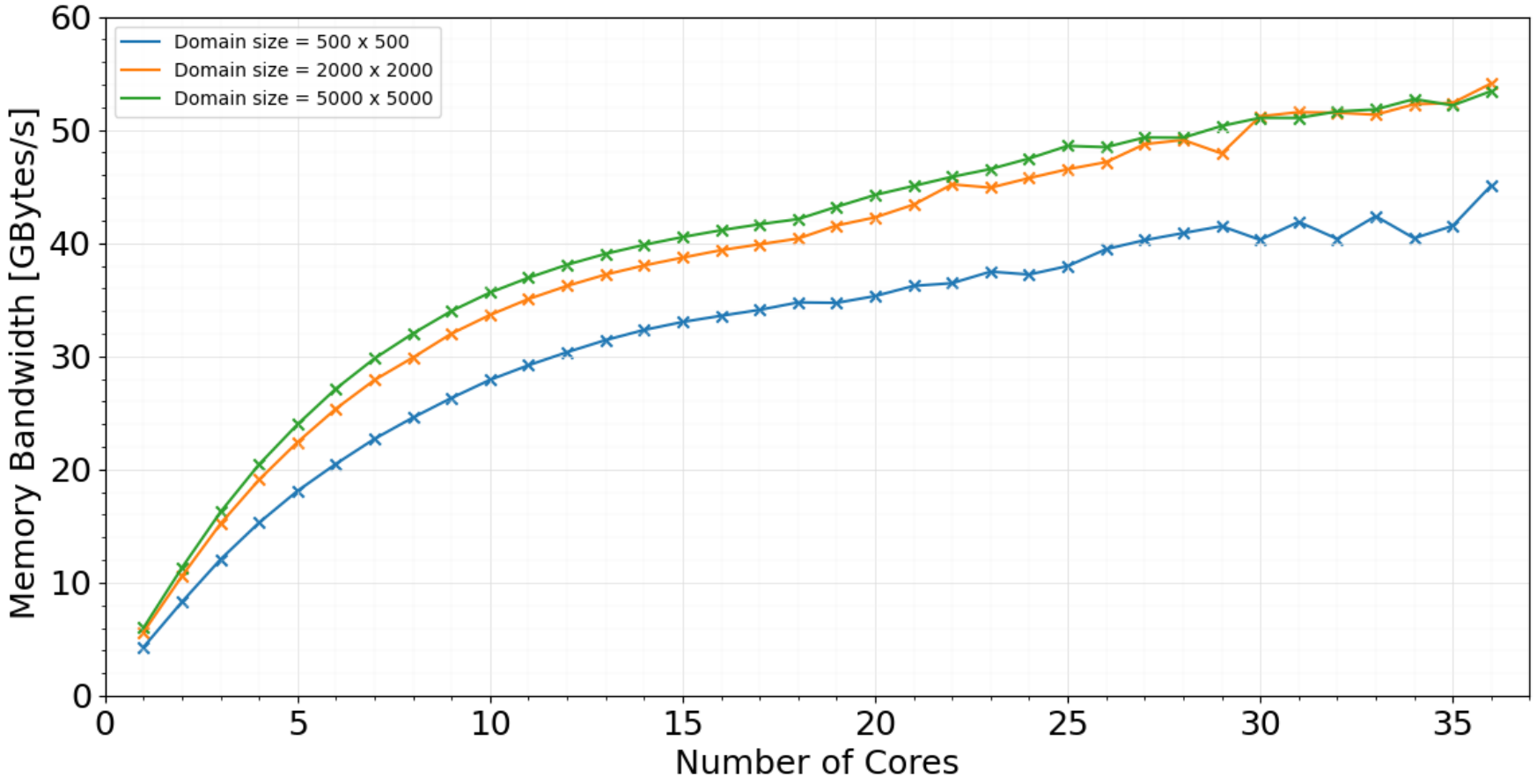


Figure12: Memory bandwidth saturation in first socket.

- Total memory bandwidth has a saturating trend in 1st NUMA domain.
- Generally, LBM solvers are known to be memory bound.[3]

Energy and power

CPU name: Intel Xeon Platinum 8360Y CPU @ 2.40GHz
CPU type : Intel Icelake SP processor
Compiler: g++
Compiler Directives: -O3 –march=native –mavx –ftree-vectorize

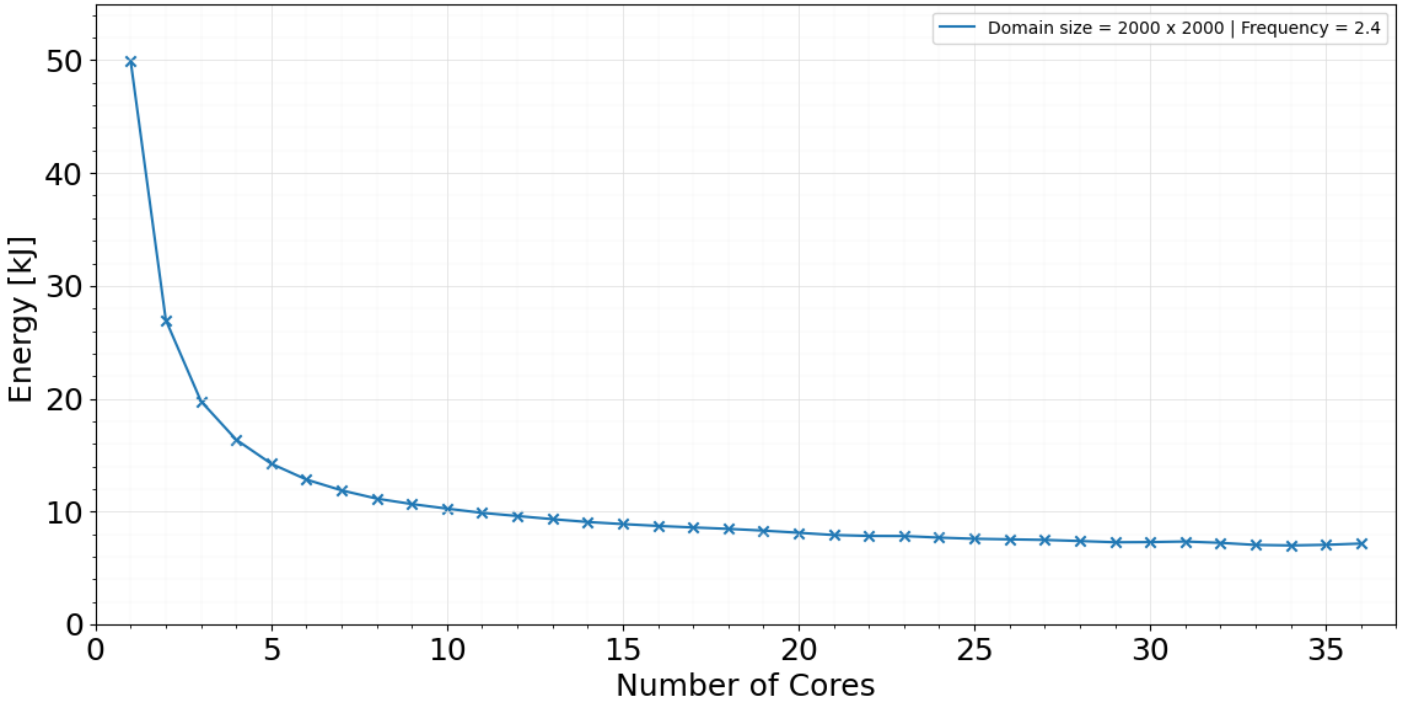


Figure13: Energy variation in first socket.

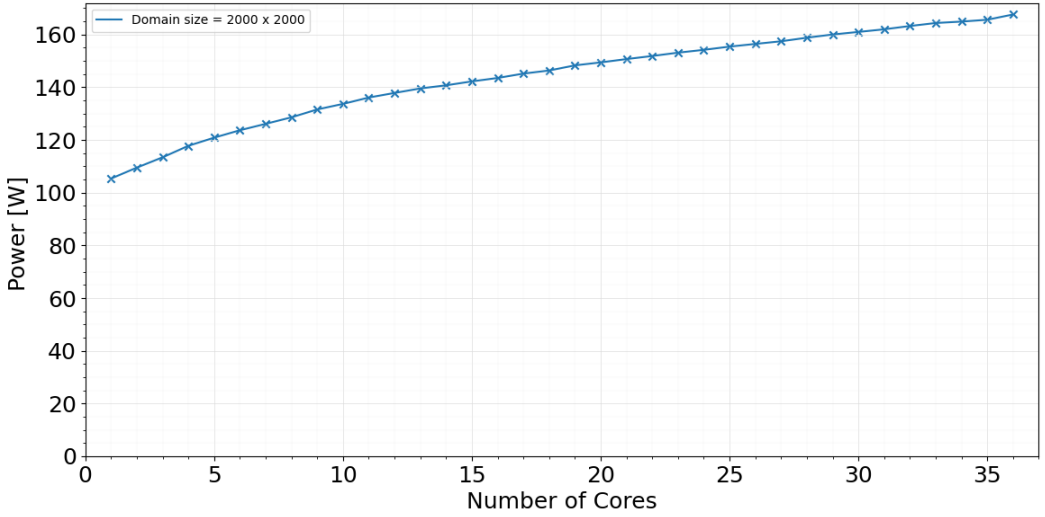


Figure14: Power variation in first socket.

- Total energy decreases with increasing core count.
- Power increase with increasing core count.

More on energy

- Even with the increasing frequencies total energy curves tend to meet at 36 core count.
- newtonRaphson function also has the same trend as the total energy.

