2019.05.24(Fri) 13:40-14:20 iWAPT2019 in IPDPS2019@Hilton Rio de Janeiro Copacabana

RIIT RESEARCH INSTITUTE FOR INFORMATION TECHNOLOGY, KYUSHU UNIVERSITY 九州大学情報基盤研究開発センター

Satoshi Ohshima (Kyushu University),
Soichiro Suzuki (RIKEN), Tatsuya Sakashita (Tamagawa University),
Masao Ogino, Takahiro Katagiri, Yoshimichi Andoh (Nagoya University)

Performance evaluation of the MODYLAS application on modern multi-core and many-core environments

Motivations

1. Accelerating our molecular dynamics application on modern and future computers.
   - viewpoint of computational science

2. Utilizing and comparing modern parallel computers and applications executed on them.
   - viewpoint of computer science

me

- What/Where is Auto-Tuning(AT) in this work?
  - We think that optimizing and evaluating programs and comparing the performance on multiple hardware is very important to AT.
    - If there are no differences, AT is not needed.
    - If there are significant differences, there are some tuning parameters and criteria.

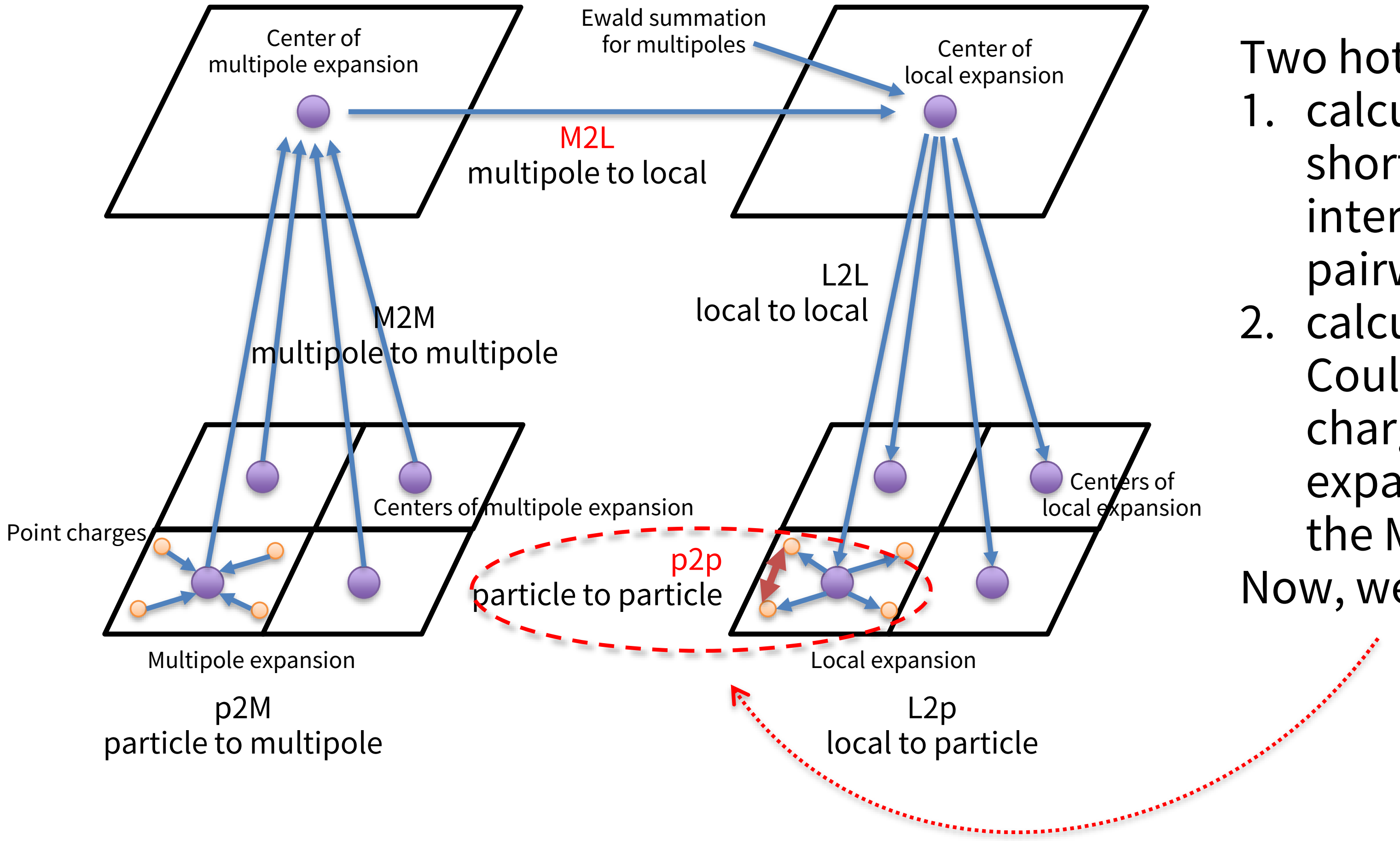
Background

- Molecular dynamics (MC) simulations
  - essential tool for research in chemistry, physics, biology, and virology
  - essential to obtain knowledge about materials at the molecular level, and in designing materials with novel functions arising from specific molecular features of the materials
- our target application: MODYLAS
  - MOlecular DYnamics software for LArge System
  - one of the infrastructure programs in the priority issue 5 supported by FLAGSHIP 2020 project (post-K computer project)
  - mainly developed by researchers in Nagoya University
  - free software published on http://www.modylas.org/
  - written in Fortran, parallelized by OpenMP and MPI
  - While there are many MD programs, MODYLAS aim to obtain good performance on large-scale computer systems, such as supercomputers.
  - previous target hardware was K computer
  - To catch up current computers (includes post-K), we have to consider wide SIMD and many cores.

The name of post-K was opened yesterday.
The name is 富岳 (Fugaku), means Mt.Fuji.

Hotspot of MODYLAS

- Operational procedures in the fast multipole method of MODYLAS:



Two hotspots:

1. calculation of the Lennard-Jones (LJ) and short-range part of the Coulombic interactions with neighboring atoms in a pairwise additive manner (p2p part)
2. calculation of long-range part of the Coulombic interactions with distant point charges by a combination of the multipole expansion and local expansion (especially, the M2L part)

Now, we focus on the p2p part.

Our previous work

- Yoshimichi Andoh, Soichiro Suzuki, Satoshi Ohshima, Tatsuya Sakashita, Masao Ogino, Takahiro Katagiri, Noriyuki Yoshii, Susumu Okazaki: A thread-level parallelization of pairwise additive potential and force calculations suitable for current many-core architectures, The Journal of Supercomputing, Vol.74, pp.2449--2469 (2018).
  - optimized MODYLAS for modern multi-core/many-core processors
  - previous work: K computer (8cores, 128bit SIMD)
  - main target: FX100 (32cores, 256bit SIMD)
  - sub target: KNC(60cores*4threads, 512bit SIMD)
  - developed new OpenMP implementations (next slide) and evaluated the performance

- In this (iWAPT2019) work, how about current Intel's processors?
  - SKX: Xeon Scalable Processor, Skylake-SP, AVX-512 (F, CD, ER, PF)
  - KNL: Xeon Phi, Knights Landing, AVX-512 (F, CD, VL, DQ, BW)

Four new algorithms developed in our previous work

maximum number of threads of original algorithm is always 40,
not enough to fill the all cores of current CPUs

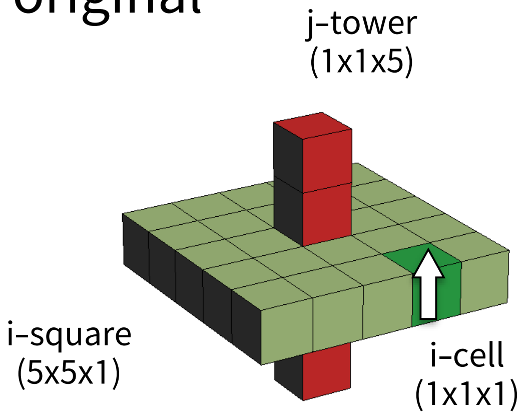
algorithm	parallel granularity of each thread	*maximum number of adoptable threads for each algorithm*				
		#total processes Nx*Ny*Nz	1 8x8x8	8 4x4x4	64 2x2x2	512 1x1x1
original	8,000 / Nt	40	40	40	40	40
code1	8,000 Nz / Nt	40Nz	320	160	80	40
code2	8,000	25NxNyNz	12,800	1,600	200	25
code3	8,000 – 200,000	(Nx+4)(Ny+4)Nz	1,152	256	72	25
code4	8000 Nz	25NxNy	1,600	400	100	25

- Average number of atoms in each subcell is assumed to be 40.
- Nx , Ny and Nz are number of subcells distributed to the MPI process along the x, y, and z axes, respectively.
- Nt is a given thread number.

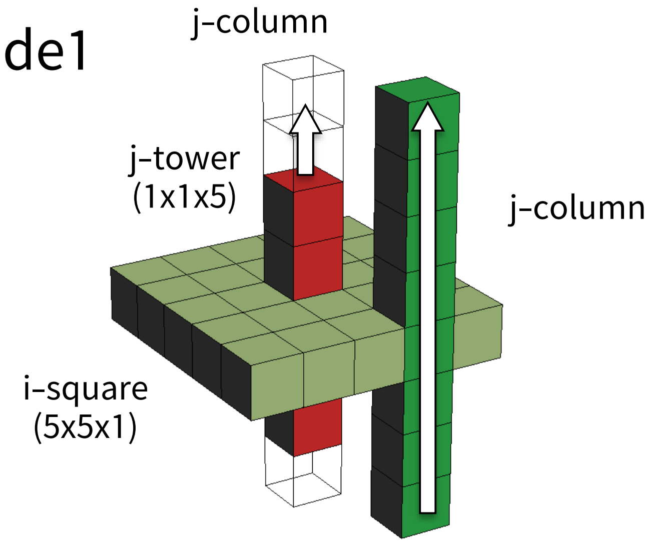
512 processes
all algorithms cannot fill the cores

