Report - Parallel Computing and Data Analysis

Marco Di Bello

August 31, 2024

1 Introduction

1.1 The aim

GADGET-2 is a powerful simulation code used for cosmological N-body and hydrodynamical studies, known for its efficiency and scalability on parallel computing systems. Our aim is to test the strong scalability of GADGET-2 by running a simulation of a 50 Mpc/h box containing 64^3 particles, using 2, 4, 8, 16, and 32 processors. This test will help us evaluate how well the code performs as we increase the number of processors while keeping the problem size constant.

1.2 Compilation

In order to compile and prepare the GADGET-2 code for execution, we followed a straightforward process adapted for our BladeRunner cluster environment. We began by updating the `lcdm_gas.Makefile` from the parameterfiles directory to include the specific compiler settings for our cluster. This involved copying the file to the main source code directory and renaming it to Makefile.

Next we performed a clean build to remove any previous object files and executables using the `make clean` command. We then compiled the code by running the `make` command, which compiled all the source code files based on the updated Makefile settings.

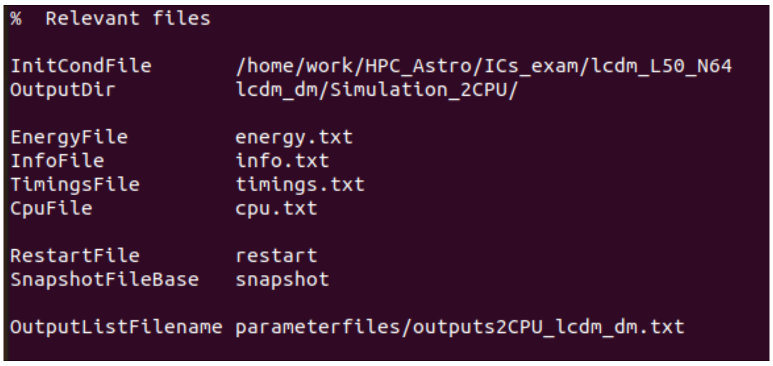
This process resulted in the creation of the `Gadget2` executable, which is now ready for use in our simulations.

2 Simulations

2.1 Parameter files

The first part of the GADGET-2 parameter file includes several crucial settings that define input/output paths and filenames for various outputs generated during the simulation. Below there is an explanation of each parameter:

- **InitCondFile**: This parameter specifies the path to the initial conditions file, which contains the starting data for the simulation. In this



```
%  Relevant files

InitCondFile       /home/work/HPC_Astro/ICs_exam/lcdm_L50_N64
OutputDir          lcdm_dm/Simulation_2CPU/

EnergyFile         energy.txt
InfoFile           info.txt
TimingsFile        timings.txt
CpuFile            cpu.txt

RestartFile        restart
SnapshotFileBase   snapshot

OutputListFilename parameterfiles/outputs2CPU_lcdm_dm.txt
```

Figure 1: Relevant files

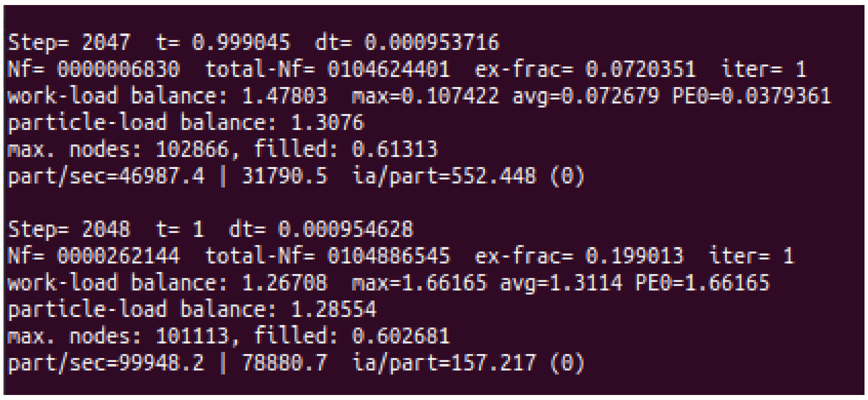
case, it is set to `/home/work/HPC_Astro/ICs_exam/lcdm_L50_N64`, indicating the file containing the initial particle positions, velocities, and other properties.