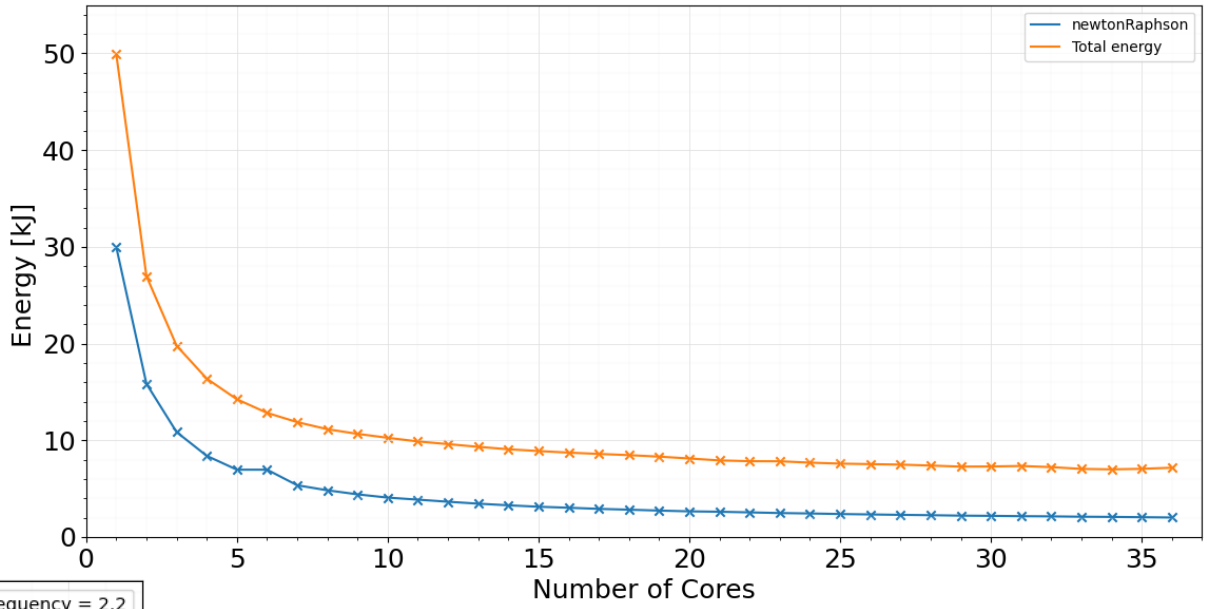


Figure15: Total energy for 2.4 GHz vs energy taken for the newtonRaphson function.

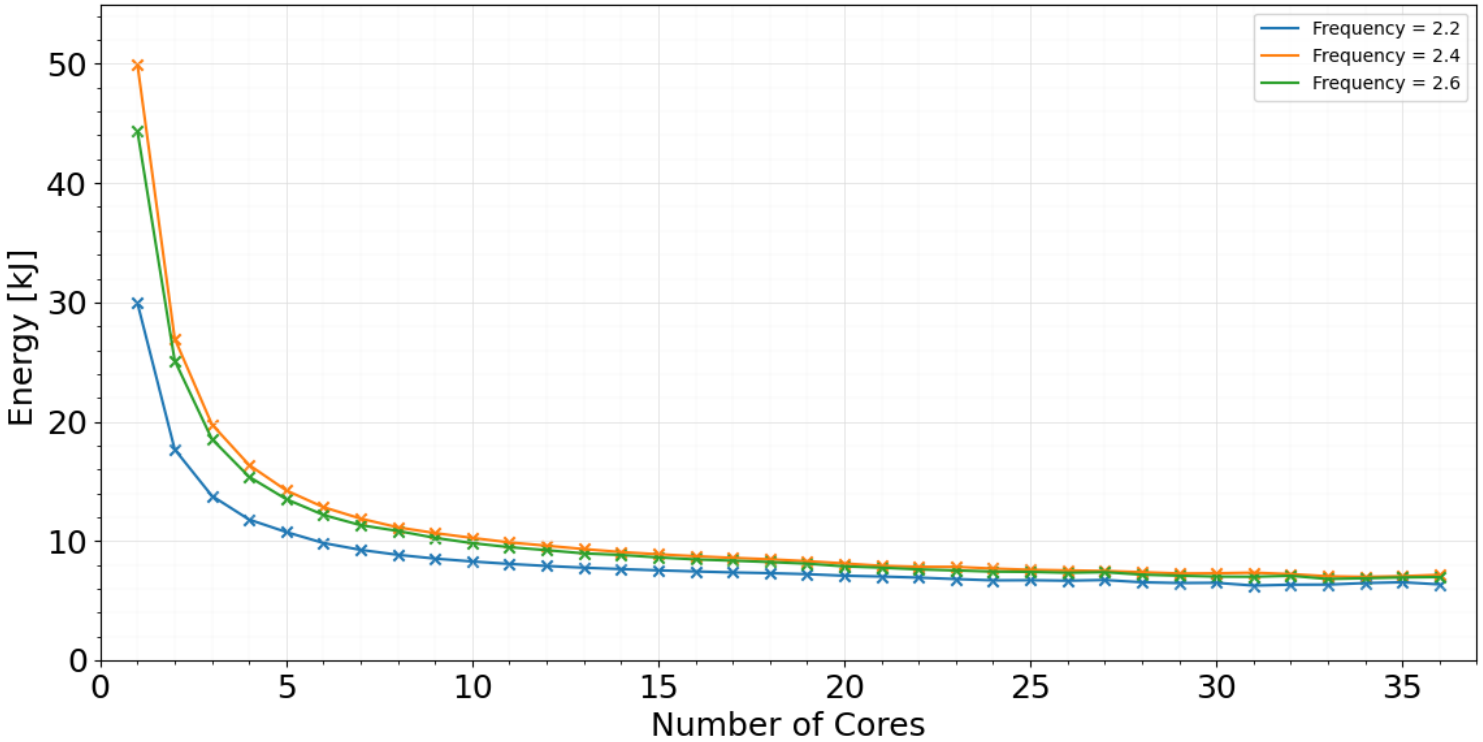


Figure16: Total energy variation for different frequencies for domain size of 2000 x 2000

Remarks and TODOs

- High OpenMP overhead can be observed according to the profiling results.
- Newton Raphson function is the most critical function in CLBM solver.
- According to the physics number of newton Raphson iteration not unique in each domain sizes.
- Solver is memory bound.
- Need to focus on parallelization.
  - Experiments with OpenMP scheduling
  - Barrier effect.
- Investigation of memory bandwidth in entire node.
- Investigation of the effect of different compilers.
- Benchmarking of 3D application.
- Investigation of optimal energy point for the present implementation.

Reference

1. **The Landscape of Parallel Computing Research A View from Berkeley,** 2006
   Electrical Engineering and Computer Sciences, University of California at Berkeley, *Electrical Engineering and Computer Sciences, University of California at Berkeley,*
2. A. A. Mohamad. **Lattice Boltzmann Method Fundamentals and Engineering Applications with Computer Codes**. Springer, 2019
3. Timm Krüger, Halim Kusumaatmaja, Alexandr Kuzmin, Orest Shardt, Goncalo Silva, and Erlend Magnus Viggen. **The Lattice Boltzmann Method**. Springer International Publishing, 2017.

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

COMPRESSIBLE LATTICE BOLTZMANN SOLVER CPU BENCHMARK

MuCoSim WS 2023/24

PHASE: 2

Sudesh Rathnayake

Advisor: Dane Lacey

Introduction: Summary

- Lattice Boltzmann Method(LBM) commonly adopt a structured grid in practice.

"Represented by a regular grid; points on grid are conceptually updated together. It has high spatial locality"' [1]

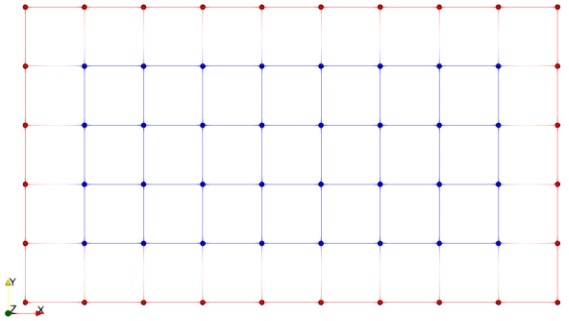


Figure 1: Exemplary structured grid

- Naturally adapted to parallel processes computing. [3].
- The "Collide and Stream" algorithms.

$$\boldsymbol{f}_i^*(\boldsymbol{x},\boldsymbol{t}) = \boldsymbol{f}_i(\boldsymbol{x},\boldsymbol{t}) + \Omega_i(\boldsymbol{x},\boldsymbol{t}) \qquad \boldsymbol{f}_i(\boldsymbol{x}+\boldsymbol{c}_i\Delta\boldsymbol{t},\boldsymbol{t}+\Delta\boldsymbol{t}) = \boldsymbol{f}_i^*(\boldsymbol{x},\boldsymbol{t})$$

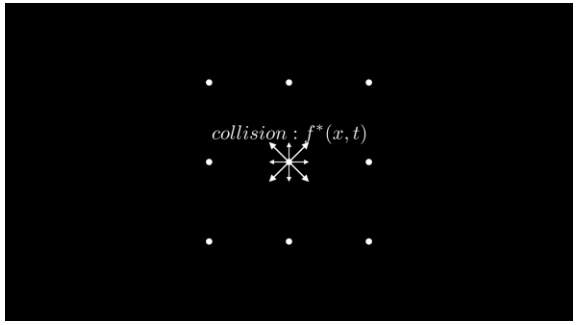


Figure 02: Collide and Stream

- Compressible LBM → $\boldsymbol{f}_i^{\boldsymbol{eq}}$ using discreate entropy function by formulating a minimization problem.
- 2 pdfs' to integrate energy equation apart from the mass, and momentum equations.
- The present solver based on Entropic Lattice Boltzmann Method (ELBM)

