Acceleration of Inlet-Exhaust Pipe Simulation Using Multi-processor Systems

Akira Kojima, Taisuke Yamamoto, Tetsuo Hironaka
Department of Computer and Network Engineering
Hiroshima City University

Outline

- Introduction
- Target Program
- Parallelizing method
- Evaluaiton
- Conclusion and future work

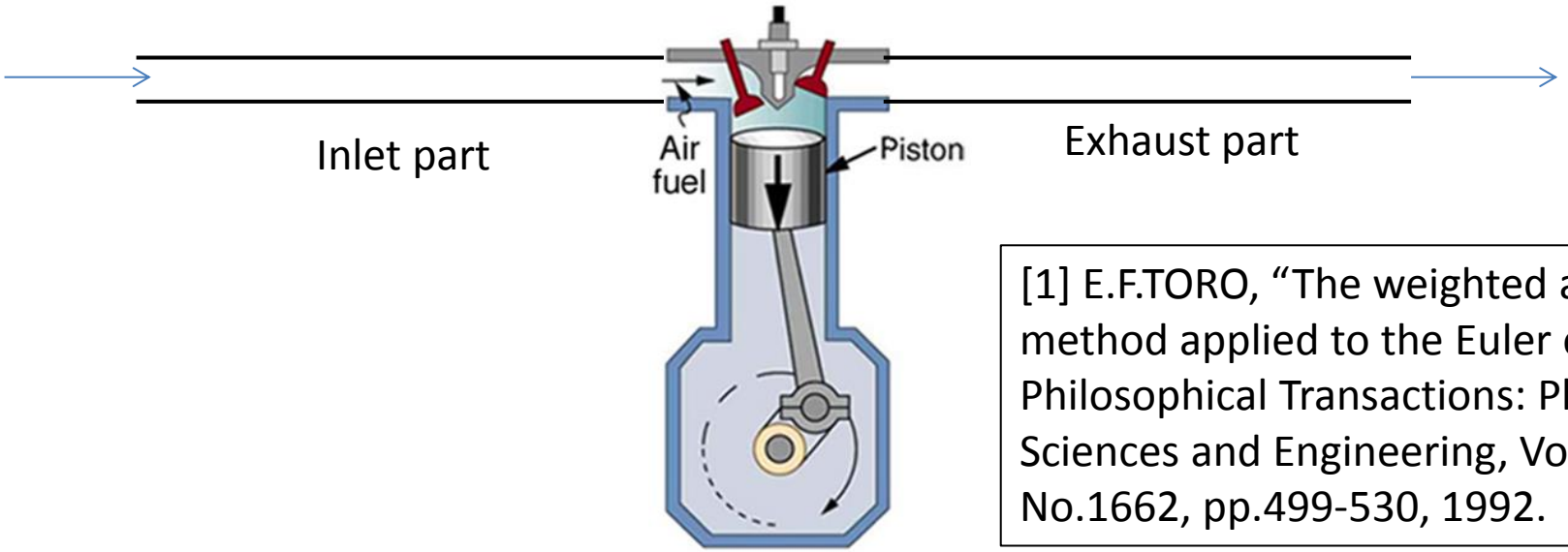
Introduction

- Numerical simulation for designing today's cars
  - Need high precision
  - Need to try many cases, parameter patterns
    → Lots of time
    → Required to speed up

- Acceleration methods
  - Multi-processor system
  - GPU
  - Hardware accelerator

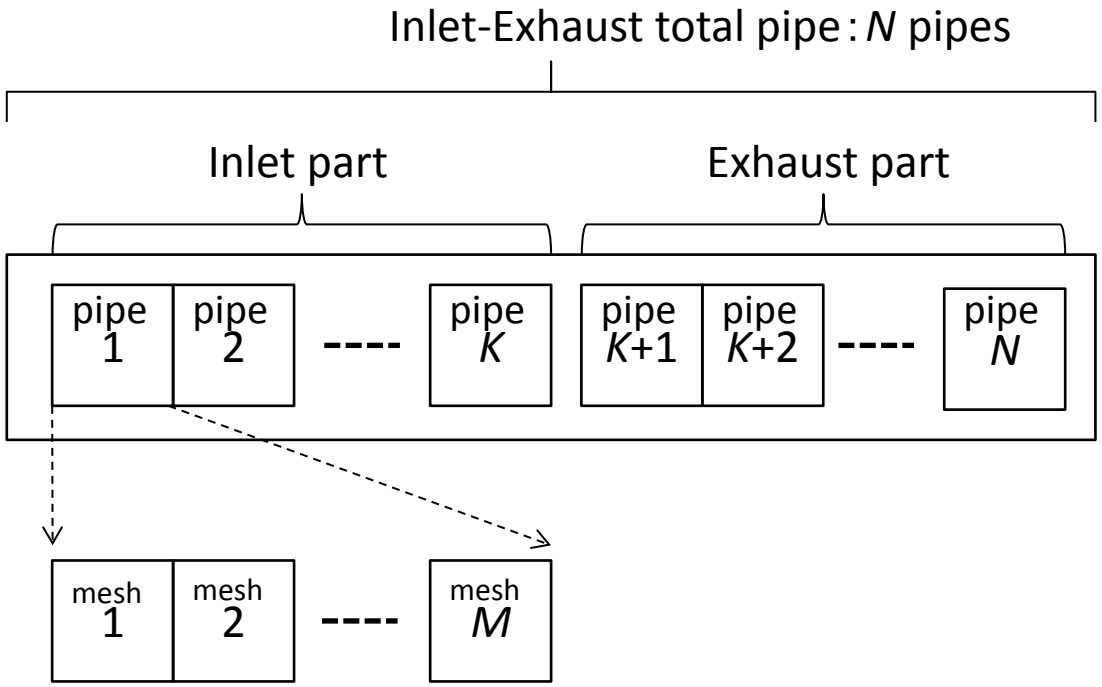
Target Program

- Inlet-Exhaust Pipe simulation
  - Euler equations for compressible perfect fluid
    - approximate equations of Navier Stokes equations
    - omit viscous term from Navier Stokes equations
  - Weighted Average Flux (WAF) method[1]



[1] E.F.TORO, "The weighted average flux method applied to the Euler equations," Philosophical Transactions: Physical Sciences and Engineering, Vol.341, No.1662, pp.499-530, 1992.