Access pattern of all five algorithms in the p2p operations

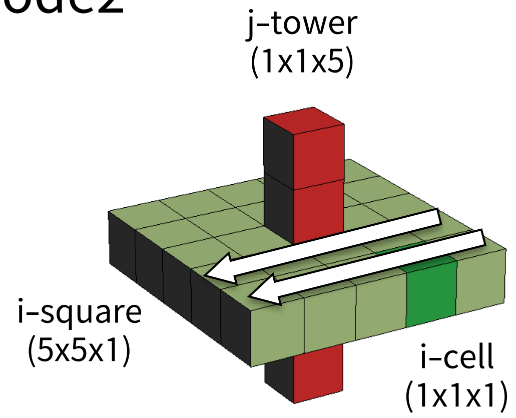
original



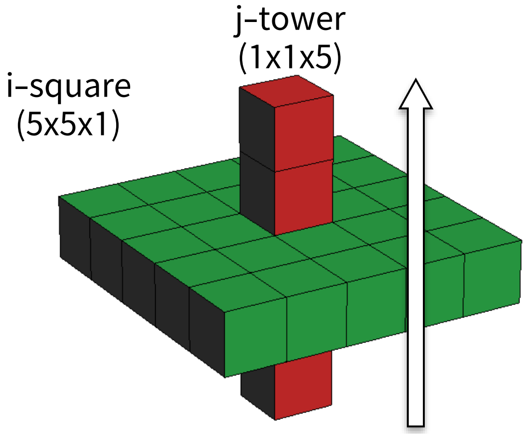
code1



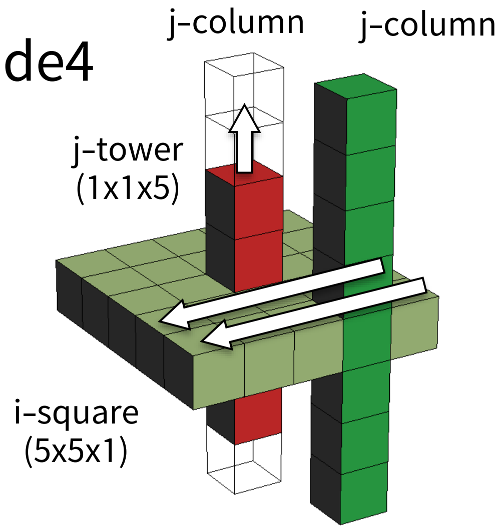
code2



code3



code4



- differences
  - order of loops
  - lengths of the OMP and SIMD target loop
  - creation tables in sequential, etc.

Example code of the original algorithm

many calculations in SIMD loop

```
!$omp do
do i0=tag(iz,iy,ix), tag(iz,iy,ix)+na_per_cell(iz,iy,ix)-1
  i=m2i(i0)
  epsilon_sqrt_i0=epsilon_sqrt(paranum(i))
  R_half_i0=R_half(paranum(i))
  chgv_i0=chgv(paranum(i))
  xi=wkxyz(1,i0)
  yi=wkxyz(2,i0)
  zi=wkxyz(3,i0)
  ic=1
  stlcx=0.d0
  stlcy=0.d0
  stlcz=0.d0
! SIMD target
  do j0=tag(jzb-2,jyb,jxb), tag(jzb+2,jyb,jxb) &
      + na_per_cell(jzb+2,jyb,jxb)-1
    rx=xi-wkxyz(1,j0)
    ry=yi-wkxyz(2,j0)
    rz=zi-wkxyz(3,j0)
    r2=rx*rx+ry*ry+rz*rz
    r2_r=1.d0/r2
    if(r2<=cutrad2) then
       eps=epsilon_sqrt_i0 * epsilon_sqrt_table(ic,iam)
    else
       eps=0d0
    endif !cut-off
    R=R_half_i0+R_half_table(ic,iam)
    Rr6=R * R * r2_r
    Rr6=Rr6 * Rr6 * Rr6
    Rr12=Rr6 * Rr6
    coef=12.d0 * eps * r2_r * (Rr12-Rr6)
    tlx=coef*rx
    tly=coef*ry
    tlz=coef*rz
    Ulj12=     eps*Rr12
    Ulj6 =-2d0*eps*Rr6
    sUlj12=sUlj12+Ulj12
    sUlj6 =sUlj6 +Ulj6
    stlcx=stlcx+tlx
    stlcy=stlcy+tly
    stlcz=stlcz+tlz
    rc =sqrt(r2)
    rc_r=1.d0/rc
    rc2_r=rc_r*rc_r
    Cij=chgv_i0*chgv_table(ic,iam)
    Cij=Cij*rc_r
    tmp=Cij*rc2_r
    tcx=tmp*rx
    tcy=tmp*ry
    tcz=tmp*rz
    Ucoulomb=Cij
    sUcoulomb=sUcoulomb+Ucoulomb
    stlcx=stlcx+tcx
    stlcy=stlcy+tcy
    stlcz=stlcz+tcz
    ic=ic+1
  enddo !j0
  w3_f(1,i0,0)=w3_f(1,i0,0)+stlcx
  w3_f(2,i0,0)=w3_f(2,i0,0)+stlcy
  w3_f(3,i0,0)=w3_f(3,i0,0)+stlcz
enddo !i0
!$omp end do
```

Execution environments

	multi-core		many-core	
	FX100	**ITO**	**OFP**	**KNCC (retired system)**
Processor	SPARC64 XIfx	Xeon Gold 6154	Xeon Phi 7150	Xeon Phi P5110
Installed System	Nagoya University FX100	Kyushu University ITO	JCAHPC Oakforest-PACS	UTokyo Experimental cluster
#processor/node	1	2 (use only 1 socket)	1	1
Frequency	2.2 GHz	3.0-3.7 GHz	1.4-1.6 GHz	1.05 GHz
#cores/socket	32 (+ 2 assistants)	18 (use only 16 cores)	68 (use only 64 cores)	60
HPL performance/1socket	1.0 TF	1.1 TF	1.6 TF	1.0 TF
Memory kind, amount/socket	HMC 32 GB	DDR4 96 GB	MCDRAM 16 GB	GDDR 8 GB
STREAM Triad/socket	210 GB/s	95 GB/s	495 GB/s	140 GB/s
Interconnect	Tofu2	IB-EDR	OPA	*
Compiler, MPI	Fujitsu TCS	Intel 18.0.0 (& OpenMPI, MVAPICH)	Intel 18.0.1	Intel 17.0.4

Performance evaluation: measurement and analysis

- We measured the execution time on target hardware and analyzed the trends, and compared them.
- Evaluation Environment: compiler, MPI, and compiler option:
  - FX100: Fujitsu TCS, frtpx -Kfast,simd=2,openmp,parallel,ocl
  - ITO: Intel 18.0.0, MVAPICH 2.2, ifort -O3 -qopenmp -align array64byte -xCORE-AVX512
  - OFP: Intel 18.0.1, ifort -O3 -qopenmp -align array64byte -xMIC-AVX512
    - Flat-mode & Quadrant-mode
  - KNCC: Intel 17.0.4, ifort -O3 -mmic -qopenmp, native execution mode

We tried -xCOMMON-AVX512 but performance improvement is not obtained.

- various combinations of #processes and #threads
  - *m* P *n* T : *m* process(es) per node, *n* thread(s) per process
    - 1P32T : 1process per node, 32 threads per process (total 32 threads per node)
    - 4P8T : 4processes per node, 8 threads per process (total 32 threads per node)

Problem settings

- Target data
  - a cubic calculation unit cell with a side length of 58.62 ˚ A contains 6,510 water molecules
  - assuming the liquid state
  - To apply the FMM, the unit cell is decomposed into 8 3 smaller cells, named subcells, according to the octree cell partitioning with three levels.
  - Number of atoms in each subcell is 40 on average.
  - Fourth order expansions were adopted for multipole and local expansions, while its order does not affect the performance of the p2p part.
  - can be executed by up to 512 processes
  - Number of processes affects execution time and its dispersion among processes, because atoms do not exist equally in the space, and atoms in each process interact with atoms in neighboring processes.
  - Therefore, we choose the longest elapsed time of all processes because the total execution time depends on the process that requires the longest execution time of all processes.

