



2012 HPC Challenge Class 2 Submission

XcalableMP for Productivity and Performance

Masahiro Nakao[†], Hitoshi Murai[‡]
Takenori Shimosaka[‡], Mitsuhsisa Sato^{†‡}

[†] Center for Computational Sciences, University of Tsukuba, Japan

[‡] RIKEN Advanced Institute for Computational Science, Japan



Outline



1. What's XcalableMP ?
2. Implementation and performance
 1. HPL
 2. RandomAccess
 3. FFT
 4. HIMENO Benchmark (Optional)
3. Conclusion

What is XcalableMP?



- XcalableMP (XMP)
 - Directive-based language extension of Fortran 2008 and C99
 - **XMP/Fortran** language and **XMP/C** language exist (same directives)
 - **Global-view** and **Local-view** programming models
 - A part of the design is based on the experiences of HPF
 - XMP/Fortran is upward compatible with the Fortran 2008
 - **Coarray syntax** is also available in XMP/C
 - **XMP specification working group** consists of members from academia, research labs, and industries
 - Univ. of Tsukuba, RIKEN AICS, Fujitsu, NEC, Hitachi, and so on
 - Omni XMP compiler 0.6 and XMP specification 1.1 are available at <http://www.xcalablemp.org>

Benchmarks



- 3 HPC benchmarks : **HPL, RandomAccess, FFT**
- Optional benchmark : **Himeno benchmark**
 - Evaluates performance of incompressible fluid analysis code
 - Typical stencil application
 - Why do we select the HIMENO benchmark ?
 - To demonstrate parallelization by XMP directives which communicate and synchronize the overlapped regions for stencil applications

Machines



	The K computer	HA-PACS
CPU	SPARC64 VIIIfx 2.0GHz 8Cores, 128GFlops	Xeon E5-2670 2.6GHz x2 8Cores x2, 332.8GFlops
Memory	DDR3 SDRAM 16GB 64GB/s/Socket	DDR3 SDRAM 128GB 51.4GB/s/Socket
Network	Torus fusion six-dimensional mesh/torus network, 5GB/s	Infiniband QDRx2rails Fat-tree network, 4GB/s



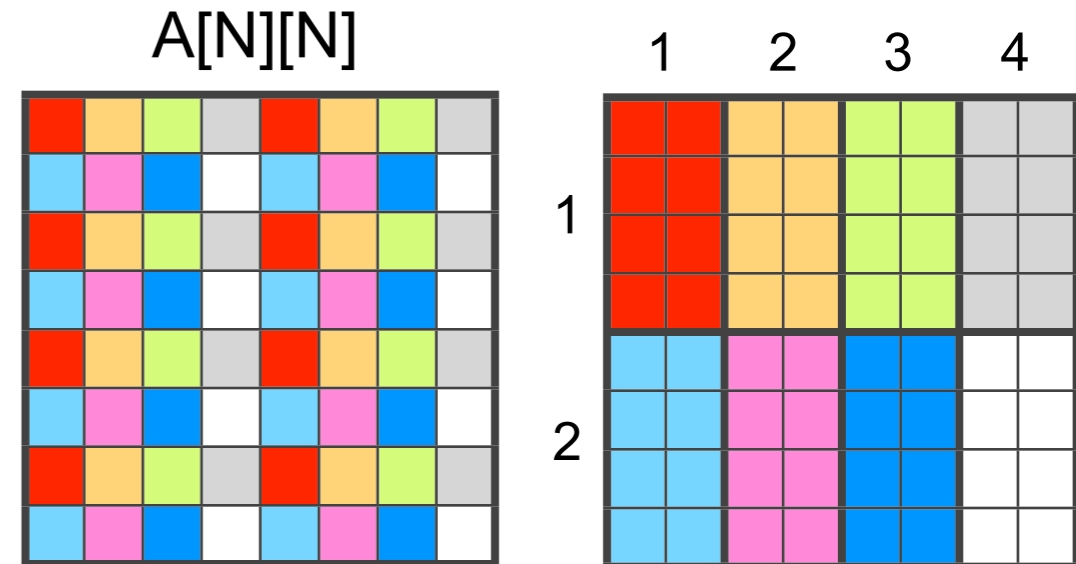
To measure the performance, we used maximum **8192 nodes** of the K computer and maximum **64 nodes** of HA-PACS

HPL(Implementation)



- Source lines of code (SLOC) is **288**, written in XMP/C
- Block-Cyclic Distribution

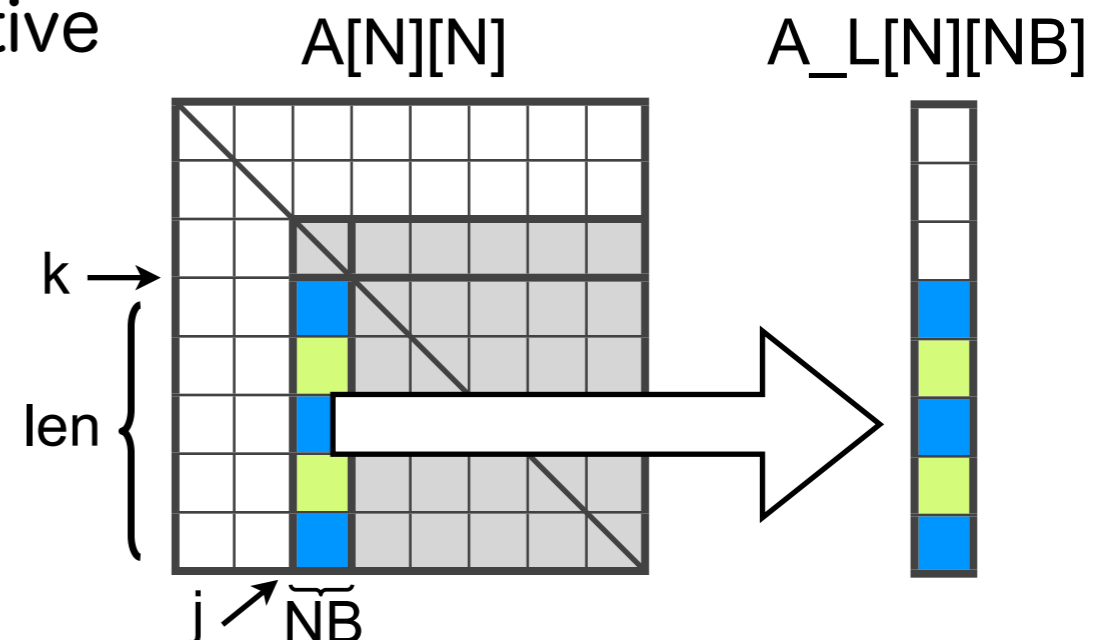
```
double A[N][N];
#pragma xmp nodes p(P,Q)
#pragma xmp template t(0:N-1, 0:N-1)
#pragma xmp distribute t(cyclic(NB), \
                        cyclic(NB)) onto p
#pragma xmp align A[i][j] with t(j,i)
```



