Link-wise Artificial Compressibility Method (part 1)

by Yifan Yang

With supervisor Thomas Zeiser 27.01.2015

Outline

1. project plan
2. Intro to LWACM
3. C implementation
4. results

seminar plan

Two presentations:

1. C implementation of Lwacm
2. code optimization and parallelization

All sources and data are available on Github:

http://github.com/cosailer/lwacm

Based on paper:

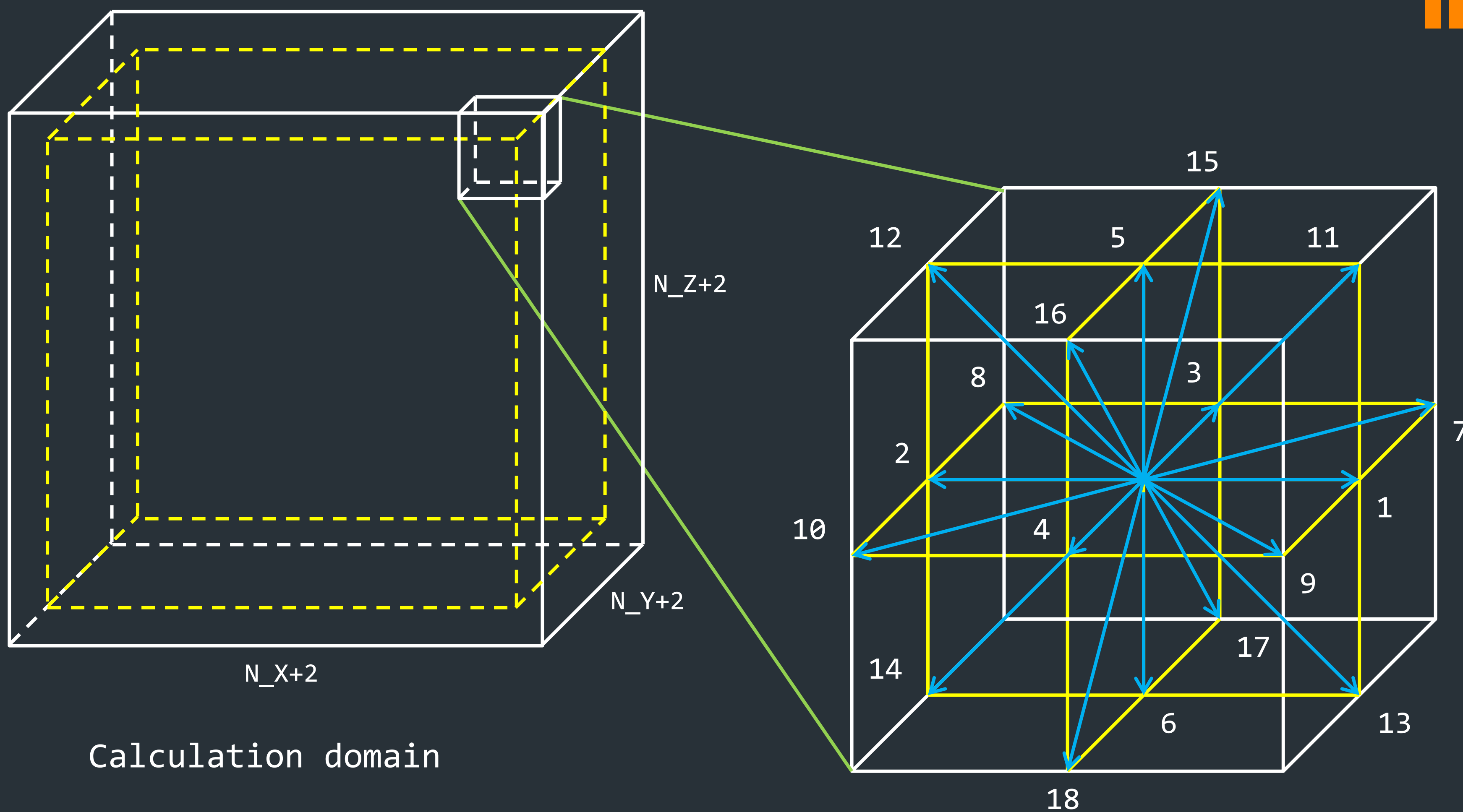
http://www.sciencedirect.com/science/article/pii/S0021999114004823

artificial compressibility method

- The artificial compressibility method (ACM), is a numerical approach for solving the incompressible Navier–Stokes equations (INSE).