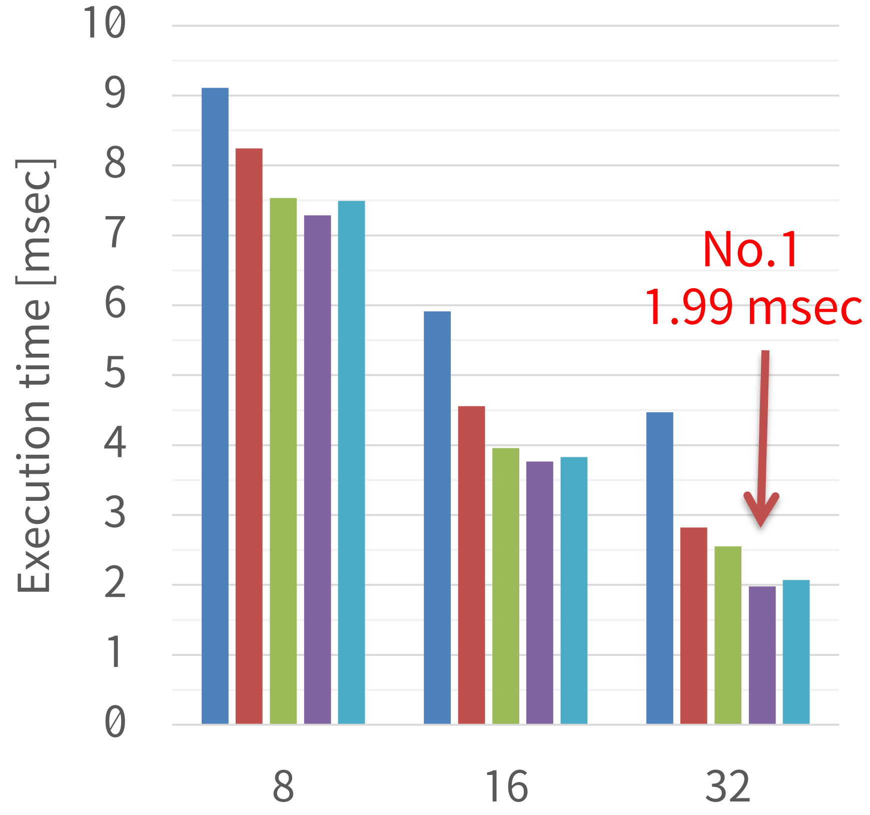
Result of our previous work

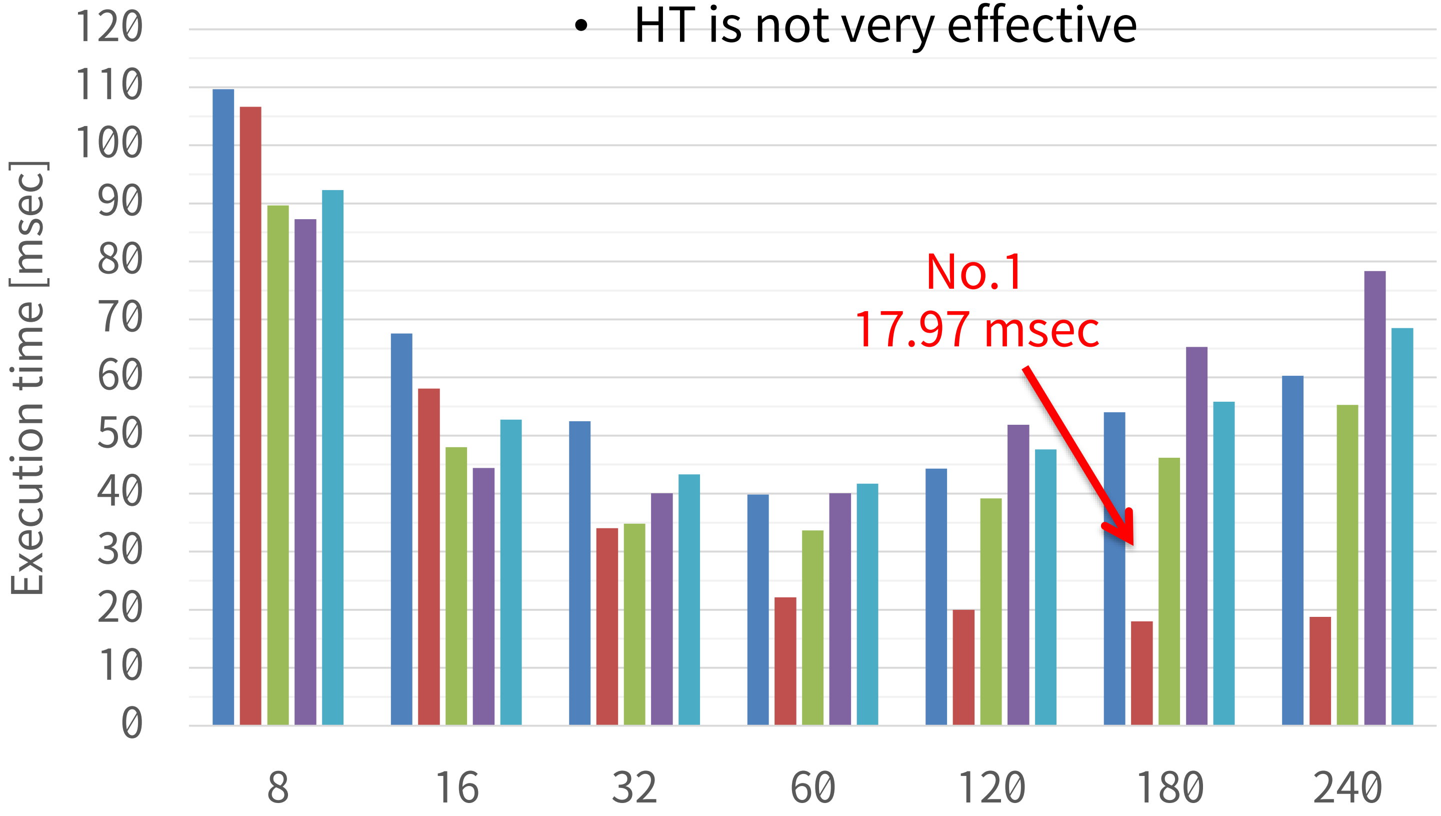
FX100

- code3 and code4 achieve good performance

KNCC

- code2 achieves good performance
- x8 execution time to FX100 (very slow)
- HT is not very effective

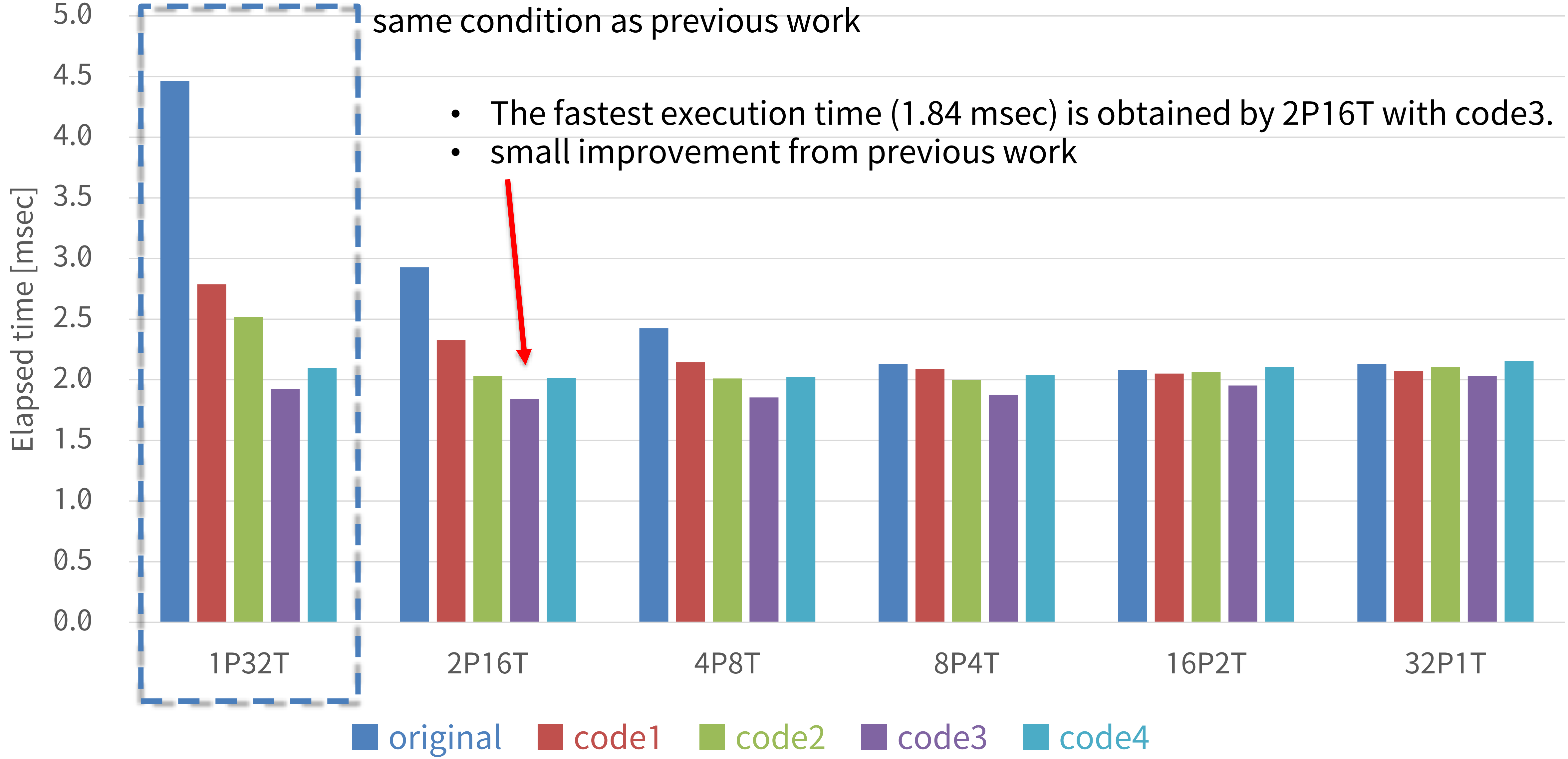




#Threads / process (= #Threads / node)

■ original ■ code1 ■ code2 ■ code3 ■ code4

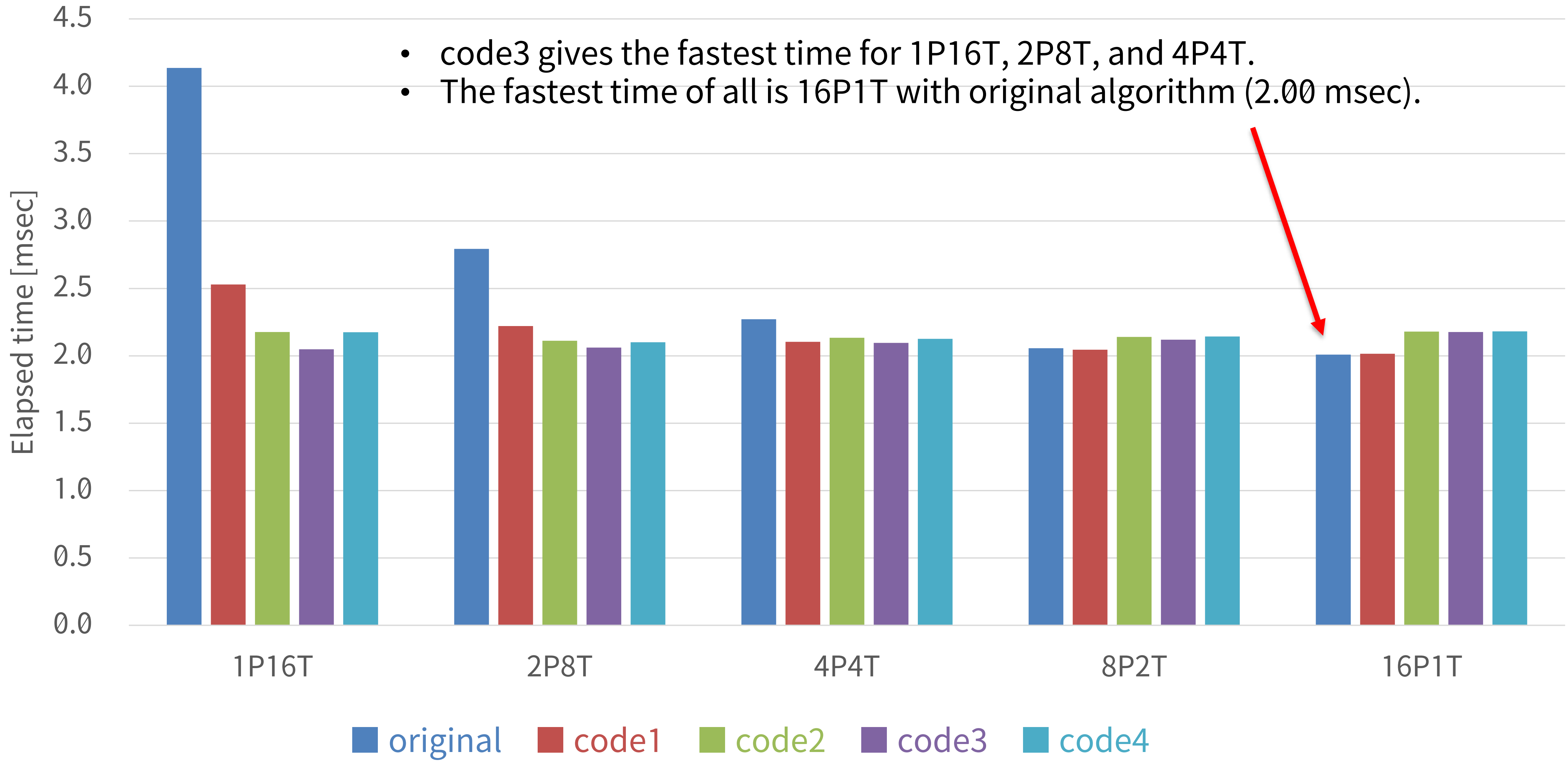
Result: Execution time of p2p part on FX100, 32 cores * 8 nodes