Programmer can use BLAS for distributed array.

- Panel Broadcast by using **gmove** directive

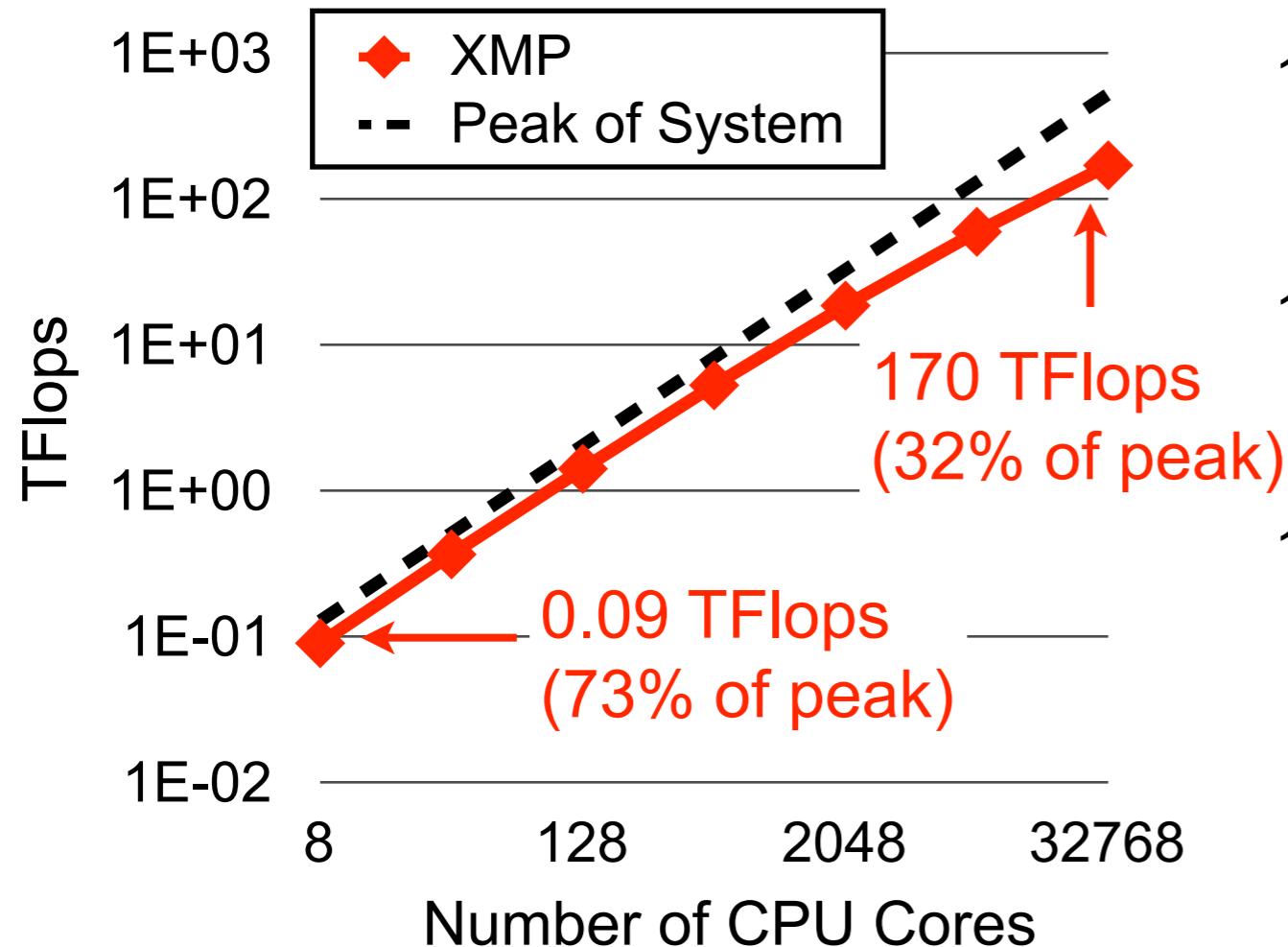
```
double A_L[N][NB];
#pragma xmp align A_L[i][*] with t(*,i)
:
#pragma xmp gmove
A_L[k:len][0:NB] = A[k:len][j:NB];
```