Link-wise formulation

- The LW-ACM is a discrete formulation of the ACM within a framework similar to the one of the LBM. It operates on a regular Cartesian spatial mesh of mesh size δx with a regular time step δt. In accordance with the established practice of LBM, we shall express all following quantities in terms of lattice units, i.e. adopt δx as unit of length and δt as unit of time



Calculation domain

D3Q19 stencil

single-step LW-ACM

```
for all time step t do
    for all mesh point x do
        for all index α do
            load ρ(x-ξα, t)and u(x-ξα, t)
            compute f(e)α(x-ξα, t)
            compute f(e,o)α(x-ξα, t)
        end for
        for all index α do
            compute fα(x, t+1)
        end for
        compute ρ(x, t+1)
        compute u(x, t+1)
        store ρ(x, t+1) and u(x, t+1)
    end for
End for
```

C implementation

>> Main array:

```
double p[2][N_X+2][N_Y+2][N_Z+2]
double u[2][N_X+2][N_Y+2][N_Z+2][3]
```

Array size can be really large if we increase domain size

\>> Main loop:

```
for( t = 0; t < T_MAX; t++)
    for( x = 1; x < N_X+1; x++)
    |    for( y = 1; y < N_Y+1; y++)
    |        for( z = 1; z < N_Z+1; z++)
    |            alpha_0_call();
    |            alpha_1_call();
    |            ...
    |            alpha_18_call();
    |
    |            // step 11, compute p(x, t+1)
    |            // step 12, compute u(x, t+1)
    |
    |__end for

    // set boundary condition
    // store p(x, t+1)
```

>> Function call for alpha loop:


```
void alpha_0_call()
{
     //load p(x-xia) and u(x-xia)
     p_load = p[t_now][x][y][z];
     u_load[0] = u[t_now][x][y][z][0];
     u_load[1] = u[t_now][x][y][z][1];
     u_load[2] = u[t_now][x][y][z][2];

     u_xi = 0;
     u_2 = u_load[0]*u_load[0] + u_load[1]*u_load[1] + u_load[2]*u_load[2];
     u_x_xi = 0;
     // step 5, compute f(e)(x-xi[0], t)
     f_e = 1.0/3.0 * p_load * ( 1 + 3*u_xi + 4.5*u_xi*u_xi - 1.5*u_2 );
     f_e_o = 3.0 * 1.0/3.0 * p_load * u_xi;  // Eq.10

     f_x_t = 3.0 * 1.0/3.0 * p[t_now][x][y][z] * u_x_xi;
     // step 9, compute f(x, t+1)
     f[0] =  f_e + 2*( omega-1/omega )*( f_x_t - f_e_o ) ;
}
```

Results

Performance Measurement

>> runtime: CPU timer, wall clock

>> time step t = (Domain size) / (N_X* N_Y* N_Z)

	Domain size	**Runtime (avg)**	**Stream results**
Atom 450	31 250 000	58.57	5232 MB/S
Exynos 5250	24 000 000	21.48	3258 MB/S
I7 2600	200 000 000	39.62	12420 MB/S

Operation per lattice update

Assume N_X = N_Y = N_Z = N

>> 543 * N^3 * t (FPOPs)

>> (338 * N^3 + 24 * N^2 + 48 * N) * t (mem OPs)

Performance

FPOPS:

$543\ N^3\ t\ /\ T$

$= 543 * \mathrm{MLUPS}$

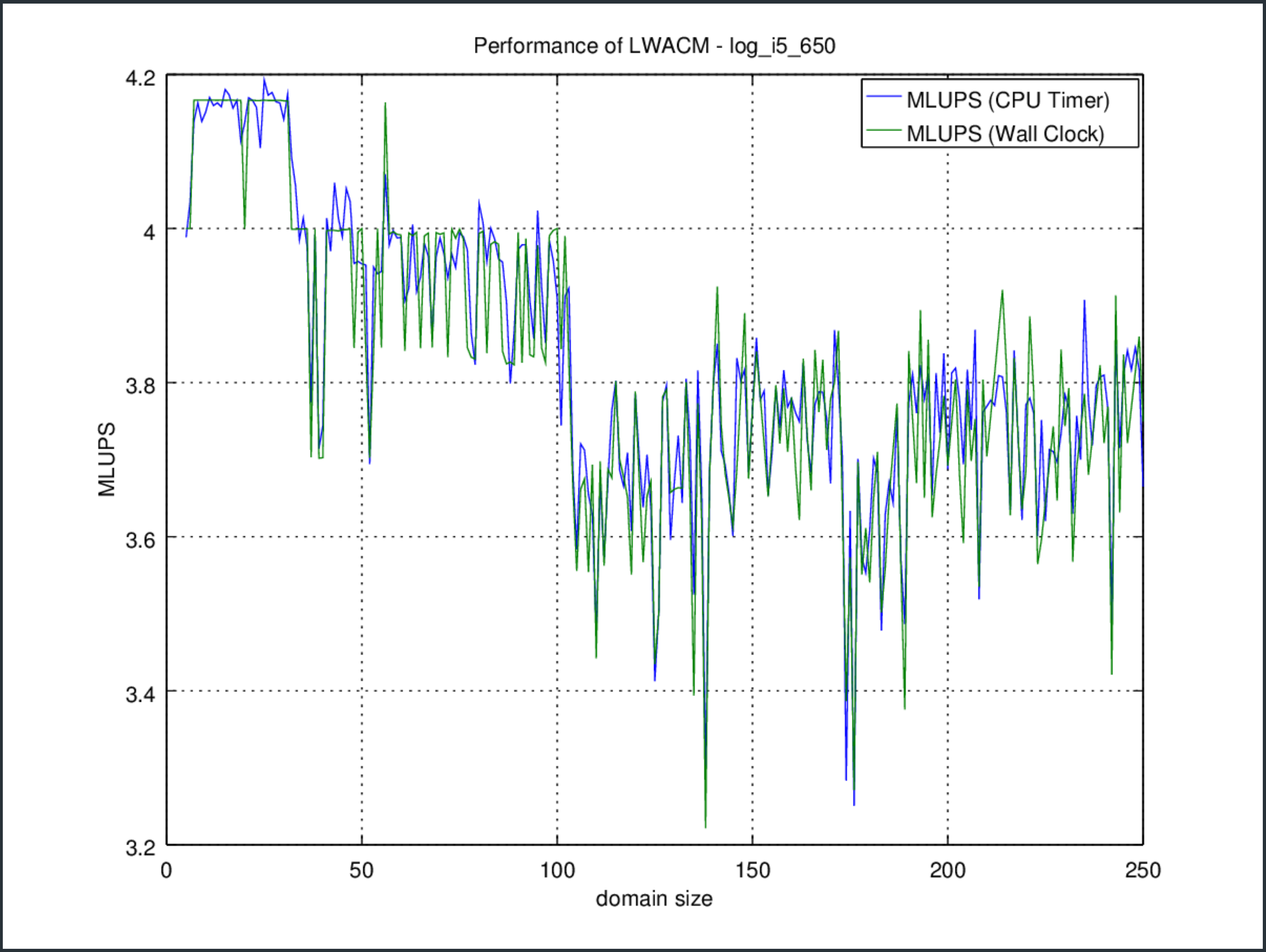
Memory speed:

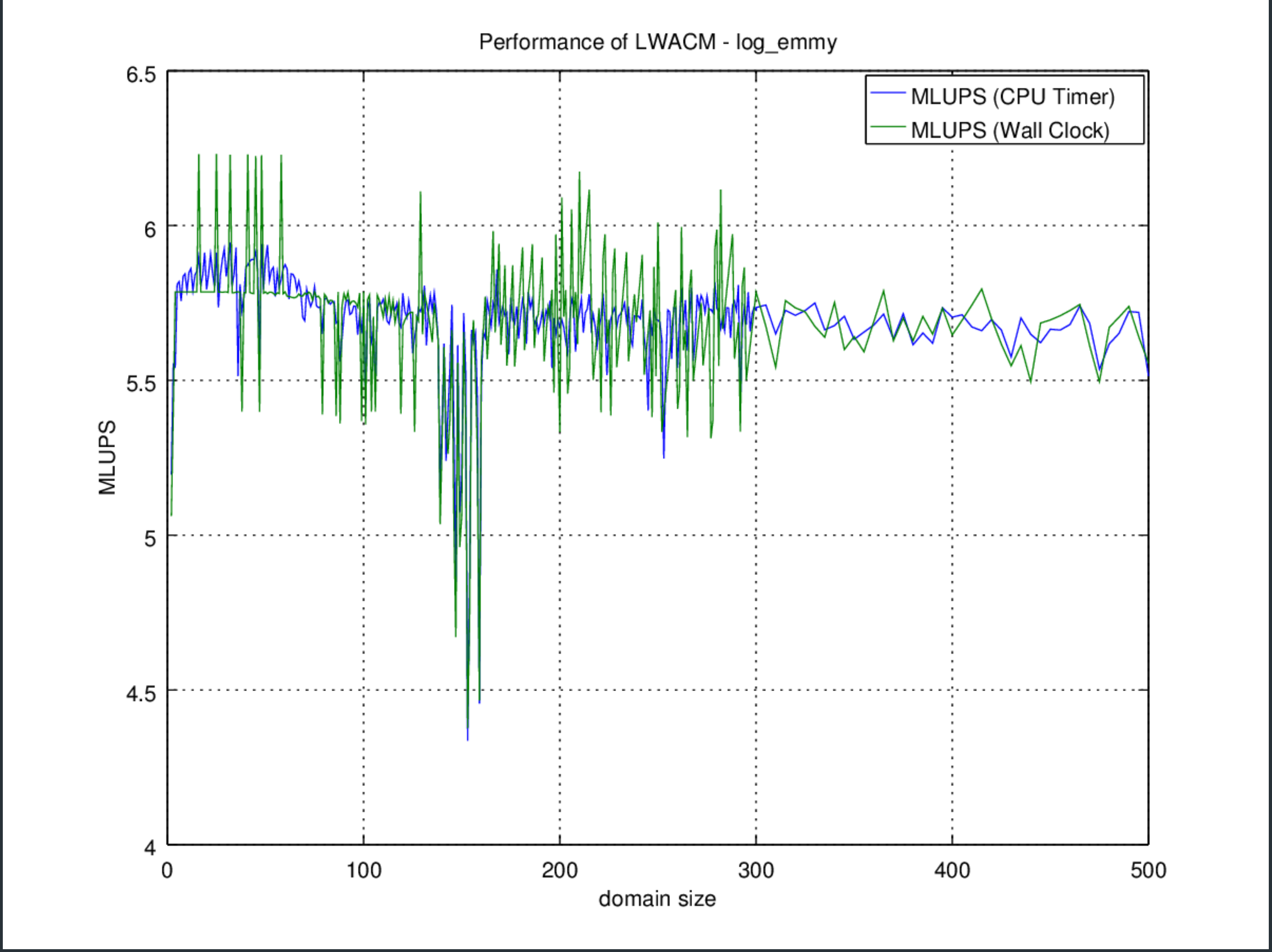
$(338\ N^3 + 24\ N^2 + 48\ N)\ t\ /\ T$

