Performance Evaluation of Acoustic FDTD(2,4) Method Using Distributed Shared Memory System mSMS

Ryoya Tabata (Kyushu Inst. Tech.), Hiroko Midorikawa (Seikei Univ.), Ki'nya Takahashi (Kyushu Inst. Tech.)

1. Introduction

In this study, mSMS [1], which is page-based distributed shared memory system is used for multi-node implementation of FDTD(2,4) method, and the performance evaluation is presented.
For efficient large-scale acoustic analysis, it is important to employ high-order FDTD method such as FDTD(2,4) method [2], which has second-order accuracy in time and fourth-order accuracy in space.
To improve the programming productivity in multiple-node environments, the PGAS language that provides a global view of programming can be used.

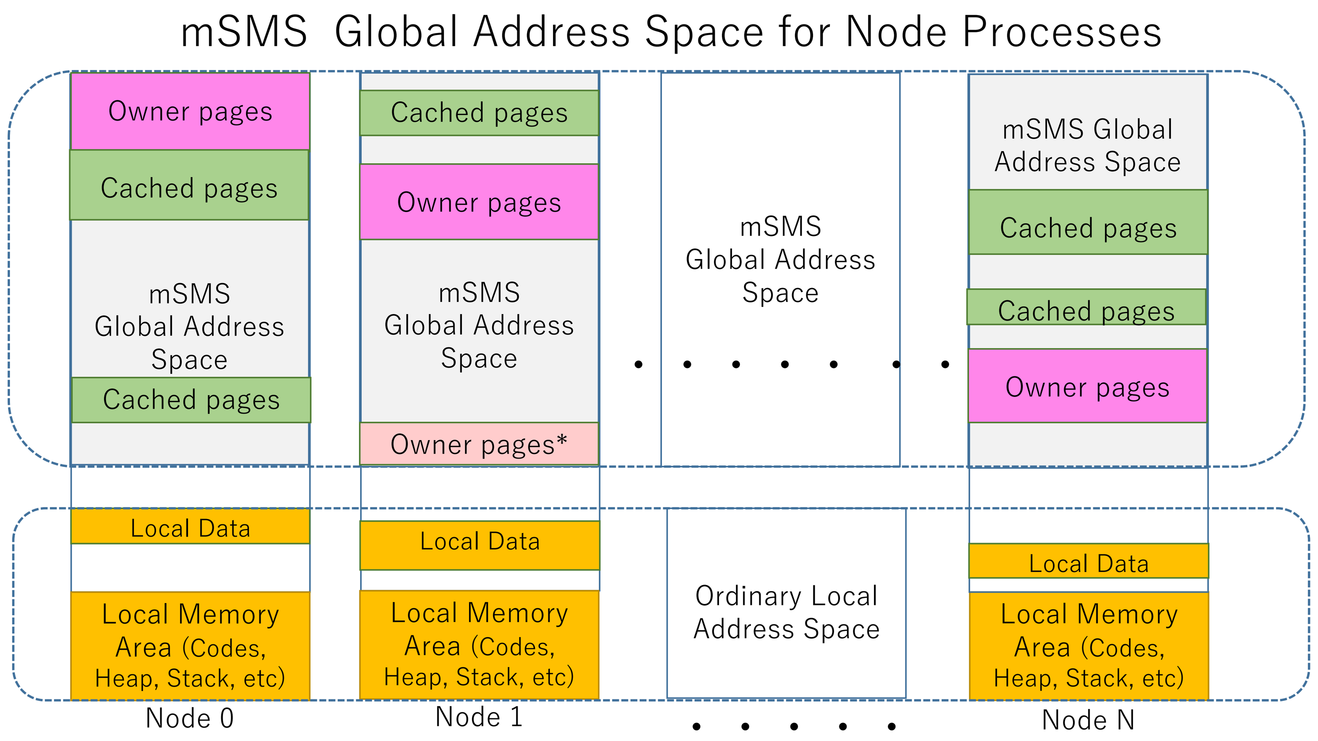

Fig. 1 mSMS Global Address Space (adapted from [1])

FDTD (2, 4) Method

- Governing equations

$$\rho \frac{\partial \vec{v}}{\partial t} = -\nabla p \qquad \vec{v}\ \text{: particle velocity}$$

$$\nabla \cdot \vec{v} = -\frac{1}{k}\frac{\partial p}{\partial t} \qquad p\ \text{: acoustic pressure}$$