- **OutputDir**: This parameter defines the directory where all output files generated during the simulation will be stored. For this run, it is set to `lcdm_dm/Simulation_2CPU/`, which means all simulation outputs will be saved in this specific folder. This saving process will need to be repeated for all directories representing the simulation with 2, 4, 8, 16, and 32 CPUs, respectively(Simulation_4CPU, Simulation_8CPU ecc..).

- **EnergyFile**: The file where the total energy of the system is periodically written during the simulation. It is stored in a file named `energy.txt`.

- **InfoFile**: This file contains general information about the simulation's progress, including parameters and runtime diagnostics. The information is recorded in `info.txt`.

- **TimingsFile**: This file logs timing information, such as the time spent on different parts of the simulation process.

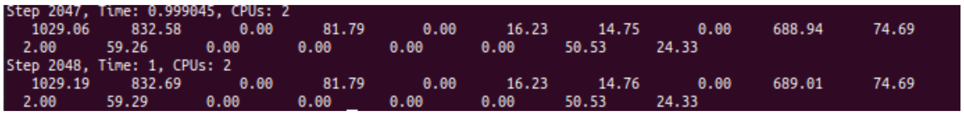


```
Step= 2047  t= 0.999045  dt= 0.000953716
Nf= 0000006830  total-Nf= 0104624401  ex-frac= 0.0720351  iter= 1
work-load balance: 1.47803  max=0.107422 avg=0.072679 PE0=0.0379361
particle-load balance: 1.3076
max. nodes: 102866, filled: 0.61313
part/sec=46987.4 | 31790.5  ia/part=552.448 (0)

Step= 2048  t= 1  dt= 0.000954628
Nf= 0000262144  total-Nf= 0104886545  ex-frac= 0.199013  iter= 1
work-load balance: 1.26708  max=1.66165 avg=1.3114 PE0=1.66165
particle-load balance: 1.28554
max. nodes: 101113, filled: 0.602681
part/sec=99948.2 | 78880.7  ia/part=157.217 (0)
```

Figure 2: Timings file tail

`work-load balance` gives the work-load balance in the actual tree walk, i.e. the largest required time of all the processors divided by the average time taken by all processors. This value is closely related to the scalability of the code; a value far from one indicates an imbalance in the distribution of the work-load among the processors. Such an imbalance can lead to inefficiencies, as some processors may be inactive while others are overloaded, ultimately affecting the overall performance and scalability of the simulation.

- **CpuFile**: The `cpu.txt` file logs CPU usage statistics during the simulation, which helps in assessing the computational efficiency of the code.



