## Application Performances on Many-core Processors

# Xeon Phi versus Kepler GPU


*Takayuki Aoki*

*Tokyo Institute of Technology*


Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology 

# TOP 500 Ranking

| Rank | Site | System | Cores | Rmax (TFlop/s) | Rpeak (TFlop/s) | Power (kW) |
|---|---|---|---|---|---|---|
| 1 | National University of Defense Technology, China | Tianhe-2 (MilkyWay-2) - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P, NUDT | 3120000 | 33862.7 | 54902.4 | 17808 |
| 2 | DOE/SC/Oak Ridge National Laboratory, United States | Titan - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x, Cray Inc. | 560640 | 17590.0 | 27112.5 | 8209 |
| 3 | DOE/NNSA/LLNL, United States | Sequoia - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom, IBM | 1572864 | 17173.2 | 20132.7 | 7890 |
| 4 | RIKEN Advanced Institute for Computational Science (AICS), Japan | K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect, Fujitsu | 705024 | 10510.0 | 11280.4 | 12660 |
| 5 | DOE/SC/Argonne National Laboratory, United States | Mira - BlueGene/Q, Power BQC 16C 1.60GHz, Custom, IBM | 786432 | 8586.6 | 10066.3 | 3945 |
| 6 | Texas Advanced Computing Center/Univ. of Texas, United States | Stampede - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi SE10P, Dell | 462462 | 5168.1 | 8520.1 | 4510 |
| 7 | Forschungszentrum Juelich (FZJ), Germany | JUQUEEN - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect, IBM | 458752 | 5008.9 | 5872.0 | 2301 |
| 8 | DOE/NNSA/LLNL, United States | Vulcan - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect, IBM | 393216 | 4293.3 | 5033.2 | 1972 |
| 9 | Leibniz Rechenzentrum, Germany | SuperMUC - iDataPlex DX360M4, Xeon E5-2680 8C 2.70GHz, Infiniband FDR, IBM | 147456 | 2897.0 | 3185.1 | 3423 |
| 10 | National Supercomputing Center in Tianjin, China | Tianhe-1A - NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050, NUDT | 186368 | 2566.0 | 4701.0 | 4040 |

MIC
Xeon Phi 31S1P
GPU K20X

MIC

GPU

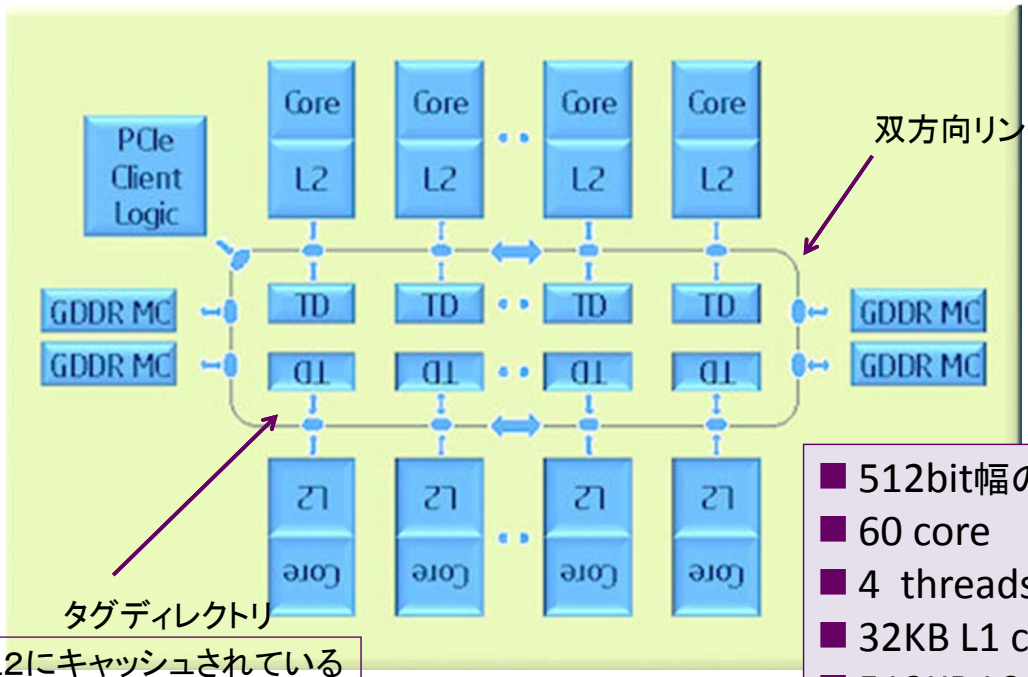
Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

