Parallel Multigrid Solvers using OpenMP/MPI Hybrid Programming Models on Multi-Core/Multi-Socket Clusters

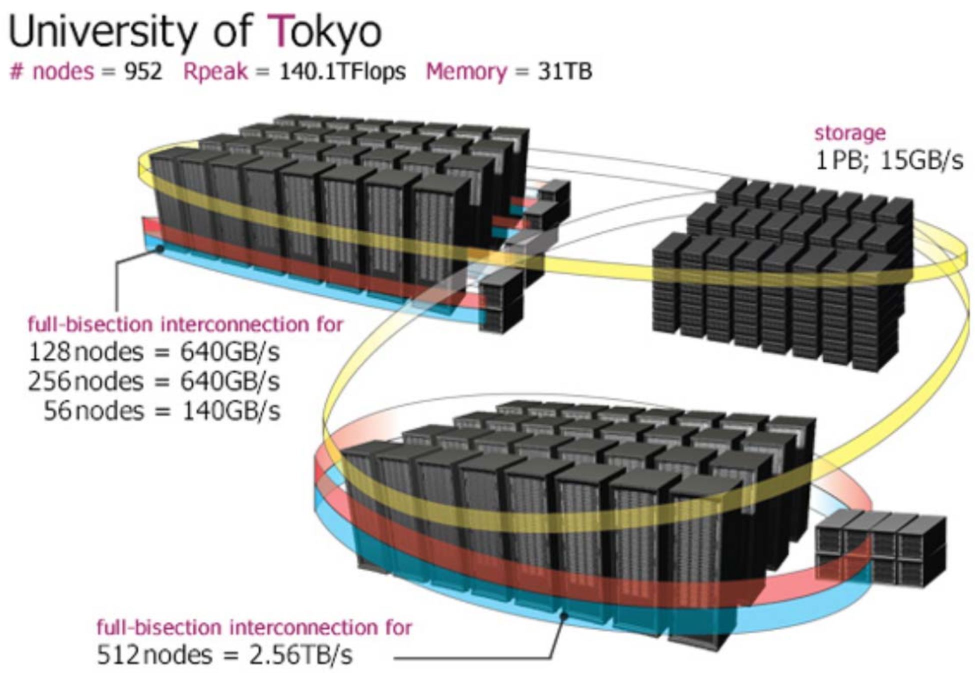
Kengo Nakajima
Information Technology Center, The University of Tokyo
Japan Science & Technology Agency (JST)

9th International Meeting High Performance Computing for Computational Science
VECPAR 2010
June 22-25, 2010, Berkeley, California, USA

T2K/Tokyo (1/2)

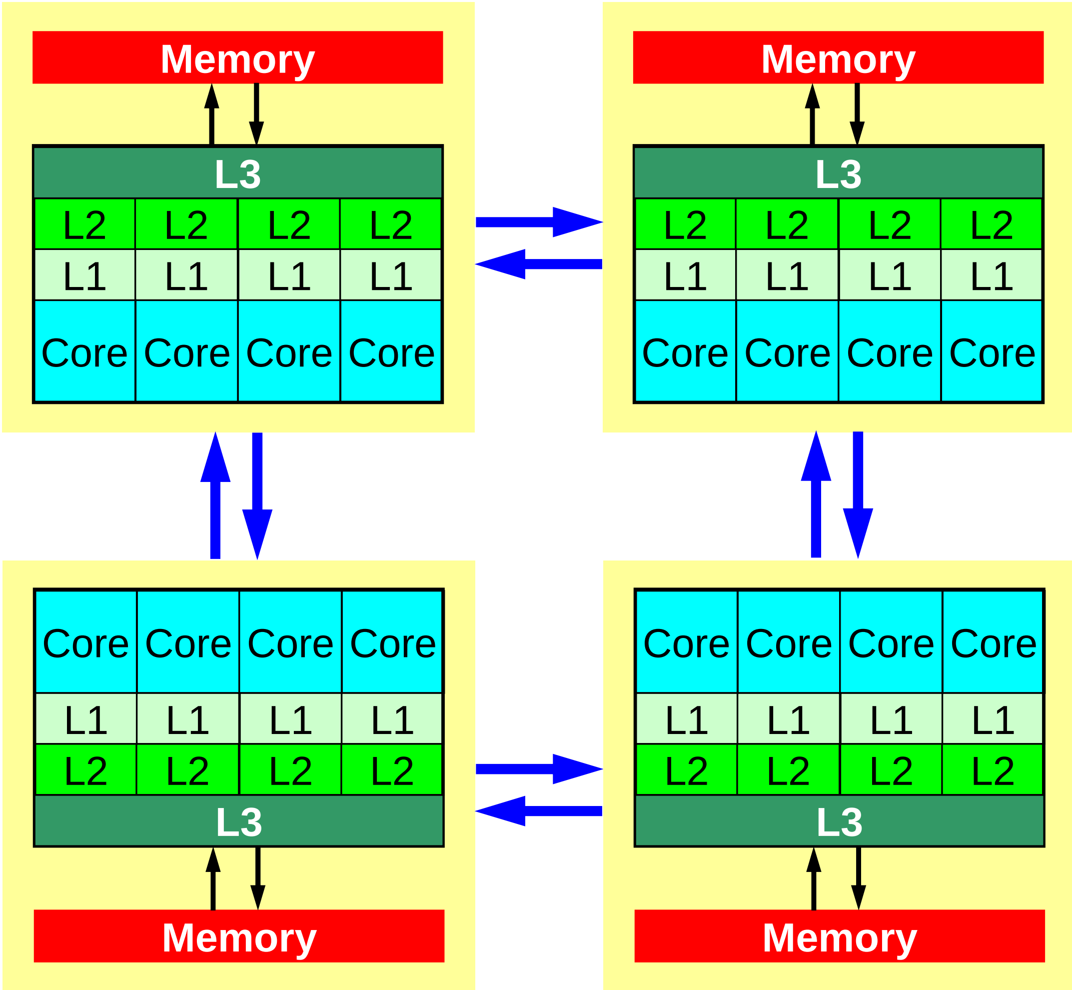
- “T2K Open Supercomputer Alliance”
  - http://www.open-supercomputer.org/
  - Tsukuba, Tokyo, Kyoto