same condition as previous work

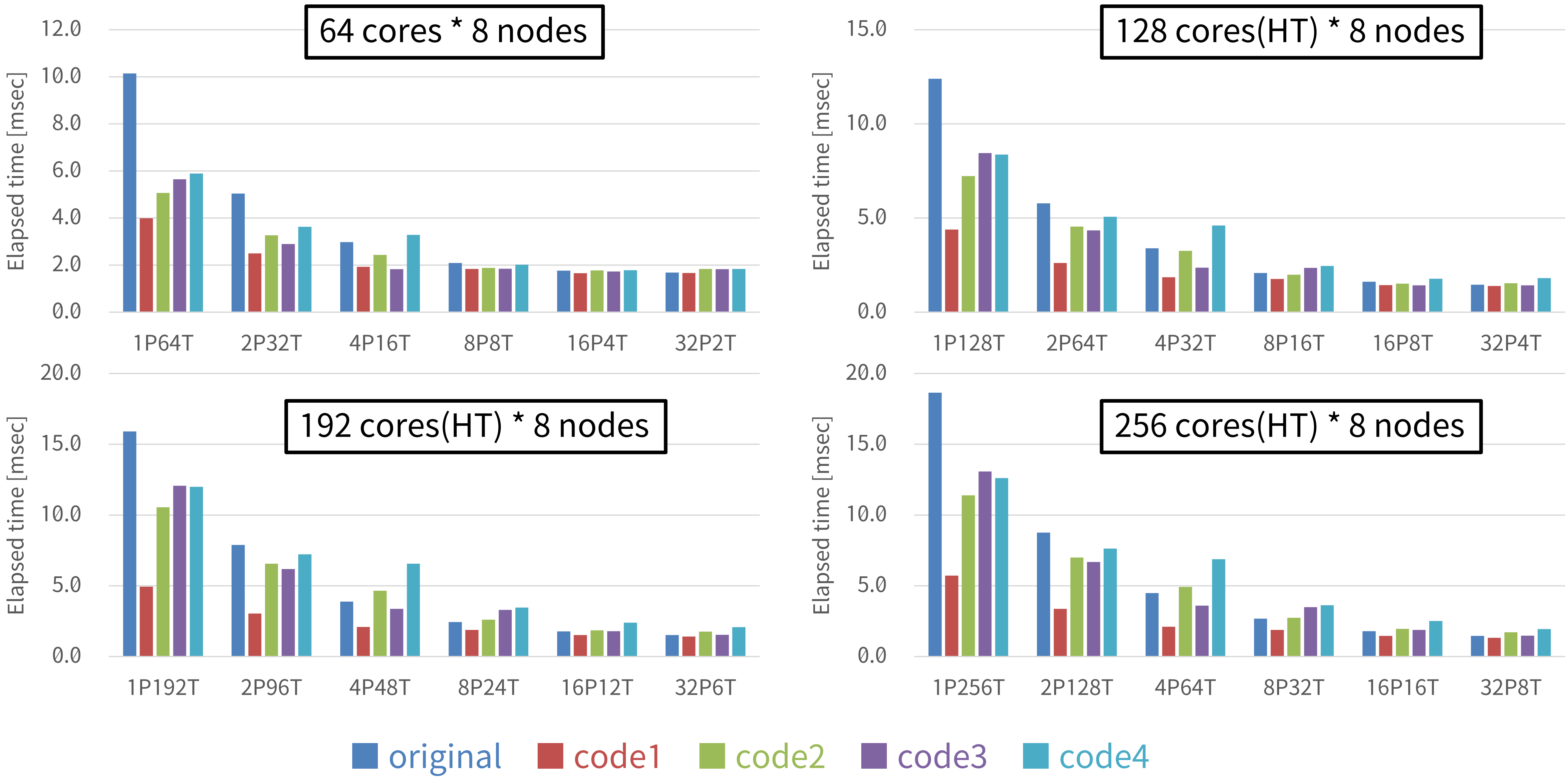
- The fastest execution time (1.84 msec) is obtained by 2P16T with code3.
- small improvement from previous work

Result: Execution time of p2p part on ITO, 16 cores * 8 nodes

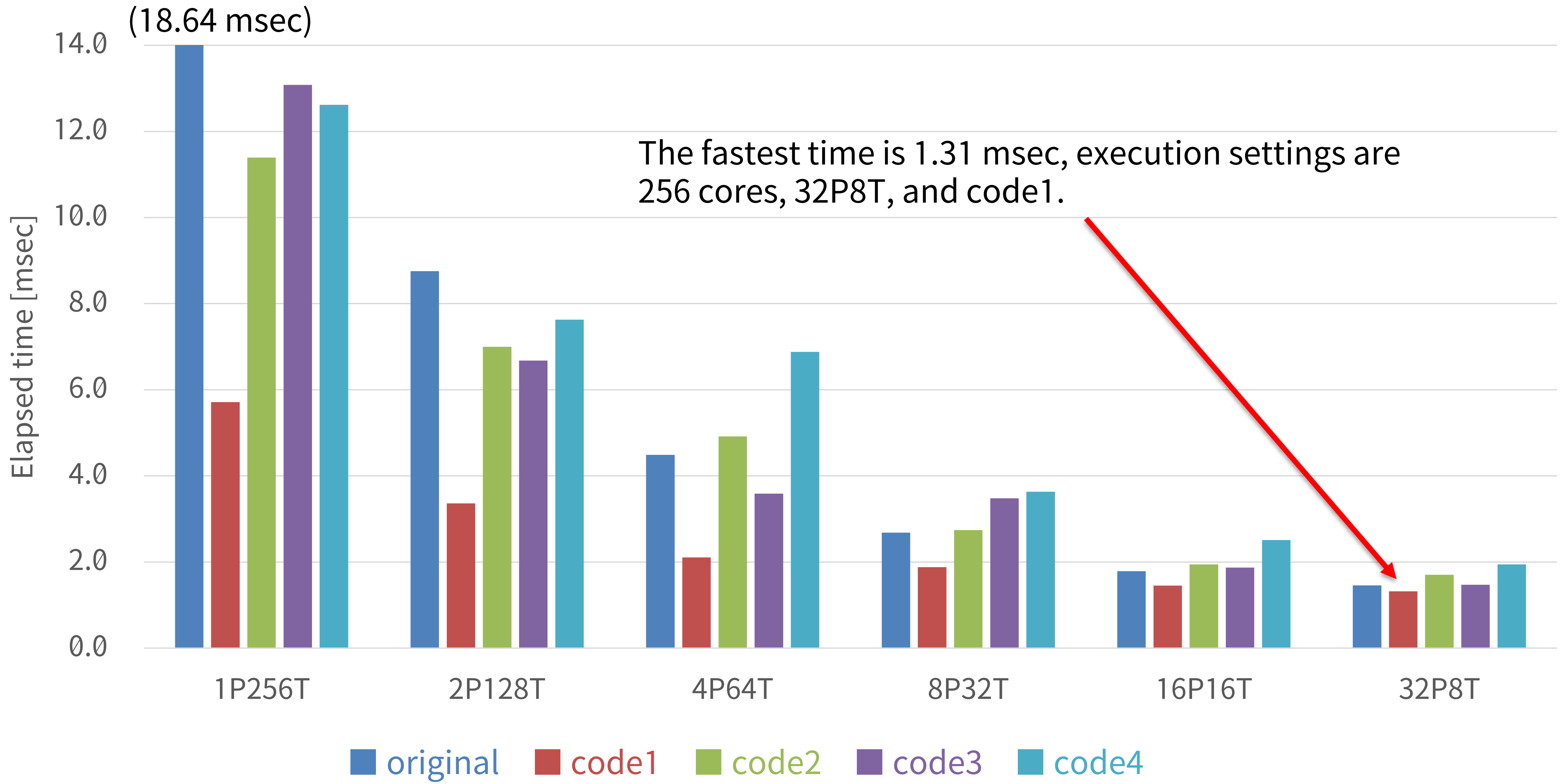


- code3 gives the fastest time for 1P16T, 2P8T, and 4P4T.
- The fastest time of all is 16P1T with original algorithm (2.00 msec).

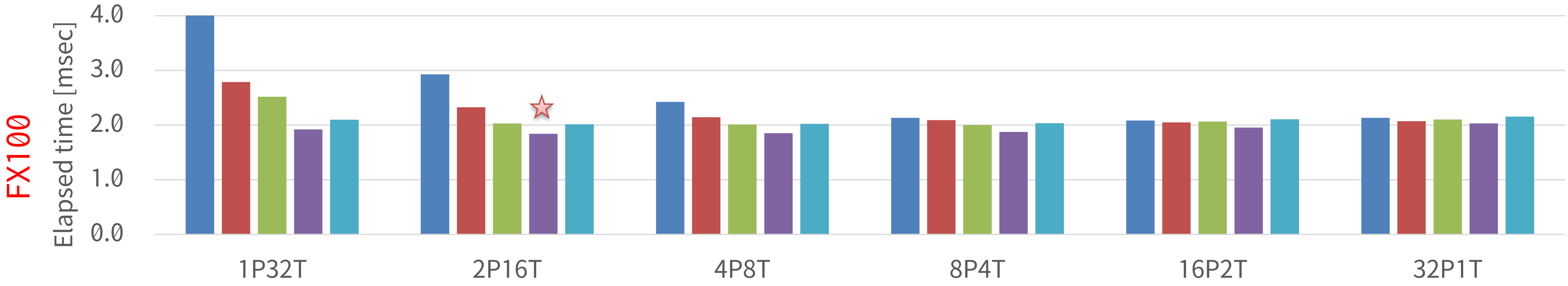
Result: Execution time of p2p part on OFP, 8 nodes