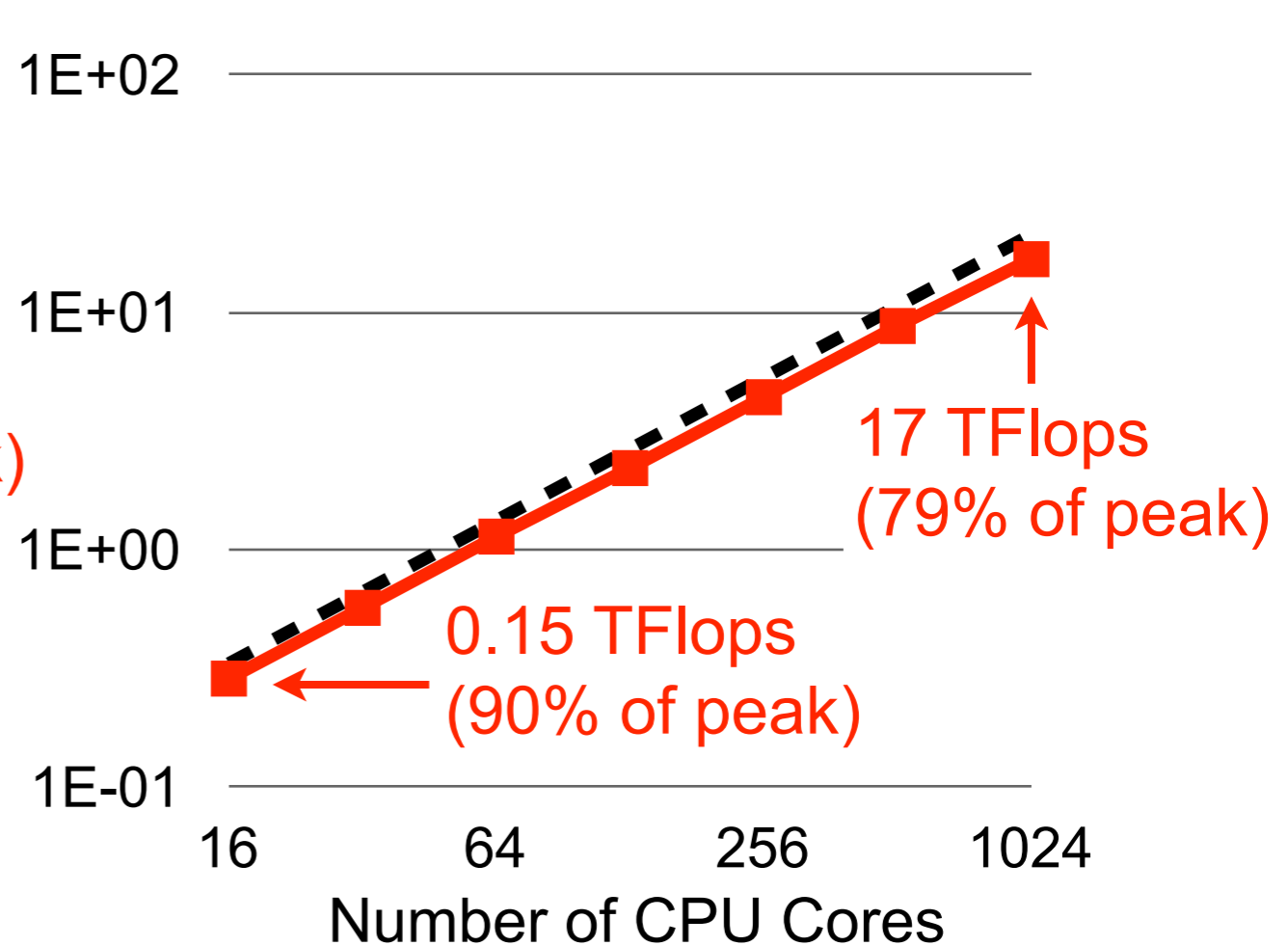
HPL (Result)



- The K computer



- HA-PACS



- On HA-PACS, the performance of the XMP implementation is good
- On the K computer, the performance is a little worse on only one node
 - In this time, we did not have enough time for tuning it ...

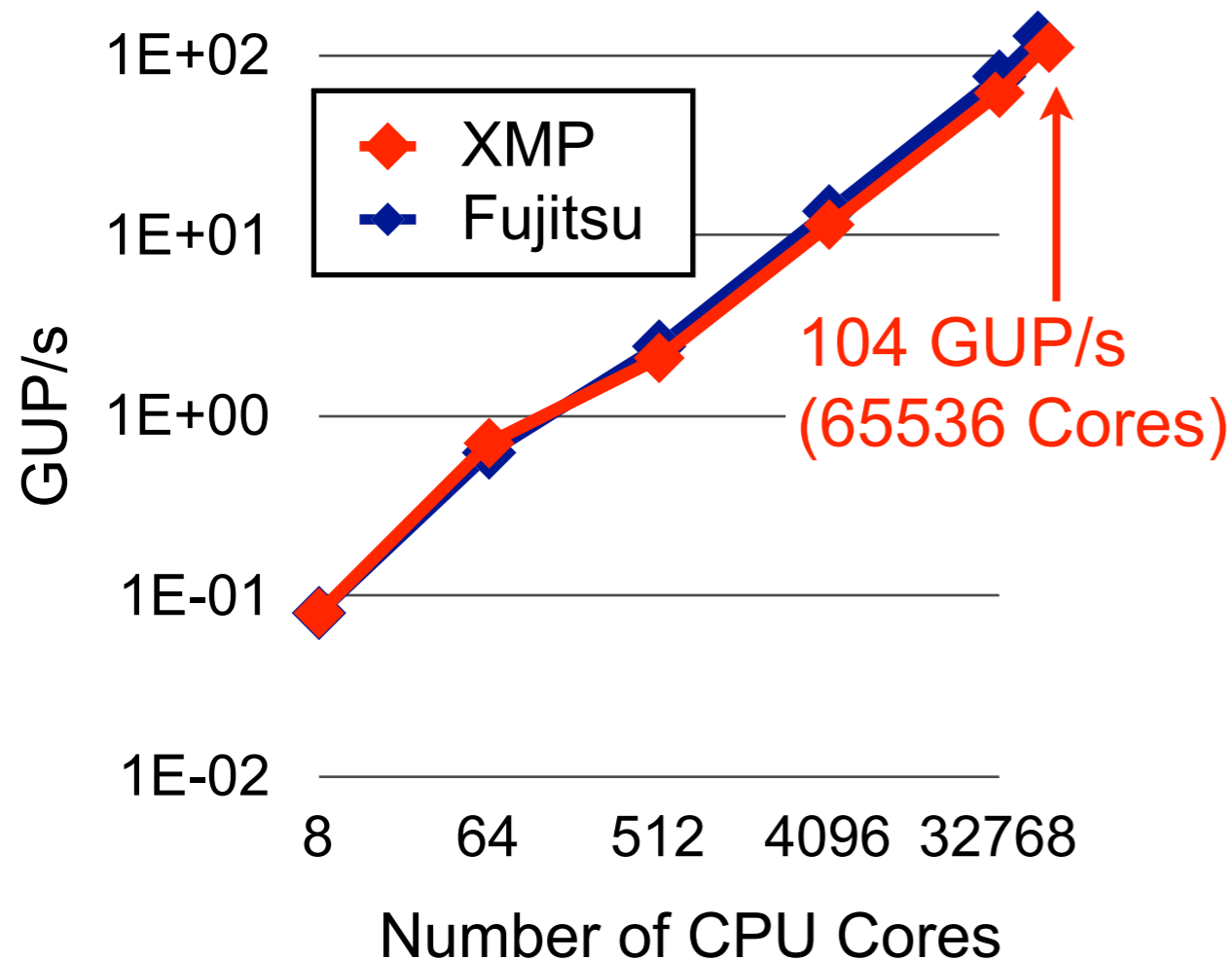
RandomAccess(Implementation)