- “T2K Open Supercomputer (Todai Combined Cluster)”
  - by Hitachi
  - op. started June 2008
  - Total 952 nodes (15,232 cores), 141 TFLOPS peak
    - Quad-core Opteron (Barcelona)
  - 53rd in TOP500 (JUN 2010)
  - Fat-Tree with Myrinet-10G



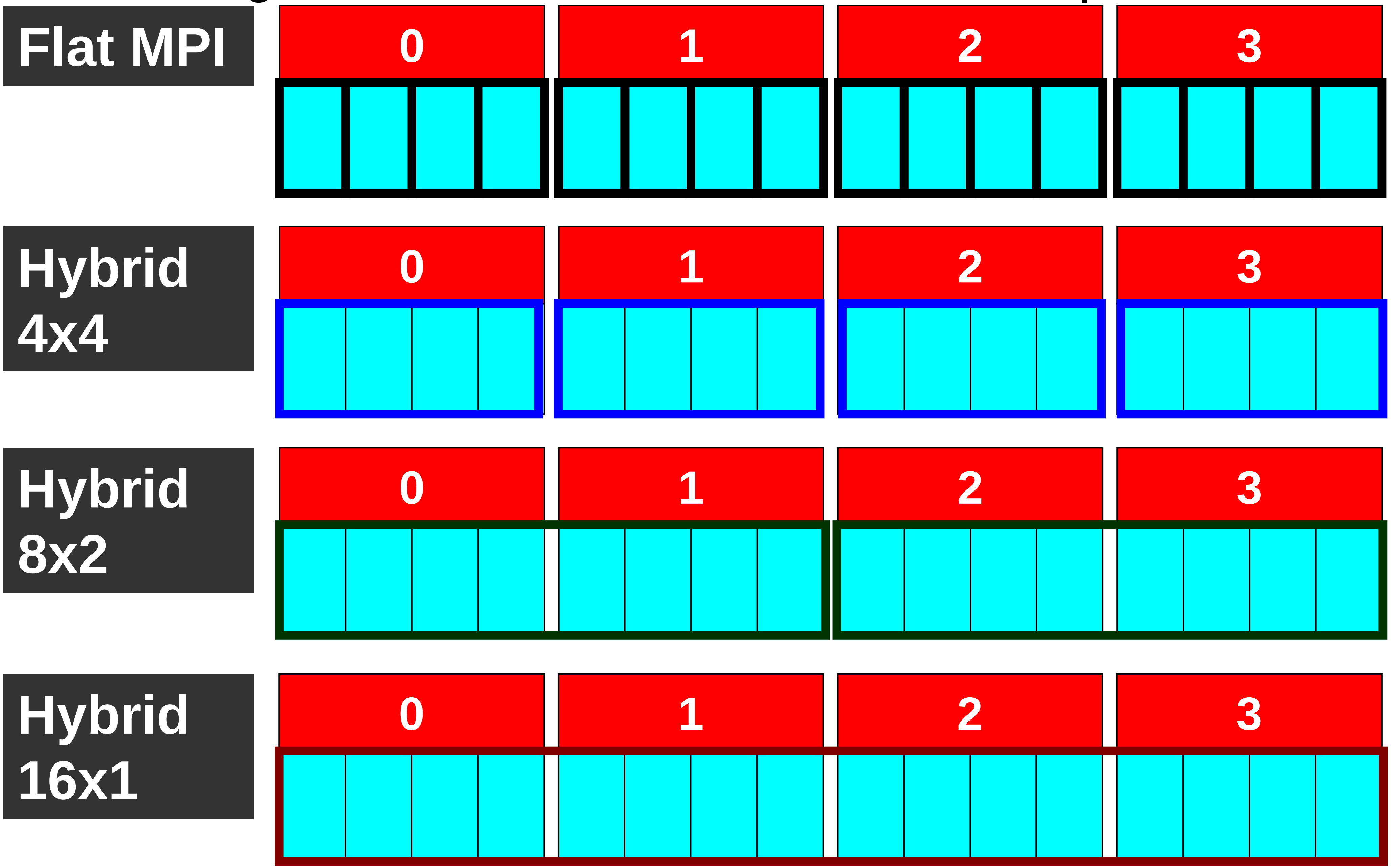
T2K/Tokyo (2/2)

- AMD Quad-core Opteron (Barcelona) 2.3GHz
- 4 “sockets” per node
  - 16 cores/node
- Multi-core, multi-socket system
- cc-NUMA architecture
  - careful configuration needed
    - local data ~ local memory



Flat MPI, Hybrid (4x4, 8x2, 16x1)