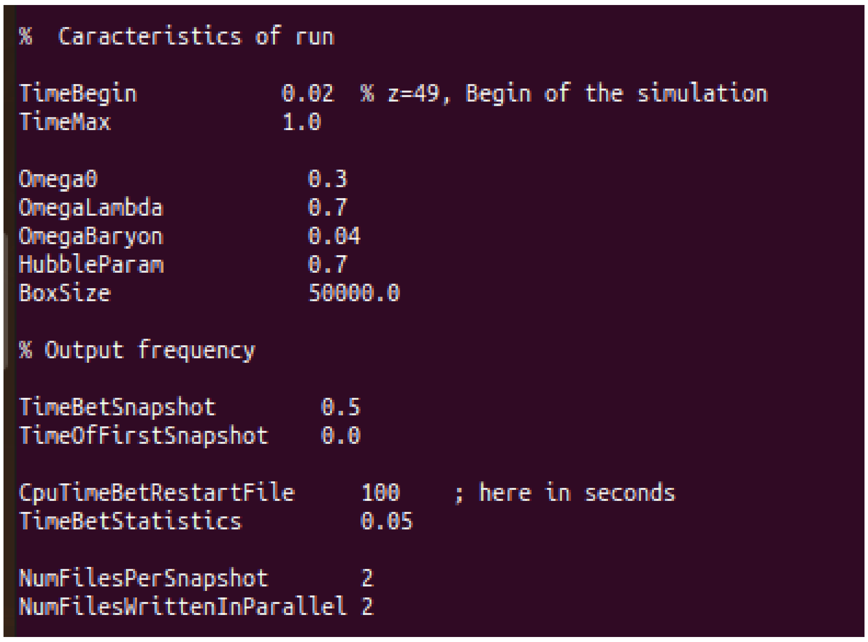
```
Step 2047, Time: 0.999045, CPUs: 2
   1029.06     832.58       0.00      81.79       0.00      16.23      14.75       0.00     688.94      74.69
  2.00      59.26       0.00       0.00       0.00       0.00      50.53      24.33
Step 2048, Time: 1, CPUs: 2
   1029.19     832.69       0.00      81.79       0.00      16.23      14.76       0.00     689.01      74.69
  2.00      59.29       0.00       0.00       0.00       0.00      50.53      24.33
```

Figure 3: Cpu file tail

In the second line, the first number informs you about the total cumulative CPU consumption of the code up to this point (in seconds), while the other numbers measure the time spent in various parts of the code. This is particularly useful when we want to test the strong scalability of the code. The number of CPUs used in the simulation is reported in the first line of each time step, which is also useful for testing scalability.

- **RestartFile**: This parameter defines the base name for files used in the restart process. If the simulation needs to be paused and later resumed, it uses files prefixed with `restart` to restore the system's state.

- **SnapshotFileBase**: The base name for snapshot files, which store the state of the system at various points in time during the simulation. These snapshots allow for analysis of the system's evolution over time. The files will have names beginning with `snapshot`.
- **OutputListFilename**: This specifies the path to a file containing a list of times (redshifts) at which snapshots of the simulation should be taken. The list is found in `parameterfiles/outputs2CPU_lcdm_dm.txt`. This saving process will need to be repeated for all numbers of CPU involved in the simulations(outputs4CPU_lcdm_dm.txt ecc..).

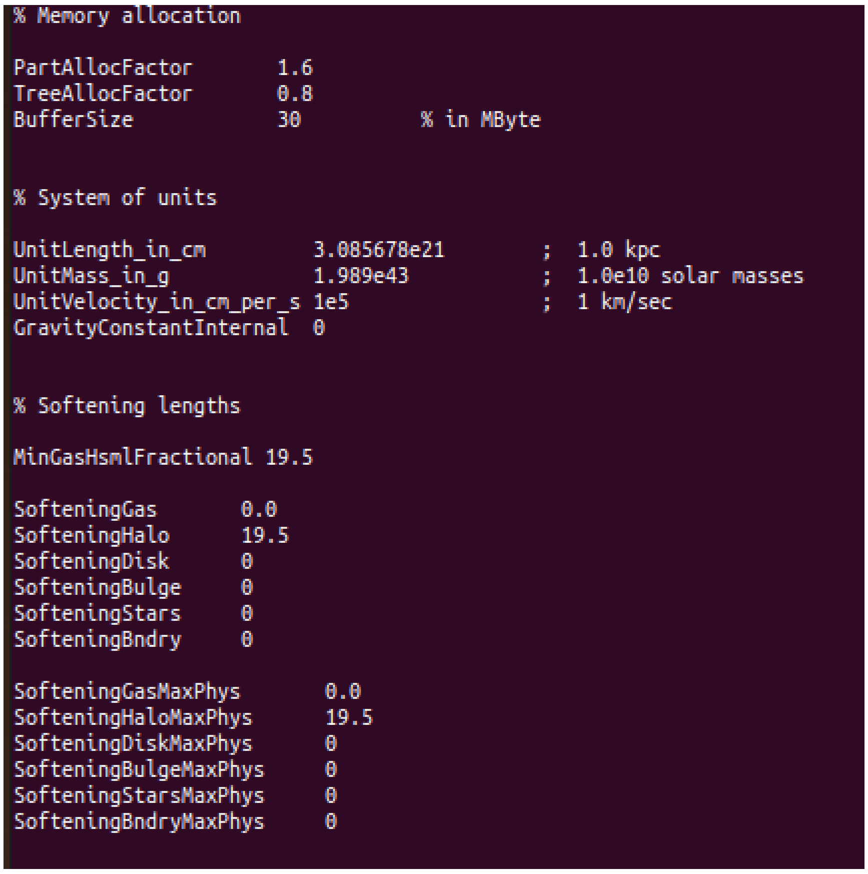


```
%  Caracteristics of run

TimeBegin           0.02  % z=49, Begin of the simulation
TimeMax             1.0

Omega0                0.3
OmegaLambda           0.7
OmegaBaryon           0.04
HubbleParam           0.7
BoxSize               50000.0

% Output frequency

TimeBetSnapshot        0.5
TimeOfFirstSnapshot    0.0

CpuTimeBetRestartFile     100    ; here in seconds
TimeBetStatistics         0.05

NumFilesPerSnapshot       2
NumFilesWrittenInParallel 2
```