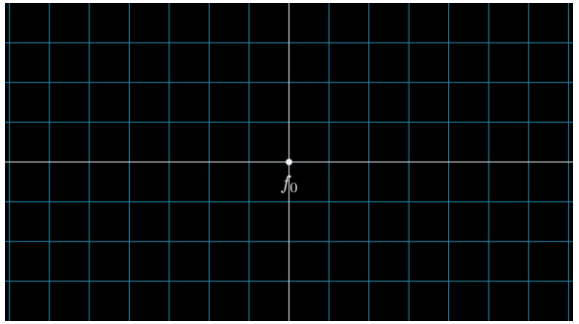


Figure 03: f and g populations in a lattice node

git@gitlab.cs.fau.de:yh54ojyn/mucosim.git

Test system & CLBM solver

Algorithm

- Solver is based on C++ with OpenMP and build using Cmake 3.23.1.
- Input parameter file defines the simulation properties.
- For benchmarking, a 2D shock tube simulation is considered.
- Stencil configuration : D2Q9 (D2Q9 X 2 in one LB node);
- Domain size : 3000 x 3000 lattice nodes

Tools

- LIKWID version : 5.3.0
- LIKWID flags : likwid-perfctr -g MEM/ENERGY/MEM_DP
  : likwid-topology

Compiler

- GNU g++
- compiler flags : -O3 –march=native –mavx –ftree-vectorize

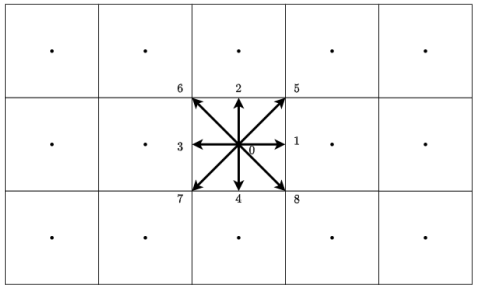


Figure 04:Lattice with D2Q9 stencil

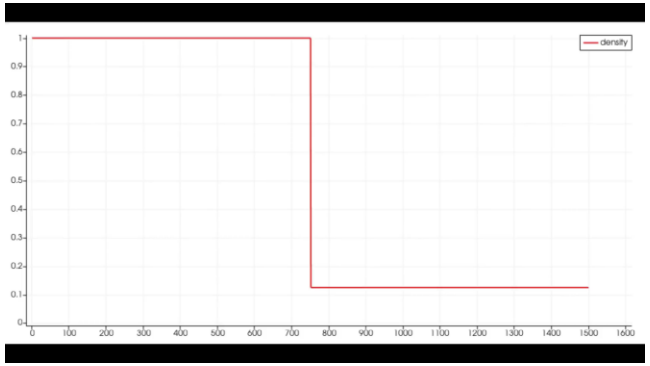


Figure 05: Results for density variation of air inside a 2D shock tube using CLBM solver

Test system

Name	Fritz
Processor	Intel Xeon Platinum 8360Y
Micro architecture	Icelake
Frequency [GHz]	2.4
Cores	72
Sockets	2 (No SMT)
NUMA domains	4
Main memory [GB]	256
Thermal design power [W]	250

Issues: Phase01

- According to single core performance, domain size of 500x500 fits to L3 cache.
- Bad scaling behaviour after the second socket.
- Total memory bandwidth and performance has a saturating trend in 1st NUMA domain.
- expf4; calculation of exponential function using math.h library is critical inside the newtonRaphson kernel.

Grouping: Function / Call Stack

Function / Call Stack	CPU Time ▼	Module	Function (I
expf64	1687.306s	libm.so.6	expf64
↖ j23 ← newtonRaphson._omp_fn.17	261.509s	CLBM	j23(node&)
↖ j13 ← newtonRaphson._omp_fn.17	255.705s	CLBM	j13(node&)
↖ j33 ← newtonRaphson._omp_fn.17	131.396s	CLBM	j33(node&)
↖ func2 ← newtonRaphson._omp_fn.17	128.771s	CLBM	func2(node&, double)
↖ j22 ← newtonRaphson._omp_fn.17	128.538s	CLBM	j22(node&)
↖ j21 ← newtonRaphson._omp_fn.17	127.912s	CLBM	j21(node&)
↖ j11 ← newtonRaphson._omp_fn.17	127.344s	CLBM	j11(node&)
↖ func3 ← newtonRaphson._omp_fn.17	126.925s	CLBM	func3(node const&, double)
↖ j12 ← newtonRaphson._omp_fn.17	126.888s	CLBM	j12(node&)
↖ func1 ← newtonRaphson._omp_fn.17	121.991s	CLBM	func1(node&, double)
calcGeq._omp_fn.5	95.709s	CLBM	calcGeq._omp_fn.5
[No call stack information]	54.618s		
stream._omp_fn.15	679.169s	CLBM	stream._omp_fn.15
collision._omp_fn.9	610.130s	CLBM	collision._omp_fn.9
calcQuasiEqG._omp_fn.8	505.590s	CLBM	calcQuasiEqG._omp_fn.8
resetDDFshitS._omp_fn.11	475.925s	CLBM	resetDDFshitS._omp_fn.11
pTensor._omp_fn.6	425.356s	CLBM	pTensor._omp_fn.6
swap._omp_fn.16	422.428s	CLBM	swap._omp_fn.16
pEqTensor._omp_fn.7	410.548s	CLBM	pEqTensor._omp_fn.7
calcGeq._omp_fn.5	323.667s	CLBM	calcGeq._omp_fn.5
calcFeq._omp_fn.4	256.802s	CLBM	calcFeq._omp_fn.4
func@0x770e4	248.515s	libm.so.6	func@0x770e4
func@0x1dfd4	227.869s	libgomp.so.1	func@0x1dfd4
setWeights._omp_fn.3	214.170s	CLBM	setWeights._omp_fn.3

Figure 6: Vtune hotspot results..

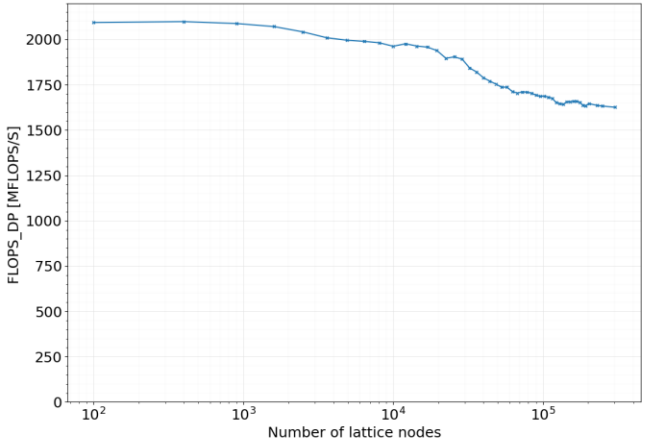


Figure 7: single core performance for fixed 2.4GHz.

BAD IMPLEMENTATION?
NEED CODE OPTIMIZATIONS?
OpenMP scheduling?

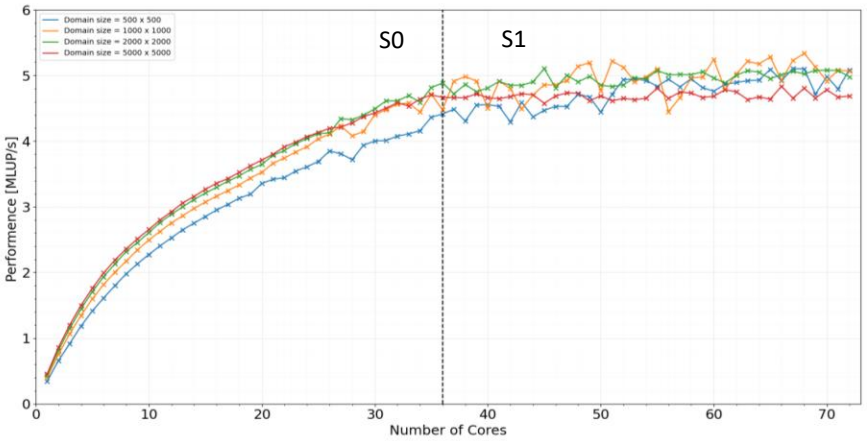


Figure 6: Performance saturation in one node. (phase:01)

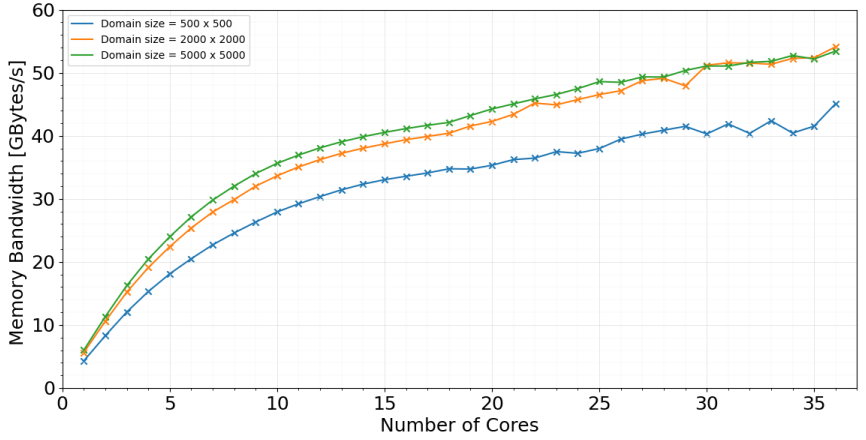


Figure 8: Memory bandwidth saturation in one node. (phase:01)

Issues: Phase01

- According to the initial profiling results; investigated possible issues in newtonRaphson kernel.

- Repeated calls for expf4 unnecessarily.
  - ✓ Since it's the hotpost, investigated the possible code optimizations.
  - ✓ Therefore, modified the newtonRapson subroutines.

- Further, experimented OpenMP scheduling chunk size.
  - ✓ High OpenMP overhead for small domain sizes.
  - ✓ Tried performance vs chunk size investigations
  - ✓ Use the static sheduling.
  - ✓ Found out that peformance is better in default settings compared to settings with chunk size.

- Ready for phase to tests.

Strong scaling.

- Better strong scaling results compared to phase 01
- Performance saturation in 1st NUMA domain.
- Maximum performance of ~26 MLUP/S with using full node.