- We have implemented 2 versions, written in XMP/C
 - For the K computer version, SLOC is **253**
 - For general cluster version, SLOC is **278**
 - Note that there are minor differences (only local operations are changed)
- Local-view programming with XMP/C coarray syntax

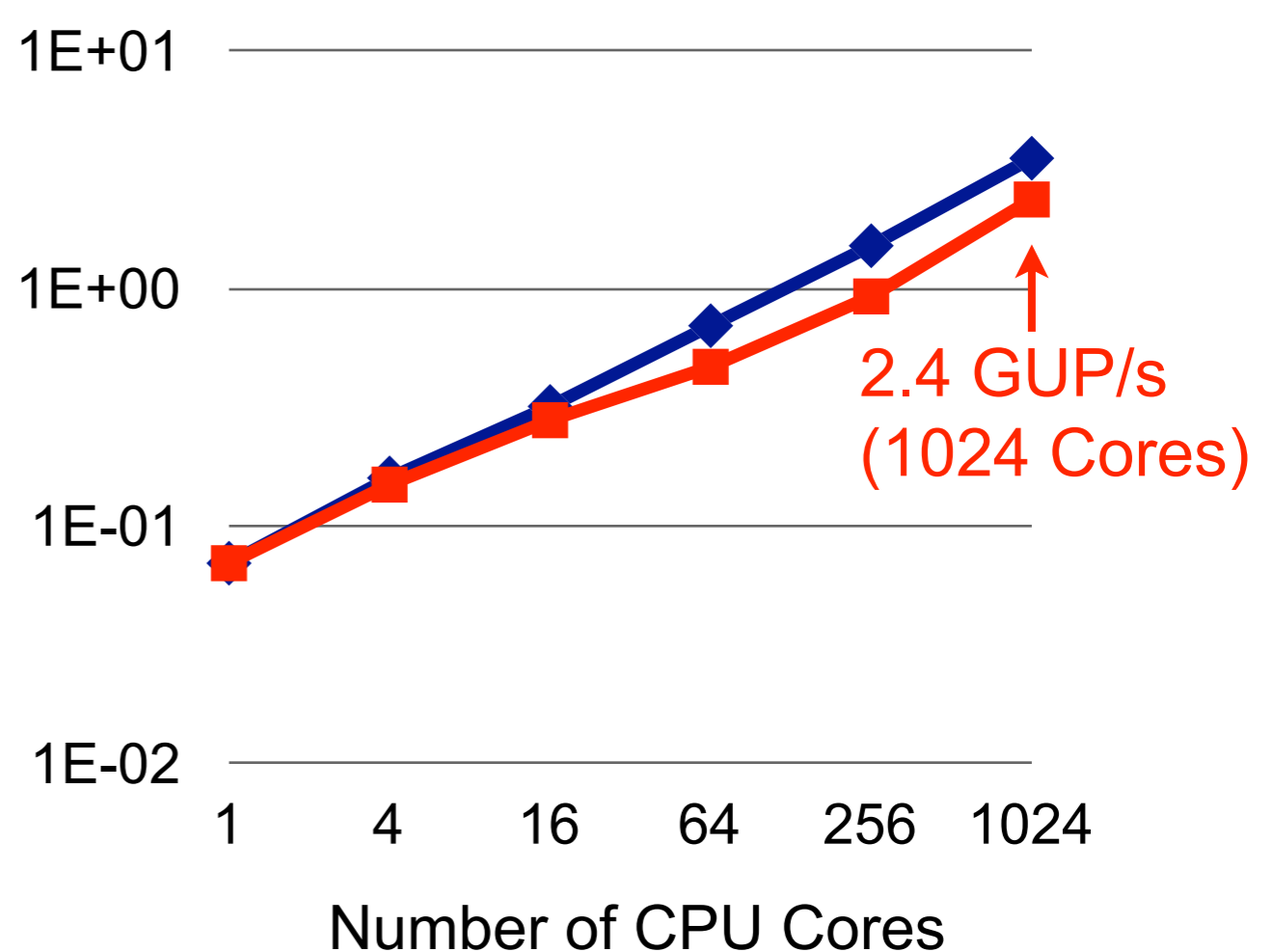
```
recv[j][0:num]:[i_partner] = send[i][0:num]; // Put Operation
#pragma xmp sync memory
#pragma xmp post(p(i_partner), 0)
:
#pragma xmp wait(p(j_partner))
```
- A point-to-point synchronization is specified with the XMP's `post` and `wait` directives to realize asynchronous behavior of this algorithm

RandomAccess(Result)

