全国共同利用施設
東京大学情報基盤センター
Information Technology Center, The University of Tokyo

東京大学
THE UNIVERSITY OF TOKYO

Flat MPI vs. Hybrid: Evaluation of Parallel Programming Models for Preconditioned Iterative Solvers on "T2K Open Supercomputer"

Kengo NAKAJIMA

Information Technology Center
The University of Tokyo

Second International Workshop on Parallel Programming Models and Systems Software for High-End Computing (P2S2), September 22, 2009, Vienna

to be held in conjunction with ICPP-09: The 38th International Conference on Parallel Processing

Topics of this Study

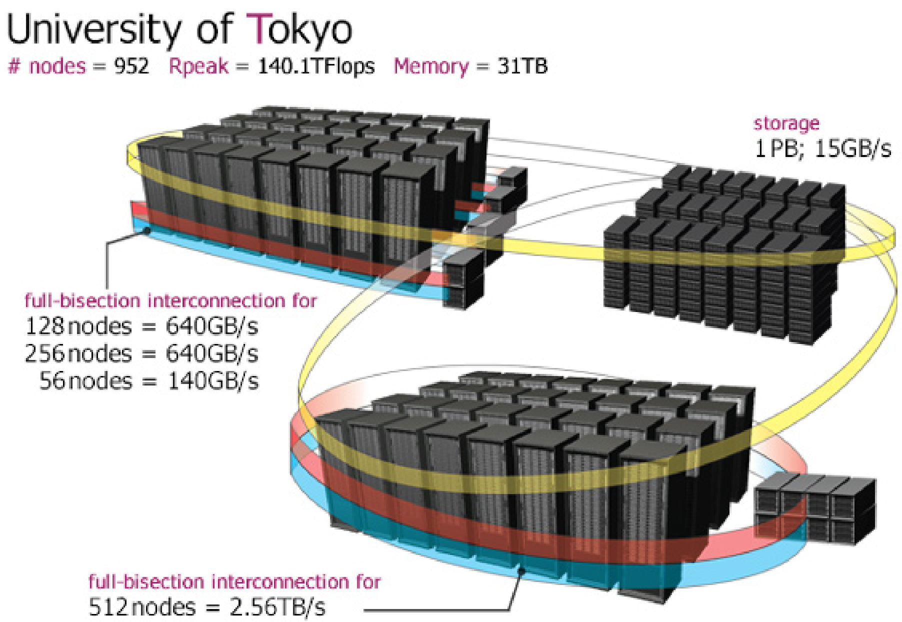
- Preconditioned Iterative Sparse Matrix Solvers for FEM Applications
- T2K Open Supercomputer (Tokyo) (T2K/Tokyo)
- Hybrid vs. Flat MPI Parallel Programming Models
- Optimization of Hybrid Parallel Programming Models
  - NUMA Control
  - First Touch
  - Further Reordering of Data

TOC

T2K/Tokyo (1/2)

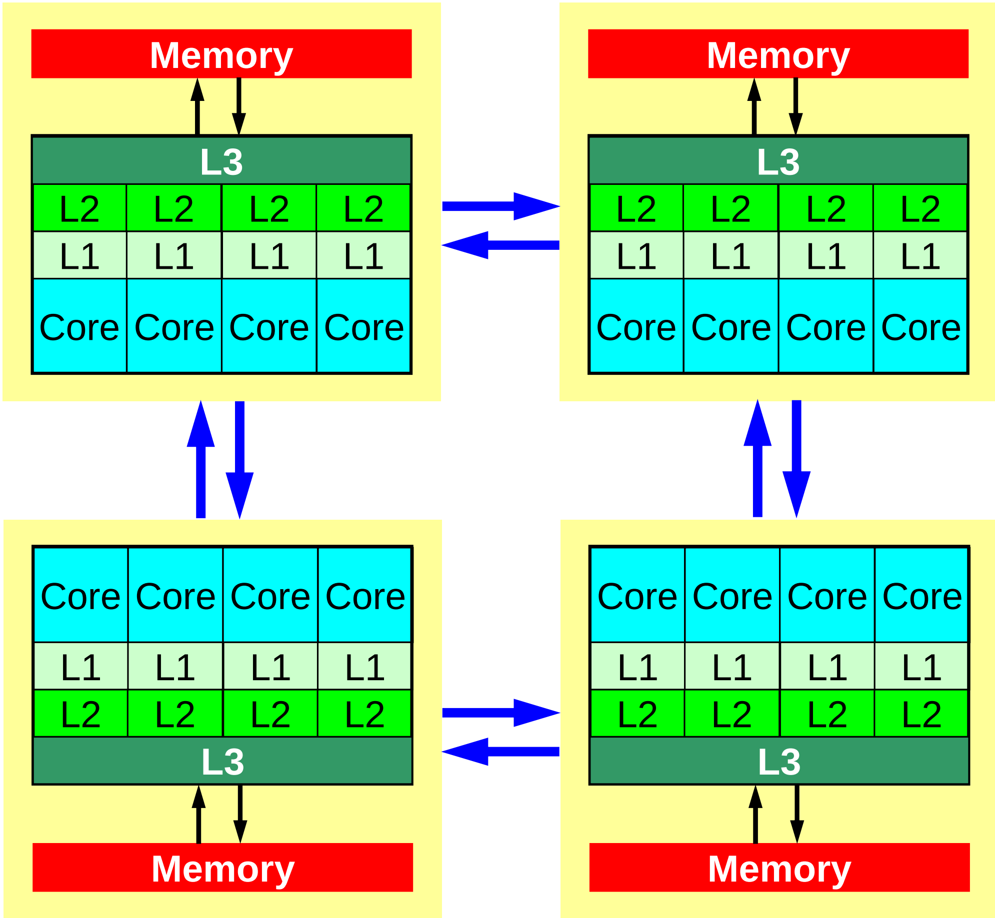
- "T2K Open Supercomputer Alliance"
  - http://www.open-supercomputer.org/
  - Tsukuba, Tokyo, Kyoto

- "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)
  - 27th in TOP500 (NOV 2008) (fastest in Japan at that time)