Data Structure



One dimensional model
Inlet-Exhaust total pipe can be divided into *N* pipes.
Each pipe can be divided into *M* meshes.
（example: *N*=100～　*M*=2～20）

Flow of Calculation Function

```
int calc_pipes( ... )
{
    ...
    for (int IP = 1; IP <= N; IP++) { // loop for pipes
        M = NUM_MESH[IP]; // number of meshes in IP pipe
        ...
        for (int IM = 0; IM < M + 1; IM++) { // loop1 for meshes
            ...
        }
        ...
        for (int IM = 0; IM <= M + 1; IM++) { // loop2 for meshes
            ...
        }
        ...
        // loop3 ~ loop8 for meshes
        ...
        for (int IM = 2; IM < M - 2; IM++) { // loop9 for meshes
            ...
        }
    }
    return 0;
}
```

Boundary, Main Routine

- On boundaries, make the next step data using the neighbor's data
- Boundary between meshes in a pipe
  - Processed in the pipe loop of the calculation function
- Boundary between pipes
  - Processed in the main routine
- Main routine
  - Main routine calls the calculation function.
  - Main routine is written in MATLAB and Visual Basic.
  - Main routine is not the target of this study.

Parallelization method

- Parallelize the outer loop for pipes
  - large granularity , reduce overhead
- Execute the inner loops for meshes sequentially
  - Not use fine grain parallelization for multi-processor
  - （in the future, SIMD instructions like AVX will be used for the inner loops for meshes）
- OpenMP
  - Compiler directives. Programmers have to care about dependency conflict.
  - Can be used with Intel compiler icc, GNU compiler g++, Microsoft compiler cl, PGI compiler
- Make local work variables for threads
  - Original sequential code uses global work variables
- Improve loop scheduling
  - Dynamic Scheduling ：fine grain chunk, even granularity of thread tasks
  - #pragma omp parallel for schedule(dynamic)

Loop scheduling (1)

- Static scheduling (OpenMP default)

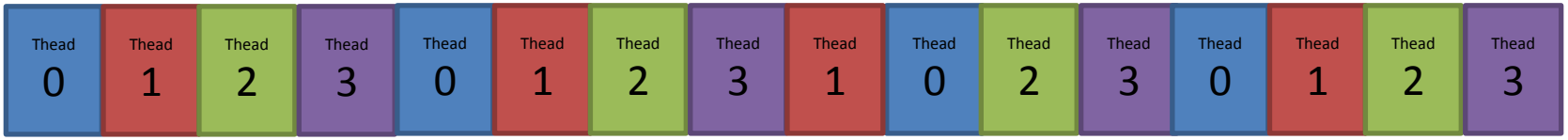
- Chunk size = Num. of loop iterations / Num. of threads

For the target program, Num. of Meshes in Pipes are different.

Thread granularity becomes uneven.

Loop scheduling (2)

- Dynamic scheduling



- Chunk size = 1 (one loop iteration)

For the target program,
thread granularity does not become uneven.
One pipe iteration has enough size of grain,
thread switching overhead does not become large percentage.

Loop scheduling (3)

- Guided scheduling

- At first, chunk size is large like static scheduling. After that, it will decrease.

- Intermediate scheduling between Static scheduling and Dynamic scheduling.

Evaluation

- Evaluate Execution time
  - Inlet 100000 calls + Exhaust 100000 calls = 200000calls
  - Num. of pipes is 118. Num of meshes is 2～20.

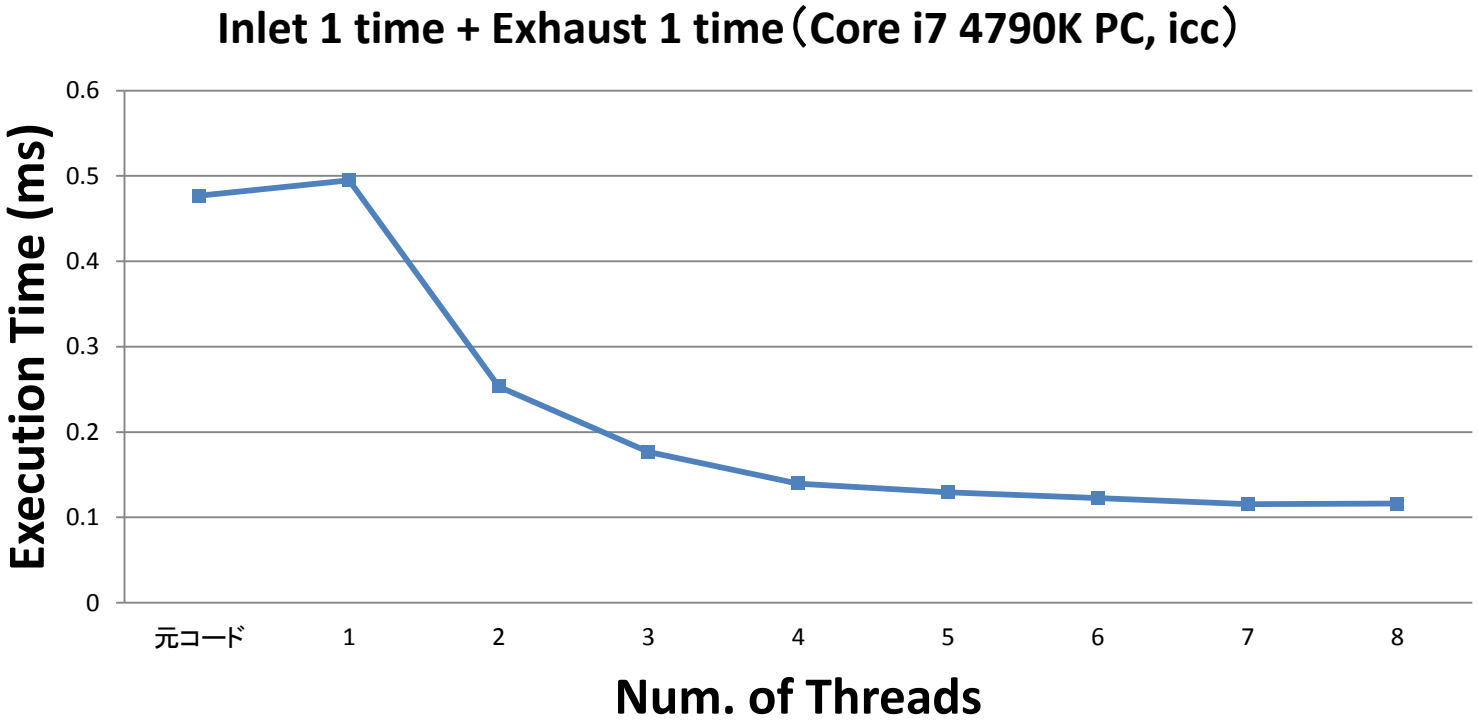
- PC Core i7 4790K 4GHz 4core
  - max : 8thread
  - CentOS 6.6 64bit, Intel Compiler icc 14.0.1 –O3