Higher Performance of HB16x1 is important

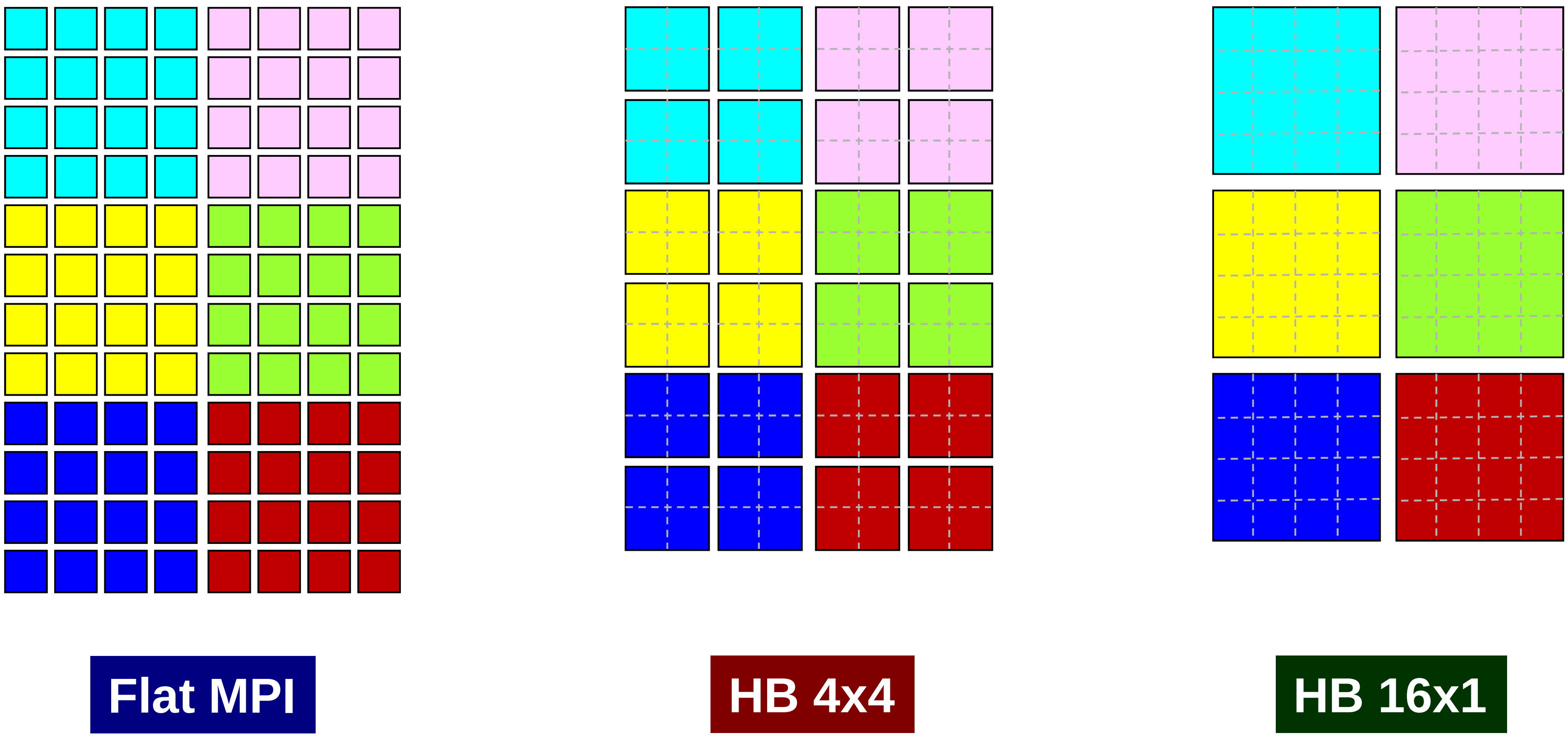


Domain Decomposition

Inter Domain: MPI-Block Jacobi

Intra Domain: OpenMP-Threads (re-ordering)