- The K computer



- HA-PACS



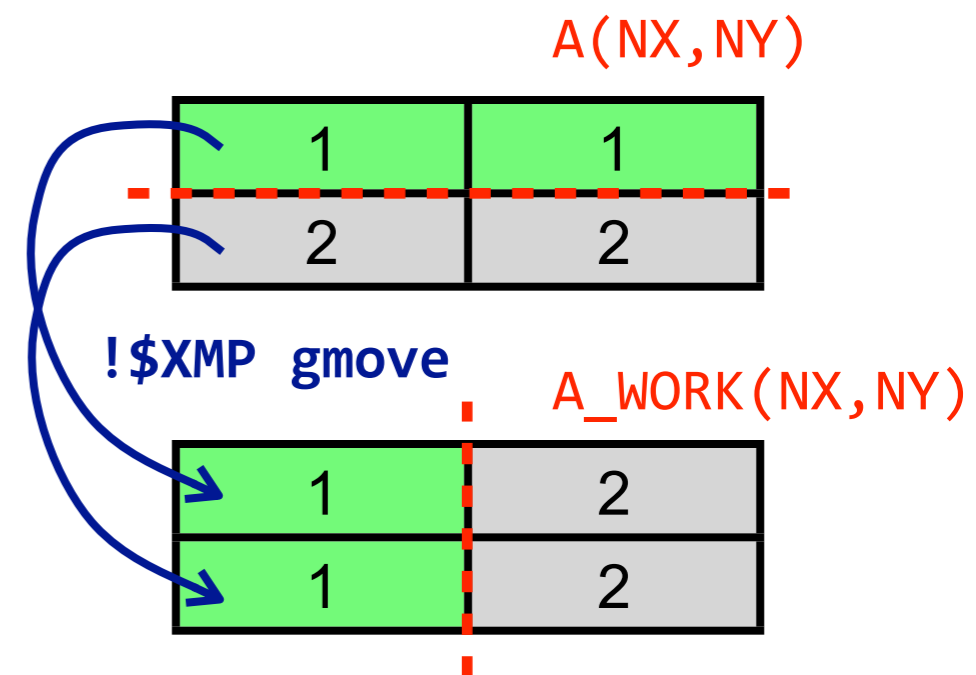
- For comparison, we also used the custom RandomAccess whose some local functions of hpcc-1.4 were optimized by Fujitsu.
- The XMP performance and scalability are good !!

FFT (Implementation)



- SLOC is 112+1522, written in XMP/Fortran + OpenMP + MPI/C
 - We rewrote only “pzfft1d.f” in fite-5.0 which is a main kernel
 - This SLOC is 112 (the SLOC of original “pzfft1d.f” is 186)
 - Our implementation uses C interface and another kernels of the original FFT in hpcc-1.4 (these SLOC is 1522)
- This implementation is a good demonstration how to mix a XMP program with a MPI program (MPI program calls subroutine written in XMP)
- Six-step FFT algorithm

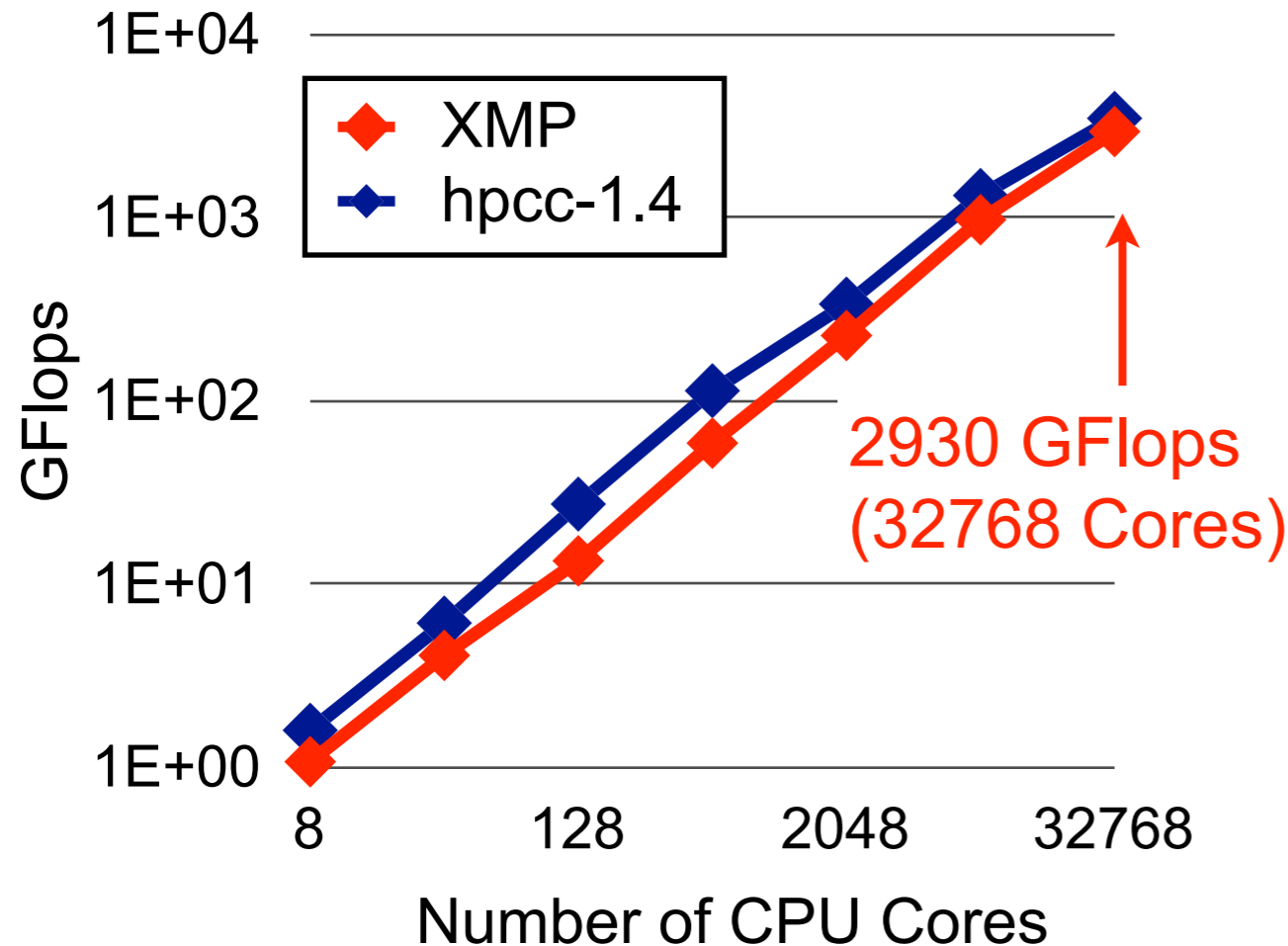
```
!$XMP distribute tx(block) onto p
!$XMP distribute ty(block) onto p
!$XMP align A(*,i) with ty(i)
!$XMP align A_WORK(i,*) with tx(i)
:
!$XMP gmove
A_WORK(1:NX,1:NY) = A(1:NX,1:NY) ! all-to-all
```