- Server Xeon E5-2680 v2 2.8GHz 10core × 2CPU
  - max : 40thread
  - CentOS 6.6 64bit, Intel Compiler icc 14.0.1 –O3

Num. of Thread and Execution time(1)

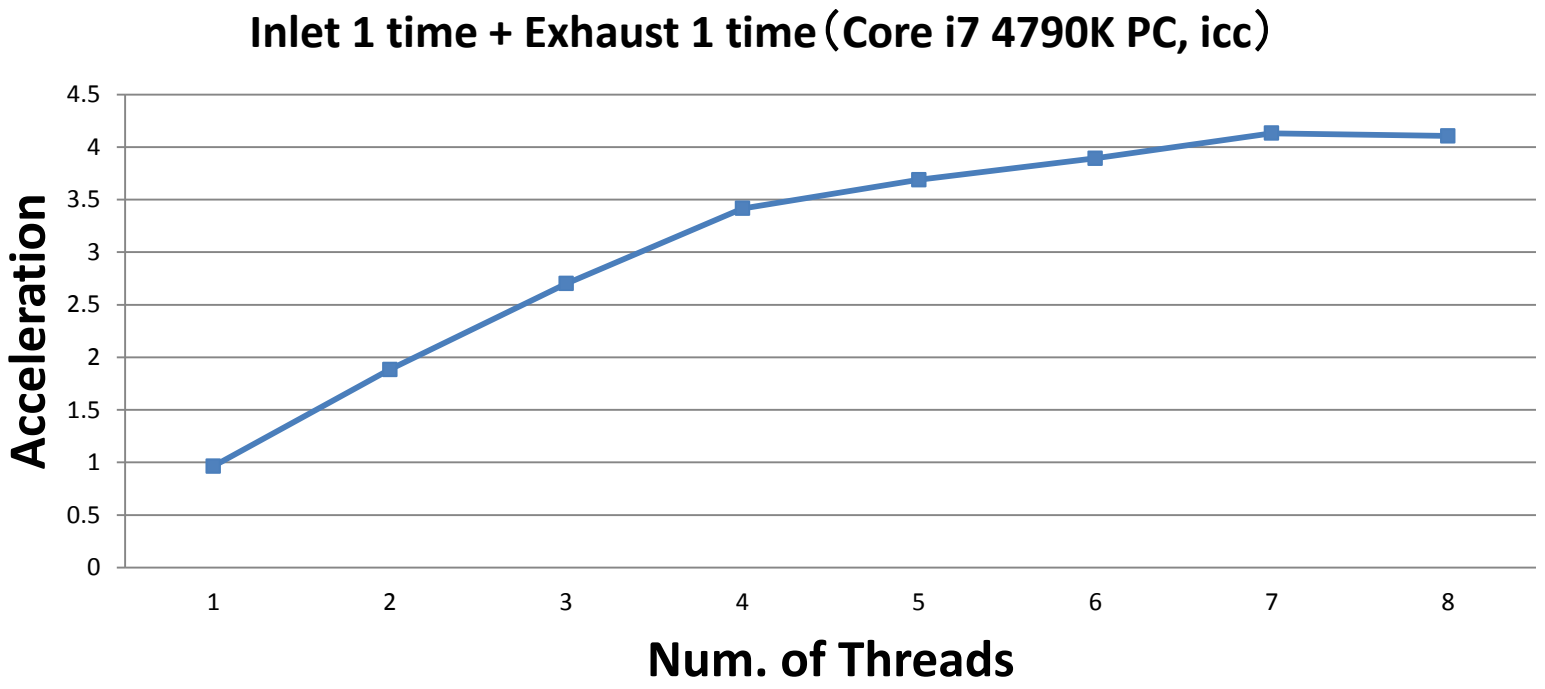
Original code : Inlet 100000 + Exhaust 100000= 47.684608 sec.