example: 6 nodes, 24 sockets, 96 cores

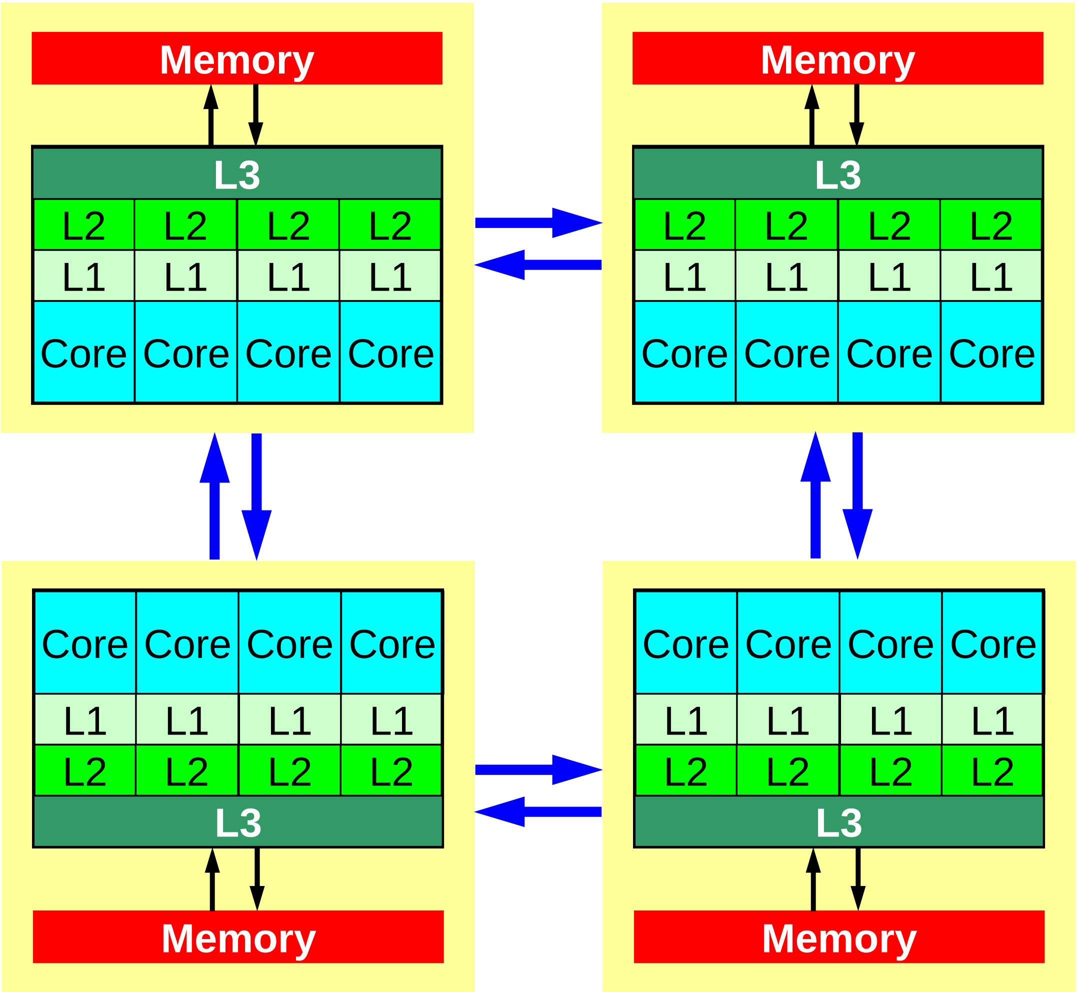


First Touch Data Placement

Local Data – Local Memory

The most common NUMA page-placement algorithm is the "first touch" algorithm, in which the PE first referencing a region of memory will have the page holding that memory assigned to it.

A very common technique in OpenMP program is to initialize data in parallel using the same loop schedule as will be used later in the computations.

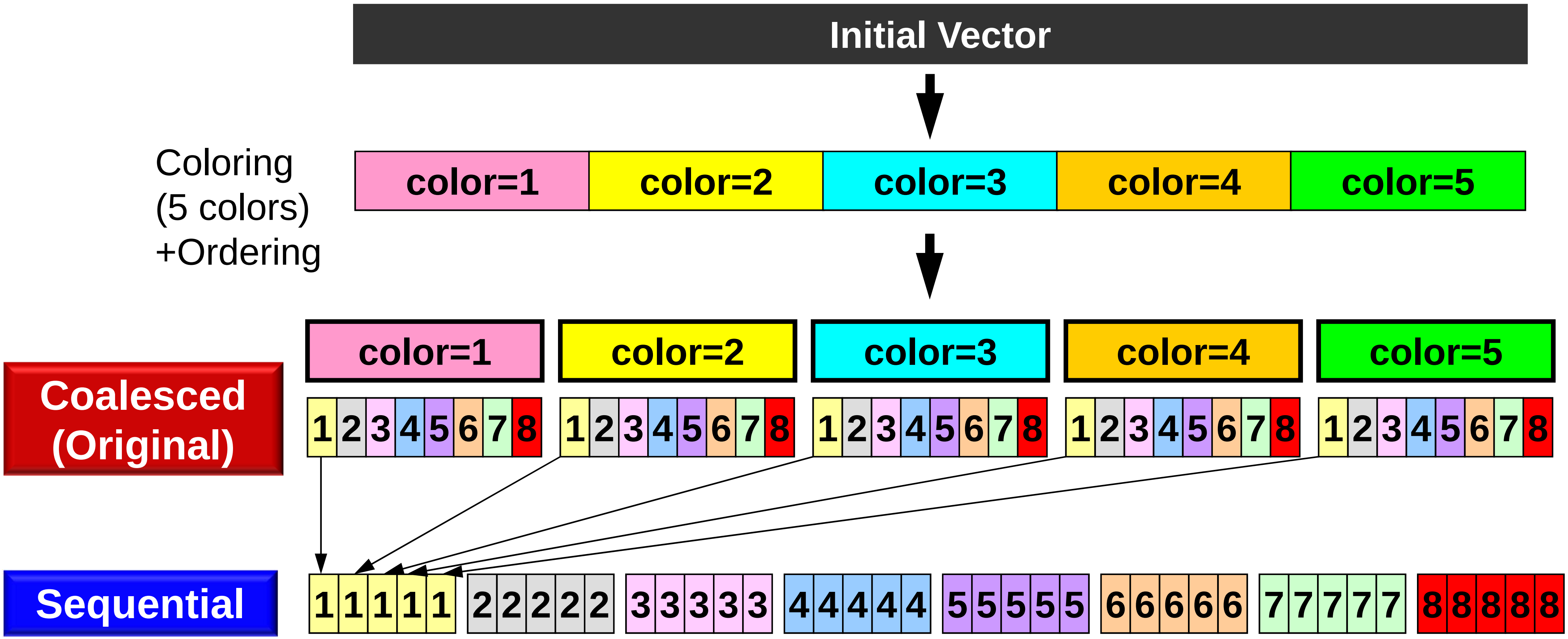


First Touch Data Placement

Method of Initialization
to be initialized as computation

```
  do lev= 1, LEVELtot
    do ic= 1, COLORtot(lev)
!$omp parallel do private(ip,i,j,isL,ieL,isU,ieU)
      do ip= 1, PEsmpTOT
      do i = STACKmc(ip,ic-1,lev)+1, STACKmc(ip,ic,lev)
        RHS(i)= 0.d0; X(i)= 0.d0; D(i)= 0.d0

        isL= indexL(i-1)+1
        ieL= indexL(i)
        do j= isL, ieL
          itemL(j)= 0; AL(j)= 0.d0
        enddo

        isU= indexU(i-1)+1
        ieU= indexU(i)
        do j= isU, ieU
          itemU(j)= 0; AU(j)= 0.d0
        enddo
      enddo
      enddo
!$omp omp end parallel do
    enddo
  enddo
```