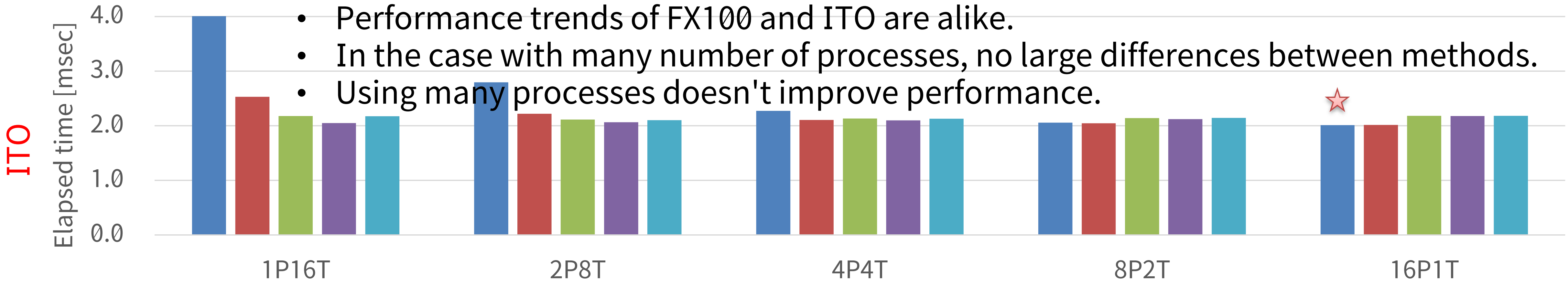


Result: Execution time of p2p part on OFP, 8 nodes

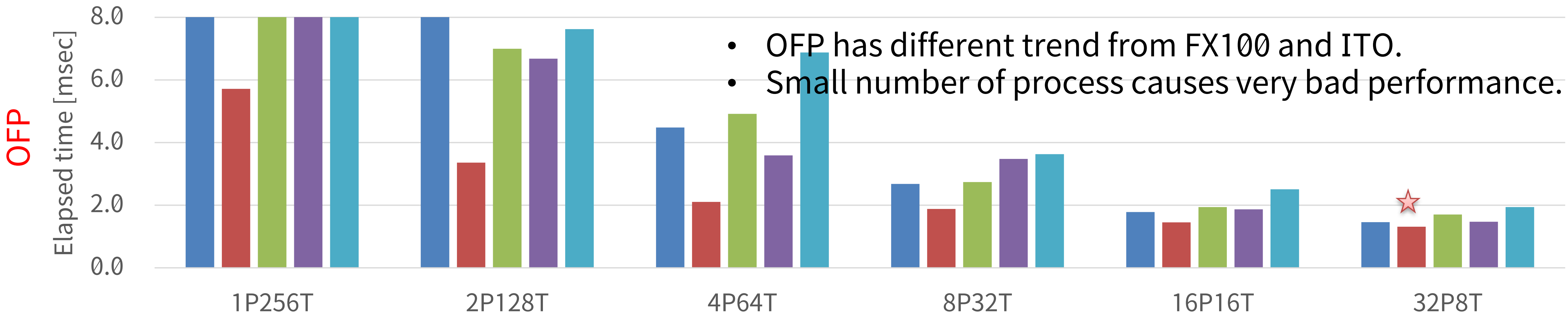


The fastest time is 1.31 msec, execution settings are 256 cores, 32P8T, and code1.





- Performance trends of FX100 and ITO are alike.
- In the case with many number of processes, no large differences between methods.
- Using many processes doesn't improve performance.



- OFP has different trend from FX100 and ITO.
- Small number of process causes very bad performance.

The fastest time and corresponding cases (8nodes)

	time	case
FX100	1.84 msec	2P16T code3
ITO	2.00 msec	16P1T original
OFP	1.31 msec	32P8T code1

- OFP obtained the fastest performance, but the differences between other environment are smaller than benchmark scores.
- (Maybe) because MODYLAS is much more difficult than simple benchmarks.

	HPL	STREAM Triad	MODYLAS (p2p)
FX100 vs OFP	1.0TF vs 1.6TF + 60%	210 GB/s vs 495 GB/s +135%	1.84 msec vs 1.31 msec -29%
ITO vs OFP	1.1TF vs 1.6TF + 45%	95 GB/s vs 495 GB/s +420%	2.00 msec vs 1.31 msec -35%

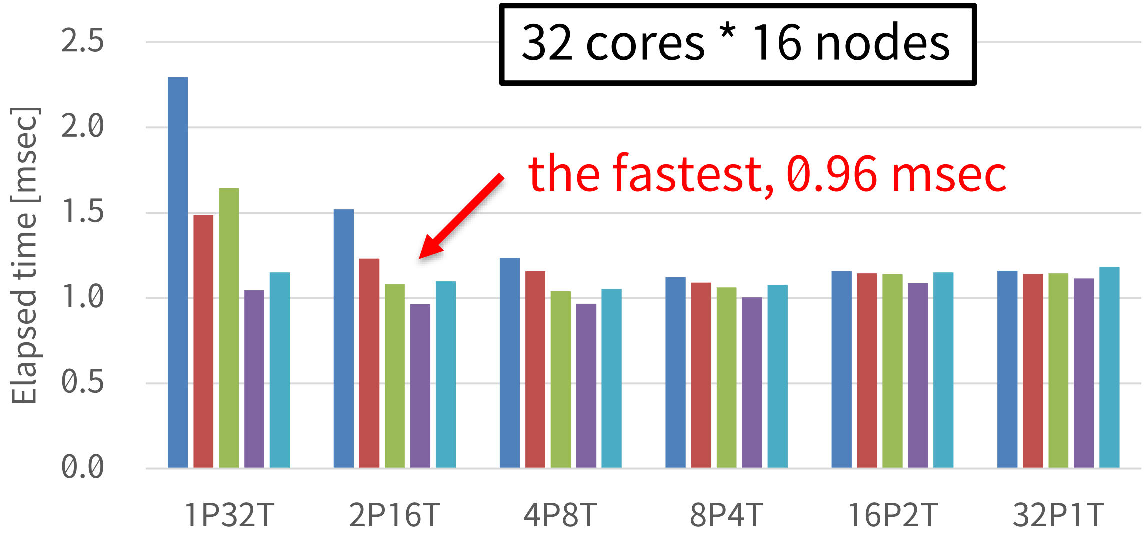
- Next, how about many nodes?

Strong scaling evaluation

- Our previous work and evaluations above use only 8 nodes.
- How about more many nodes?
- limitation: target data can be executed by up to 512 processes
- We measured the execution time on 16 ,32, and 64 nodes, and compared them.

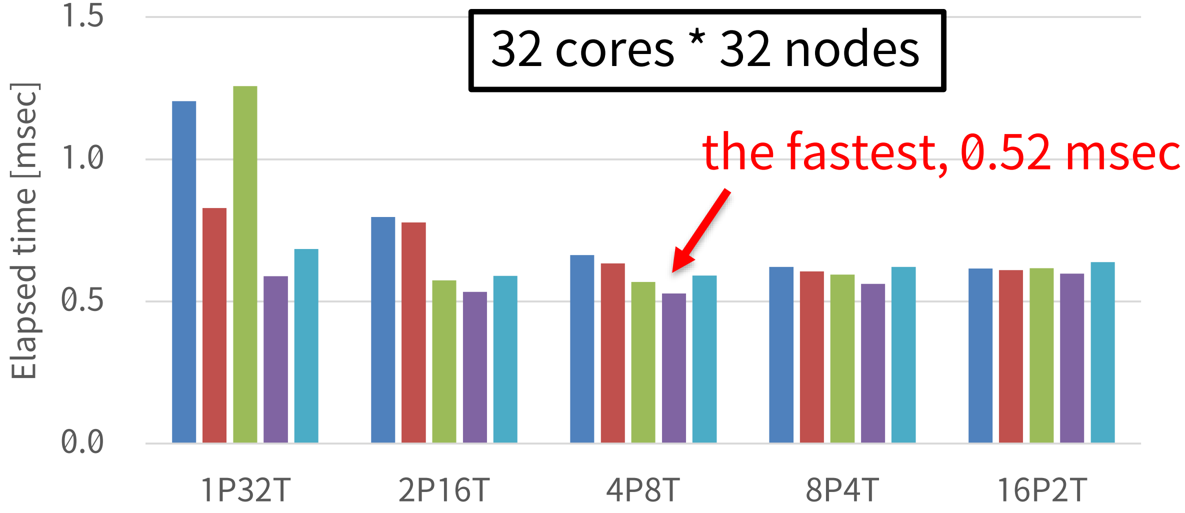
Result: Execution time of p2p part on FX100, 16, 32, and 64 nodes

- 2P or 4P with code3 is fast
- the shortest time will be reduced more than 64 nodes