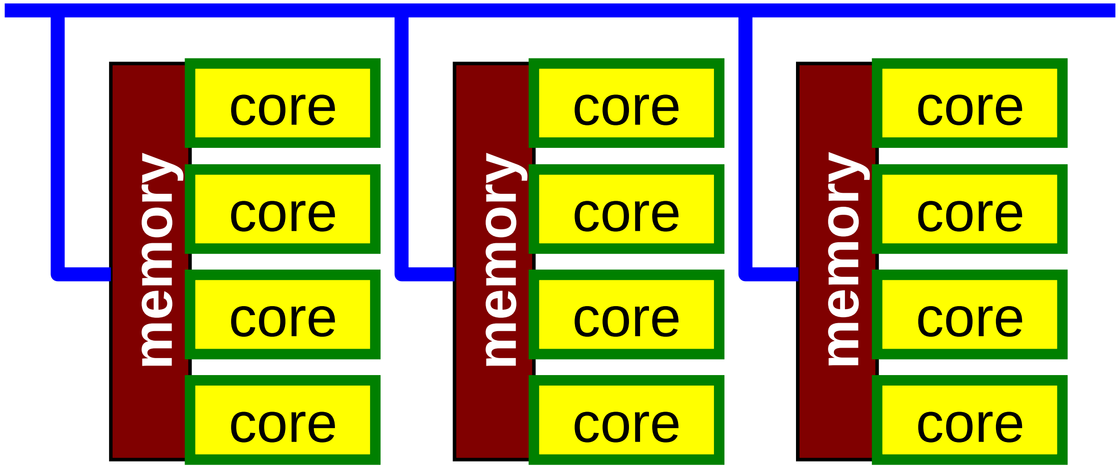
T2K/Tokyo (2/2)

- 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
  - To reduce memory traffic in the system, it is important to keep the data close to the cores that will work with the data (e.g. NUMA control).

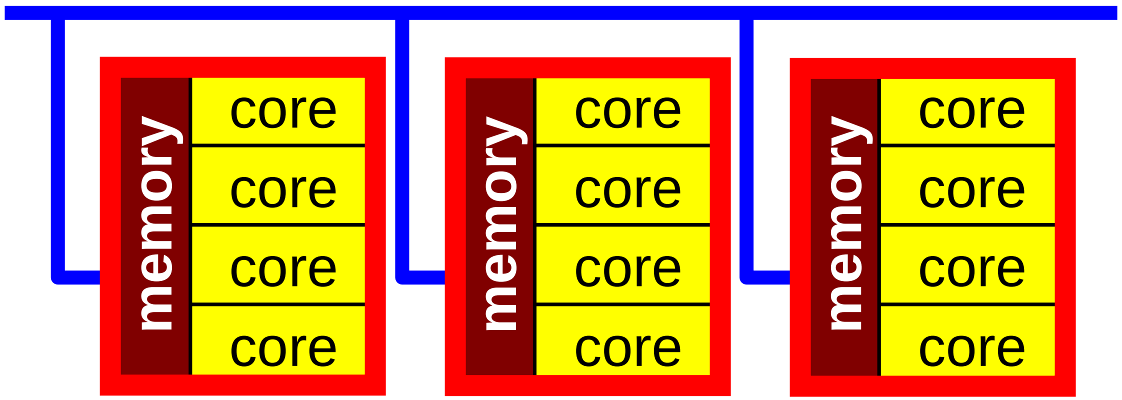


Flat MPI vs. Hybrid

Flat-MPI : Each PE -> Independent



Hybrid : Hierarchal Structure



Flat MPI vs. Hybrid

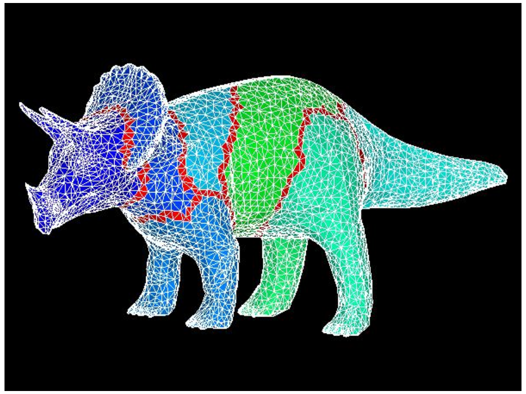
- Performance is determined by various parameters

- Hardware
  - core architecture itself
  - peak performance
  - memory bandwidth, latency
  - network bandwidth, latency
  - their balance
- Software
  - types: memory or network/communication bound
  - problem size

Sparse Matrix Solvers by FEM, FDM …

```
for (i=0; i<N; i++) {
  for (k=Index(i-1); k<Index(i); k++{
    Y[i]= Y[i] + A [k]*X[Item[k]];
  }
}
```

- Memory-Bound
  - indirect accesses
  - Hybrid (OpenMP) is more memory-bound
- Latency-Bound for Parallel Computations
  - comm.’s occurs only at domain boundaries
  - small amount of messages

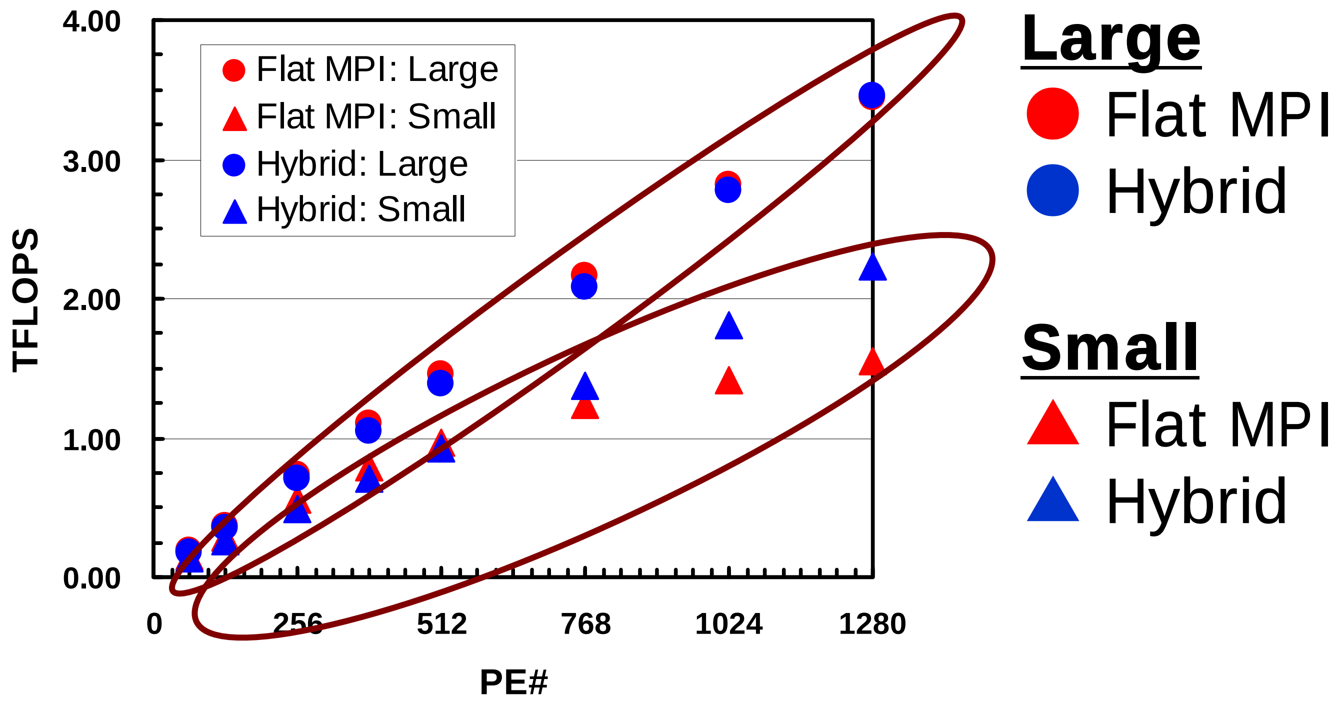


- Exa-scale Systems
  - $O(10^8)$ cores
  - Communication Overhead by MPI Latency for > 10^8-way MPI’s
  - **Expectations for Hybrid**
    - **1/16 MPI processes for T2K/Tokyo**