# Processor SPECIFICATIONS

| | Xeon E5-2670 | Xeon Phi 5110P | Tesla M2050 | Tesla K20c | Tesla K20X | GeForce GTX Titan |
|---|---|---|---|---|---|---|
| Performance (SP) | 333 GFlops | 2022 GFlops | 1030 GFlops | 3520 GFlops | 3.95 GFlops | 4500 GFlops |
| Memory Bandwidth | 51.2 GB/s | 320 GB/s (ECC off) | 148 GB/s (ECC off) | 208 GB/s (ECC off) | 250 GB/s (ECC off) | 288 GB/s (nonECC) |
| Memory Size | --- | 8 GB | 3 GB | 5 GB | 6 GB | 6 GB |
| number of cores | 8 | 60/61 | 448 | 2496 | 2688 | 2688 |
| Clock Speed | 2.6 GHz | 1.053 GHz | 1.15 GHz | 0.706 GHz | 0.732 GHz | 0.837 GHz |

Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology


# Architecture of Xeon Phi

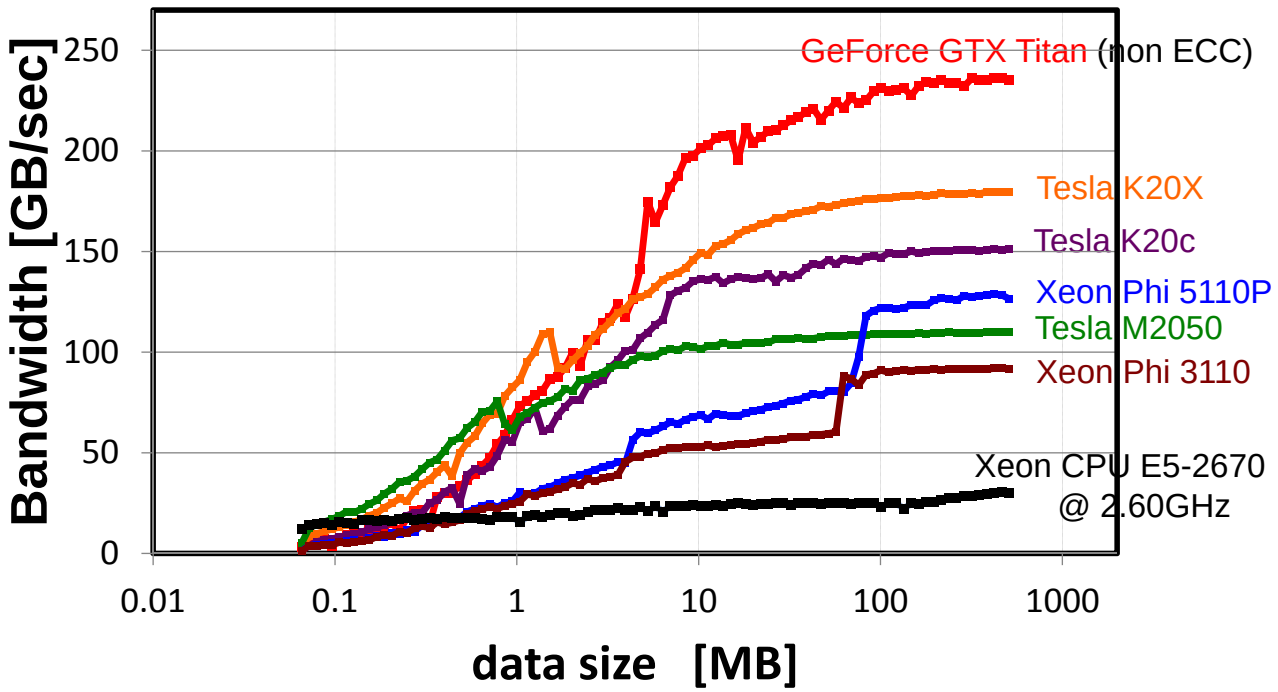



5110P

- 512bit幅のSIMD命令
- 60 core
- 4 threads/core
- 32KB L1 cache/core
- 512KB L2 cache/core

Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

# Memory Bandwidth



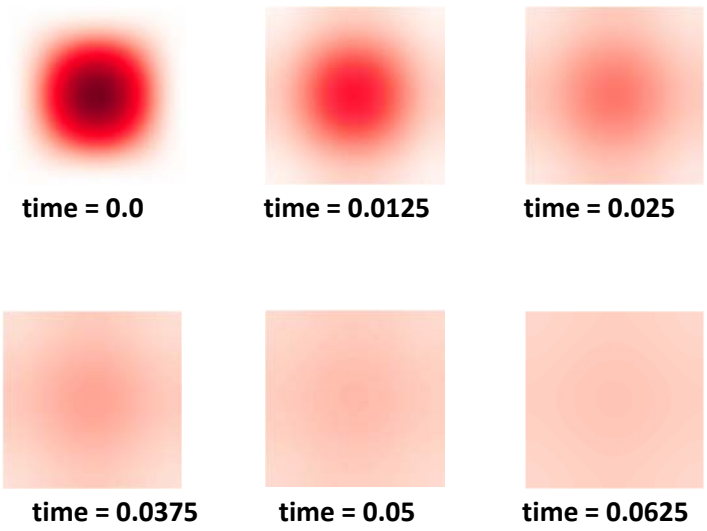

Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

# Performance of Diffusion Eq.

**A primitive stencil application accessing neighbor 7 points using 2nd-order Finite Difference Method**

$$\frac{\partial f}{\partial t} = \kappa \nabla^2 f$$

$$= \kappa \left( \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2} \right)$$

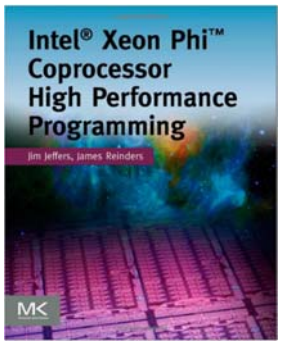



Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

# Discretization

$$\frac{f_{i,j,k}^{n+1}-f_{i,j,k}^{n}}{\Delta t}=\kappa\left(\frac{f_{i+1,j,k}^{n}-2f_{i,j,k}^{n}+f_{i-1,j,k}^{n}}{\Delta x^2}+\frac{f_{i,j+1,k}^{n}-2f_{i,j,k}^{n}+f_{i,j-1,k}^{n}}{\Delta y^2}+\frac{f_{i,j,k+1}^{n}-2f_{i,j,k}^{n}+f_{i,j,k-1}^{n}}{\Delta z^2}\right)$$

$$=c_c f_{i,j,k}^{n}+c_e f_{i+1,j,k}^{n}+c_w f_{i-1,j,k}^{n}+c_s f_{i,j+1,k}^{n}+c_n f_{i,j-1,k}^{n}+c_t f_{i,j,k+1}^{n}+c_b f_{i,j,k-1}^{n}$$

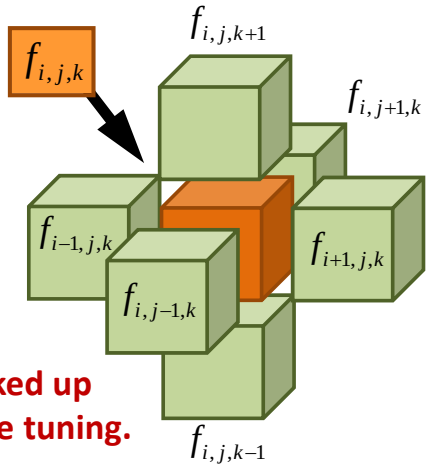



$$c_c=1-\kappa\Delta t\left(\frac{2}{\Delta x^2}+\frac{2}{\Delta y^2}+\frac{2}{\Delta z^2}\right)$$

$$c_e=c_w=\frac{\kappa\Delta t}{\Delta x^2}$$

$$c_s=c_n=\frac{\kappa\Delta t}{\Delta y^2}$$

$$c_t=c_b=\frac{\kappa\Delta t}{\Delta z^2}$$




**Intel Xeon Phi Coprocessor**
**High Performance Programming**

**Our code is picked up for performance tuning.**

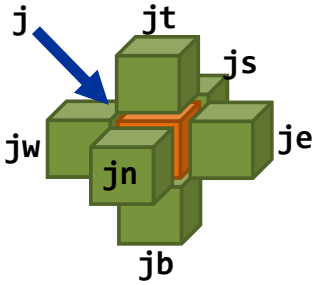
**Authors: James Jeffers, James Reinders (Intel)**

Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

# Source Code

```
float   f[256*256*256],    fn[256*256*256];

for(jz=0; jz < nz; jz++) {
    for(jy=0; jy < ny; jy++) {
        for(jx=0; jx < nx; jx++) {
            j = nx*ny*jz + nx*jy + jx;
            je = j+1;   jw = j-1;  jn = j+nx;   js =j-nx;
            jt = j + nx*ny;  jb = j - nx*ny;

            if (jx == nx-1) je = j;
            if (jx == 0   ) jw = j;
            if (jy == ny-1) jn = j;
            if (jy == 0   ) js = j;
            if (jz == nz-1) jt = j;
            if (jz == 0   ) jb = j;

            fn[j] = cc*f[j]
                  + ce*f[je] + cw*f[jw]
                  + cn*f[jn] + cs*f[js]
                  + ct*f[jt] + cb*f[jb];
        }
    }
}
```

boundary condition




Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

# Xeon/Xeon Phi (1)

## Standard (simply directive)  in the book (Intel)

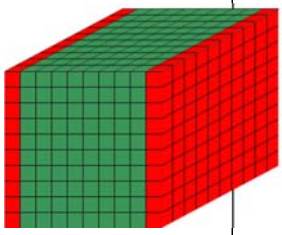
```
#pragma omp parallel  {
#pragma omp for collapse (2)
        for (int jz=0; jz < nz; jz++) {
            for (int jy=0; jy < ny; jy++) {
#pragma simd
                for (int jx=0; jx < nx; jx++){
                        j = nx*ny*jz + nx*jy + jx;
                        je = j+1;   jw = j-1;
                        jn = j+nx;   js =j-nx;
                        jt = j + nx*ny;  jb = j - nx*ny;

                        fn[j] = cc*f[j]
                              + ce*f[je] + cw*f[jw]
                              + cn*f[jn] + cs*f[js]
                              + ct*f[jt] + cb*f[jb];
                }
            }
        }
```

Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

# Xeon/Xeon Phi (2)

## Peeling in the x-direction  in the book (Intel)

```
#pragma omp parallel  {
#pragma omp for collapse (2)
        for (int jz = 0; jz < nz; jz++) {
            for (int jy = 0; jy < ny; jy++) {

                jx = 0;  . . .
#pragma simd
                for (int jx = 1; jx < nx-1; jx++){
                        j = nx*ny*jz + nx*jy + jx;
                        je = j+1;   jw = j-1;
                        jn = j+nx;   js =j-nx;
                        jt = j + nx*ny;  jb = j - nx*ny;

                        fn[j] = cc*f[j]
                              + ce*f[je] + cw*f[jw]
                              + cn*f[jn] + cs*f[js]
                              + ct*f[jt] + cb*f[jb];
                }
                jx = nx - 1;  . . .
            }
        }
```

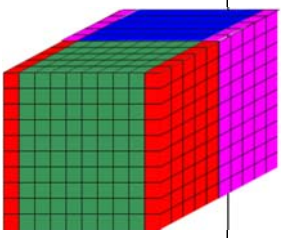


Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

# Xeon/Xeon Phi (3)

## +Tiling in the y-direction in the book (Intel)

```
#pragma omp parallel  {
#pragma omp for collapse (2)
        for(jy_b=0; jy_b < ny; jy_b += YBF){
            for(jz=0; jz < nz; jz++){
                nymax = iy_b + YBF;
                for(jy = jy_b; jy < nymax; iy++){

                jx = 0;  . . .
#pragma simd
                for (int jx = 1; jx < nx-1; jx++){
                        j = nx*ny*jz + nx*jy + jx;

                        fn[j] = cc*f[j]
                              + ce*f[je] + cw*f[jw]
                              + cn*f[jn] + cs*f[js]
                              + ct*f[jt] + cb*f[jb];
                }
                jx = nx - 1;  . . .
            }
        }
```



Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

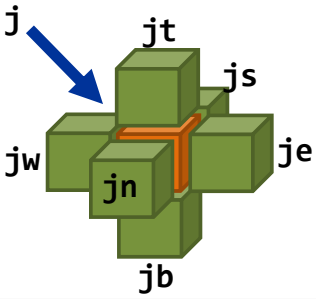
# GPU code

## Standard

```
        int jx = blockDim.x * blockIdx.x + threadIdx.x;
        int jy = blockDim.y * blockIdx.y + threadIdx.y;

        if (jx == nx-1) jxp = jx; if (jx == 0) jxm = jx;
        if (jy == ny-1) jyp = jy; if (jy == 0) jym = jy;
#pragma unroll
        for (int jz=0; jz < nz; jz++) {
                jzm = (jz == 0    ) ? jz : jz-1;
                jzp = (jz == nz-1) ? jz : jz+1;

                j = nx*ny*jz + nx*jy + jx;

                fn[j] = cc*f[j]
                      + ce*f[je] + cw*f[jw]
                      + cn*f[jn] + cs*f[js]
                      + ct*f[jt] + cb*f[jb];
        }
        __syncthreads();
```

boundary condition

boundary condition

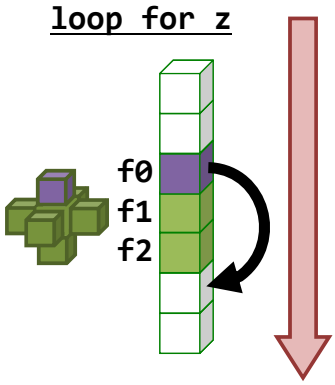



Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

# GPU tuning (1)

## + Register reuse (reuse variable)

```
        int jx = blockDim.x * blockIdx.x + threadIdx.x;
        int jy = blockDim.y * blockIdx.y + threadIdx.y;
        . . .                                   } boundary condition
        jz  = BLOCK_Z * blockIdx.z;
        jzm = (iz == 0) ? 0 : jz - 1;
        f0  = f[jzm*nx*ny + jy*nx + jx];
        f1  = f[jz *nx*ny + jy*nx + jx];
#pragma unroll
        for(int k = 0; k < BLOCK_Z; k++) {
                int jz = BLOCK_Z * blockIdx.z + k;
                jzp = (jz == nz-1) ? jz : jz+1;
                f2  = f[jzp *nx*ny + jy*nx + jx];

                fn[j] = cc*f1
                      + ce*f[je] + cw*f[jw]
                      + cn*f[jn] + cs*f[js]
                      + ct*f2    + cb*f0;

                f0 = f1;  f1 = f2;
        }
```