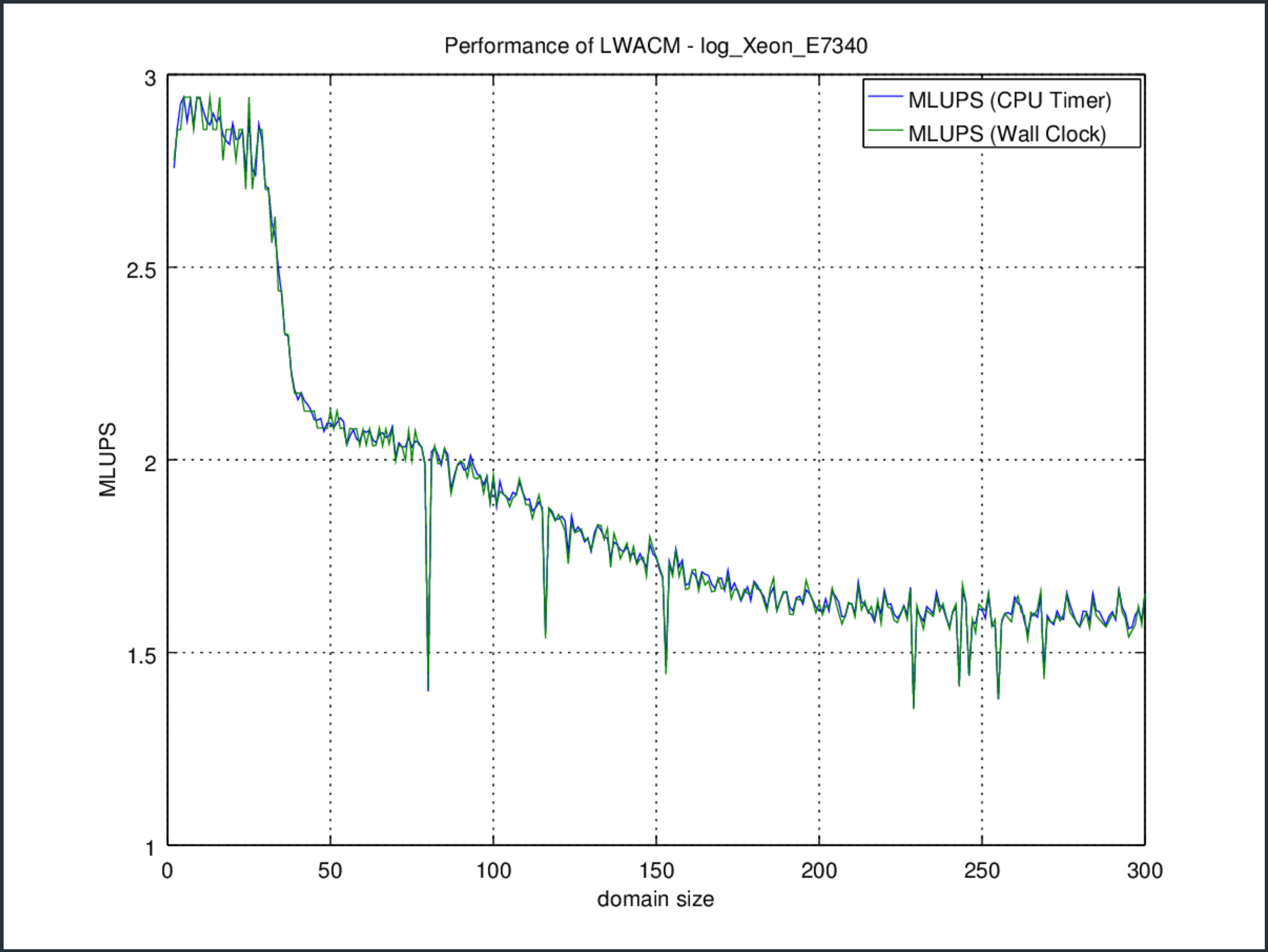
$= (338\ D + 24\ N^2\ t + 48\ N\ )\ /\ T$