Weak Scaling Results on ES

GeoFEM Benchmarks [KN 2003]

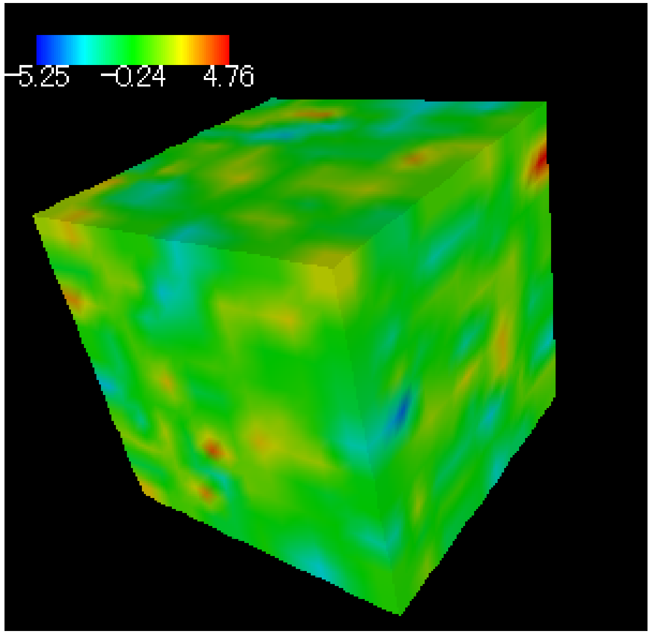
- Generally speaking, hybrid is better for large number of nodes
- especially for small problem size per node
  - "less" memory bound



- Background
  - Why Hybrid ?
- **Target Application**
  - **Overview**
  - **HID**
  - **Reordering**
- Preliminary Results
- Remarks

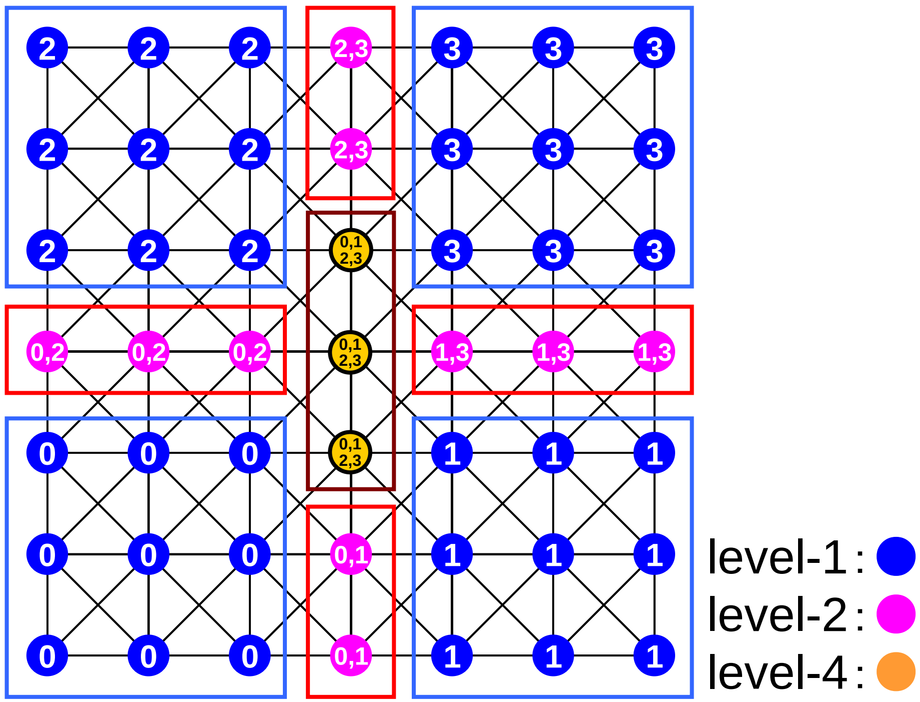
Target Application

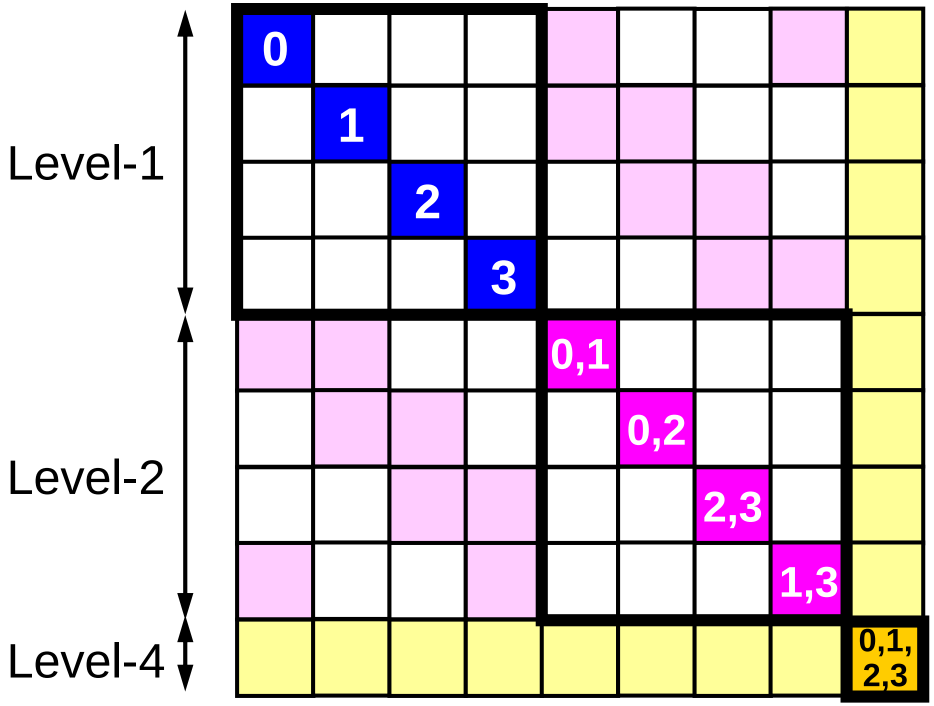
- 3D Elastic Problems with Heterogeneous Material Property
  - $E_{max}=10^3$, $E_{min}=10^{-3}$, $\nu=0.25$
    - generated by "sequential Gauss" algorithm for geo-statistics [Deutsch & Journel, 1998]
  - 128^3 tri-linear hexahedral elements, 6,291,456 DOF
    - Strong Scaling
- (SGS+CG) Iterative Solvers
  - Symmetric Gauss-Seidel
  - HID-based domain decomposition
- T2K/Tokyo
  - 512 cores (32 nodes)
- FORTARN90 (Hitachi) + MPI
  - Flat MPI, Hybrid (4x4, 8x2, 16x1)