8Threads : Inlet 100000+Exhaust 100000= 11.616966 sec. 4.1times faster



Num. of Thread and Execution time(2)

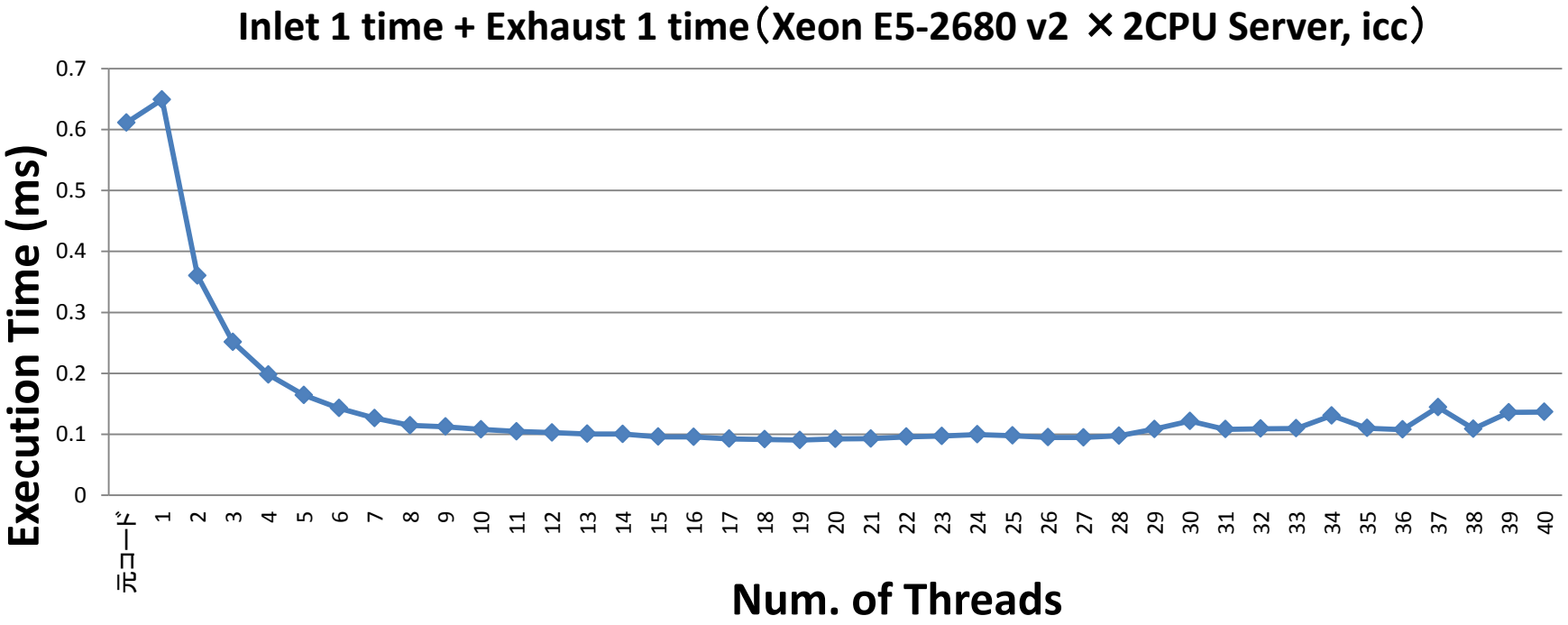
8-threads code is 4.1 times faster than the original sequential code.



Num. of Thread and Execution time(3)

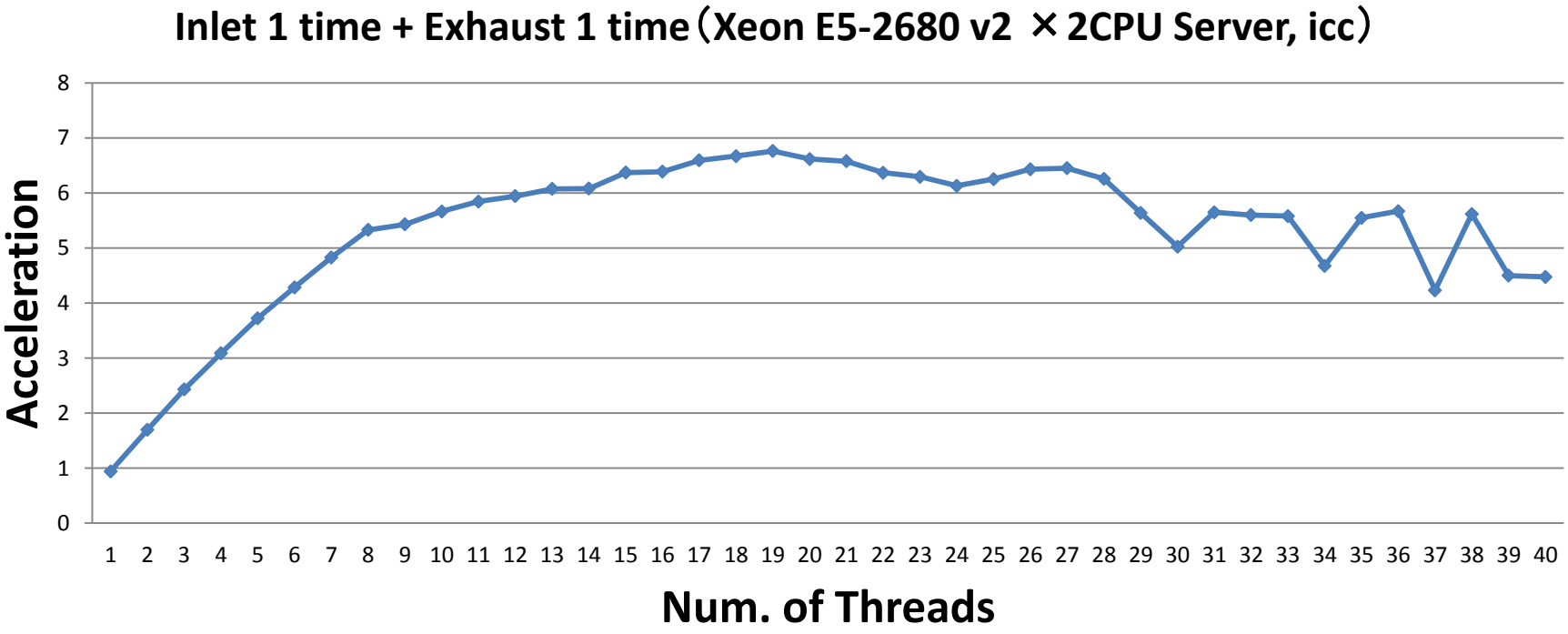
Original code : Inlet 100000+Exhaust 100000= 61.109010 sec.