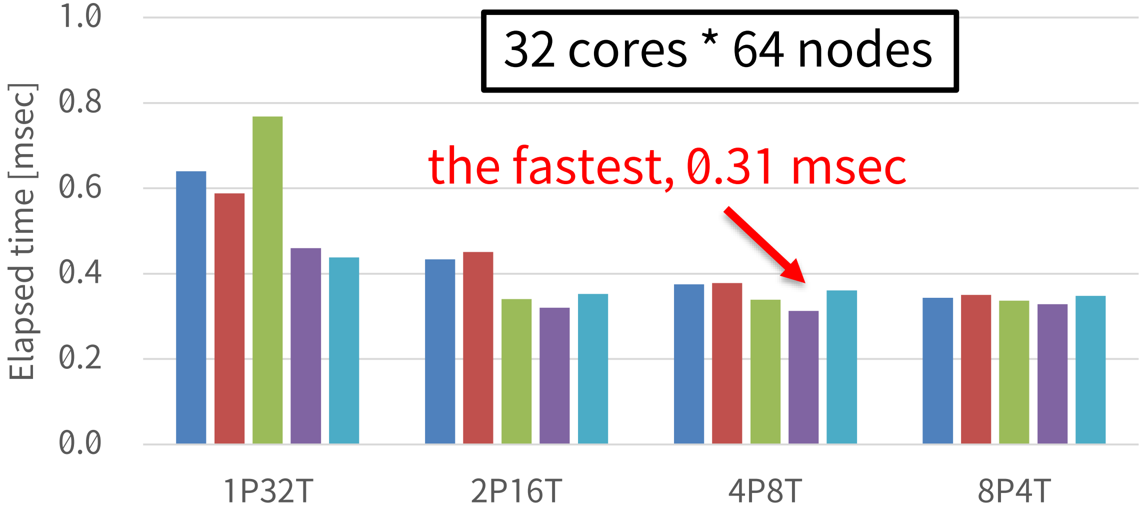
32 cores * 16 nodes

the fastest, 0.96 msec



32 cores * 32 nodes

the fastest, 0.52 msec



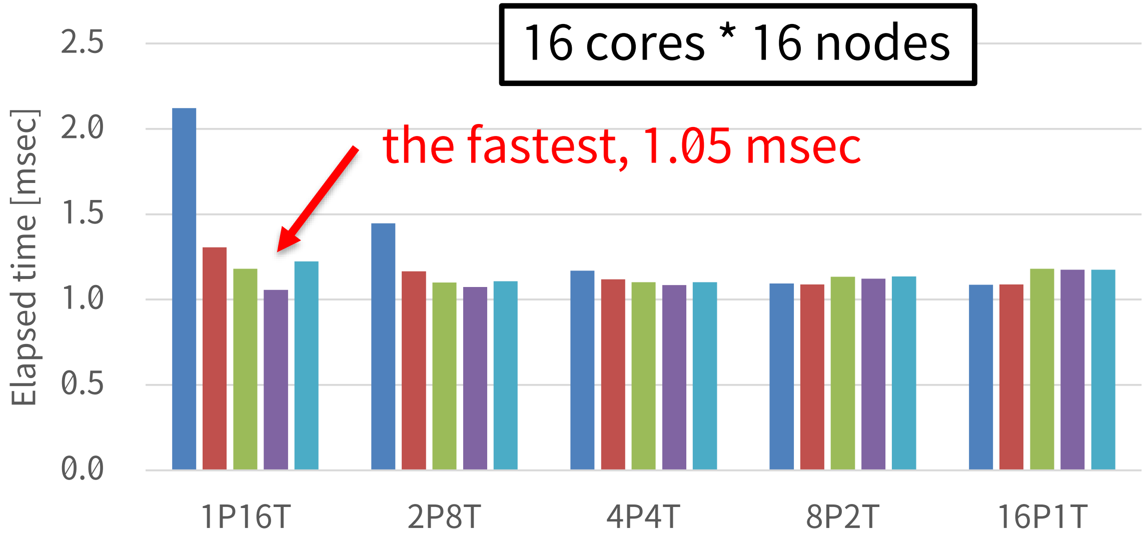
32 cores * 64 nodes

the fastest, 0.31 msec

■ original ■ code1 ■ code2 ■ code3 ■ code4

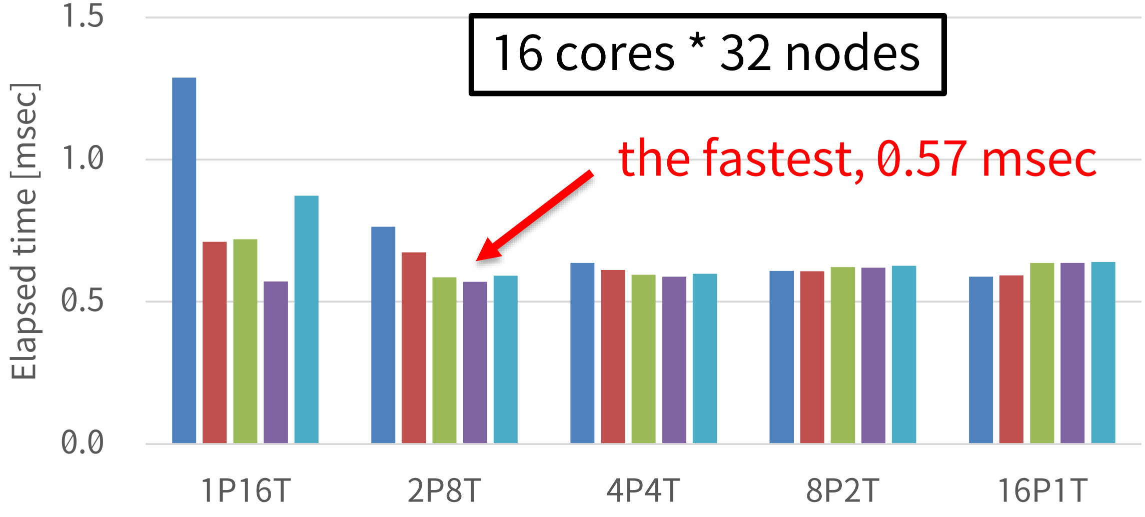
Result: Execution time of p2p part on ITO, 16, 32, and 64 nodes

- 1P or 2P with code3 is fast
- the fastest time will be reduced more than 64 nodes
- the trends are similar to FX100