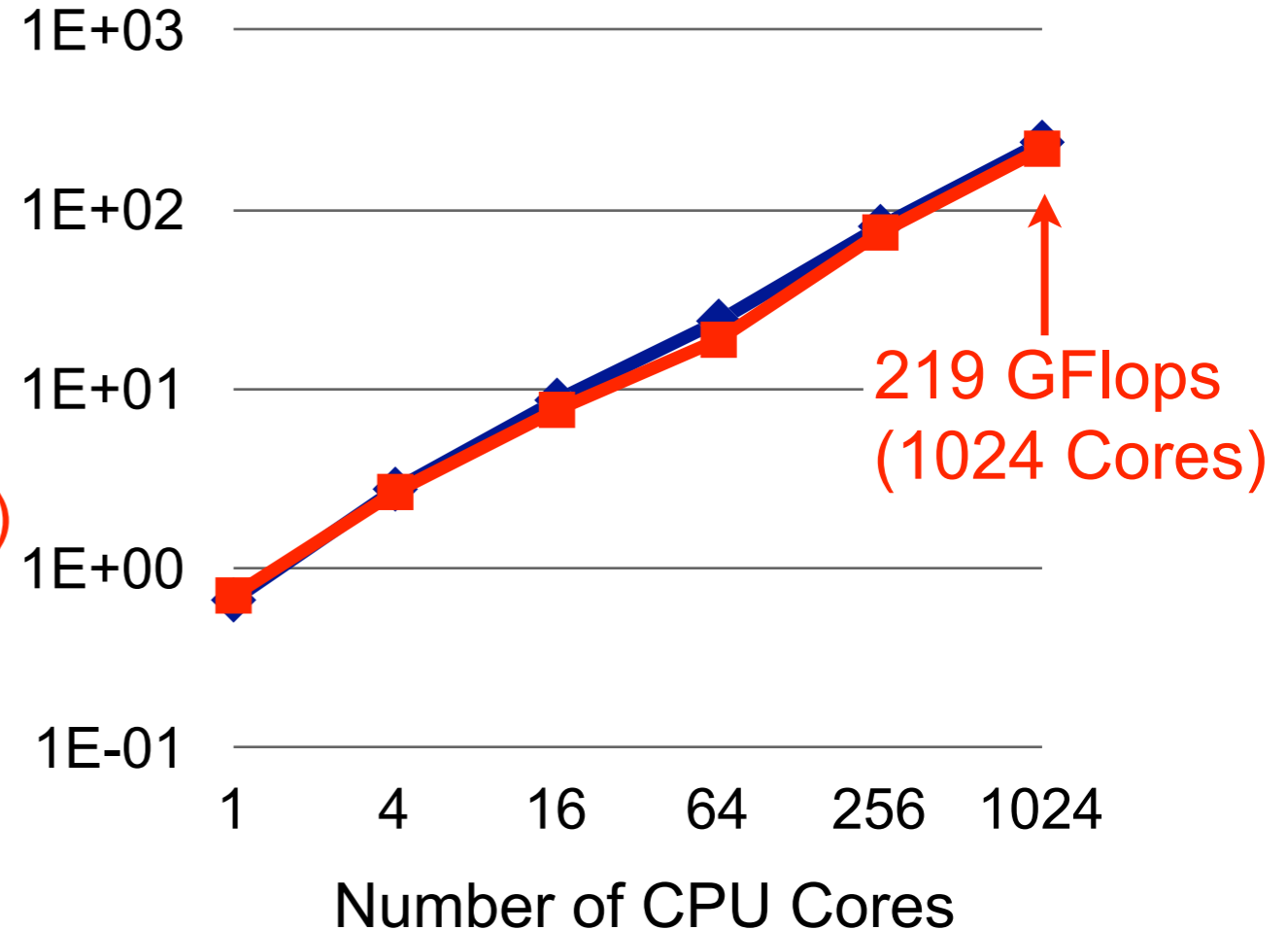
FFT (Result)



- The K computer



- HA-PACS



- The XMP performance and scalability are good !!

HIMENO(Implementation)

- SLOC is **115**, written in **XMP/Fortran**
 - SLOC of the original is **612**, written in **MPI/Fortran**
- To synchronize overlapped region, we use XMP shadow and reflect directives

```
!$xmp shadow p(0,1,1)
:
!$xmp reflect (p)
!$xmp loop (J,K) on t(*,J,K) reduction (+:GOSA)
do K = 2, kmax-1
  do J = 2, jmax-1
    do I = 2, imax-11
      SS = (S0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
      GOSA = GOSA + SS * SS
      :
    enddo
  enddo
enddo
```