Figure 4: Other parameters

These parameters in the Gadget2 parameters file define key aspects of the simulation and its output. `TimeBegin` and `TimeMax` set the initial and final times for the simulation, corresponding to certain cosmological redshifts(`z=49` in this case). The density parameters (`Omega0, OmegaLambda, OmegaBaryon`) describe the contributions of matter, dark energy, and baryons to the total energy density of the universe. `HubbleParam` indicates the Hubble constant, while `BoxSize` determines the physical size of the simulation volume. Particle coordinates need to lie in the interval [0, BoxSize] in the initial conditions file(in this case `50 000 Kpc=50 Mpc`).

The output frequency settings, including `TimeBetSnapshot`, `TimeOfFirstSnapshot`, and `CpuTimeBetRestartFile`, control how often data snapshots and restart files are created during the simulation. `TimeBetStatistics` governs how frequently simulation statistics are recorded. Finally, `NumFilesPerSnapshot` and `NumFilesWrittenInParallel` define the file structure for output data, specifying how many files are used for each snapshot and how many can be written simultaneously.



```
% Memory allocation

PartAllocFactor       1.6
TreeAllocFactor       0.8
BufferSize            30          % in MByte


% System of units

UnitLength_in_cm         3.085678e21        ;  1.0 kpc
UnitMass_in_g            1.989e43           ;  1.0e10 solar masses
UnitVelocity_in_cm_per_s 1e5                ;  1 km/sec
GravityConstantInternal  0


% Softening lengths

MinGasHsmlFractional 19.5

SofteningGas       0.0
SofteningHalo      19.5
SofteningDisk      0
SofteningBulge     0
SofteningStars     0
SofteningBndry     0

SofteningGasMaxPhys       0.0
SofteningHaloMaxPhys      19.5
SofteningDiskMaxPhys      0
SofteningBulgeMaxPhys     0
SofteningStarsMaxPhys     0
SofteningBndryMaxPhys     0
```

Figure 5: Gravitational softening

- **Memory Allocation**: PartAllocFactor and TreeAllocFactor determine how much memory is allocated for particles and tree structures, respectively, to manage computational resources effectively. BufferSize sets the size of the communication buffer used for parallel processing, with 30 MB allocated in this case. The buffer should be large enough to accommodate a ‘fair’ fraction of the particle mix in order to minimise work-load imbalance losses. Sizes between a few to 100 MB offer enough room.

- **System of Units**: The parameters define unit conversions for length, mass, and velocity.