20Threads : Inlet 100000+Exhaust 100000= 9.038148 sec. 6.6 times faster

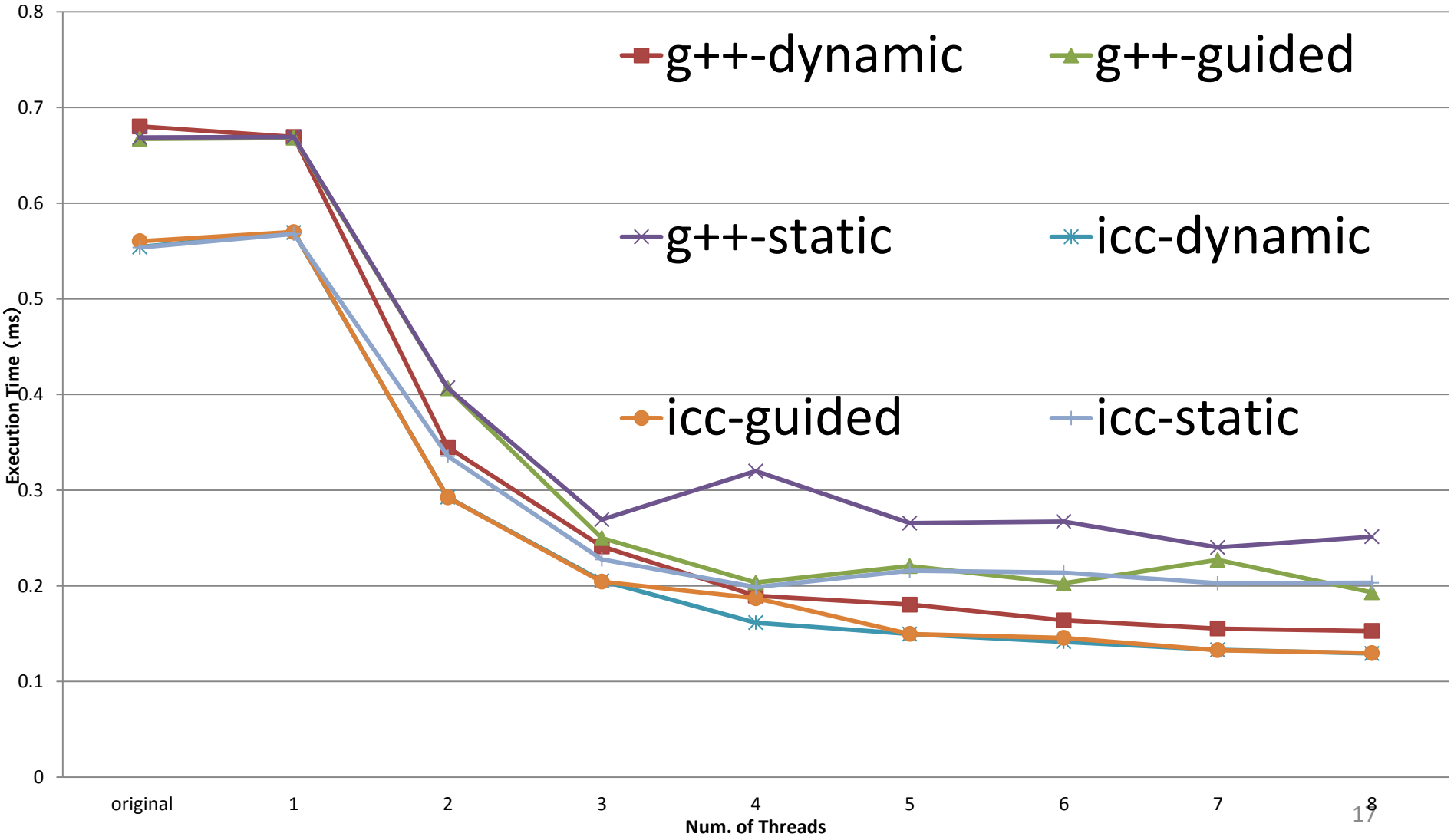


Num. of Thread and Execution time(4)

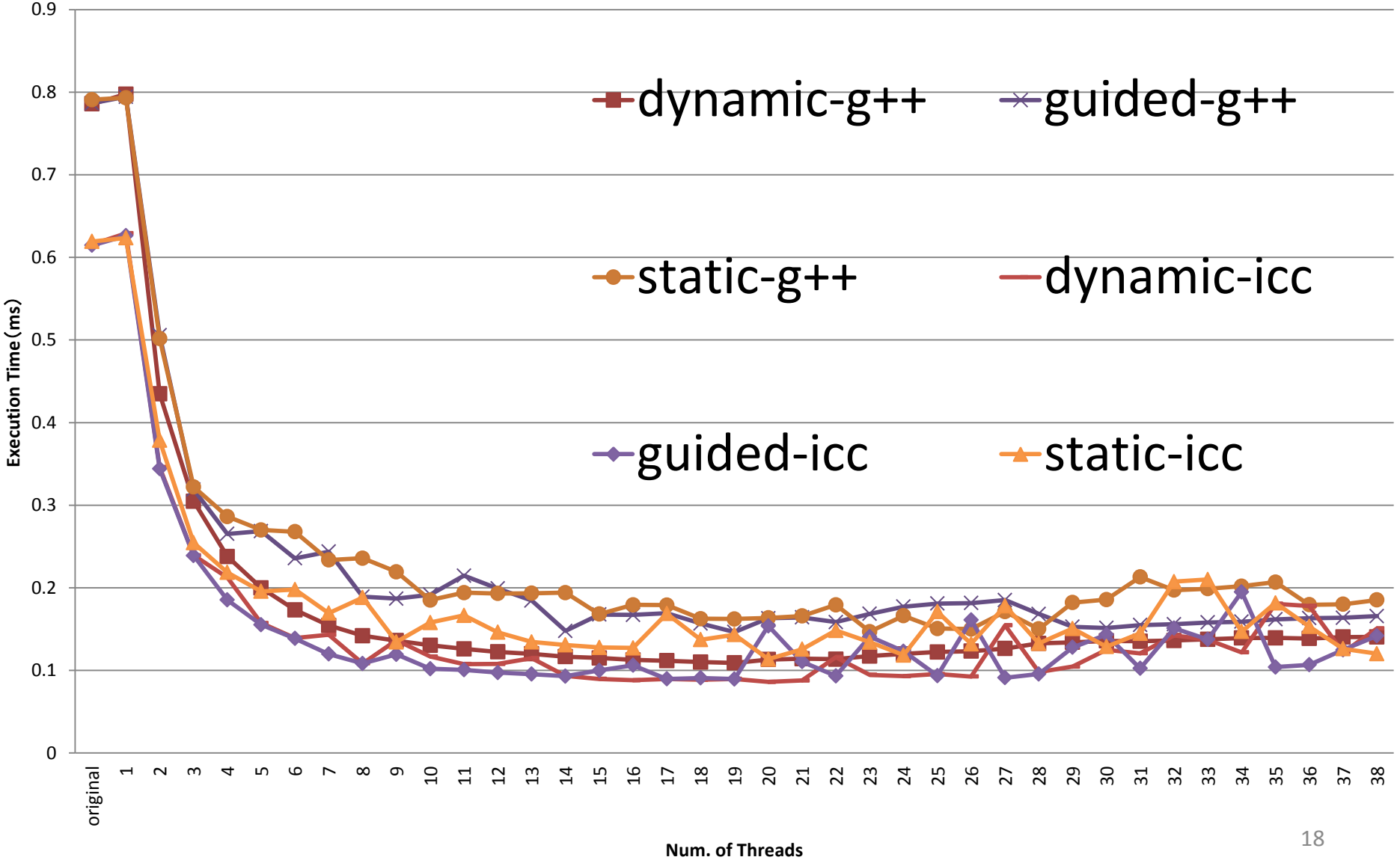
20-threads code is 6.6 times faster than the original sequential code.