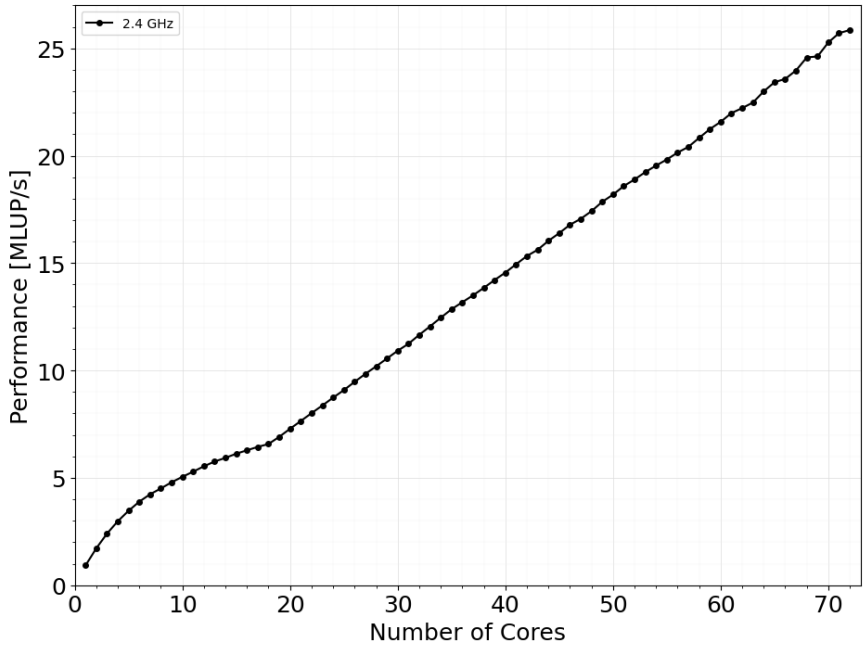


Figure 10: Memory bandwidth measurements in one node at 2.4 GHz

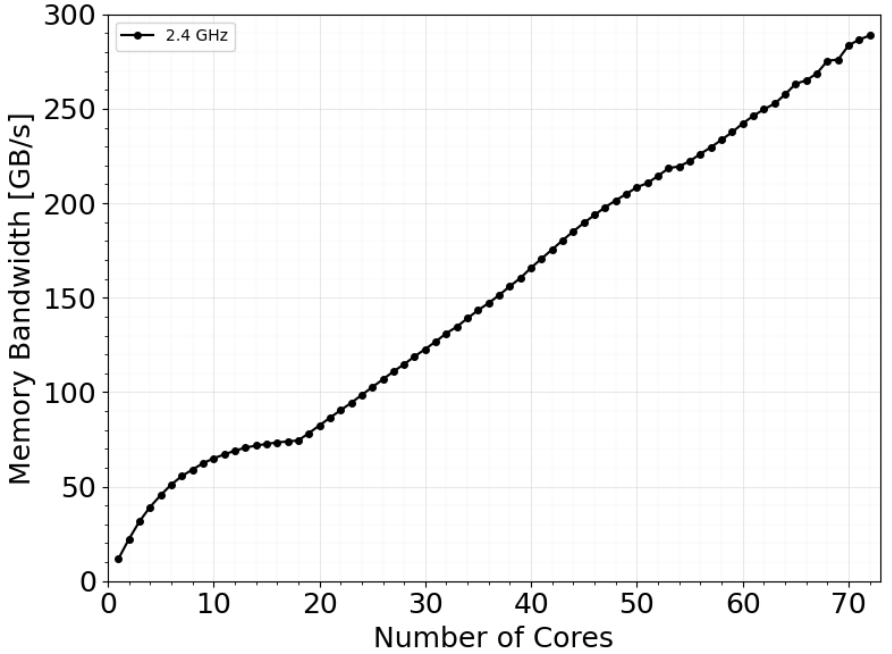


Figure 9: Performance measurements in one node at 2.4 GHz

- Memory Bandwidth also saturates in 1st NUMA domain. (74.5 GB/s).
- Single core performance is ~3x compared to phase 01.

Profiling

Elapsed Time: 114.938s

- CPU Time: 7763.840s
- Total Thread Count: 72
- Paused Time: 0s

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time	% of CPU Time
stream._omp_fn.18	CLBM	769.654s	9.9%
collision._omp_fn.12	CLBM	685.792s	8.8%
resetDDFshitS._omp_fn.14	CLBM	631.249s	8.1%
swap._omp_fn.23	CLBM	591.541s	7.6%
calcQuasiEqG._omp_fn.10	CLBM	576.830s	7.4%
[Others]	N/A*	4508.774s	58.1%

**N/A is applied to non-summable metrics.*

Figure 11: Vtune Hotspot results.

- Now the stream and collision kernels plays a critical role as in standard LBM schemes.
- According to literature, there was a strong focus for the root finding scheme considering compressible LBM performance [4].

- Focused kernels considering energy measurements
  - ✓ Stream
  - ✓ Collision
  - ✓ calcQuasiEqG
  - ✓ newtonRaphson

Welcome | hotspots_sudesh_72.f0454.nhr.fau.de

Hotspots

Analysis Configuration | Collection Log | Summary | Bottom-up | Caller/Callee | Top-down Tree | Flame Graph | Platform

Grouping: Function / Call Stack

Function / Call Stack	CPU Time	Module	Function (Full)	Source File	Start Address
stream._omp_fn.18	769.654s	CLBM	stream._omp_fn.18		0x404ce0
collision._omp_fn.12	685.792s	CLBM	collision._omp_fn.12		0x4042d0
resetDDFshitS._omp_fn.14	631.249s	CLBM	resetDDFshitS._omp_fn.14		0x404570
swap._omp_fn.23	591.541s	CLBM	swap._omp_fn.23		0x406e60
calcQuasiEqG._omp_fn.10	576.830s	CLBM	calcQuasiEqG._omp_fn.10		0x403e40
expf64	574.681s	libm.so.6	expf64		0xfb40
calc_parameters_from_PDF._omp_fn.5	513.967s	CLBM	calc_parameters_from_PDF._omp_fn.5		0x406fe0
pTensor._omp_fn.8	472.088s	CLBM	pTensor._omp_fn.8		0x403900
pEqTensor._omp_fn.9	459.827s	CLBM	pEqTensor._omp_fn.9		0x403ba0
func@0x1dfd4	456.211s	libgomp.so.1	func@0x1dfd4		0x1dfd4
calcFeq._omp_fn.6	312.084s	CLBM	calcFeq._omp_fn.6		0x403380
calcGeq._omp_fn.7	307.521s	CLBM	calcGeq._omp_fn.7		0x403520
allfuncs	305.787s	CLBM	allfuncs(node const&, double, double*)		0x409d50
newtonRaphson._omp_fn.24	297.915s	CLBM	newtonRaphson._omp_fn.24		0x409fb0
setWeights._omp_fn.4	269.078s	CLBM	setWeights._omp_fn.4		0x403280
calcRelaxationFactors._omp_fn.11	222.741s	CLBM	calcRelaxationFactors._omp_fn.11		0x4041b0
pow	171.325s	libm.so.6	pow		0xf3f0
func@0x1de24	52.950s	libgomp.so.1	func@0x1de24		0x1de24
func@0x401bb0	25.340s	CLBM	func@0x401bb0		0x401bb0
func@0x770e4	21.810s	libm.so.6	func@0x770e4		0x770e4
createFlags._omp_fn.0	8.784s	CLBM	createFlags._omp_fn.0		0x402d20
func@0x1ca74	6.797s	libgomp.so.1	func@0x1ca74		0x1ca74