- Spatial difference operator with fourth-order accuracy

$$\boldsymbol{d_{4x}} f^n(x,y,z) = \boldsymbol{d_{2x}} f^n(x,y,z) + \frac{1}{24\Delta x}\left[3f^n\left(x+\frac{\Delta x}{2},y,z\right) - 3f^n\left(x-\frac{\Delta x}{2},y,z\right) - f^n\left(x+\frac{3\Delta x}{2},y,z\right) + f^n\left(x-\frac{3\Delta x}{2},y,z\right)\right].$$

```c
#include <sms.h>
int main(int argc, char const *argv[]) {
  double (*matrix)[y_size][x_size];
  matrix=sms_mapalloc(dim,div,sizeof(double),0,sms_nprocs);
  sms_startup(&argc, &argv); // start mSMS system
  bz = NZ/sms_nprocs; // block size for one node
  sz = sms_rank * bz; ez = (sms_rank + 1) * bz;// z-division
  sms_sync(); // execution-synch.
  for ( int time_step = 0; time_step < NT; i++) { // main loop of fdtd(2,4) method
  #pragma omp parallel for
  for (k = sz; k < ez; k++) for (j = sy; j < ey; j++) for (i = sx; i < ex; i++) {
    qx[k][j][i]-=CQ*(K1*(p[k][j][i+1]-p[k][j][i])-K2*(p[k][j][i+2]-p[k][j][i-1]));
    qy[k][j][i]-=CQ*(K1*(p[k][j+1][i]-p[k][j][i])-K2*(p[k][j+2][i]-p[k][j-1][i]));
    qz[k][j][i]-=CQ*(K1*(p[k+1][j][i]-p[k][j][i])-K2*(p[k+2][j][i]-p[k-1][j][i]));
  }
  sms_sync_drop(); // execution-synch. & discard cache pages
  #pragma omp parallel for
  for (k = sz; k < ez; k++) for (j = sy; j < ey; j++) for (i = sx; i < ex; i++) {
  p[k][j][i]-=CP*(K1*(qx[k][j][i]-qx[k][j][i-1])-K2*(qx[k][j][i+1]-qx[k][j][i-2]))
            +CP*(K1*(qy[k][j][i]-qy[k][j-1][i])-K2*(qy[k][j+1][i]-qy[k][j-2][i]))
            +CP*(K1*(qz[k][j][i]-qz[k-1][j][i])-K2*(qz[k+1][j][i]-qz[k-2][j][i]));
  }
  sms_sync_drop();
  }
  sms_shutdown();// finalize mSMS system
}
```