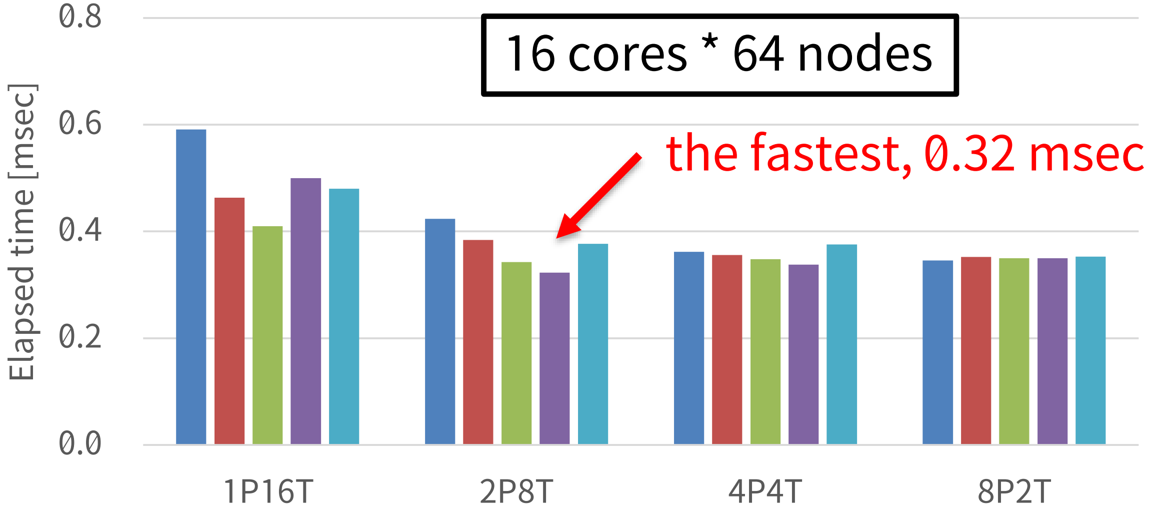
16 cores * 16 nodes

the fastest, 1.05 msec



16 cores * 32 nodes

the fastest, 0.57 msec

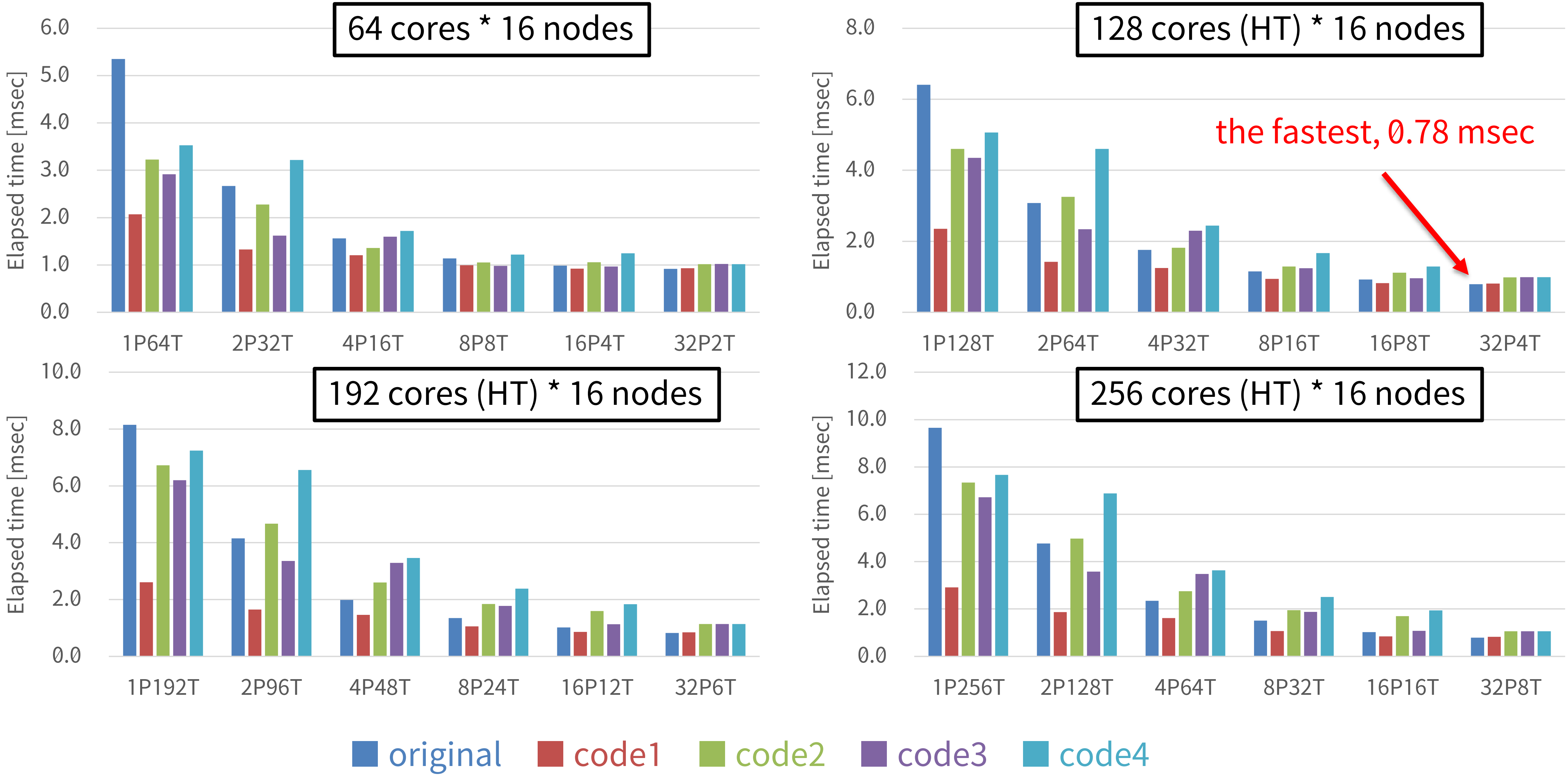


16 cores * 64 nodes

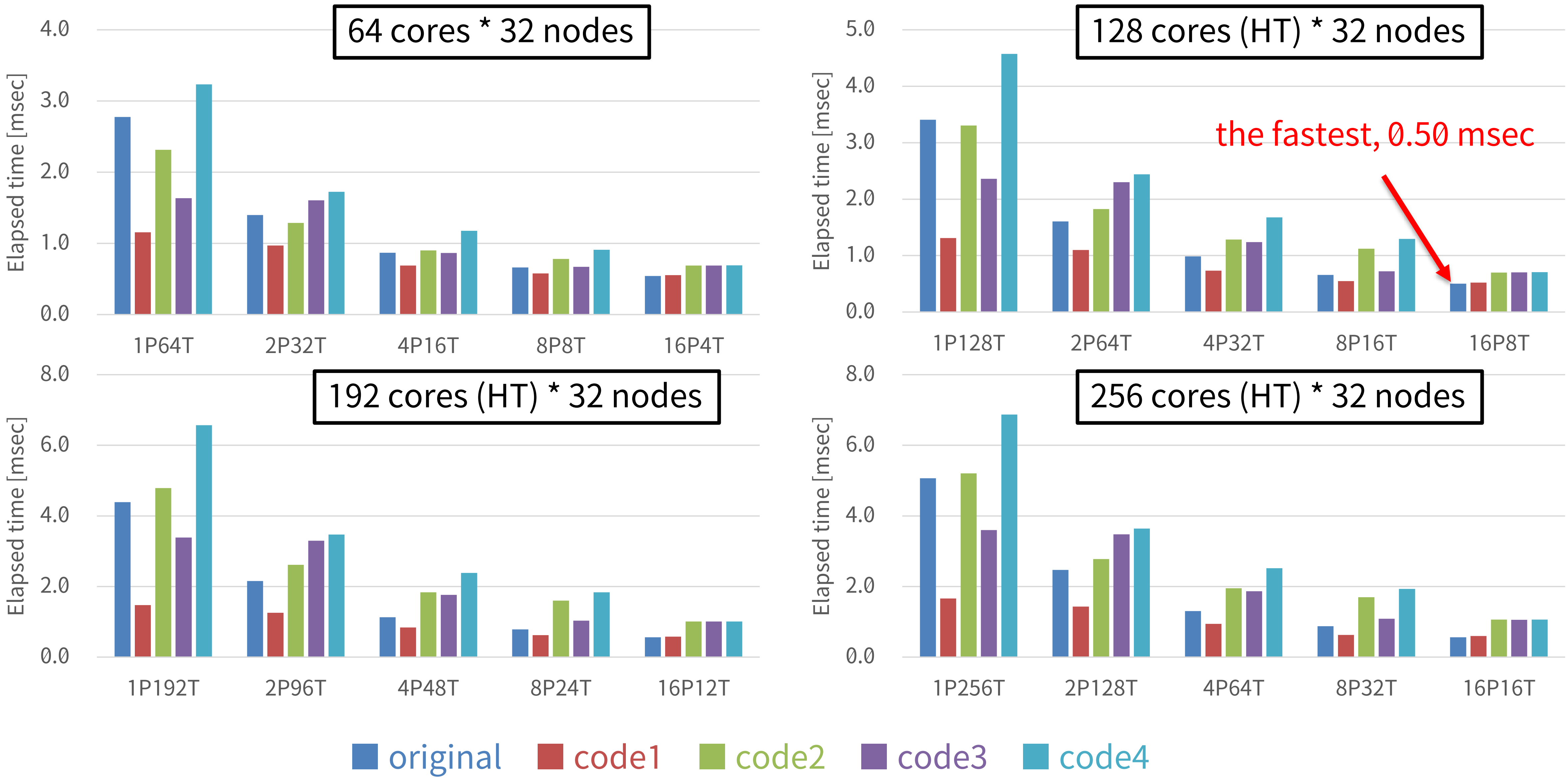
the fastest, 0.32 msec

🟦 original 🟥 code1 🟩 code2 🟪 code3 🟦 code4

Result: Execution time of p2p part on OFP, 16 nodes



Result: Execution time of p2p part on OFP, 32 nodes



64 cores * 32 nodes

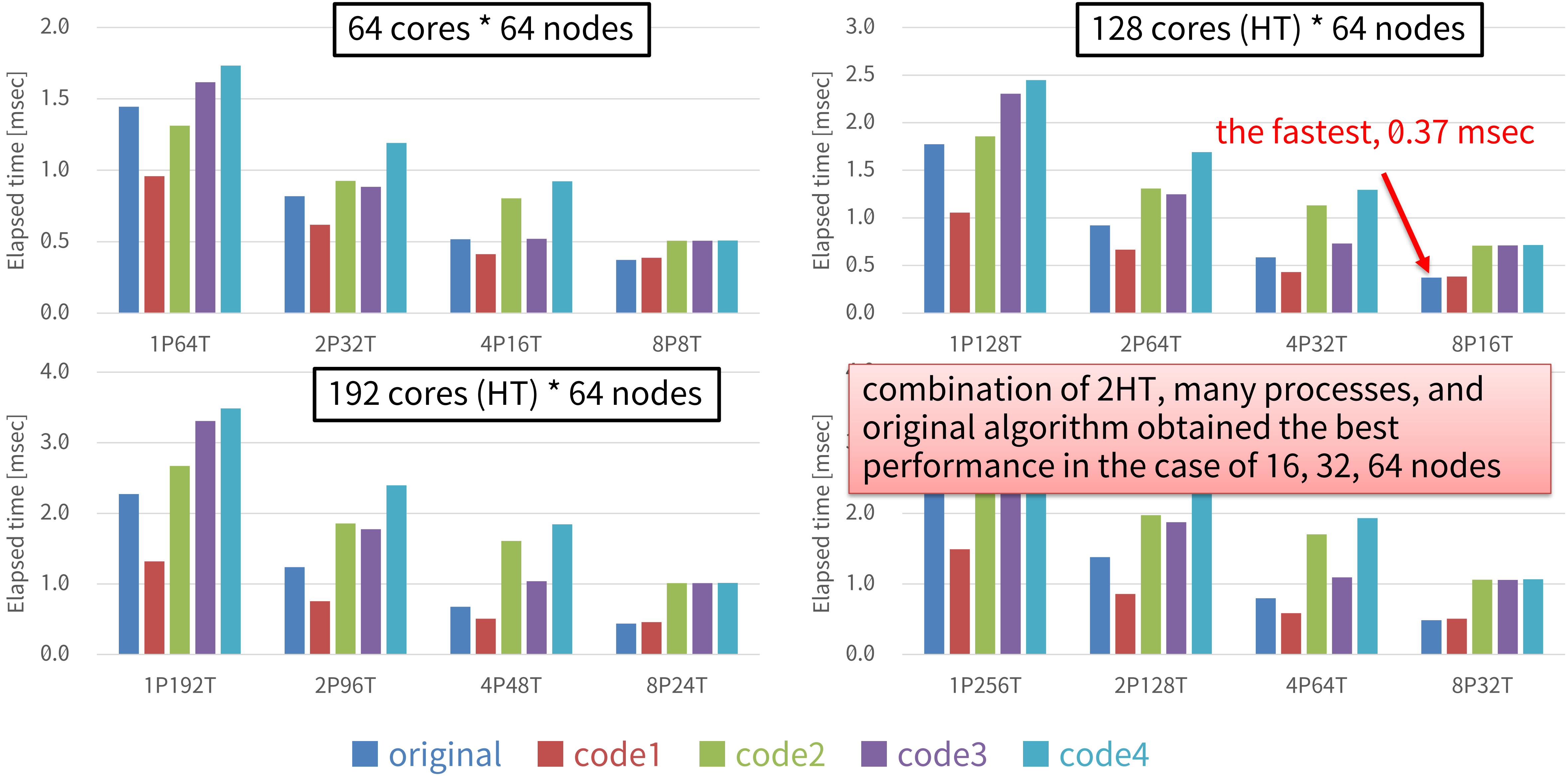
128 cores (HT) * 32 nodes

192 cores (HT) * 32 nodes

256 cores (HT) * 32 nodes

the fastest, 0.50 msec

Result: Execution time of p2p part on OFP, 64 nodes



64 cores * 64 nodes

128 cores (HT) * 64 nodes

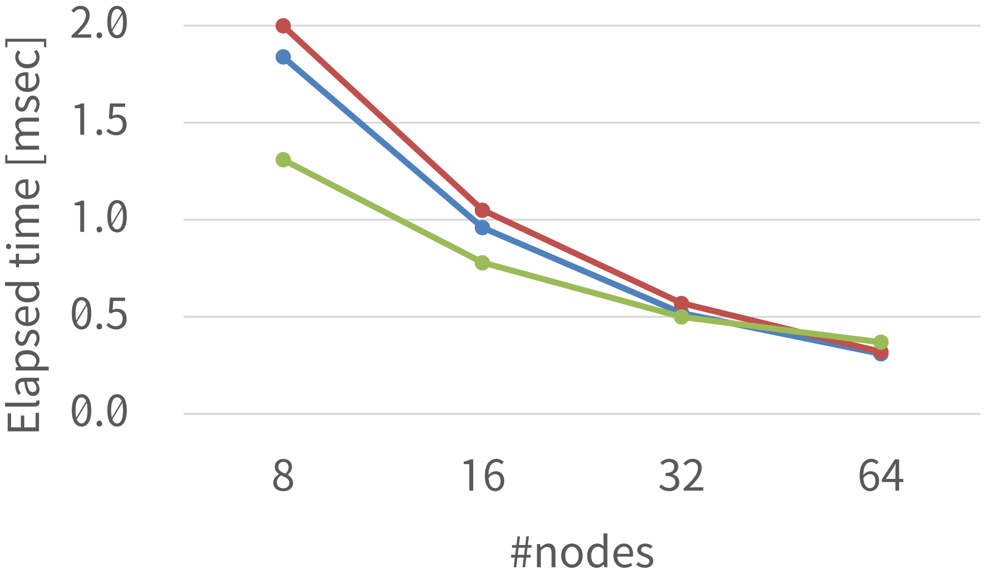
the fastest, 0.37 msec

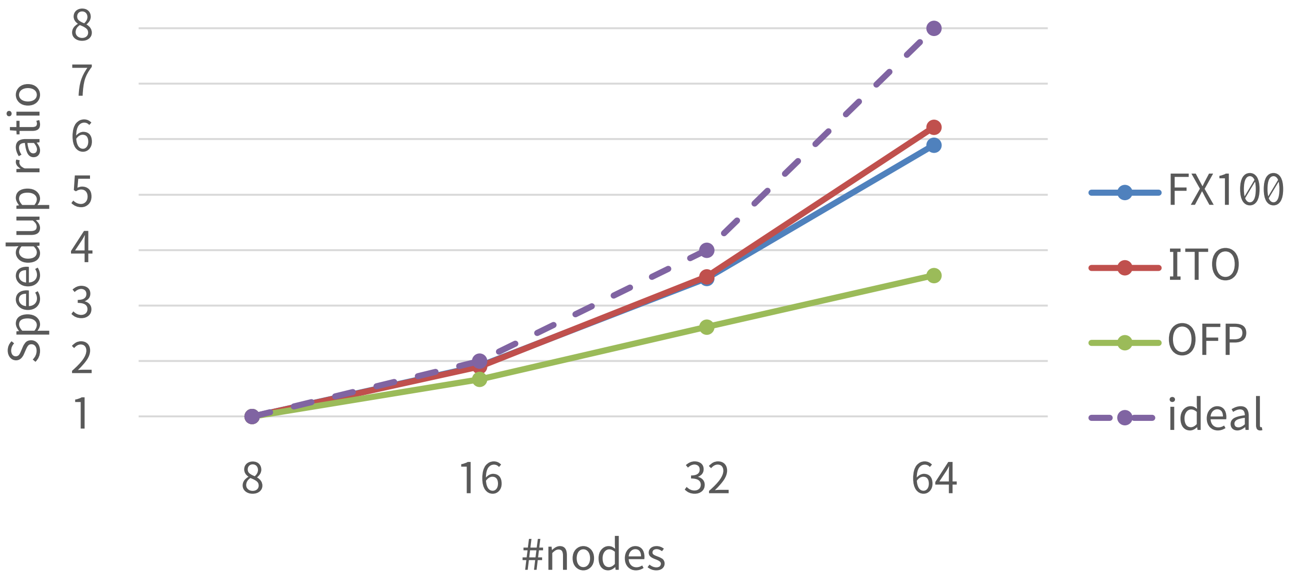
192 cores (HT) * 64 nodes

combination of 2HT, many processes, and original algorithm obtained the best performance in the case of 16, 32, 64 nodes