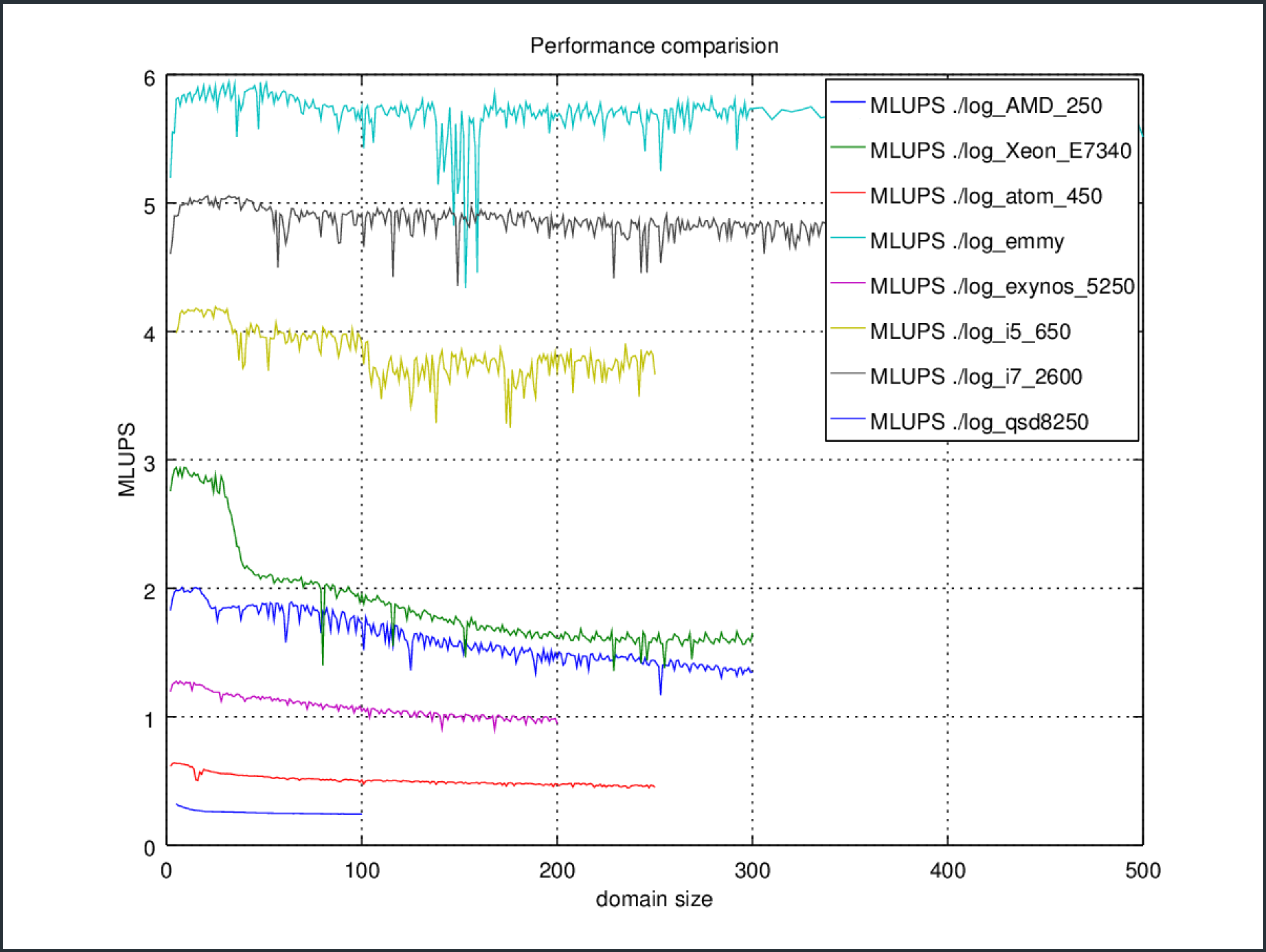
$= 338 * \mathrm{MLUPS}$

D: domain size T: program runtime


Performance of LWACM - log_i5_650
MLUPS (CPU Timer)
MLUPS (Wall Clock)
MLUPS
domain size
4.2
4
3.8
3.6
3.4
3.2
0
50
100
150
200
250


Performance of LWACM - log_emmy
MLUPS (CPU Timer)
MLUPS (Wall Clock)
MLUPS
domain size
6.5
6
5.5
5
4.5
4
0
100
200
300
400
500


Performance of LWACM - log_Xeon_E7340
MLUPS (CPU Timer)
MLUPS (Wall Clock)
MLUPS
domain size
3
2.5
2
1.5
1
0
50
100
150
200
250
300


Performance comparision
MLUPS
domain size
0
1
2
3
4
5
6
0
100
200
300
400
500
MLUPS ./log_AMD_250
MLUPS ./log_Xeon_E7340
MLUPS ./log_atom_450
MLUPS ./log_emmy
MLUPS ./log_exynos_5250
MLUPS ./log_i5_650
MLUPS ./log_i7_2600
MLUPS ./log_qsd8250

Thank you for your time !

See you in part 2