HID: Hierarchical Interface Decomposition [Henon & Saad 2007]

- Multilevel Domain Decomposition
  - Extension of Nested Dissection
- Non-overlapped Approach: Connectors, Separators
- Suitable for Parallel Preconditioning Method

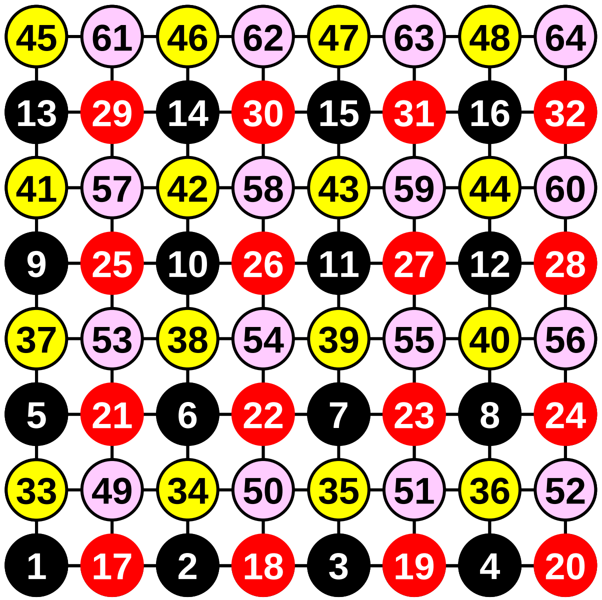


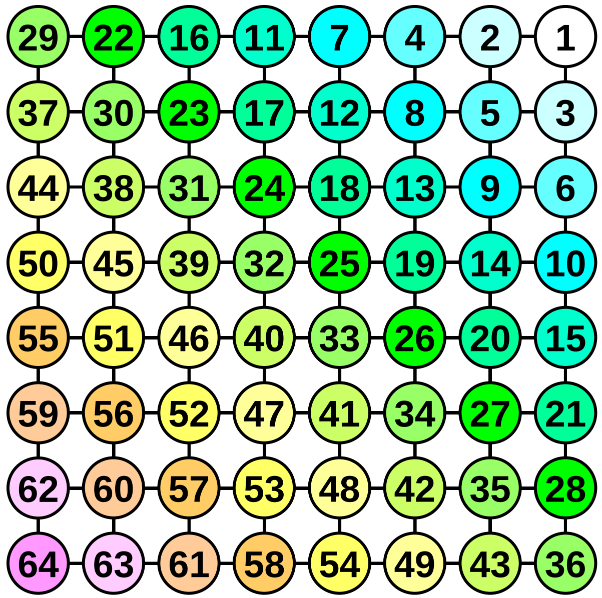


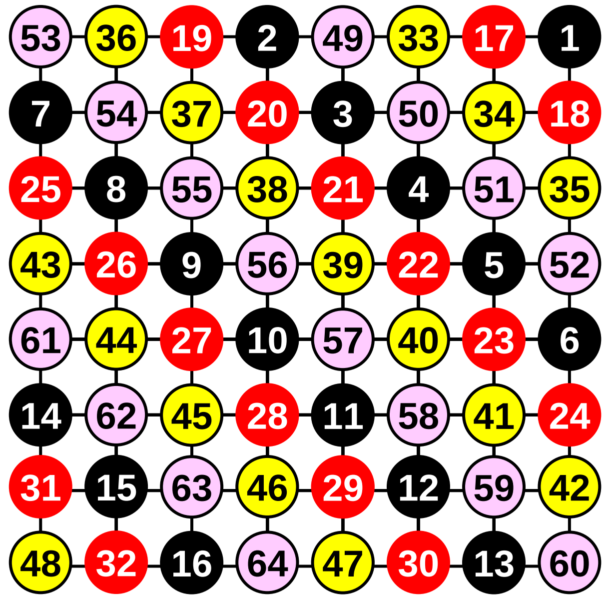
Parallel Preconditioned Iterative Solvers on an SMP/Multicore node by OpenMP

- DAXPY, SMVP, Dot Products
  - Easy
- Factorization, Forward/Backward Substitutions in Preconditioning Processes
  - Global dependency
  - Reordering for parallelism required: forming independent sets
  - Multicolor Ordering (MC), Reverse-Cuthill-Mckee (RCM)
  - Works on “Earth Simulator” [KN 2002,2003]
    - both for parallel/vector performance
- CM-RCM (Cyclic Multi Coloring + RCM)
  - robust and efficient
  - elements on each color are independent