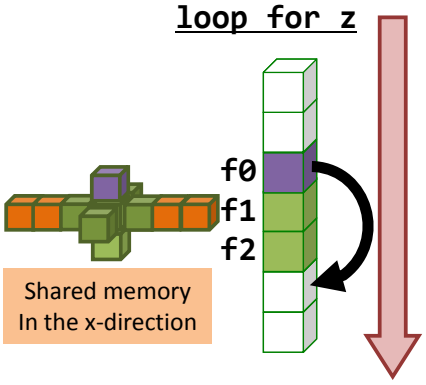

Reuse of the variable

Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

# GPU tuning (2)

## + Shared memory use

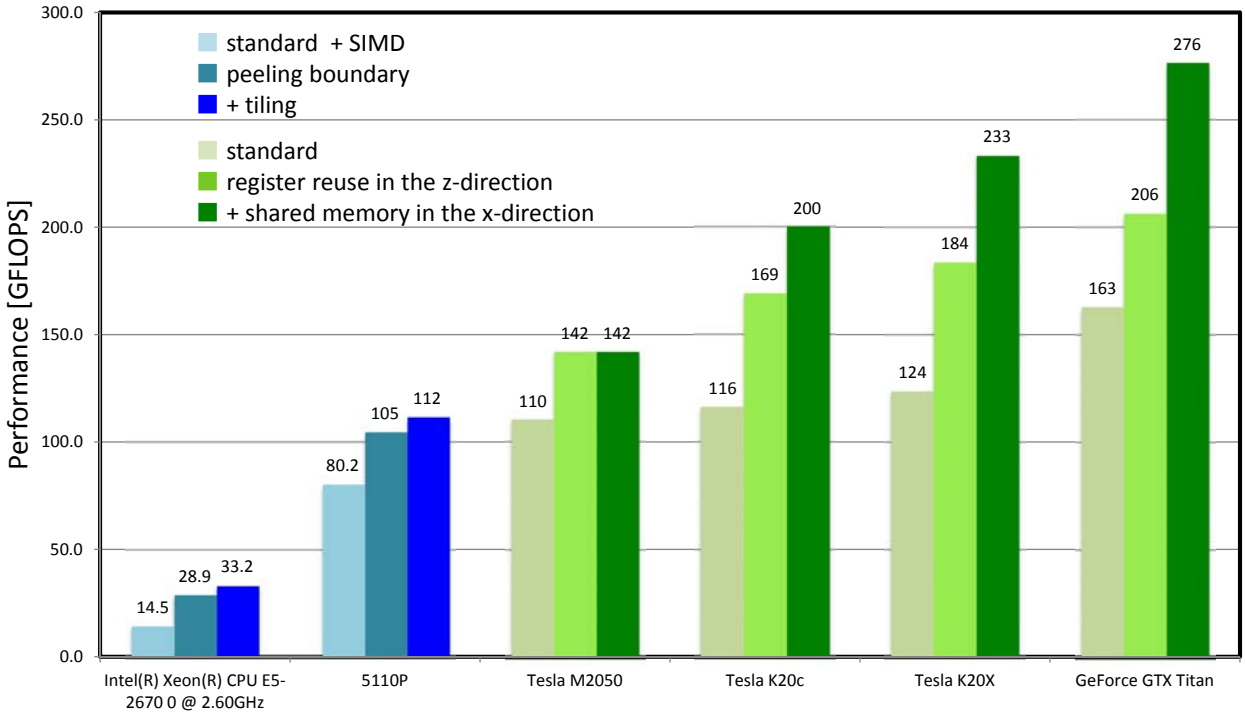
```
   int jx = blockDim.x * blockIdx.x + threadIdx.x;
   int jy = blockDim.y * blockIdx.y + threadIdx.y;

   __shared__ float fs[NX];          // shared memory

   jz  = BLOCK_Z * blockIdx.z;
   jzm = (iz == 0) ? 0 : jz - 1;
   f0  = f[jzm*nx*ny + jy*nx + jx];
   f1  = f[jz *nx*ny + jy*nx + jx];
#pragma unroll
   for(int k = 0; k < BLOCK_Z; k++) {
       int jz = BLOCK_Z * blockIdx.z + k;
       jzp = (jz == nz-1) ? jz : jz+1;
       f2  = f[jzp *nx*ny + jy*nx + jx];
       fs[jx] = f1;   __syncthreads();

       fn[j] = cc*f1
             + ce*fs[jxp] + cw*fs[jxm]
             + cn*f[jn] + cs*f[js]
             + ct*f2    + cb*f0;

       f0 = f1;  f1 = f2;
   }
```