Figure 12: Vtune call stack

Energy and power measurements; main

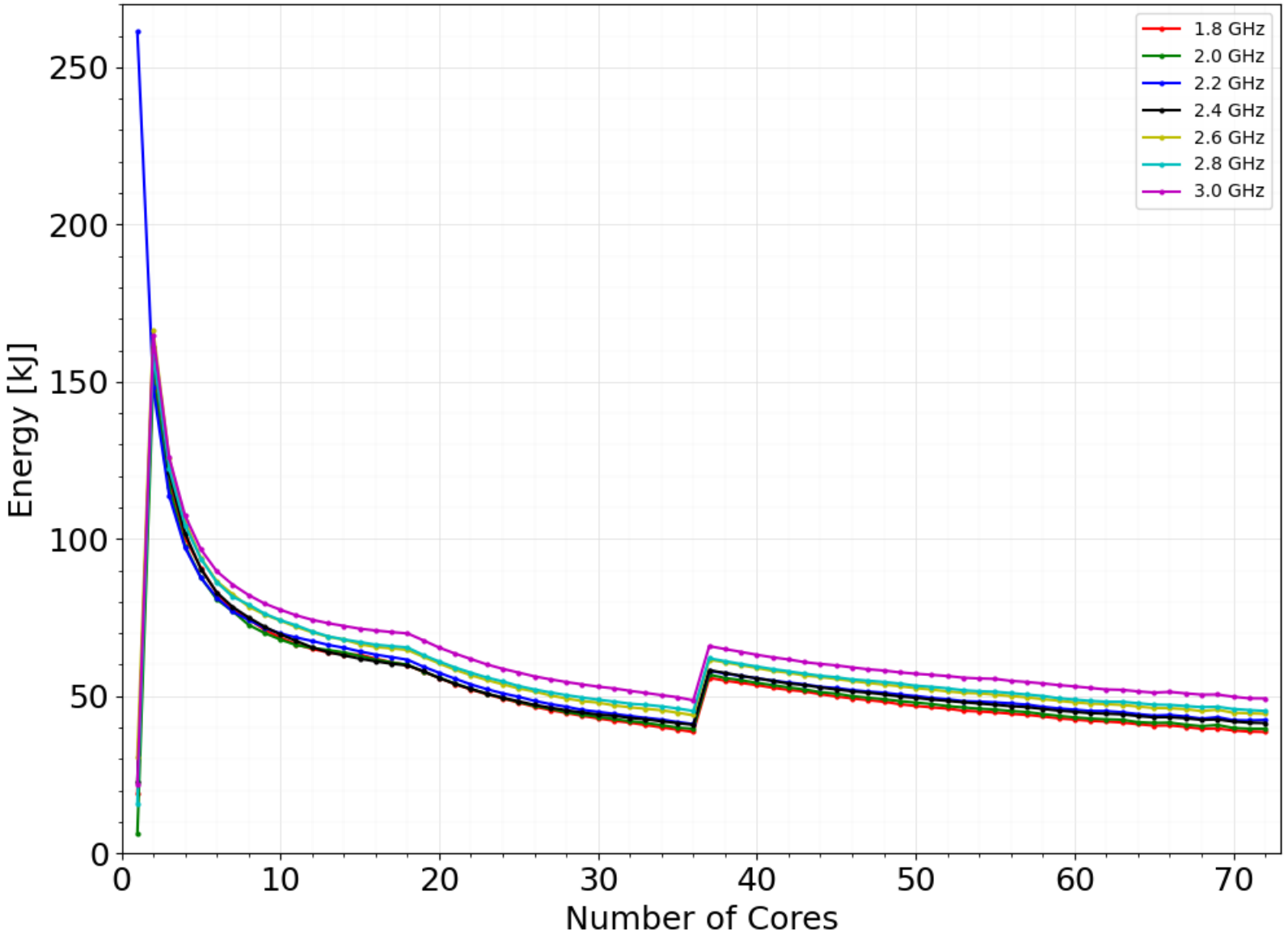


Figure 13: Energy measurements in one node

- An unusual behaviour can be observed at single core power measurement.
  - ✓ This behaviour persists for several data samples
  - ✓ However, for 2.2 GHz, the intended energy measurements can be obtained.
- The energy values have a saturating optimum point at 1st NUMA domain before it reaches a second optimum point at 1st socket.
- Sudden energy jump when moving to second socket before decrease in a quite linear fashion in second socket.
- The frequencies (GHz) 1.8, 2.0, 2.2 and 2.4 shows the lowest CPU energy measurements in first socket.

Energy and power measurements; main

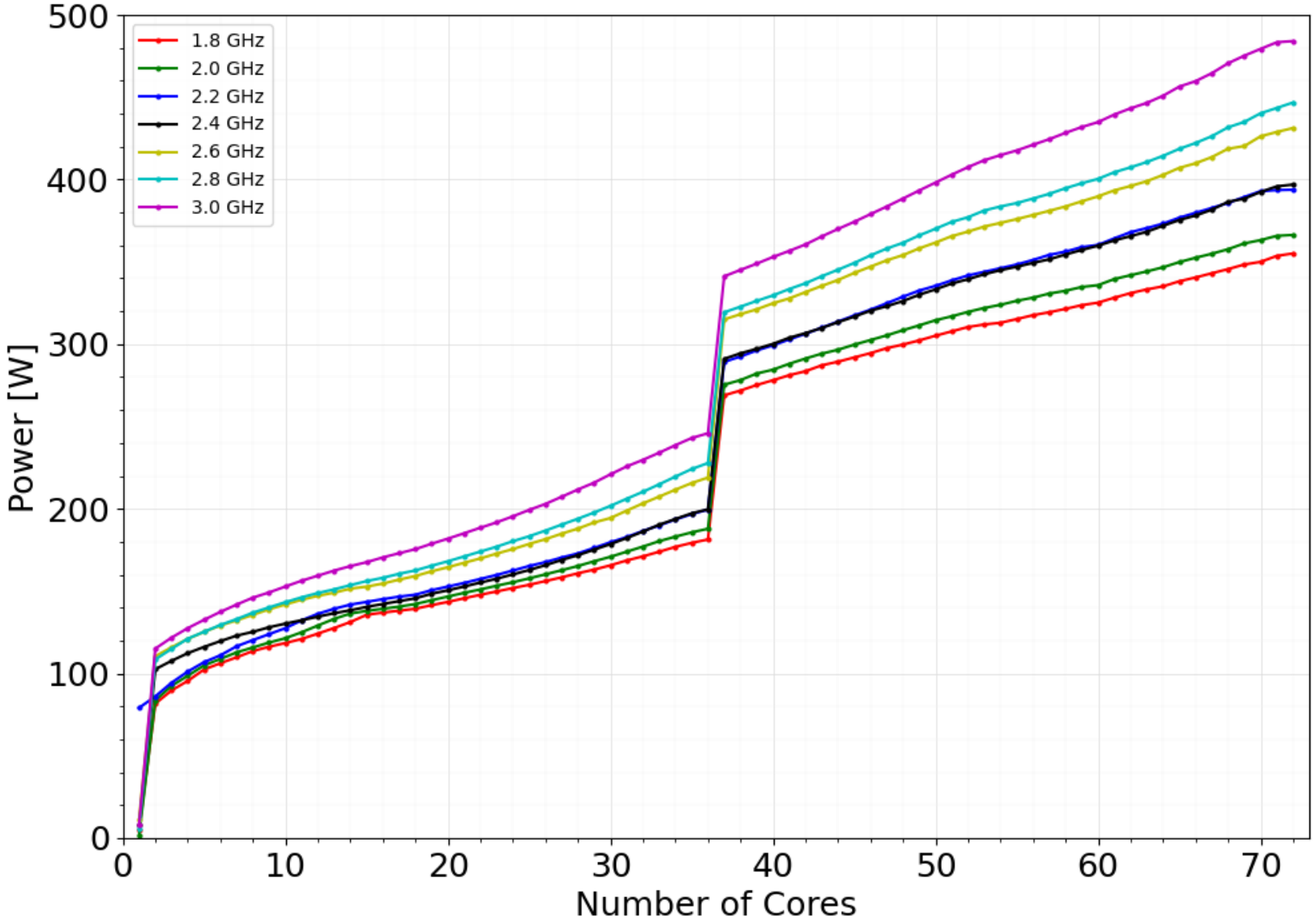


Figure 14: Power measurements in one node

- An unusual behaviour also can be observed at first core power measurment.
- LIKWID-POWERMETER also produced the same results.
- Sudden power jump when moving to second socket.

DRAM;Energy and power measurements; main

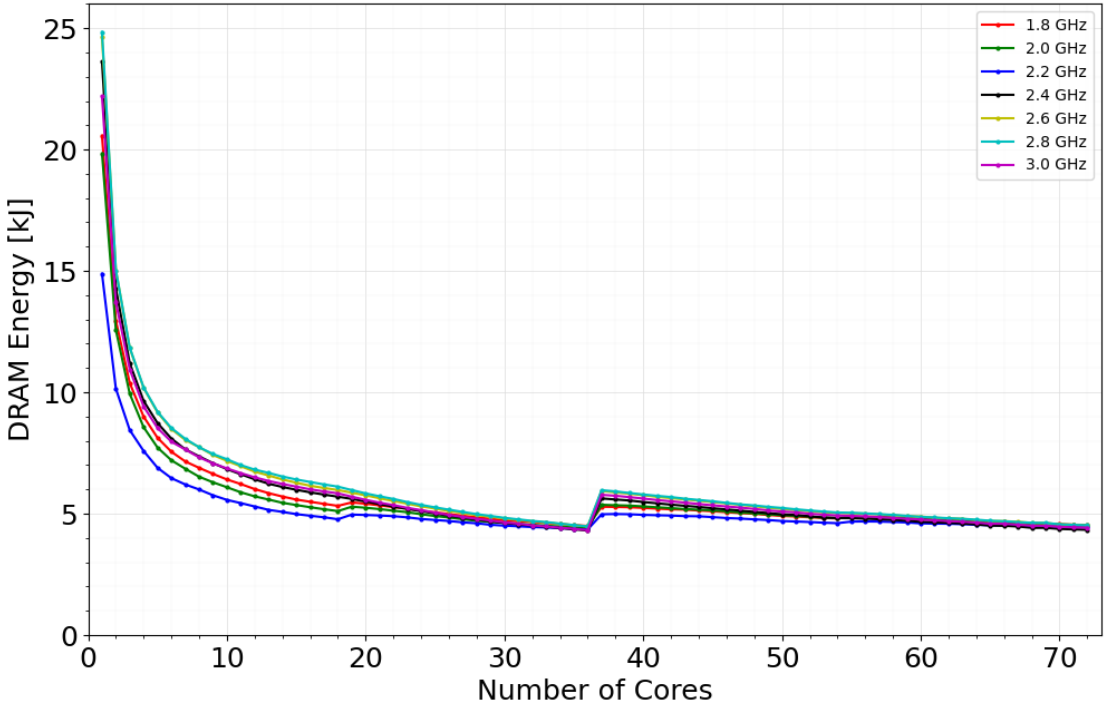


Figure 15: DRAM Energy measurements in one node

- First NUMA domain there is a saturating trend considering DRAM power.

DRAM energy measurements do not show an unexpected behaviour at single core measurements all the time.

2.2 GHz seems to be the lowest energy curve for a wider domain.

The DRAM energy contribution to the total energy is in the range of 5%-11%.