Ordering Methods





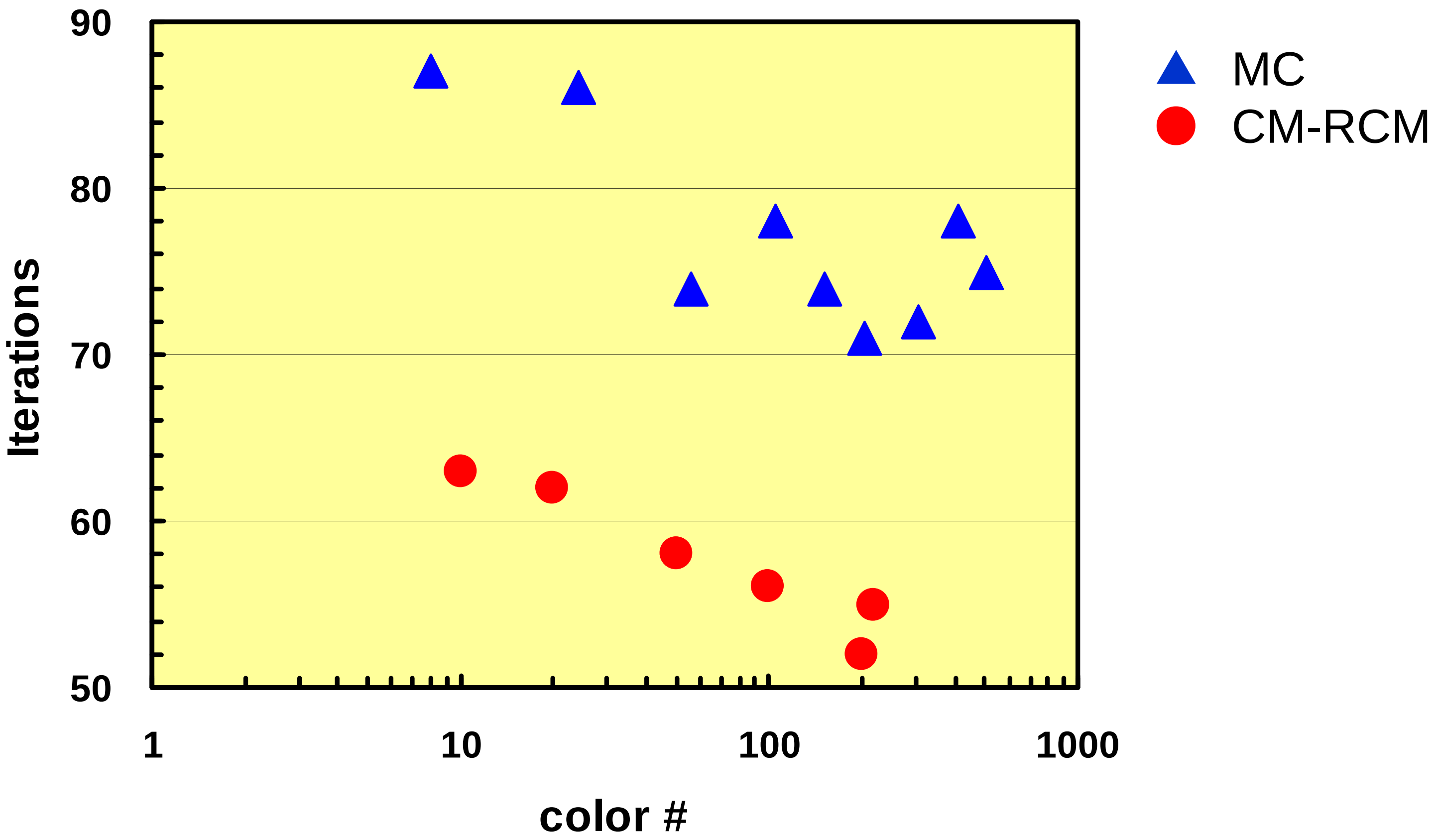


MC (Color#=4)
Multicoloring

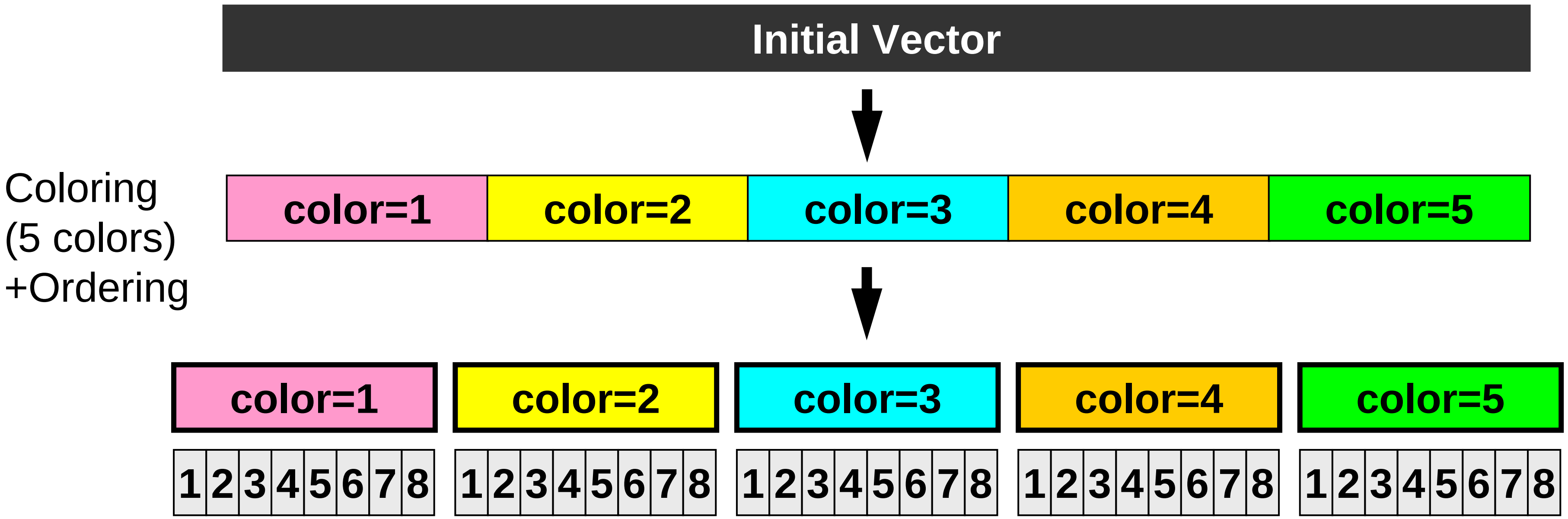
RCM
Reverse Cuthill-Mckee

CM-RCM (Color#=4)
Cyclic MC + RCM

Effect of Ordering Methods on Convergence



Re-Ordering by CM-RCM 5 colors, 8 threads

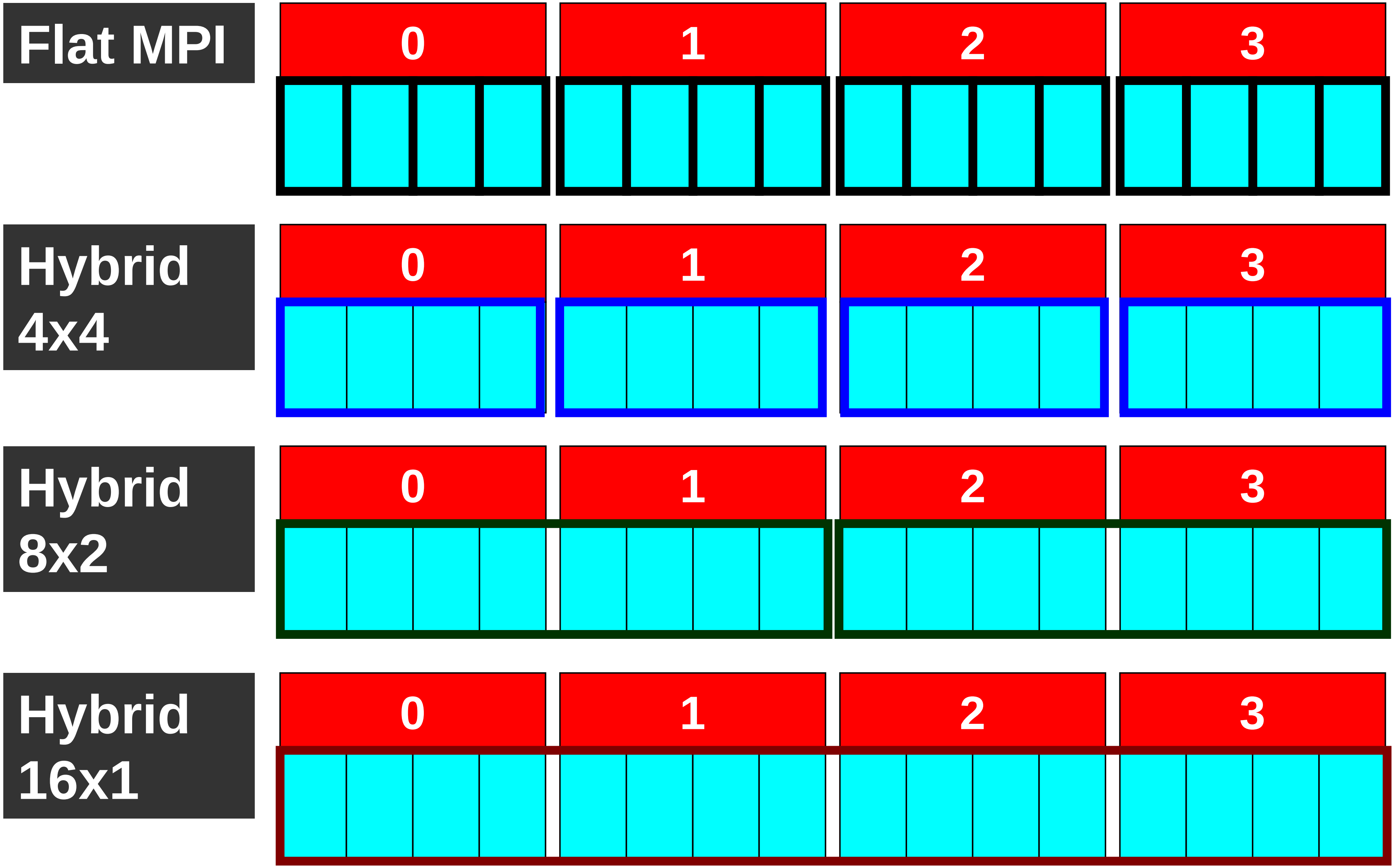


Elements in each color are independent, therefore parallel processing is possible. => divided into OpenMP threads (8 threads in this case)

Because all arrays are numbered according to “color”, discontinuous memory access may happen on each thread.

- Background
  - Why Hybrid ?
- Target Application
  - Overview
  - HID
  - Reordering
- **Preliminary Results**
- Remarks

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



CASES for Evaluation

- **Focused on optimization of HB8x2, HB16x1**

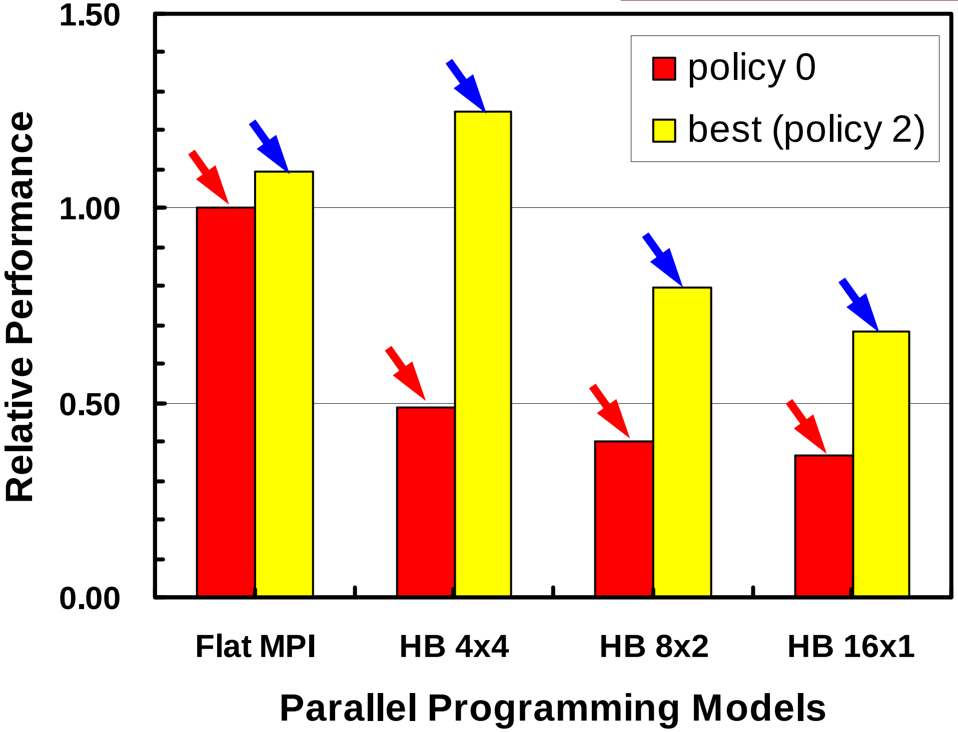
- CASE-1
  - initial case (CM-RCM)
  - for evaluation of NUMA control effect
    - specifies local core-memory configulation
- CASE-2 (Hybrid only)
  - First-Touch
- CASE-3 (Hybrid only)
  - Further Data Reordering + First-Touch
- NUMA policy (0-5) for each case

Results of CASE-1, 32 nodes/512cores

computation time for linear solvers

Normalized by Flat MPI (Policy 0)

Policy ID	Command line switches
0	no command line switches
1	`--cpunodebind=$SOCKET` `--interleave=all`
2	`--cpunodebind=$SOCKET` `--interleave=$SOCKET`
3	`--cpunodebind=$SOCKET` `--membind=$SOCKET`
4	`--cpunodebind=$SOCKET` `--localalloc`
5	`--localalloc`



Method	Iterations	Best Policy CASE-1
Flat MPI	1264	2
HB 4x4	1261	2
HB 8x2	1216	2
HB 16x1	1244	2

e.g. mpirun –np 64 –cpunodebind 0,1,2,3 a.out

First Touch Data Placement

ref. “Patterns for Parallel Programming” Mattson, T.G. et al.