Shared memory
In the x-direction

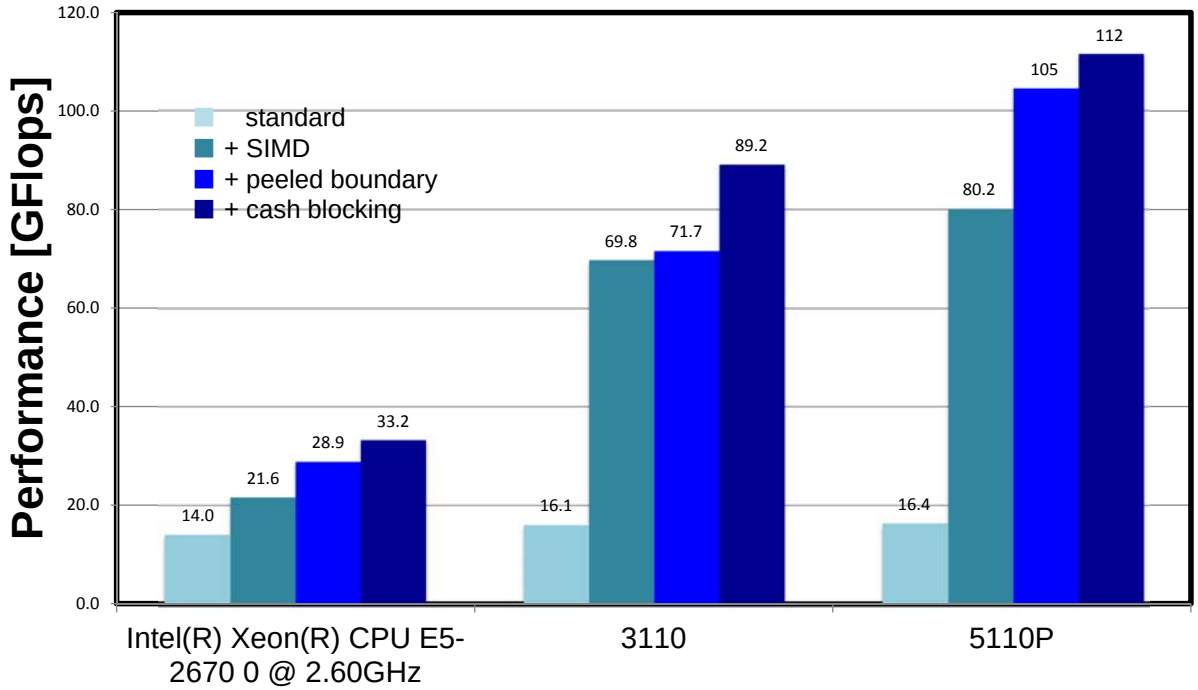
Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

# Performances




Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology 

# MIC Performances




Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

# Compressible Fluid Computation

## 3-dimensional Euler Equation:

$$\frac{\partial \rho}{\partial t}+\frac{\partial E}{\partial x}+\frac{\partial F}{\partial y}+\frac{\partial G}{\partial z}=0$$

$$U=\begin{pmatrix}\rho\\ \rho u\\ \rho v\\ \rho w\\ e\end{pmatrix}\quad E=\begin{pmatrix}\rho u\\ \rho uu+p\\ \rho vu\\ \rho wu\\ (e+p)u\end{pmatrix}\quad F=\begin{pmatrix}\rho v\\ \rho uv\\ \rho vv+p\\ \rho wv\\ (e+p)v\end{pmatrix}\quad G=\begin{pmatrix}\rho w\\ \rho uw\\ \rho vw\\ \rho ww+p\\ (e+p)w\end{pmatrix}$$