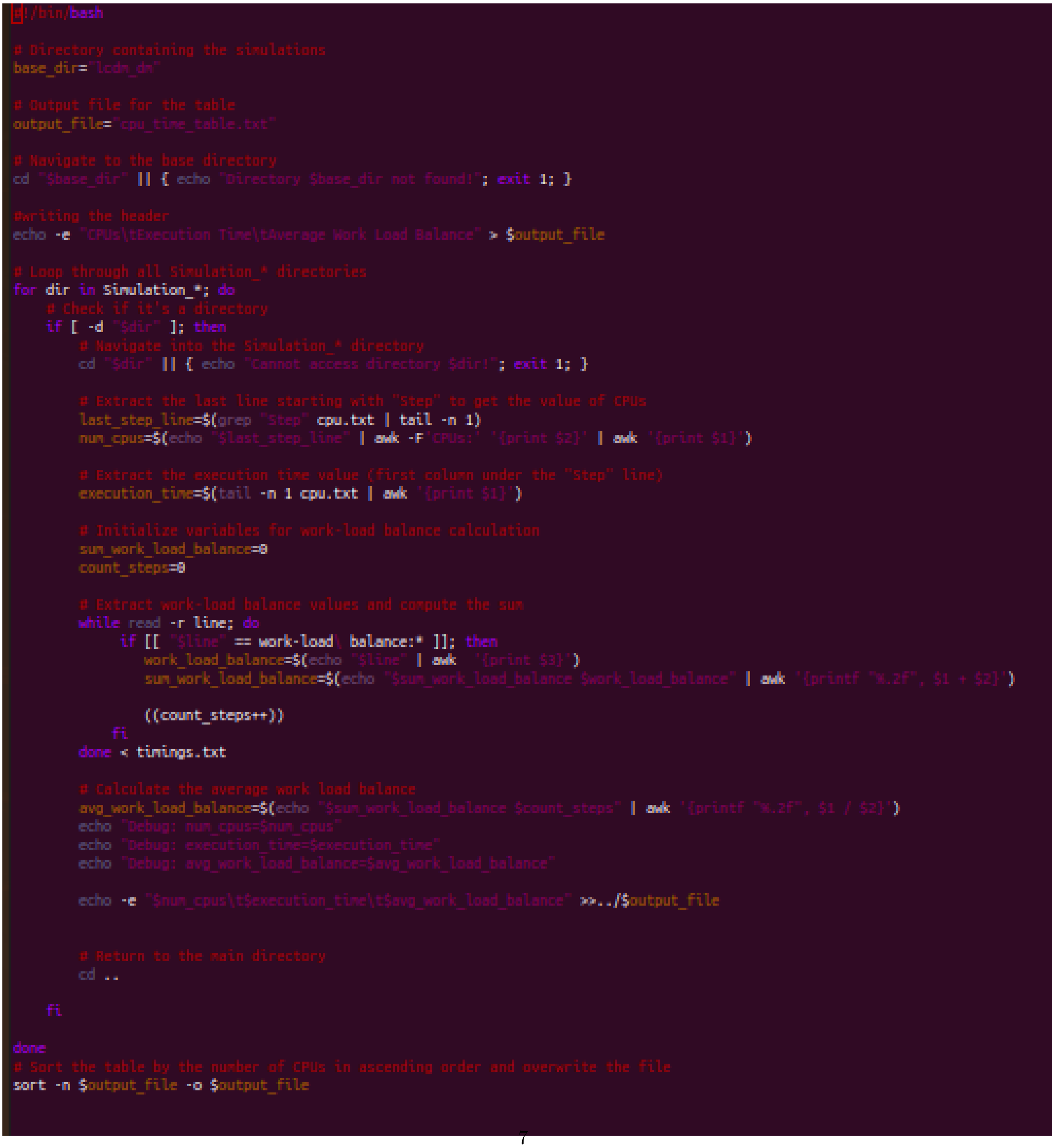
- **Softening Lengths**: This defines the softening lengths for different particle types, controlling the range over which the gravitational forces are smoothed. `MinGasHsmlFractional` specifies the minimum smoothing length for gas particles(halo particles in this case), and `SofteningHalo` determines the softening length for halo particles. Other softening parameters are set to 0, because in the simulation are involved only dark matter halos. In cosmological simulations, one sometimes wants to start a simulation with a softening ϵ_{com} that is fixed in comoving coordinates, but at a certain redshift one wants to freeze the resulting growth of the physical softening ϵ_{max} at maximum value. These maximum softening lengths are specified by the `Softening*MaxPhys` parameters. The units of measurement in the simulation are in *kpc/h*. Therefore, $L = 50\,\text{Mpc/h} = 50,000\,\text{kpc/h}$ in the code units. The softening length typically ranges between 1/30 and 1/50 of the mean inter-particle separation. Thus, with a box size L of 50,000 kpc/h and $N = 64^3$ particles, the softening length is calculated as:

$$\epsilon = \frac{1}{40} \times \left(\frac{L}{N^{1/3}} \right) \tag{2.1.1}$$

The result is `19.5 kpc`, which is the value used in every simulation since the problem size remains fixed.

3 Bash shell script

In order to collect the data necessary for testing the strong scalability of GADGET, we use a bash script that allows us to extract the required data from the output files generated in each simulation. As mentioned in the parameter files section, the necessary data will include the total execution time and the number of CPUs used, which can be found in the file cpu.txt. Another useful metric will be the average work-load balance for each simulation, which will give us an idea of how the work-load is distributed in each simulation.



```bash
#!/bin/bash

# Directory containing the simulations
base_dir="lcdm_dm"

# Output file for the table
output_file="cpu_time_table.txt"

# Navigate to the base directory
cd "$base_dir" || { echo "Directory $base_dir not found!"; exit 1; }

#writing the header
echo -e "CPUs\tExecution Time\tAverage Work Load Balance" > $output_file

# Loop through all Simulation_* directories
for dir in Simulation_*; do
    # Check if it's a directory
    if [ -d "$dir" ]; then
        # Navigate into the Simulation_* directory
        cd "$dir" || { echo "Cannot access directory $dir!"; exit 1; }

        # Extract the last line starting with "Step" to get the value of CPUs
        last_step_line=$(grep "Step" cpu.txt | tail -n 1)
        num_cpus=$(echo "$last_step_line" | awk -F'CPUs:' '{print $2}' | awk '{print $1}')

        # Extract the execution time value (first column under the "Step" line)
        execution_time=$(tail -n 1 cpu.txt | awk '{print $1}')

        # Initialize variables for work-load balance calculation
        sum_work_load_balance=0
        count_steps=0

        # Extract work-load balance values and compute the sum
        while read -r line; do
             if [[ "$line" == work-load\ balance:* ]]; then
                work_load_balance=$(echo "$line" | awk  '{print $3}')
                sum_work_load_balance=$(echo "$sum_work_load_balance $work_load_balance" | awk '{printf "%.2f", $1 + $2}')

                ((count_steps++))
            fi
        done < timings.txt

        # Calculate the average work load balance
        avg_work_load_balance=$(echo "$sum_work_load_balance $count_steps" | awk '{printf "%.2f", $1 / $2}')
        echo "Debug: num_cpus=$num_cpus"
        echo "Debug: execution_time=$execution_time"
        echo "Debug: avg_work_load_balance=$avg_work_load_balance"

        echo -e "$num_cpus\t$execution_time\t$avg_work_load_balance" >>../$output_file


        # Return to the main directory
        cd ..

    fi

done
# Sort the table by the number of CPUs in ascending order and overwrite the file
sort -n $output_file -o $output_file
```

Figure 6: Bash script

- `output_file="cpu_time_table.txt"`
  The variable `output_file` is defined at the beginning of the script. It specifies the name of the output file where the results of the simulations will be stored.