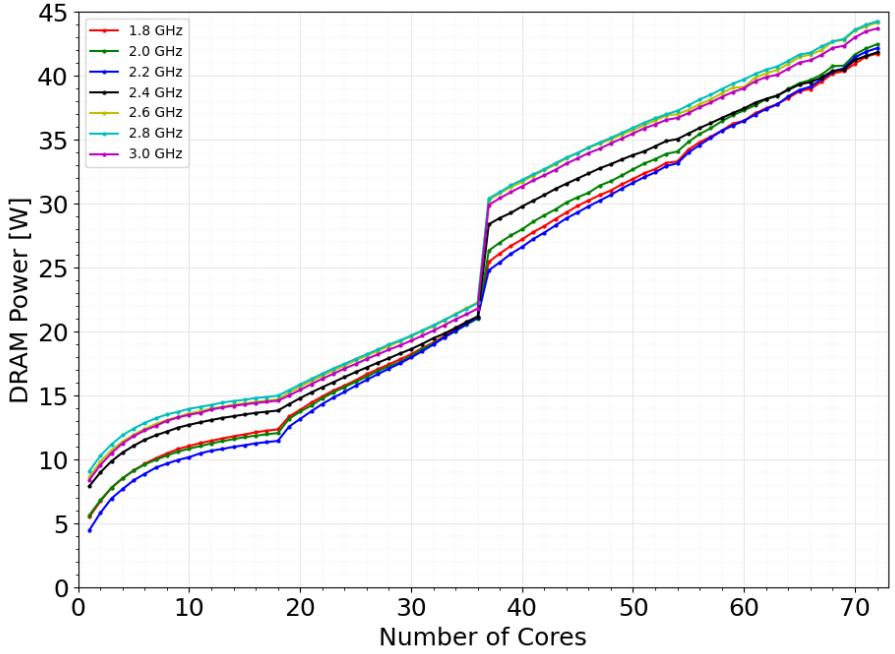


Figure 16: DRAM Power measurements in one node

Energy Delay Product (EDP)

- Z-plot considers dissipated energy versus any suitable performance metric for a given program.
  - ✓ EDP is the gradient of a line passing through in z-plot.
- First socket measurements are taken for the Z-plot.

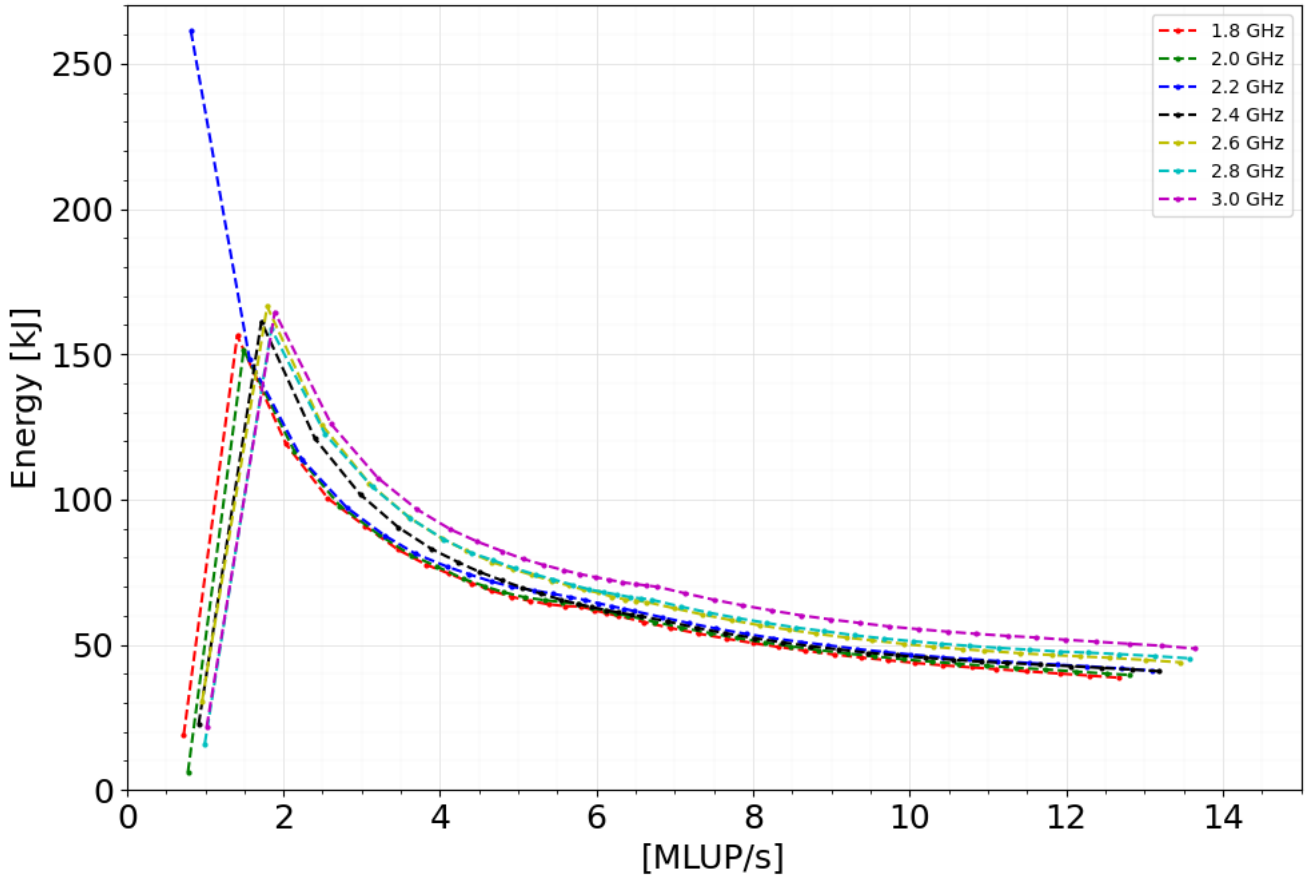


Figure 18: Z-plot considering one socket.

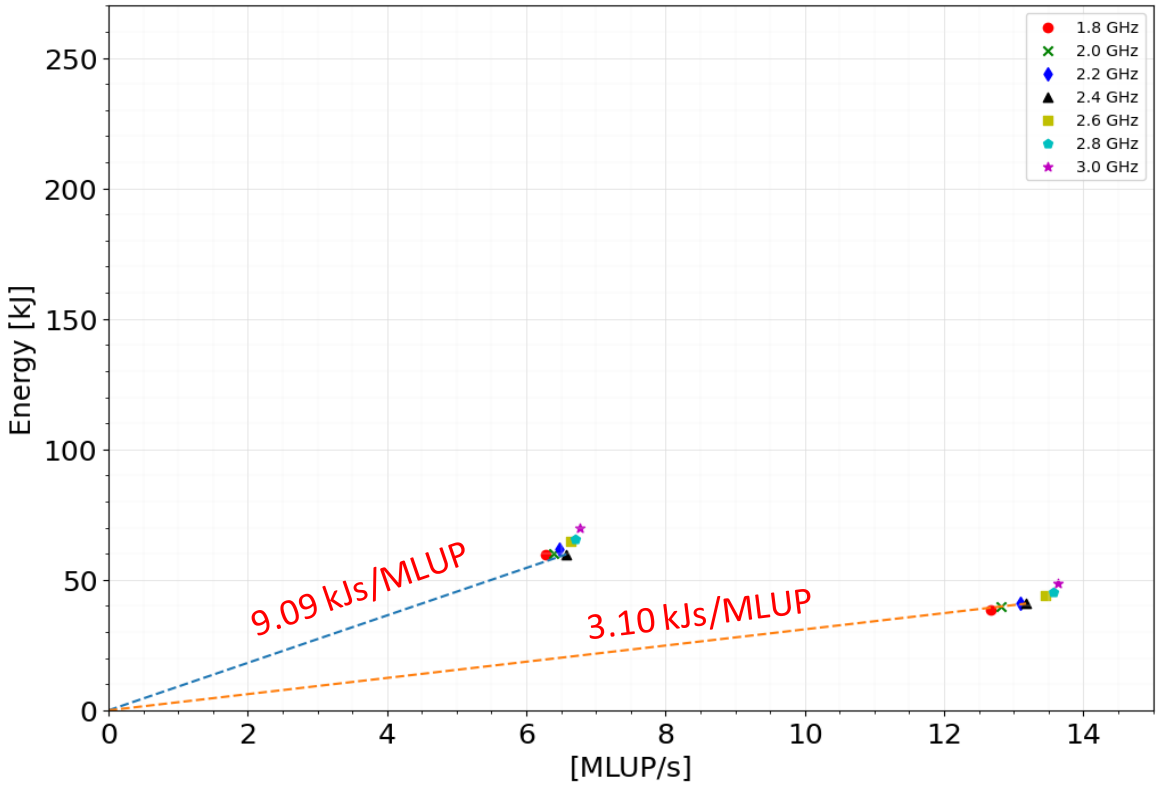


Figure 17: EDP related to 18 and 36 cores at 2.4 GHz

- This verifies the earlier point relating energy with first four frequencies.

Hotspots measurements

- Considered the stream, collision, calcQuasiEqG and newtonRaphson kernels.
- Observed the performance, memory bandwidth saturations and energy contributions.