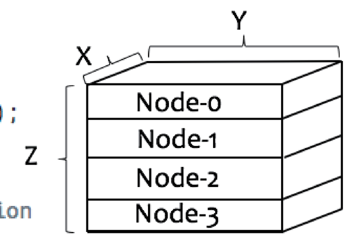

Fig. 2 Skelton FDTD (2, 4) code parallelized with mSMS

2. Performance Evaluation

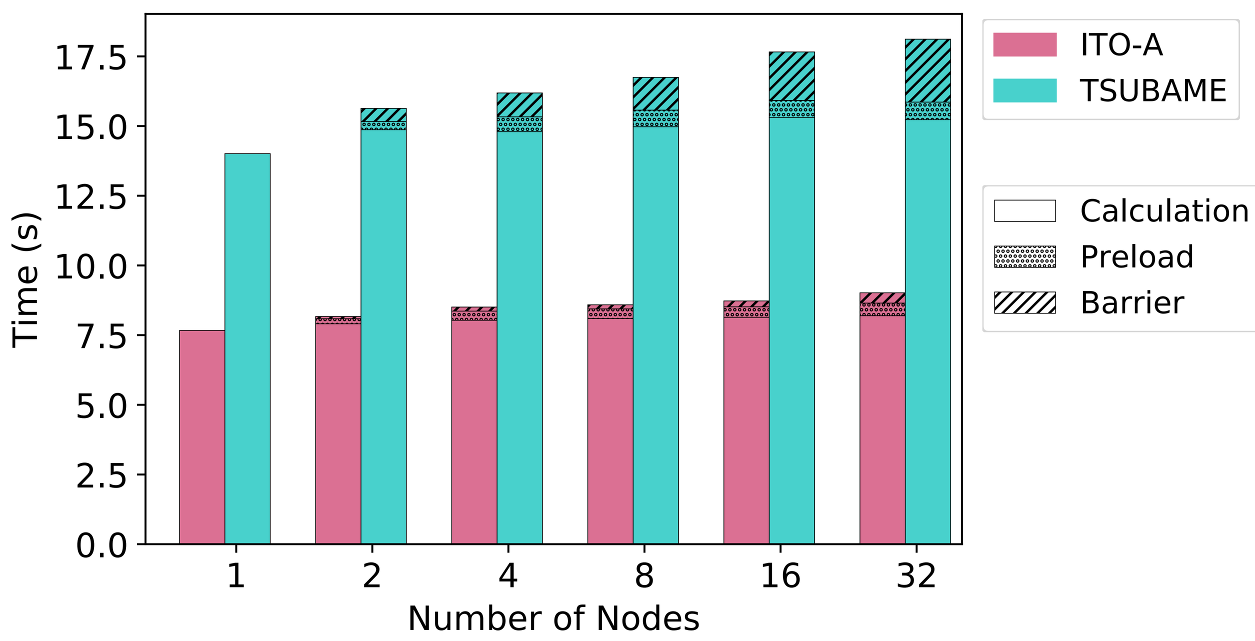

Fig. 3 Weak scaling results
(average execution time per 10 simulation steps for 5 runs)

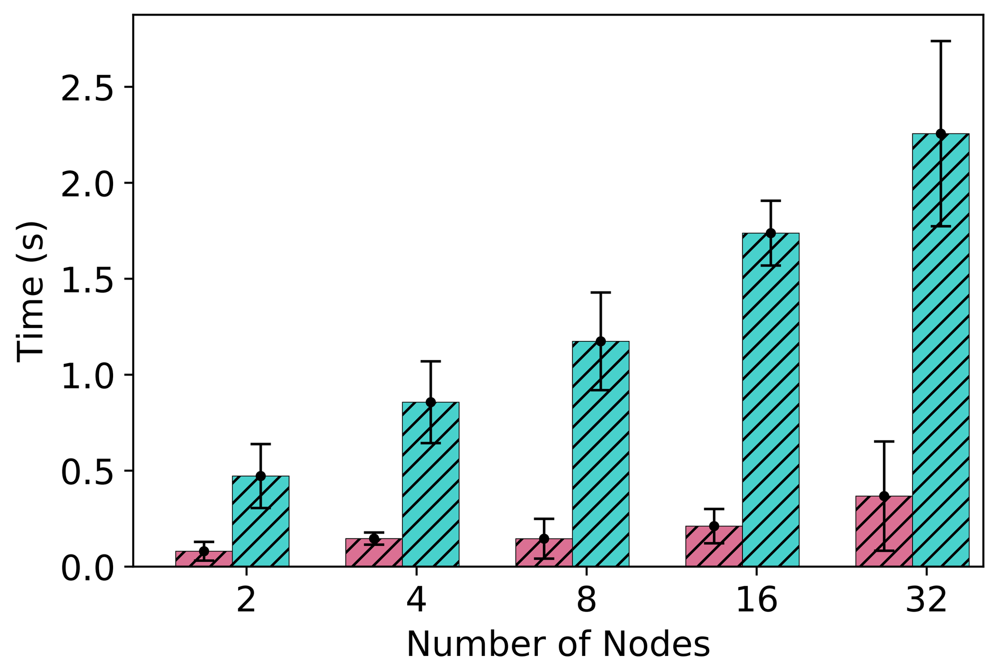

Fig. 4 Barrier time (error bars show the standard deviation)