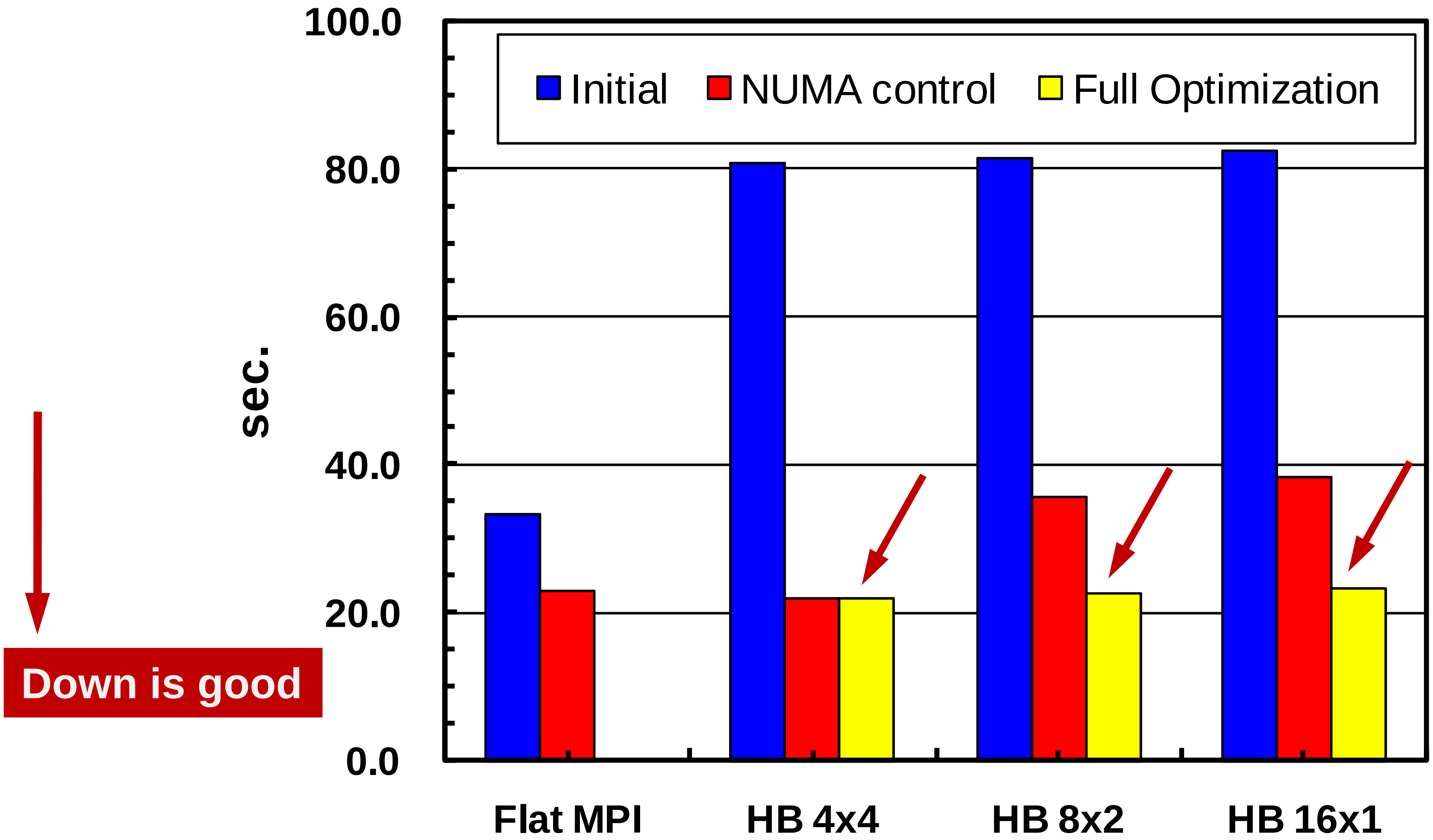
Further Re-Ordering for Continuous Memory Access: Sequential 5 colors, 8 threads



Effect of F.T. + Sequential Data Access

16,777,216= 64×64^3 cells, 64 cores, CM-RCM(2)

Time for Linear Solvers, HB 4x4 is the fastest



Strong Scaling

- 512x256x256= 33,554,432 cells
- Up to 1,024 cores (64 nodes)

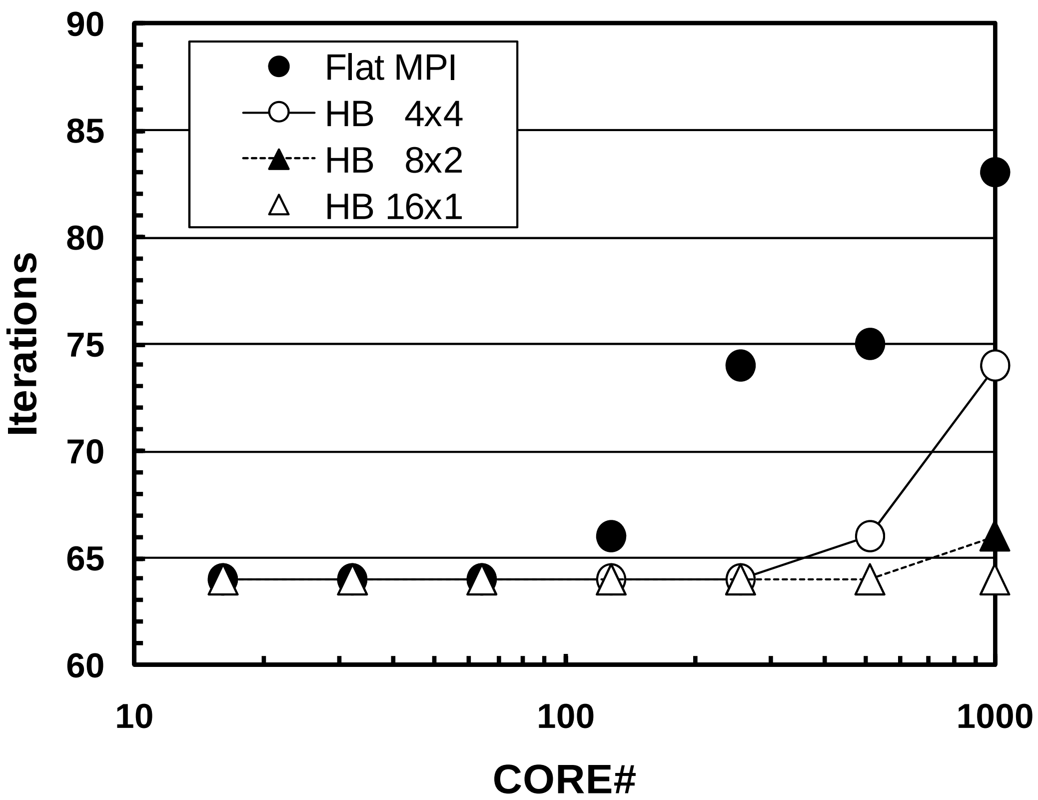
- CM-RCM(2)