The fastest cases

	8 nodes	16 nodes	32 nodes	64 nodes
FX100	1.84 msec 2P16T, code3	0.96 msec 2P16T, code3	0.52 msec 4P8T, code3	0.31 msec 4P8T, code3
ITO	2.00 msec 16P1T, original	1.05 msec 1P16T, code3	0.57 msec 2P8T, code3	0.32 msec 2P8T, code3
OFP	1.31 msec 32P8T, code1	0.78 msec 32P4T, original	0.50 msec 16P8T, original	0.37 msec 8P16T, original





Performance trends and Auto-Tuning

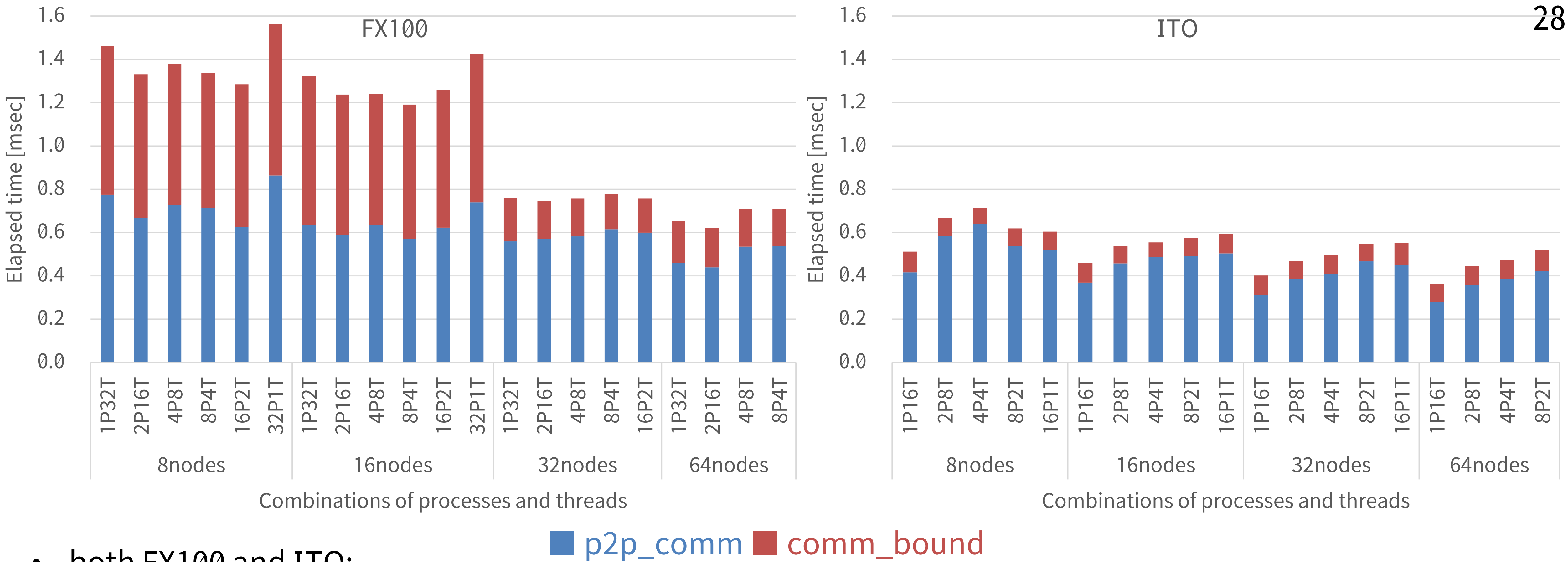
- FX100 and ITO have similar times and trends, but OFP is different.
  - FX100 & ITO: many processes/node don't improve performance, code3 is good
  - OFP: many processes/node improves performance, original is good
    - maybe because of the cost of thread management and synchronization of OpenMP
- OFP is the fastest of three target HWs at 8, 16, and 32 nodes, but the slowest at 64 nodes. Scalability is bad.
  - concrete reason is not clear

- It is confirmed that #processes, #threads, #nodes, and p2p algorithms are the important performance parameters and execution conditions of MODYLAS.
  - This result helps someone who want to use MODYLAS on other environment including post-K.
  - Making the tuning easy is our another future work. ppOpen-AT will be useful to this work.
    - http://ppopenhpc.cc.u-tokyo.ac.jp/ppopenhpc/

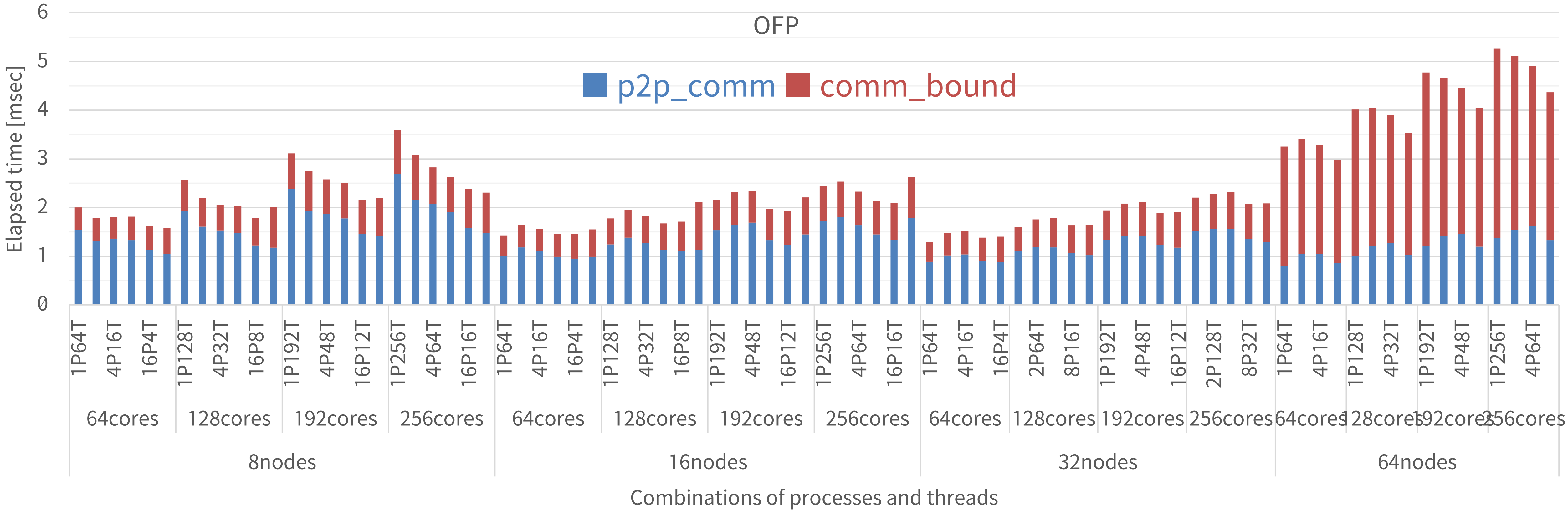
Considering the time of MPI communication

- measurement targets include only p2p calculation, MPI communication is not included
  - Generally, the more MPI processes are used, the more MPI communication time is needed.
  - Especially on OFP…
    - many #processes/node achieved fast p2p performance
      <=> many total #processes causes performance slowdown
    - Is using many #processes good or bad?

- We measured the two point-to-point communication parts of MODYLAS.
  - p2p_comm: communicates the coordinates of the atoms in halo areas
  - comm_bound: exchanges the ownership of the atoms by each processes



- both FX100 and ITO:
  - (there are some exceptions,)
  - the more nodes are used, the shorter communication time
  - the more processes/node are used, the longer communication time
- FX100: 8 nodes and 16 nodes requires very long comm_bound time.
  - maybe because of performance trend of Tofu? (Tofu is optimized for 12*n nodes)
- ITO: achieved faster time than FX100 and OFP at all #nodes
- Many nodes and processes reduce the amount of communication, but times of communication are not very difference. Latency will limit reducing the time of communication.



- OFP needs longer communication times than FX100 and ITO (around 0.4 – 0.6 msec at 64 nodes). Especially, 64 nodes requires very long comm_bound.
- OFP achieved fast p2p calculation time, total time of calculation and communication is very longer than FX100 and ITO.
- Honestly, the reason why OFP needs long communication time is not clear. Additional investigation is required.
- Moreover, in the case of 8 nodes, the more processes are used, the less p2p_comm time is required. Similarly, in the case of 64 nodes, the more processes are used, the less comm_bound time is required. Reducing the number of threads improve the communication time well?

Conclusion

- evaluated the execution times of the p2p part of MODYLAS (one of the dominant part for every input data case) on 8 nodes
  - in addition to the our previous work, combinations of #processes and #threads are expanded and new target hardware are added: ITO (Skylake-SP) and OFP (Knights Landing)
  - FX100 and ITO achieved similar performance and its trends, FX100 is a little bit faster than ITO.
  - OFP achieved the higher performance. When the number of processes is small, performance is very low. (Probably, many number of threads causes slowdown.)
- moreover, measured performance on 16, 32, and 64 nodes (strong scaling)
  - FX100 and ITO achieved similar performance and its trends
  - OFP achieved bad scalability: the fastest on 8, 16, and 32nodes, the slowest on 64 nodes
- considered the communication time
  - ITO achieved the fastest and OFP achieved the slowest communication time
  - FX100 is slower than ITO a little, optimization of Tofu (12*n nodes) should be considered

Future work

- using other input data (large water molecule other kind of molecules)
  - dominant parts are same as this work, performance trends may be changed
- large number of nodes (with large molecules), other computer systems
  - more than 100 nodes
  - Cascade Lake, post-K (Supercomputer Fugaku)
- optimizing other parts or whole MODYLAS application
  - Can we achieve high performance both p2p and communication?

Acknowledgements

- Prof. S. Okazaki at Nagoya University for providing a developing version of the MODYLAS program for this research
- supported by Joint Usage/Research Center for Interdisciplinary Large-Scale Information Infrastructures in Japan (Proposal number: jh150015, jh160040, jh170024, and jh180050)
- supported by the FLAGSHIP 2020, MEXT within the priority study 5: Development of new fundamental technologies for high-efficiency energy creation, conversion/storage, and use (Proposal number: hp170241 and hp180209)
- benchmark calculations: the Information Technology Center of Nagoya University, Research Institute for Information Technology of Kyushu University, and JCAHPC
- supported by JSPS KAKENHI Grant Number JP16K21094, JP17H05290, and JP18K19782
- funded by MEXT's program for the Development and Improvement for the Next Generation Ultra High-Speed Computer System, under its Subsidies for Operating the Specific Advanced Large Research Facilities