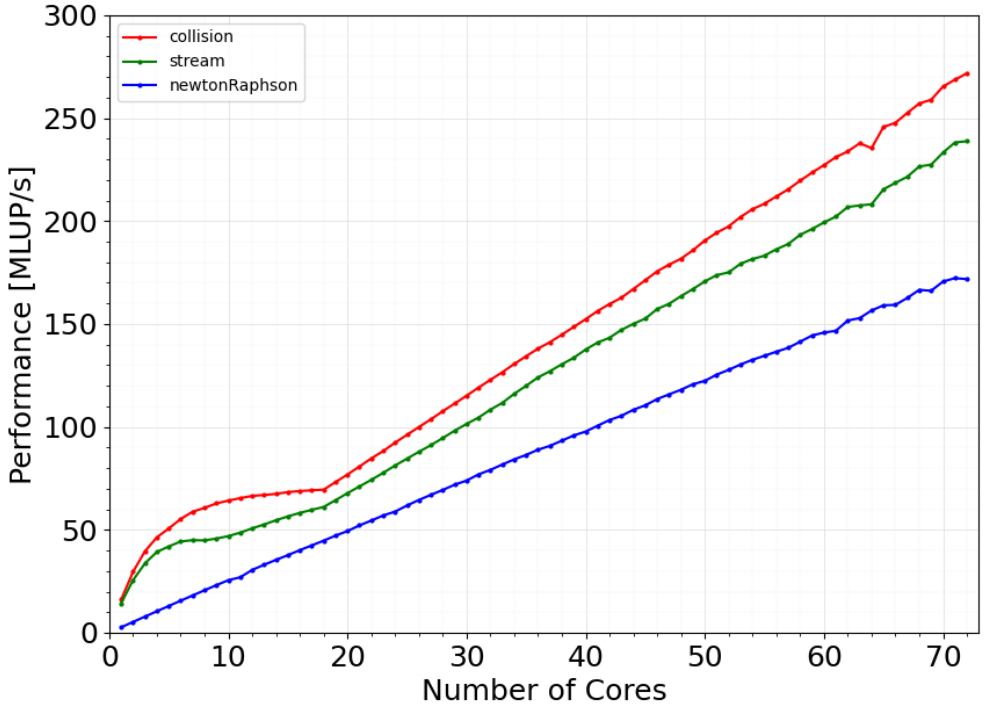


Figure 19: Performance vs. core count

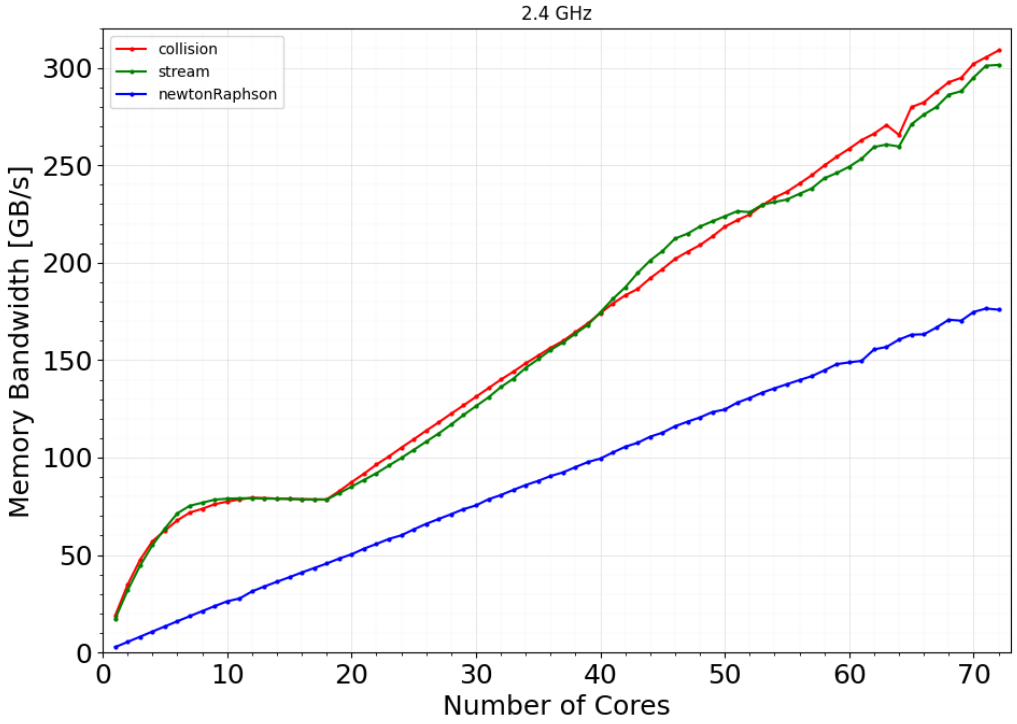
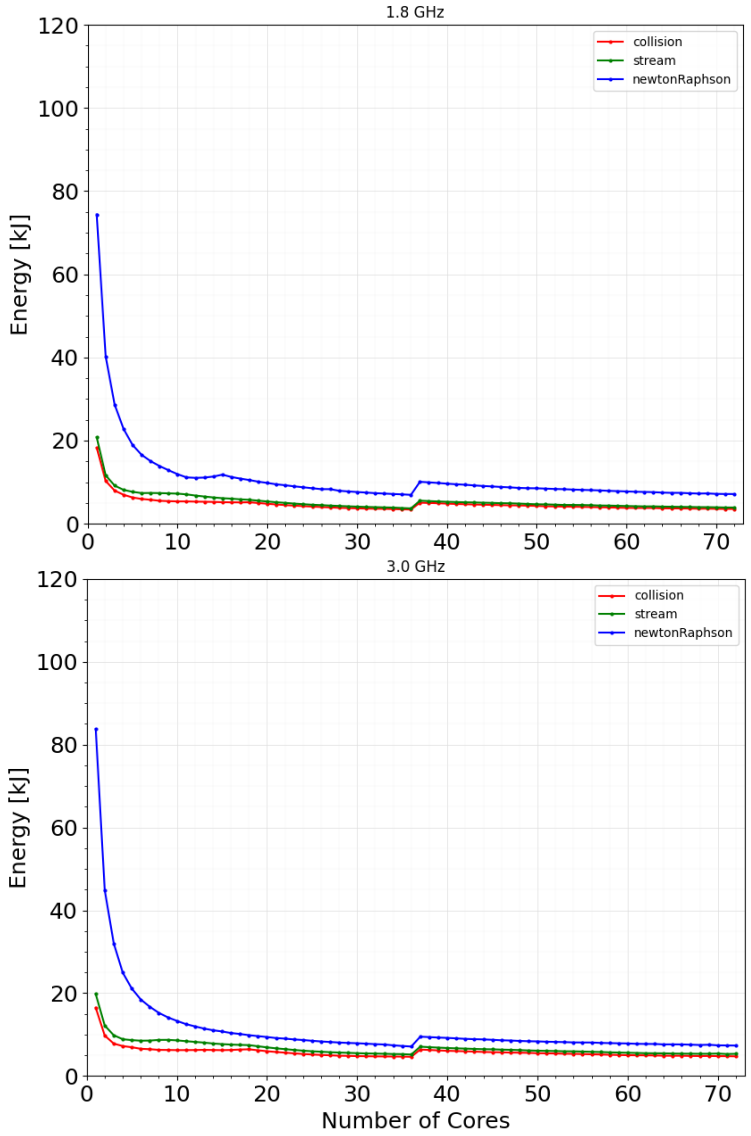


Figure 20: Memory bandwidth vs. core count

- newtonRaphson kernel does not show any saturation in full node.
- collision and streaming shows the saturating trend inside the first NUMA doamin.
- Note that here, MLUP/s have high values. Because we calcultate the same metric with respect to the hotspot runtime. (total runtime >> hotspot runtime; for 2.7 Billion lattice updates)
- Cannot use Flop/s for the representation. Because stream does not have floating point operations.

Hotspots measurements…

- The previously observed, unexpected measurement point at single cores does not appear in energy plots.



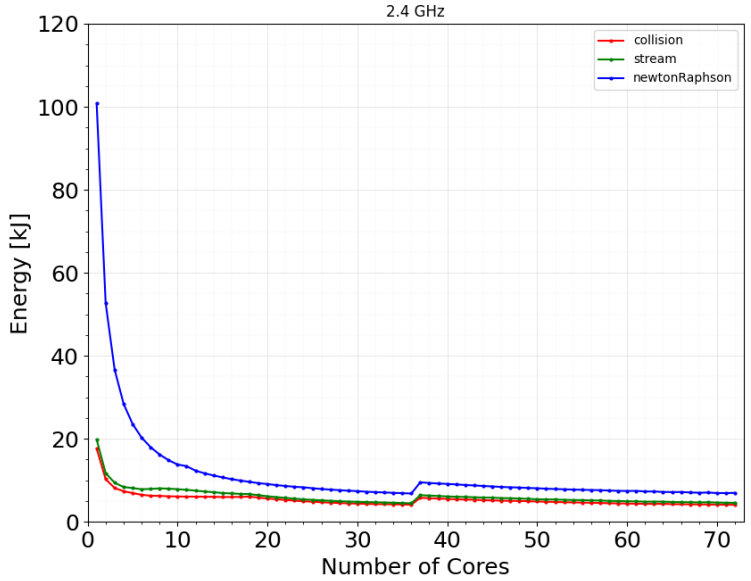


Figure 21: Energy vs. core count

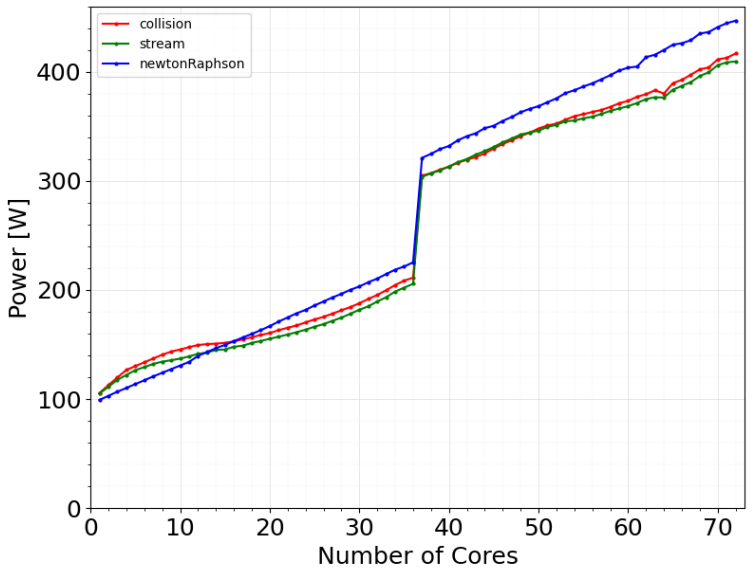


Figure 22: Power vs. core count

- All three kernels contributes 9% - 16% to the main energy measurements.
- newtonRaphson has the highest contribution while showing a scalable performance within the node.
- Both the highest hotspot ranks seems to be having similar variation in terms of power and energy.