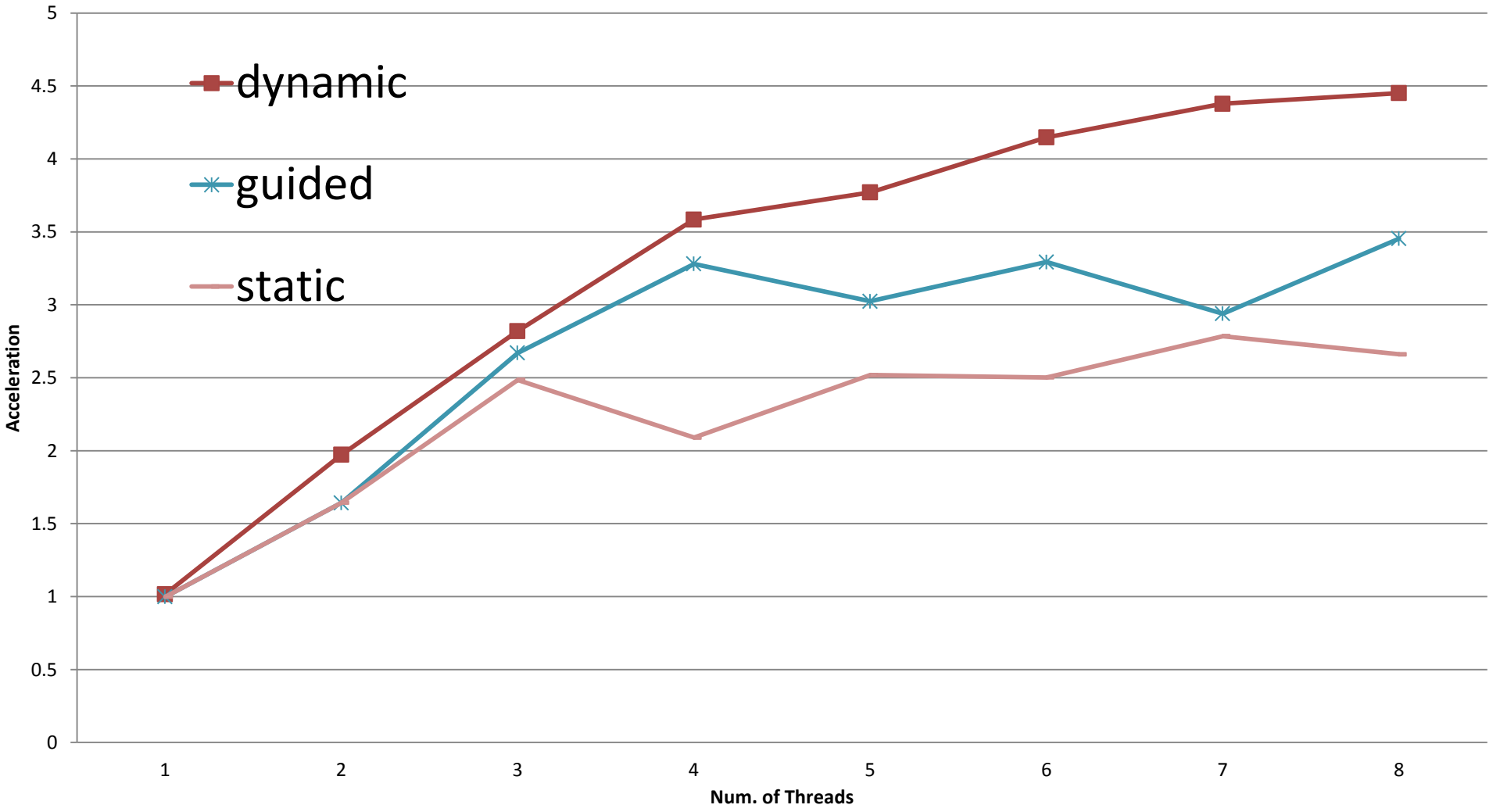
Execution Time (Core i7 PC)