Strong Scaling (T2K)

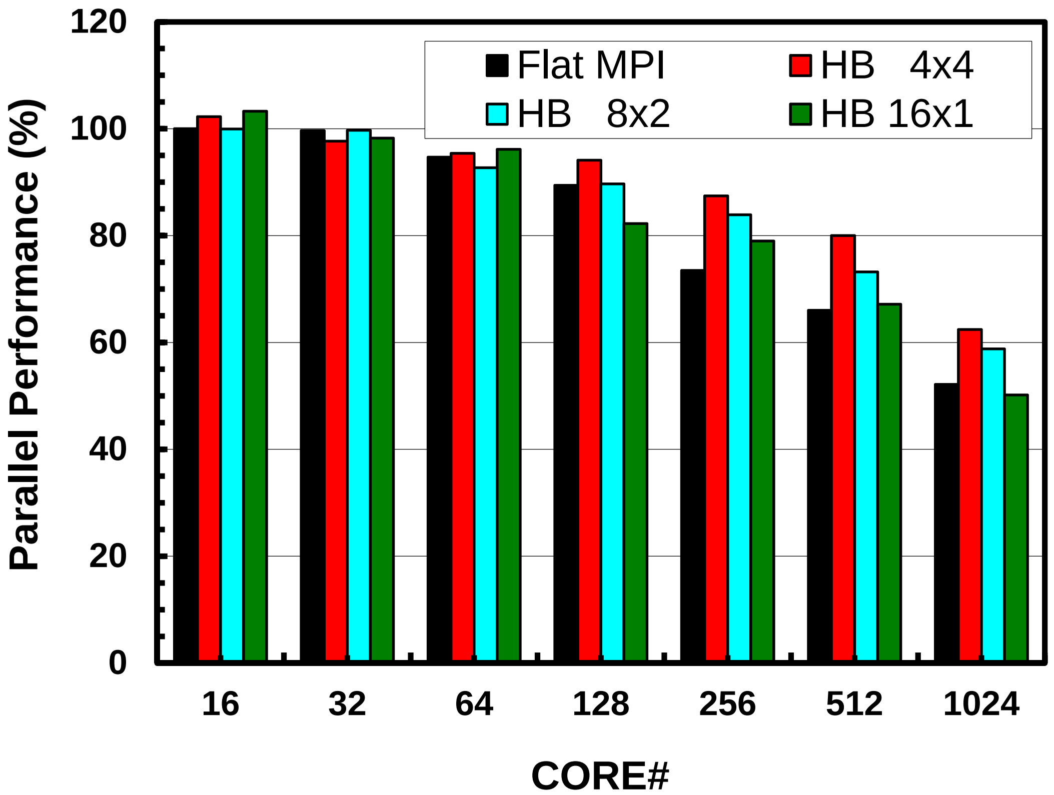
512x256x256= 33,554,432 cells

based on performance of Flat MPI with 16 cores

Iterations



Performance



OpenMP Overhead on SEND/RECV

- ⇒more nodes/cores
  - smaller number of vertices per thread
  - more OpenMP overhead
    - more significant for higher-levels of MG operations

```
!C
!C-- SEND
      do neib= 1, NEIBPETOT
        istart= levEXPORT_index(lev-1,neib) + 1
        iend  = levEXPORT_index(lev  ,neib)
        inum  = iend - istart + 1
!$omp parallel do private (ii)
        do k= istart, iend
              ii   = 3*EXPORT_ITEM(k)
          WS(3*k-2)= X(ii-2)
          WS(3*k-1)= X(ii-1)
          WS(3*k  )= X(ii  )
        enddo
!$omp end parallel do
        call MPI_ISEND (WS(3*istart-2), 3*inum, MPI_DOUBLE_PRECISION,   &
     &                 NEIBPE(neib), 0, SOLVER_COMM, req1(neib), ierr)
      enddo
```