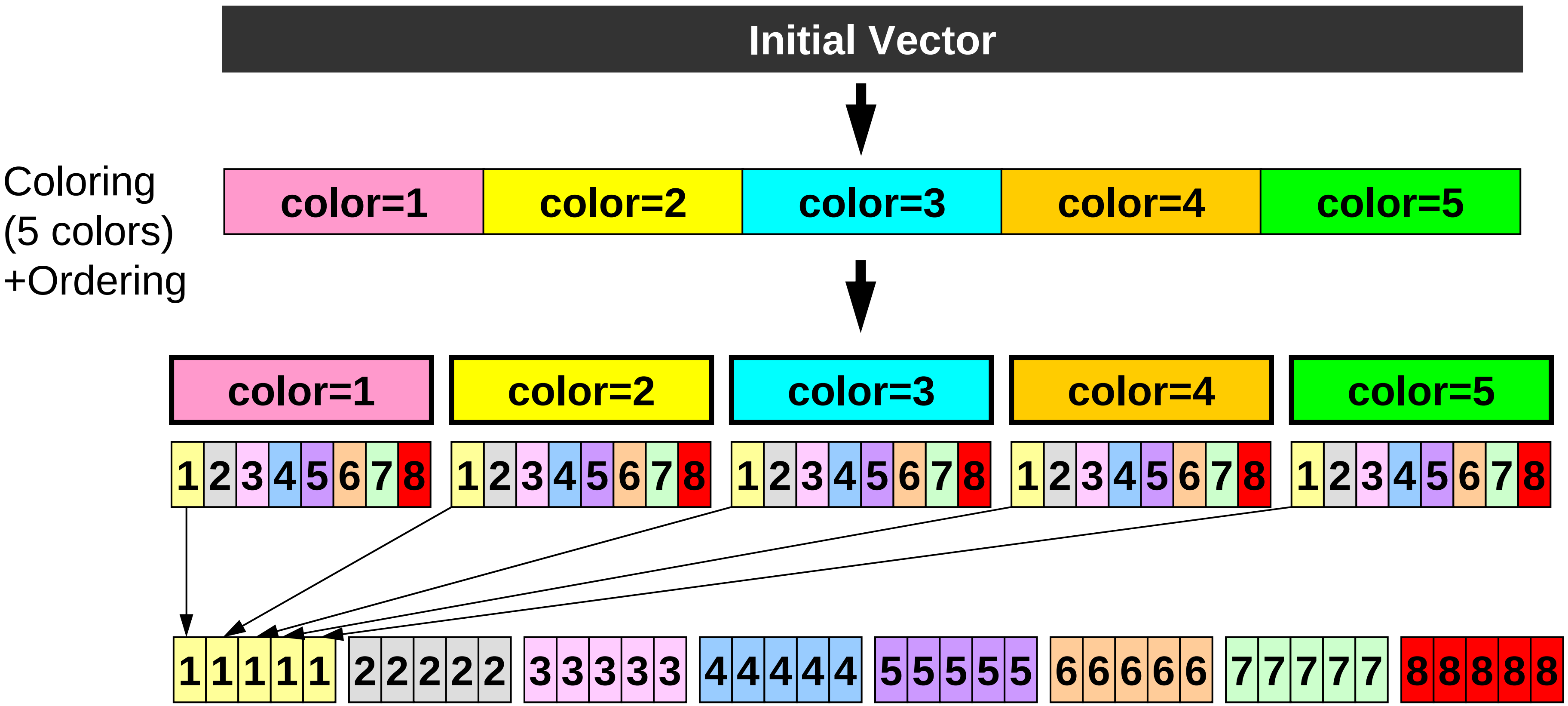
To reduce memory traffic in the system, it is important to keep the data close to the PEs that will work with the data (e.g. NUMA control).

On NUMA computers, this corresponds to making sure the pages of memory are allocated and “owned” by the PEs that will be working with the data contained in the page.

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.

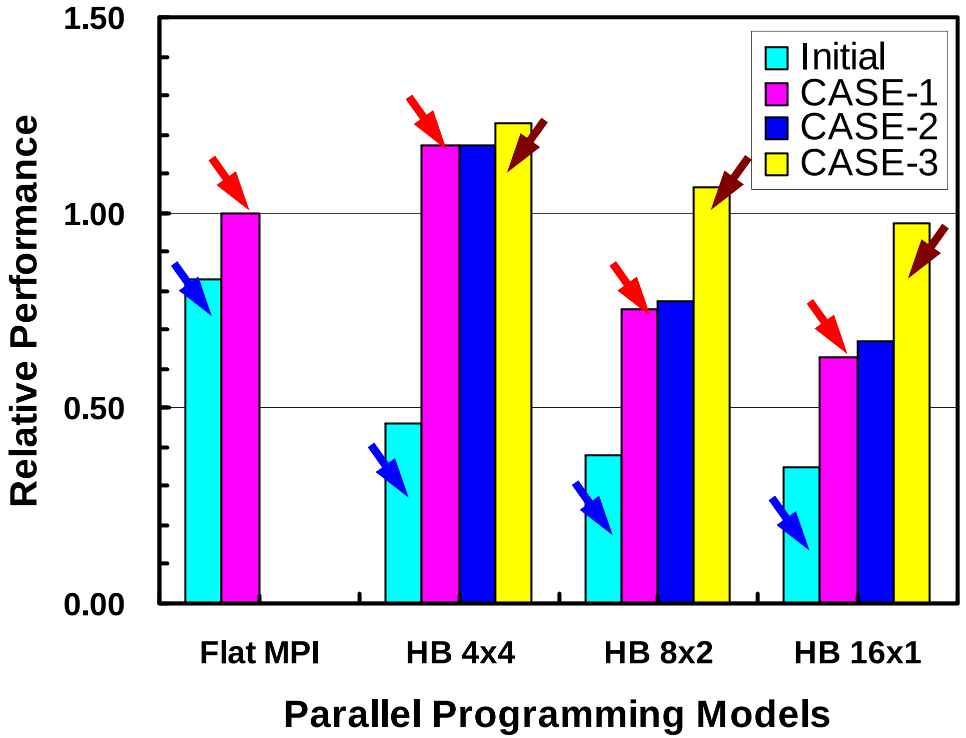
Further Re-Ordering for Continuous Memory Access 5 colors, 8 threads