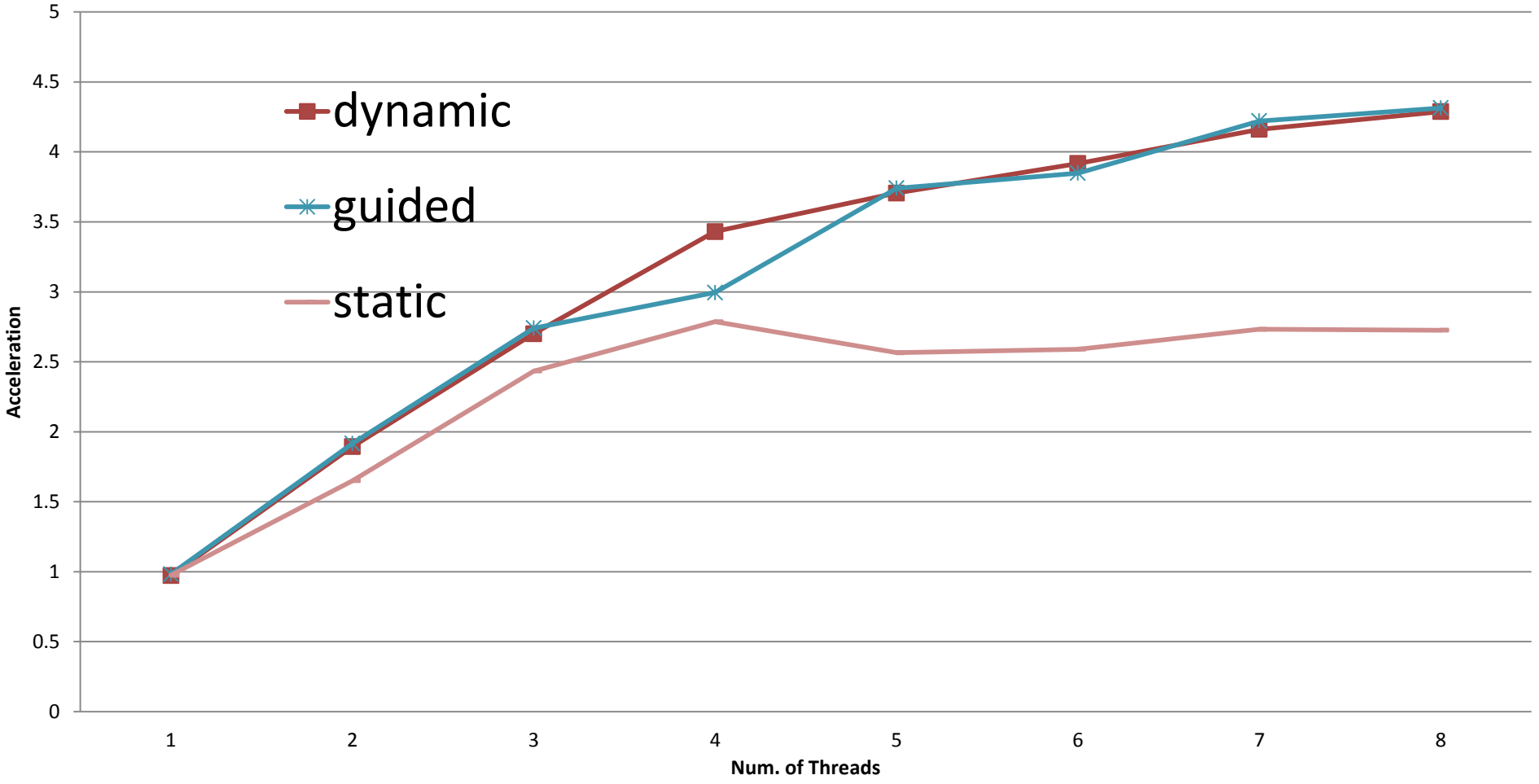
Execution Time (Xeon Server)



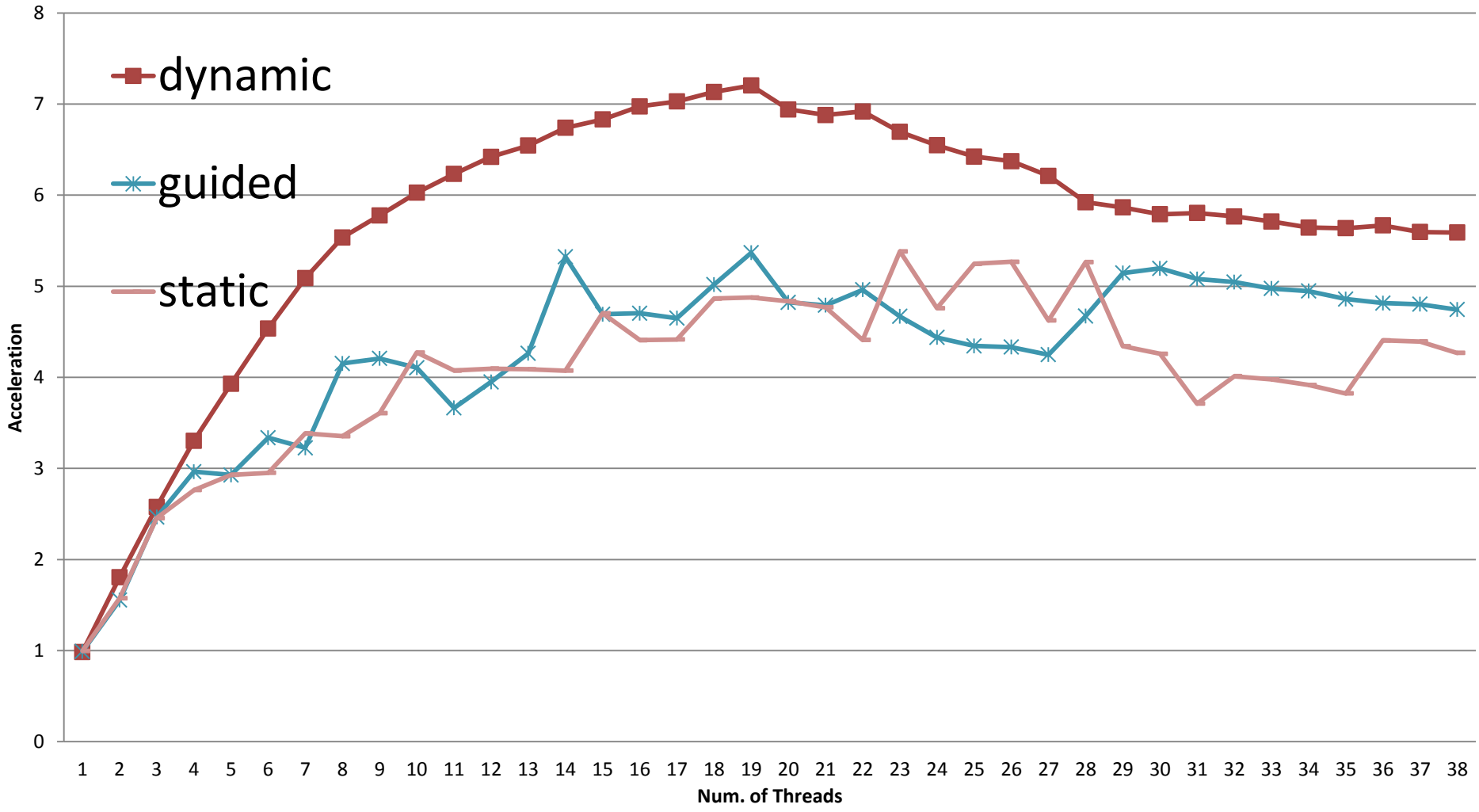
Acceleration (Core i7 PC g++)