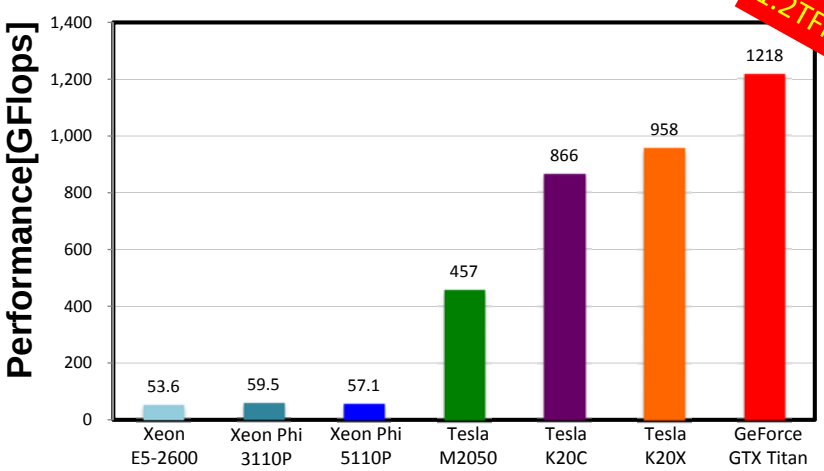
**Conservative form**

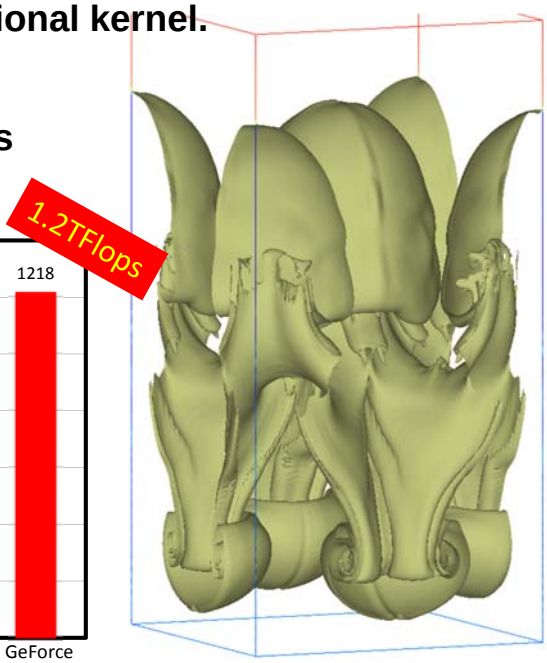
Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

# CPU, MIC, GPU Performances

## Kepler GPU Tuning

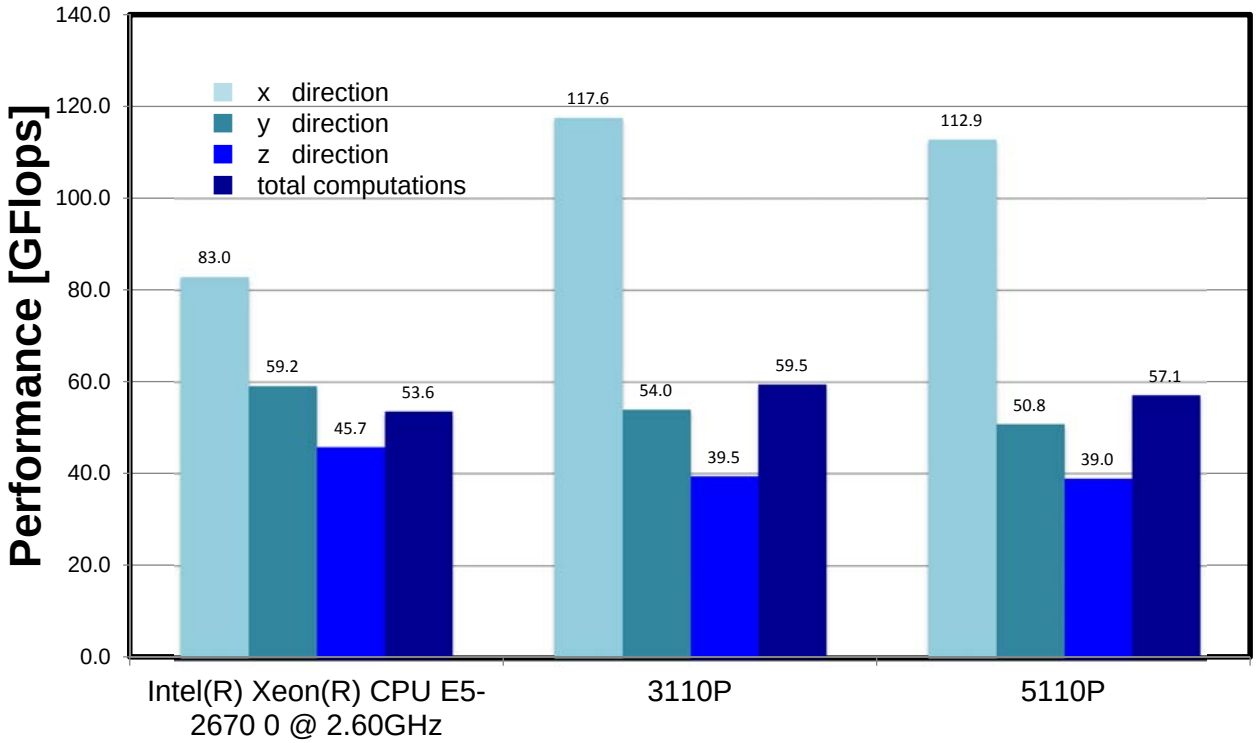
- Shared Memory Use in the x-directional kernel.
- Super function unit
- Loop unrolling
- Variable reuse in the y- and z- loops
- Reduction of branch diverges




Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology

# MIC Performances of
## 3D Compressible Flow Computation




Copyright © Takayuki Aoki , Global Scientific Information and Computing Center, Tokyo Institute of Technology