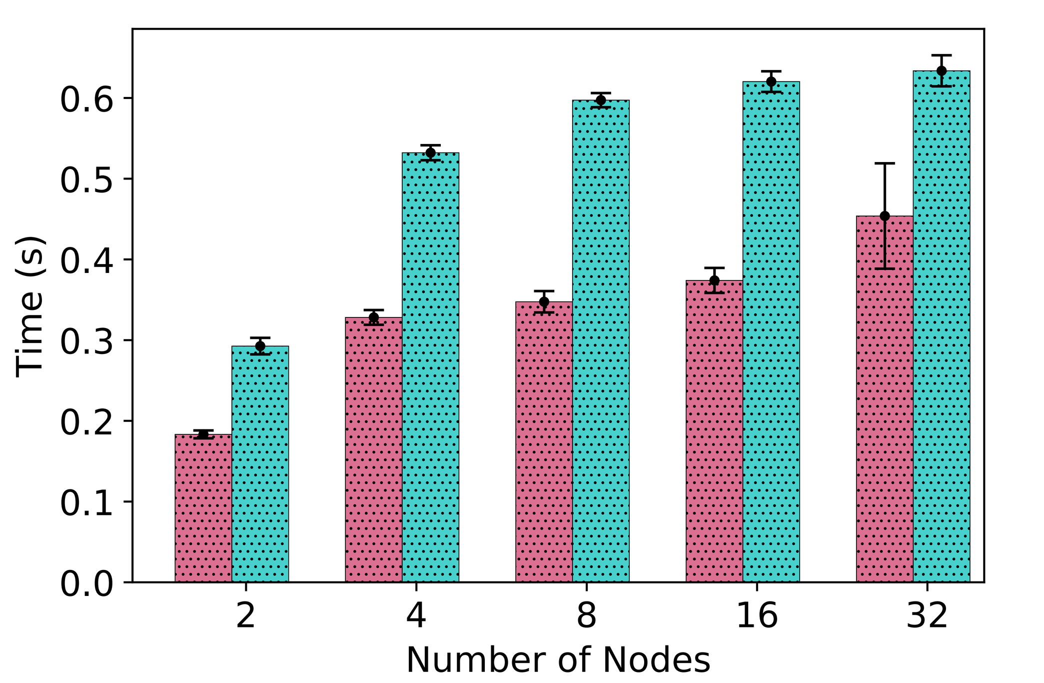

Fig. 5 Preload time (error bars show the standard deviation)

Experimental Environment and Conditions

TSUBAME 3.0 (Intel Xeon E5-2680 v4, 14 core, 2.4GHz × 2 / node, Intel Omni-Path 100Gb/s × 4, Intel MPI 2018.1.163)
ITO Subsystem A (Intel Xeon Gold 6154, 18 core, 3.0 GHz × 2 / node, InfiniBand EDR 4x 100Gb/s, MVAPICH2-X 2.2)
mSMS version: r121, Calculation Precision: double, Calculation Mesh Size: 1024^3/node, OpenMP Threads Number: 24
Preload API is used for halo exchange, which is an alternative to remote page fetching by SIGSEGV signal handler.

Remote data preload API

Efficient explicit data prefetching (preloading continuous pages with specified size) is possible by using preload API.

```c
size_t g_stld[3]= { NZ,NY,NX}; // global data size
size_t e_size[3]= { load_ez - ez,NY,NX}; // prefetch data size
if( time_step == 0)
  // prefetch e_size data from &matrix[ez][0][0] pointer to cache page
  sms_preload_array(&matrix[ez][0][0], g_stld, e_size, 3, sizeof(double), 1);
else{
  // overwrite cache page
  sms_overload_array(&matrix[ez][0][0], g_stld, e_size, 3, sizeof(double),1);
}
```

3. Application Example

Helmholtz Resonator Model with PML (Perfectly Matched Layer) [3] Boundary Condition

Even in the complex boundary problem, mSMS can achieve decent results with few code modifications.
The below results are without using Preload API (no explicit code optimization for data transfer).