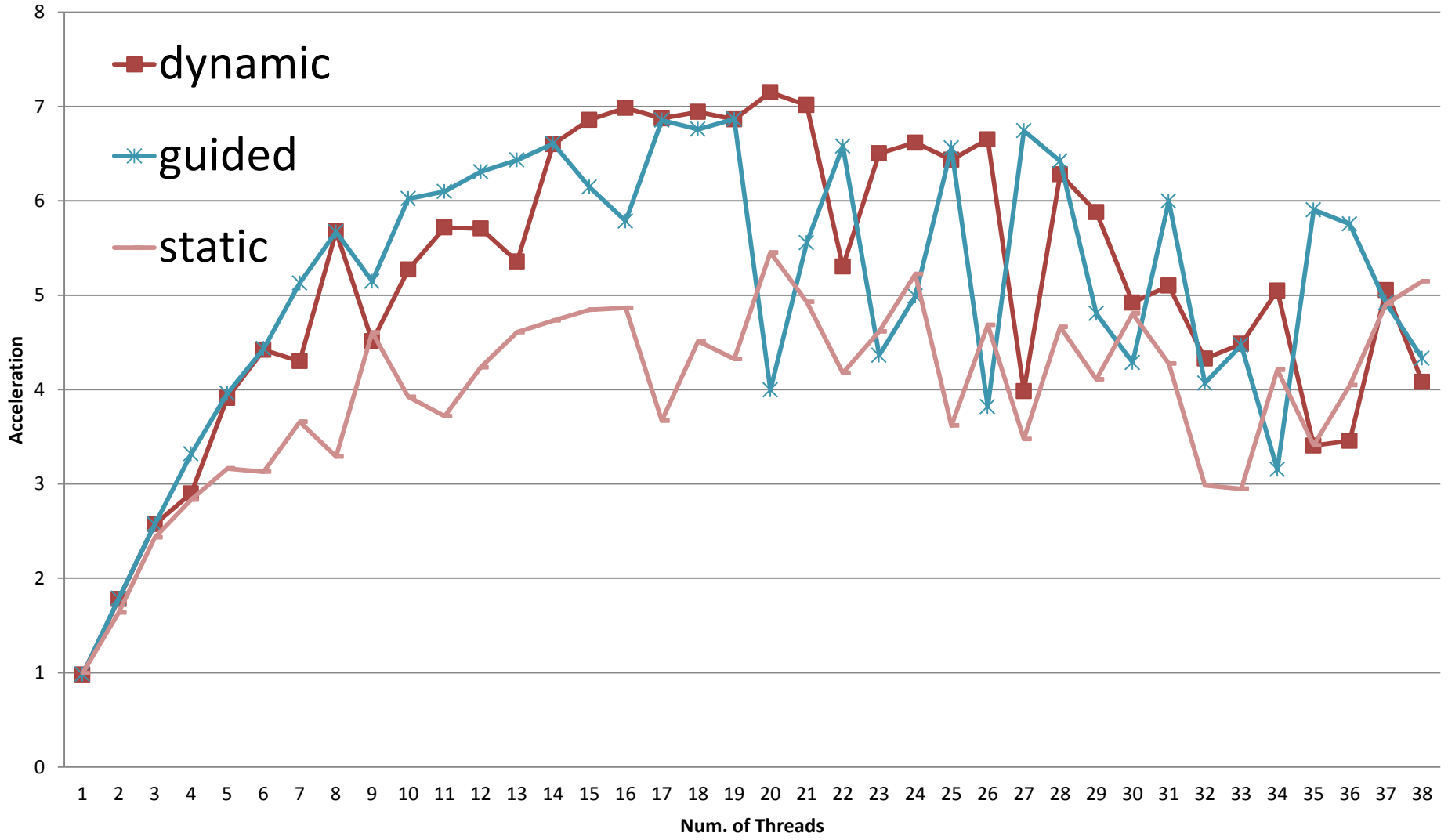
Acceleration (Core i7 PC icc)



Acceleration (Xeon Server g++)



Acceleration (Xeon Server icc)



Conclusion

- Acceleration of Inlet-Exhaust Pipe Simulation
  - Used Multi-processor Systems
  - Parallelize the outer loop for pipes
  - Enough and even size of Thread Task Granularity
  - Core i7 4790K PC 4.2 times faster than the original sequential code
  - Xeon E5-2680 v2 x2CPU Server 6.6 times faster than the original code

- Future work
  - Use SIMD instructions for the inner mesh loops, fine grain parallelism
  - Acceleration with GPU