Roofline modelling

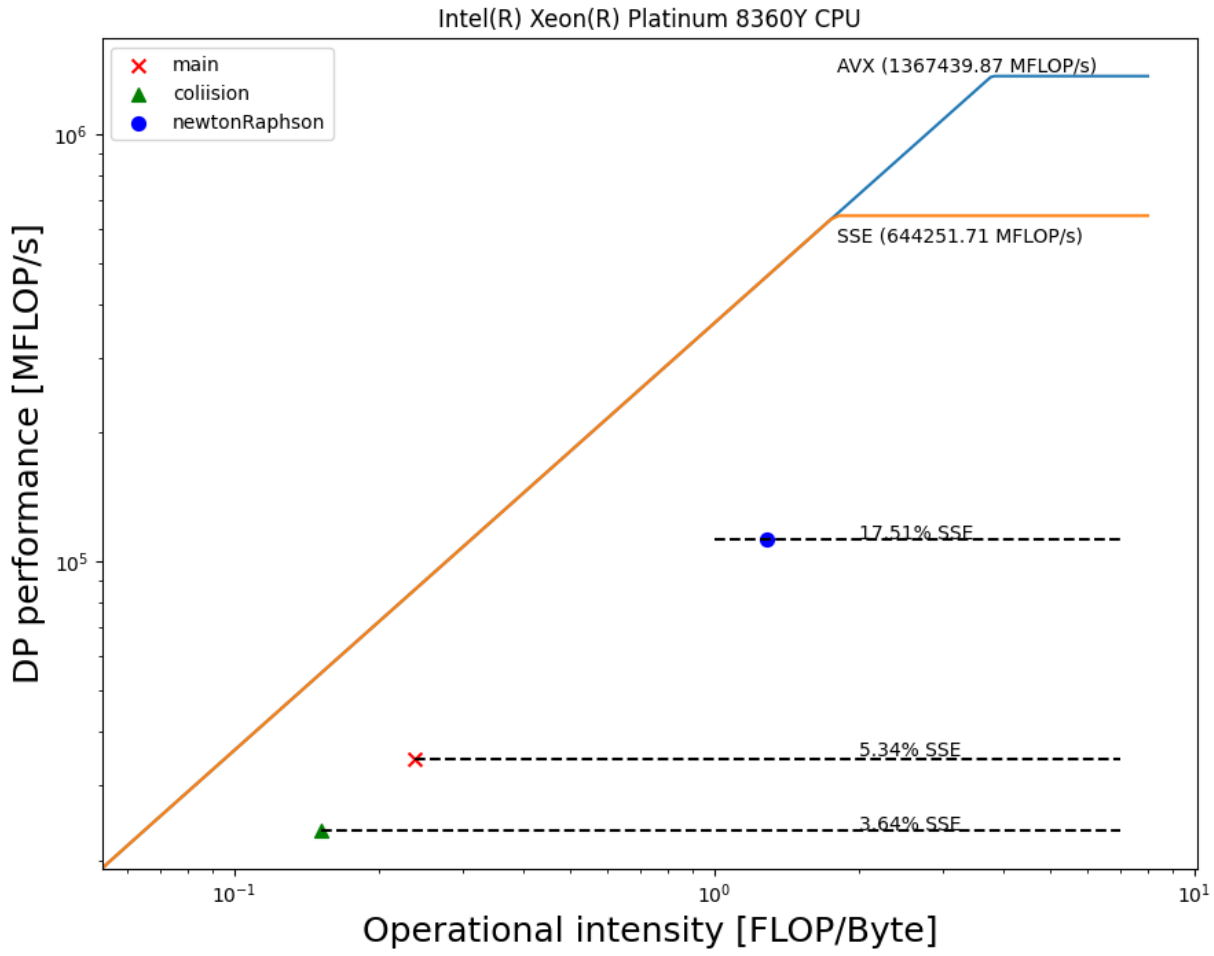


Figure 22: Roofline diagram.

- LIKWID-bench tool is utilized to construct the empherical roofline diagram [7].
- Maximum performance is related to the AVX FLOPS.
- Maximum data throughput is related to the likwid-bench load_avx
- Here the performance metric is double precision performance.
- Stream kernal does not have any flops.
- CLBM application is close to memory bound.

Remarks.

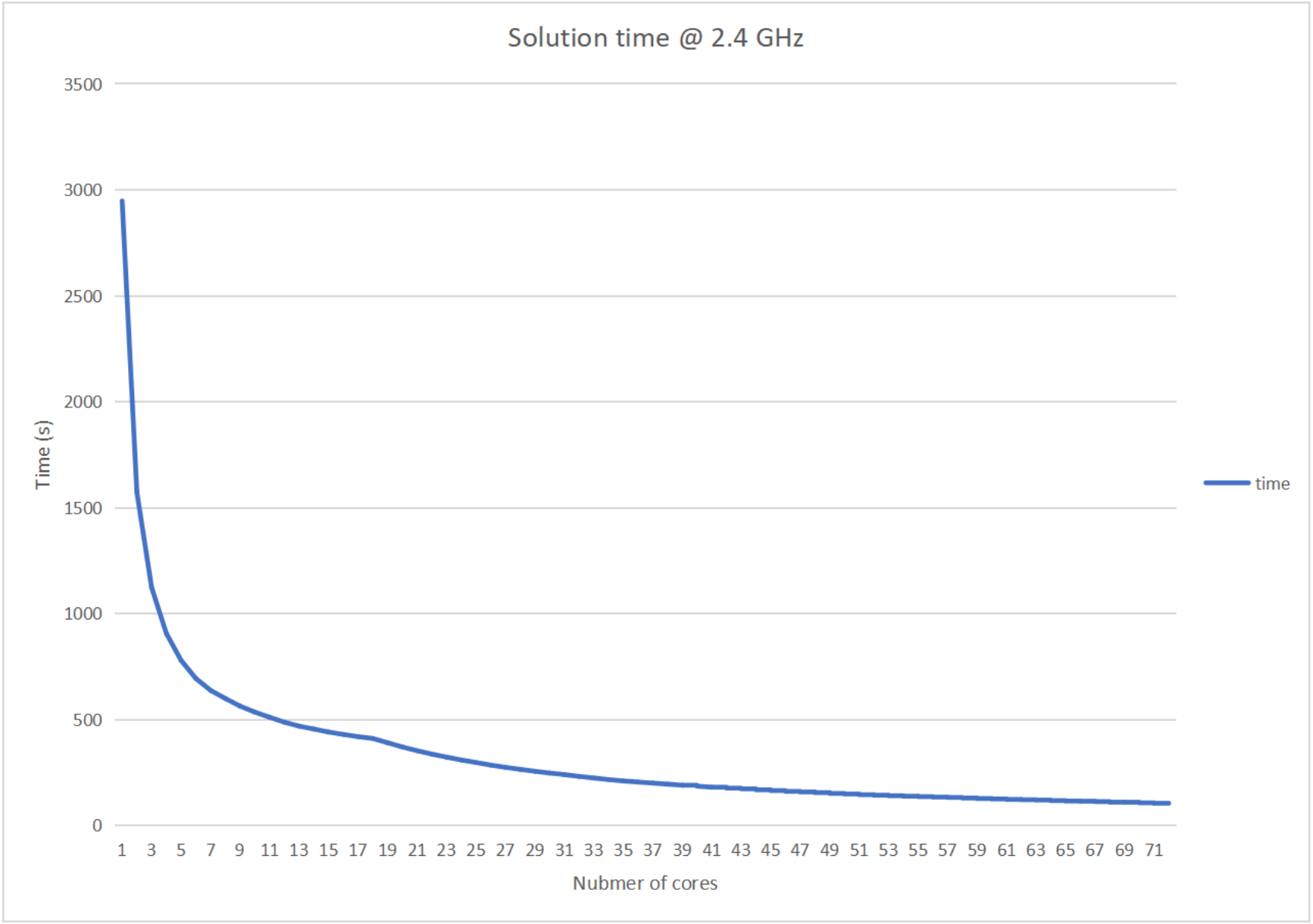
- This is the first kind of benchmarking effort considering LBM in compressible supersonic flow
- CLBM 2D test application shows scalable performance.
- Solver is close to memory bound. According to the roofline analysis, the application has arbitrary horizontal roofs.
  - ✓ May be more opportunity to increase bandwidth utilization, for example by using SIMD instructions in the newtonRaphson routine.
- Optimal energy can be observed at the first socket in fritz node considering 1.8 GHz, 2.0 GHz, 2.2 GHz and 2.4 GHz
  - ✓ EDP results verified the point above.
- DRAM energy contribution to the total energy is 5%-11%.
- Root finding scheme has less significance in terms of performance, even though root finding algorithm was the highlight in previous literature regarding, this LBM regime.

Reference

1. The Landscape of Parallel Computing Research A View from Berkeley, 2006 Electrical Engineering and Computer Sciences, University of California at Berkeley, Electrical Engineering and Computer Sciences, University of California at Berkeley,
2. A. A. Mohamad. Lattice Boltzmann Method Fundamentals and Engineering Applications with Computer Codes. Springer, 2019
3. Timm Krüger, Halim Kusumaatmaja, Alexandr Kuzmin, Orest Shardt, Goncalo Silva, and Erlend Magnus Viggen. The Lattice Boltzmann Method. Springer International Publishing, 2017.
4. Jonas Latt, Christophe Coreixas, Joël Beny, and Andrea Parmigiani. Efficient supersonic flow simulations using lattice boltzmann methods based on numerical equilibria. Philosoph ical Transactions of the Royal Society A:Mathematical, Physical and Engineering Sciences,378(2175):20190559, jun 2020. 5.
5. https://blogs.fau.de/hager/archives/tag/energy
6. Ayesha Afzal,Georg Hager,Gerhard Wellein,SPEChpc 2021 Benchmarks on Ice Lake and Sapphire Rapids Infiniband Clusters: A Performance and Energy Case Study, SC-W '23: Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis,November 2023, Pages 245–1254,https://doi.org/10.1145/3624062.3624197
7. https://github.com/RRZE-HPC/likwid/wiki/Tutorial:-Empirical-Roofline-Model

Thank you!

Appendix


Solution time @ 2.4 GHz
3500
3000
2500
2000
1500
1000
500
0
Time (s)
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71
Nubmer of cores
time

OpenMP_chunk_size