- `echo -e "CPUs\tExecution Time\tAverage Work Load Balance" > $output_file`
  This command writes the header line to the output file. The -e option in the echo command allows the interpretation of special characters that follow a backslash . Here, `\t` is used to insert tab spaces between the column titles ("CPUs", "Execution Time", and "Average Work Load Balance"). The `>` symbol redirects the output to the `$output_file`, effectively creating or overwriting the file.

- `last_step_line=$(grep "Step" cpu.txt | tail -n 1)`
  This line searches for all lines containing the word `"Step"` in the `cpu.txt` file using the `grep` command. The output of `grep` is then piped (`|`) to the `tail` command, which retrieves the last line of this output. The resulting line is stored in the variable `last_step_line`. This line typically contains information about the last time step of the simulation, which is crucial for extracting the total execution time.

- `num_cpus=$(echo "$last_step_line" | awk -F'CPUs:' '{print $2}' | awk '{print $1}')`
  This line uses `awk` to extract the number of CPUs from the last line starting with "Step" in the `cpu.txt` file. The first `awk` command splits the line at the "CPUs:" field delimiter and retrieves the second part (`$2`). The second `awk` command extracts the first word from the remaining string, which corresponds to the number of CPUs used in the simulation.

- `execution_time=$(tail -n 1 cpu.txt | awk '{print $1}')`
  This command uses `awk` to extract the execution time from the last line of `cpu.txt`. Here, `awk` simply prints the first field (`$1`), which is assumed to be the execution time value.

- `work_load_balance=$(echo "$line" | awk '{print $3}')`
  During the loop, `awk` is used to extract the third field (`$3`) from lines that contain the work-load balance information in `timings.txt`. This value is then added to the cumulative sum `sum_work_load_balance`.

- `avg_work_load_balance=$(echo "$sum_work_load_balance $count_steps" | awk '{printf "%.2f", $1 / $2}')`
  Finally, the average work-load balance is computed by dividing the cumulative sum of the work-load balance by the number of steps. The `awk` command here uses `printf` to format the result to two decimal

places, ensuring that the average is presented in a consistent numeric format.

- `echo -e "$num_cpus\t$execution_time\t$avg_work_load_balance" >>../$output_file`
  After processing the data for each simulation, this line appends the results (number of CPUs, execution time, and average work load balance) to the `cpu_time_table.txt` file. The `>>` operator appends the content to the file without overwriting the existing data.

- `sort -n $output_file -o $output_file`
  Finally, the script sorts the rows in the output file based on the number of CPUs in ascending order. The `-n` option ensures numerical sorting, and the `-o` option writes the sorted output back to the original file `$output_file`.

So the script extracts the necessary data to test scalability and places it in a table within the `lcdm_dm` directory, which contains the output directories of the simulations. The `cpu_time_table.txt` table will consist of three columns, containing the number of CPUs, the execution time of the code, and the average work-load balance for each simulation.

4 Final results

To test the strong scalability of Gadget2, we conducted simulations using 2 different nodes for each test, maintaining a consistent reference scale when increasing the number of processors. The parameters were kept the same across all simulations, with the exception of `NumFilesWrittenInParallel`, which was adjusted to match the number of CPUs used in each simulation.

Additionally, the `TreeAllocFactor` and `PartAllocFactor` were modified for the 32 CPU simulation. In this case, the original parameters led to the error "maximum number 10485 of tree-nodes reached," indicating that the program had hit the maximum limit of tree nodes it could handle. Although this error is specific to the tree structure, increasing the BufferSize can help prevent potential memory management bottlenecks caused by insufficient buffers. Therefore, the BufferSize was also increased to 50 MB, but only for the 32 CPU simulation.