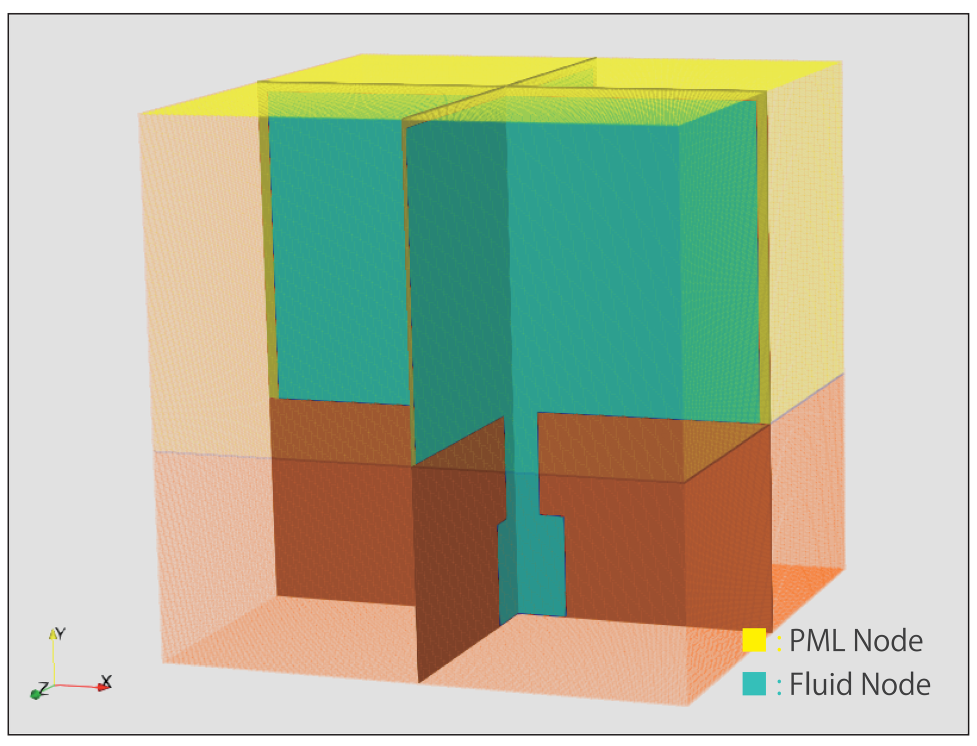

Fig. 6 Helmholtz Resonator Model (PML layer num: 20, mesh size: 1024^3, no calculation in orange region)

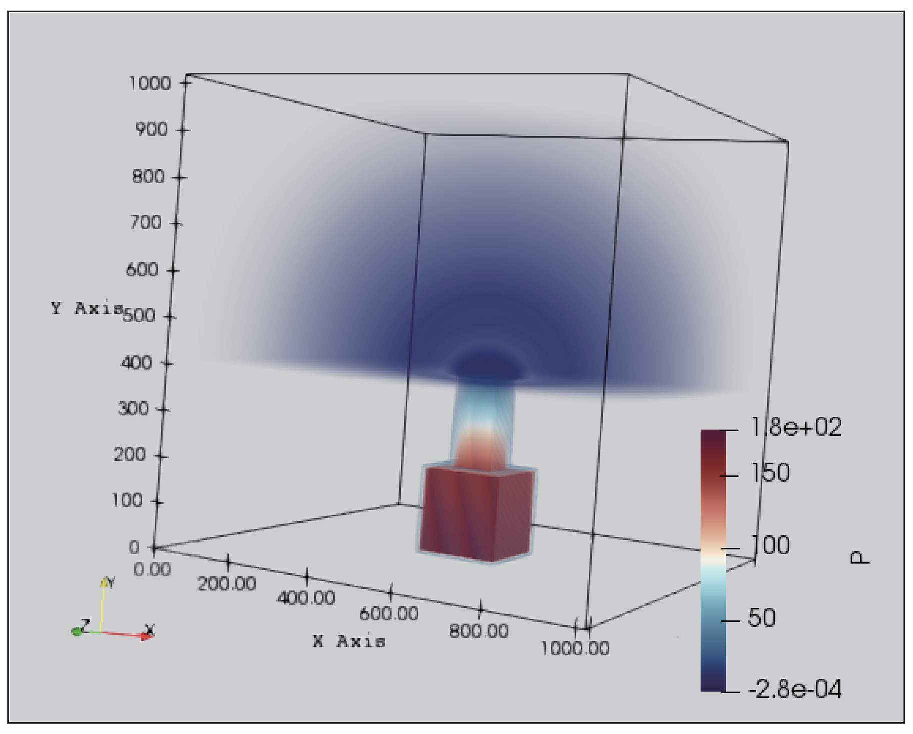

Fig. 7 Acoustic Pressure Distribution (time step : 5000)