OpenMP Overhead on SEND/RECV

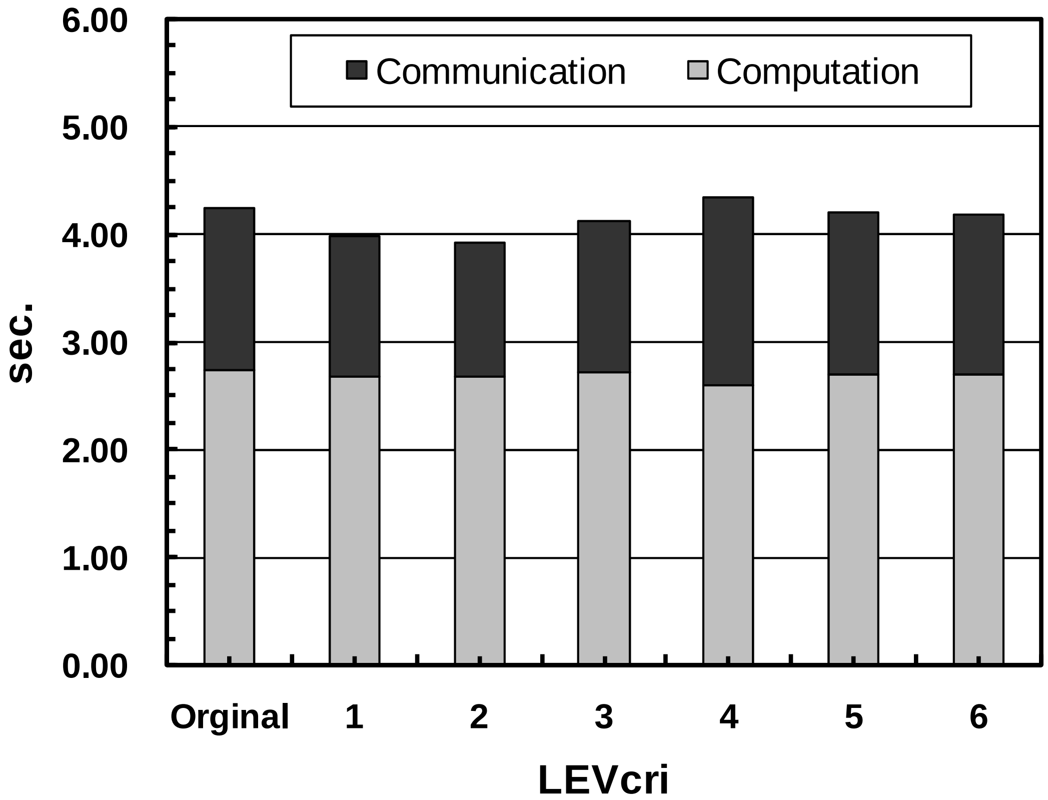
- Serial computation for memory copy in SEND/RECV
- Optimization
  - OpenMP: only for the finest mesh

```
!C
!C-- SEND
      do neib= 1, NEIBPETOT
        istart= levEXPORT_index(lev-1,neib) + 1
        iend  = levEXPORT_index(lev  ,neib)
        inum  = iend - istart + 1

        do k= istart, iend
              ii   = 3*EXPORT_ITEM(k)
          WS(3*k-2)= X(ii-2)
          WS(3*k-1)= X(ii-1)
          WS(3*k  )= X(ii  )
        enddo

        call MPI_ISEND (WS(3*istart-2), 3*inum, MPI_DOUBLE_PRECISION,   &
     &                  NEIBPE(neib), 0, SOLVER_COMM, req1(neib), ierr)
      enddo
```

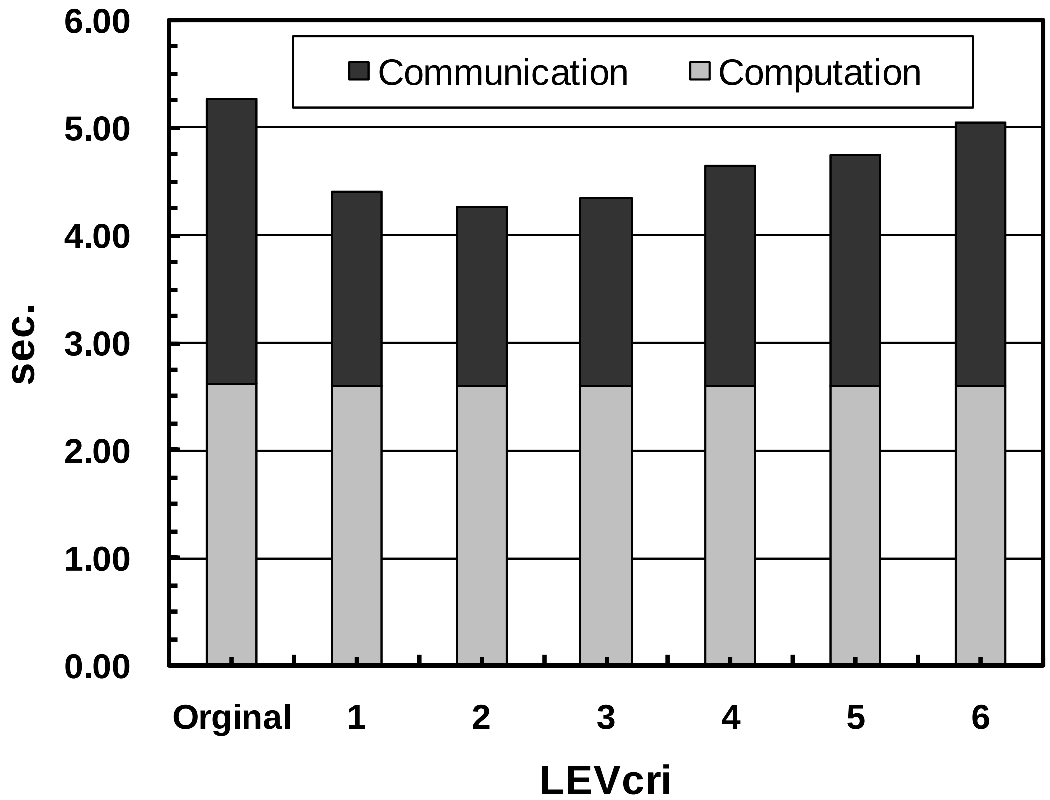
Effect of Optimization at 1,024 cores

- If "**LEVEL (level of multigrid) ≥ LEVcri**", serial memory copy (without OpenMP) is applied.
- **"LEVcri=1"** means NO OpenMP
- **"LEVcri=2" (OpenMP only at the finest level) : best**