← Define overlapped region

← Sync. overlapped region

← Work mapping

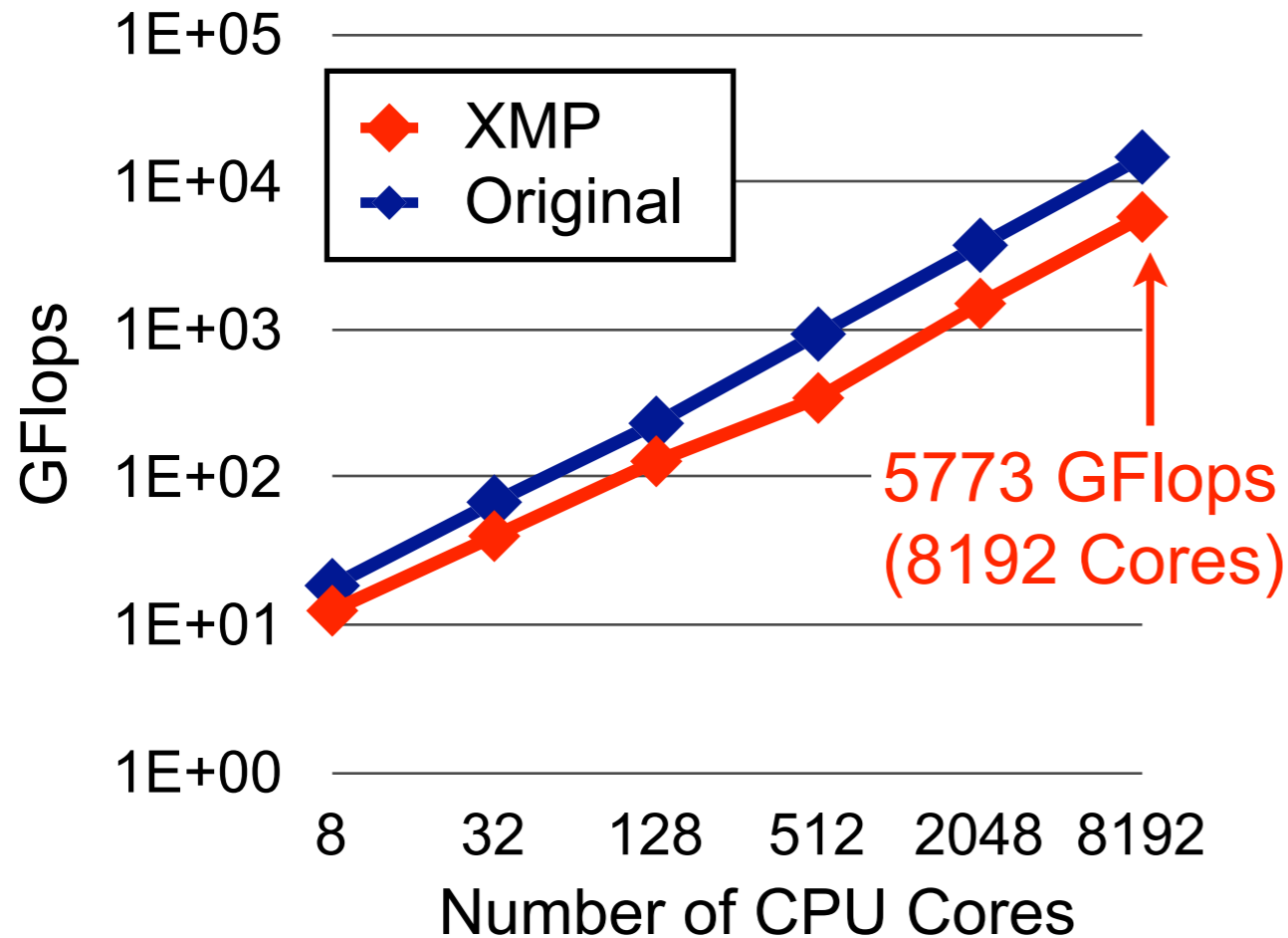
Note:

Only add XMP directives into the sequential Himeno benchmark basically.

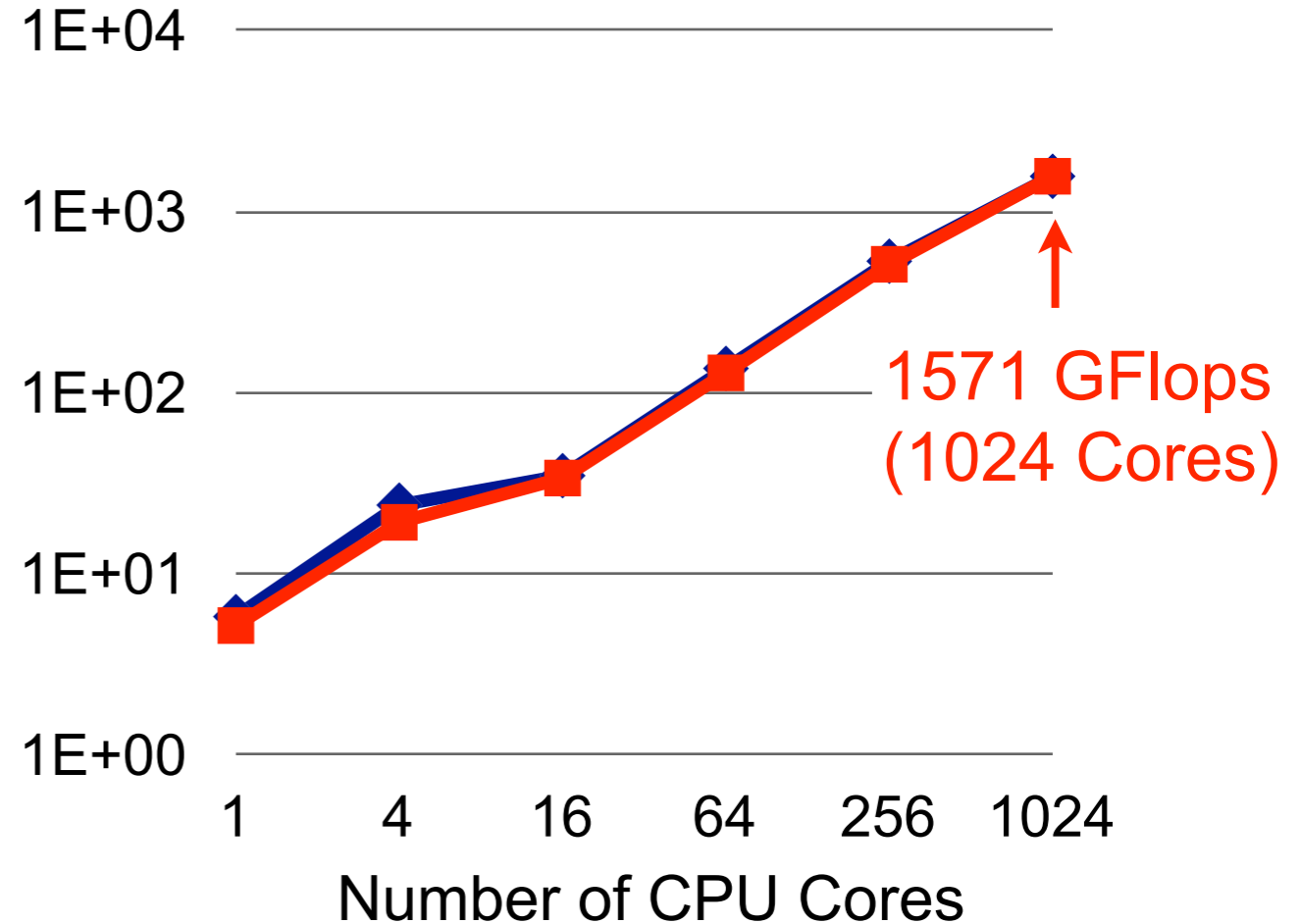
HIMENO(Result)