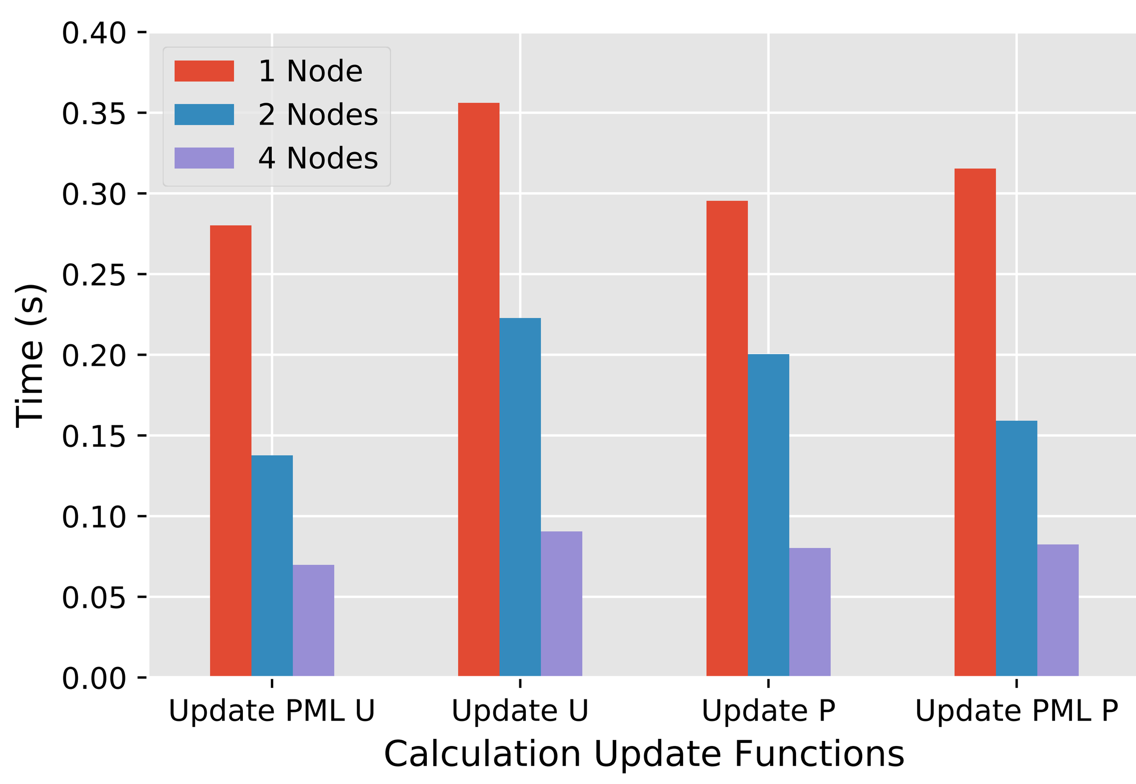

Fig. 8 Average execution time per timestep for each calculation update function (measured on ITO-A)

4. Conclusion and Future Work

▶ Conclusion

We demonstrated the nearly ideal weak scaling performance of the FDTD(2,4) method parallelized with mSMS and OpenMP, in spite of using a simple implementation.

▶ Future Work

- Overlapping computation/communication by using a non-blocking preload API.
- Implementation and performance evaluation of FDTD(2,4) method with mSMS+GPGPU.
- Incorporating spatial and temporal blocking (non-redundant) into the FDTD(2,4) method.

5. References

[1] Hiroko Midorikawa, Kenji Kitagawa, and Yugo Sakaguchi: "mSMS : PGAS Runtime with Efficient Thread-based Communication for Global-view Programming", 2019 IEEE International Conference on Cluster Computing, pp.1-2. DOI: 10.1109/CLUSTER.2019.8891009.
[2] Yuuki Sendo, Hironori Kudo, Tatsuya Kashiwa, and Tadao Ohtani: "The FDTD(2, 4) method for highly accurate acoustic analysis in three-dimensional space", Electronics and Communications in Japan (Part III: Fundamental Elec tronic Science) 86 (11 2003), pp. 30–37. DOI: 10.1002/ecjc.10076.
[3] J. Berenger: "A perfectly matched layer for the absorption of electromagnetic waves". Journal of Computational Physics 114 (2): 185–200. DOI:10.1006/jcph.1994.1159.