HB 4×4



HB 16×1

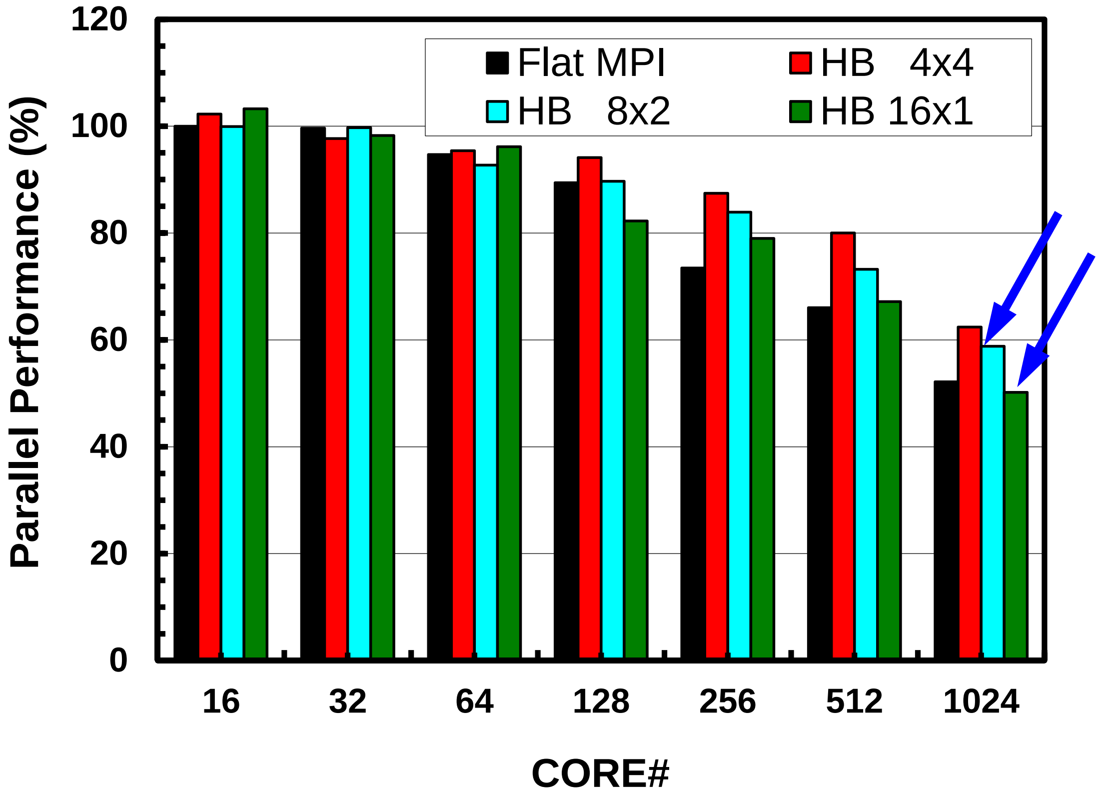


Memory Copy Processes in Comm. Optimized

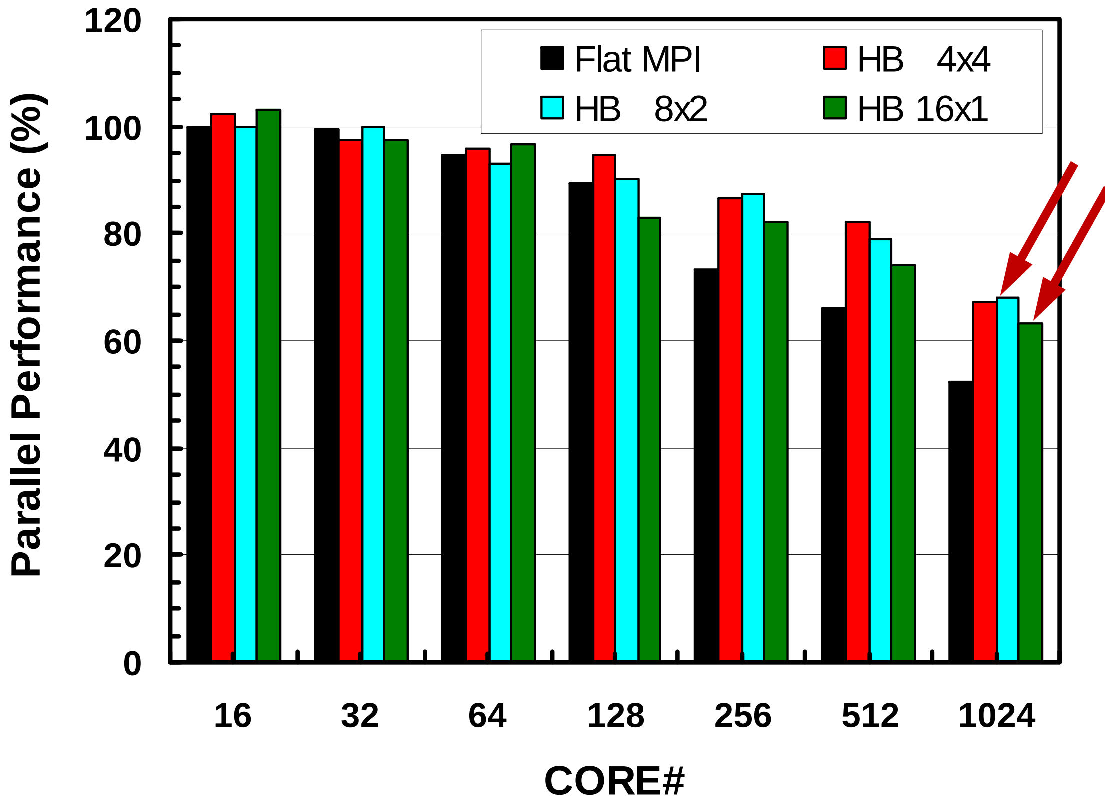
512x256x256= 33,554,432 cells

based on performance of Flat MPI with 16 cores

Before



After



Strong Scale: Parallel Performance

512x256x256= 33,554,432 cells

based on performance of Flat MPI with 16 cores

(Improved coarse-grid smoother does not work well because cost per iteration is larger for many-core cases)