- The K computer

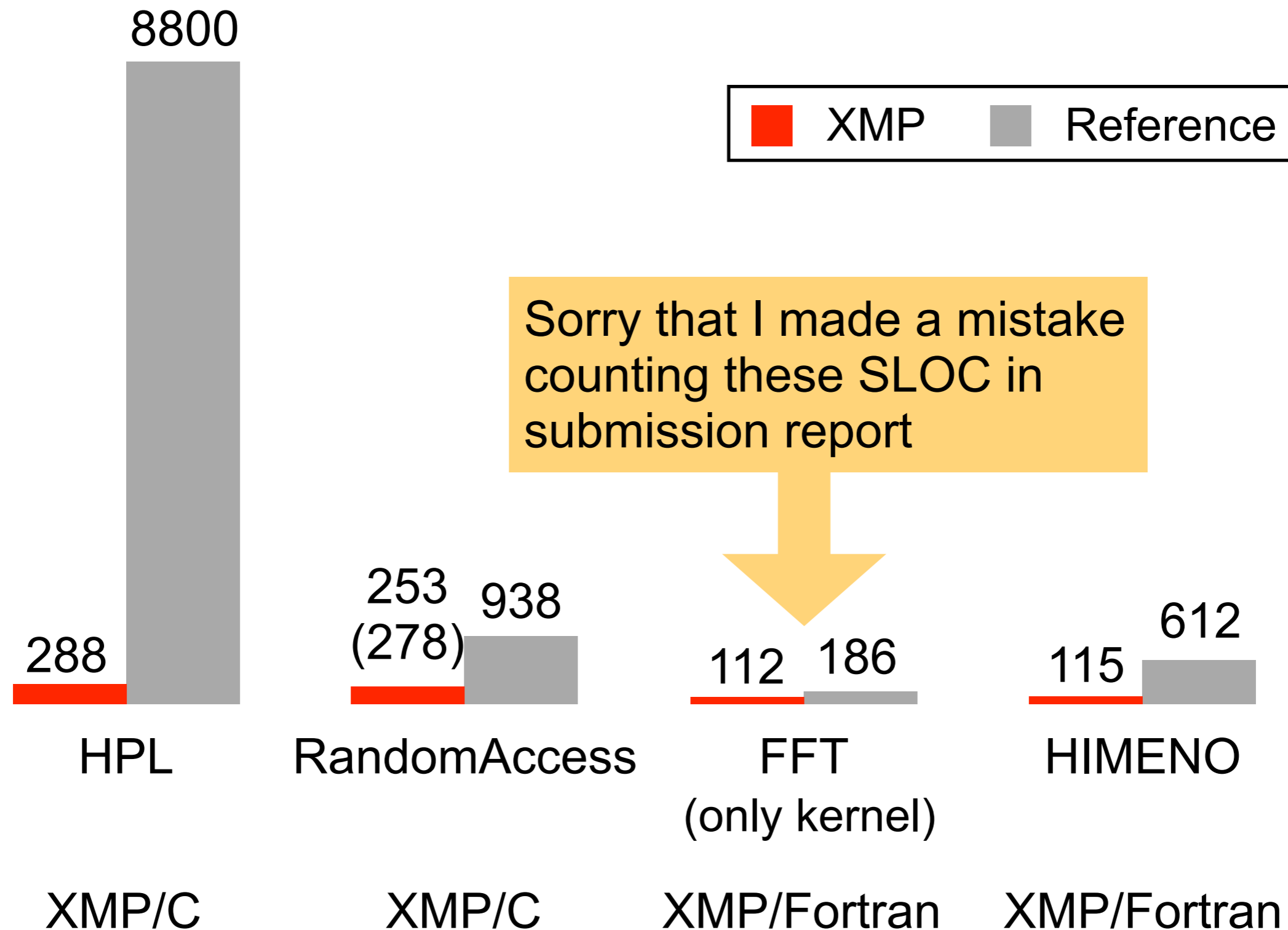


- HA-PACS



- On the HA-PACS, the XMP performance and scalability are good !!
- On the K computer, we are tuning it now ...

Summary (SLOC)




Summary(Performance)

- The K computer

Benchmark	MAX Cores	Performance
HPL	32,768	170 TFlops
RandomAccess	65,536	104 GUP/s
FFT	32,768	2930 GFlops
HIMENO	8,192	5773 GFlops

These results are better than submission



- HA-PACS

Benchmark	MAX Cores	Performance
HPL	1,024	17 TFlops
RandomAccess	1,024	2.4 GUP/s
FFT	1,024	219 GFlops
HIMENO	1,024	1517 GFlops

- Best performance and SLOC

Benchmark	MAX Cores	Performance	SLOC
HPL	32,768	170 TFlops	288
RandomAccess	65,536	104 GUP/s	258(278)
FFT	32,768	2930 GFlops	112 + 1522
HIMENO	8,192	5773 GFlops	115

Thank you for your attention !!

For more information, please visit

#3613 Center for Computational Sciences, University of Tsukuba

#2247 RIKEN AICS (Advanced Institute for Computational Science)