CPUs	**Execution Time (s)**	**Average Work Load Balance**
2	1029.19	1.20
4	883.44	1.37
8	527.36	1.63
16	384.27	2.33
32	271.22	2.98

Table 1: Scalability test results for Gadget2 with a simulation box of $L = 50$ Mpc/h and $N = 64^3$ particles.

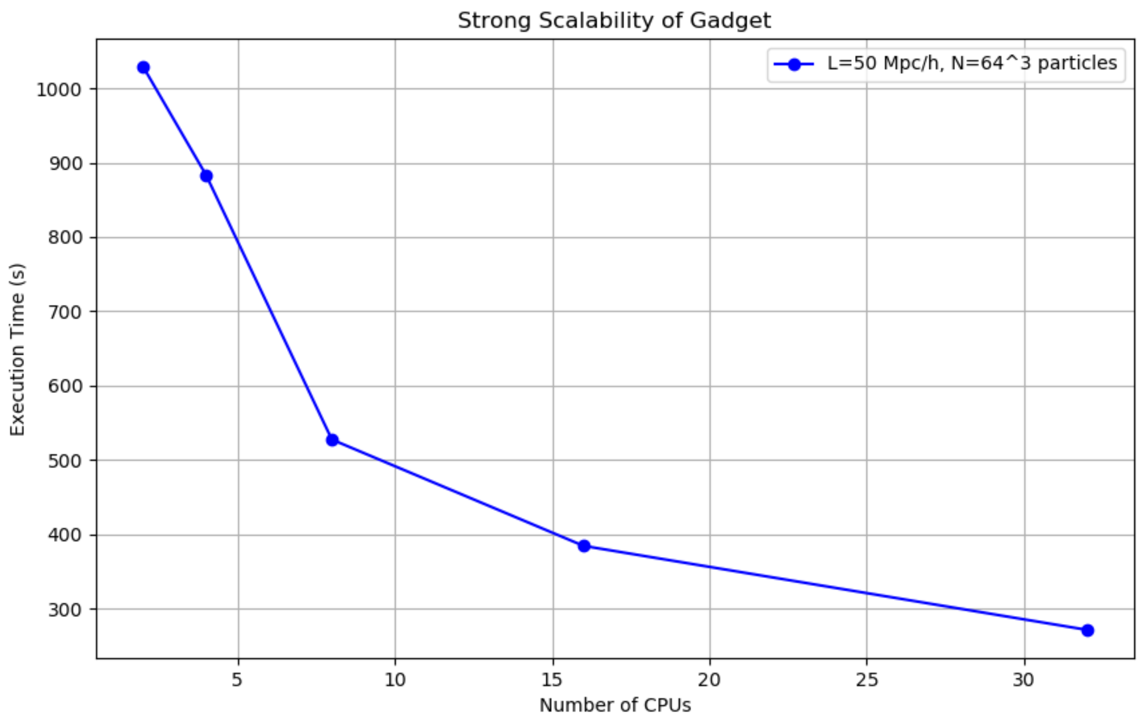


Figure 7: Strong scalability plot

In the context of strong scalability, a system is said to have good strong scalability if the execution time decreases proportionally with the increase in the number of CPUs, while maintaining a fixed problem size. Ideally, if we double the number of CPUs, the execution time should halve, indicating that the workload is being evenly distributed across the processors.

However, the data provided reveals that while the execution time does decrease with more CPUs, the decrease is not perfectly proportional. This is reflected in the third column of the table, which shows the Average Work

Load Balance.. As the number of CPUs increases, the Work Load Balance value moves further away from 1.0, indicating that the workload is not evenly distributed among the processors. In particular, for the simulation with 32 CPUs, we have a Work Load Balance value of 2.98, which is significantly higher compared to the value of 1.20 obtained with 2 CPUs.

When the Work Load Balance is significantly greater than 1, it suggests that some processors are doing more work than others, leading to inefficiencies and underutilization of the available computational resources. This imbalance in workload distribution can result in a less efficient use of additional CPUs, and is a primary reason why the scalability is not ideal in this case.

Contents