Improvement: CASE-1 ⇒ CASE-3

Normalized by the Best Performance of Flat MPI

32nodes, 512cores
196,608 DOF/node

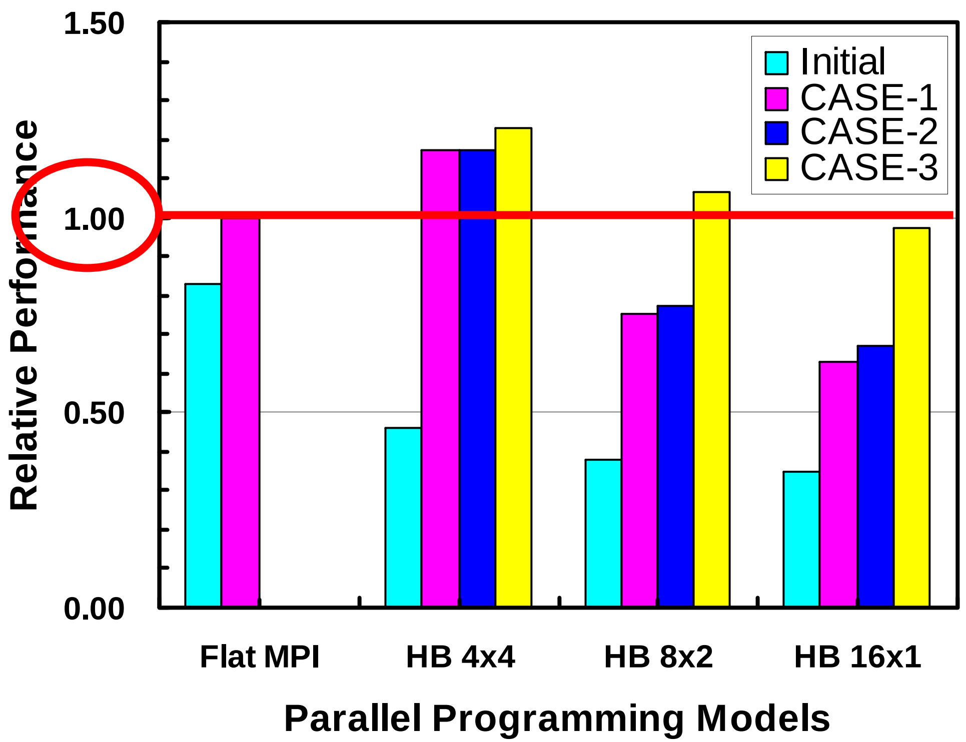


CASE-1: NUMA control
CASE-2: + F.T.
CASE-3: + Further Reordering

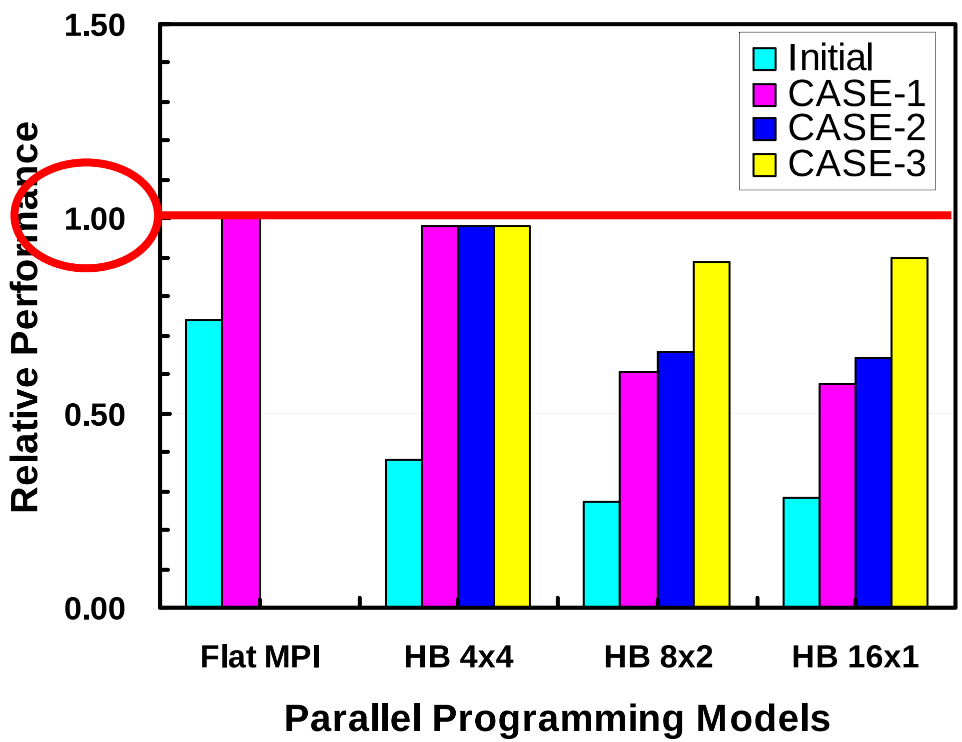
Improvement: CASE-1 ⇒ CASE-3

Normalized by the Best Performance of Flat MPI

32nodes, 512cores
196,608 DOF/node



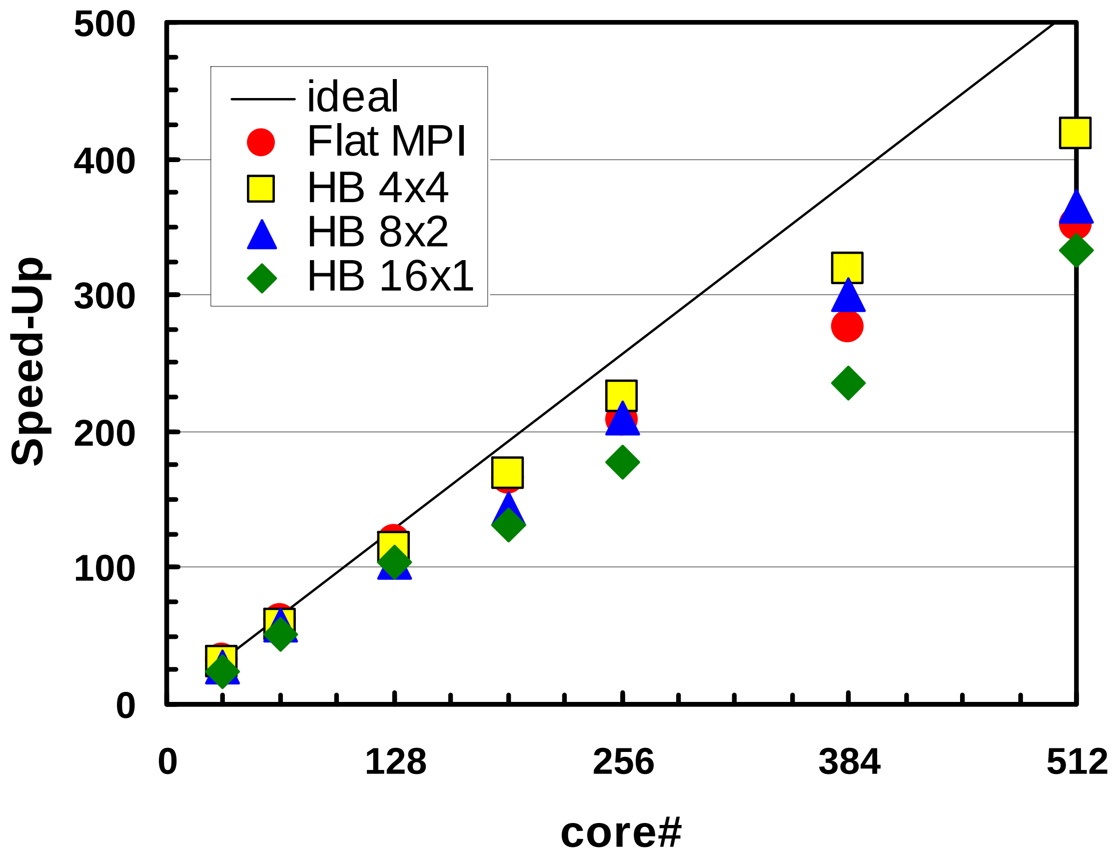
8nodes, 128cores
786,432 DOF/node



Strong Scalability (Best Cases)

32~512 cores

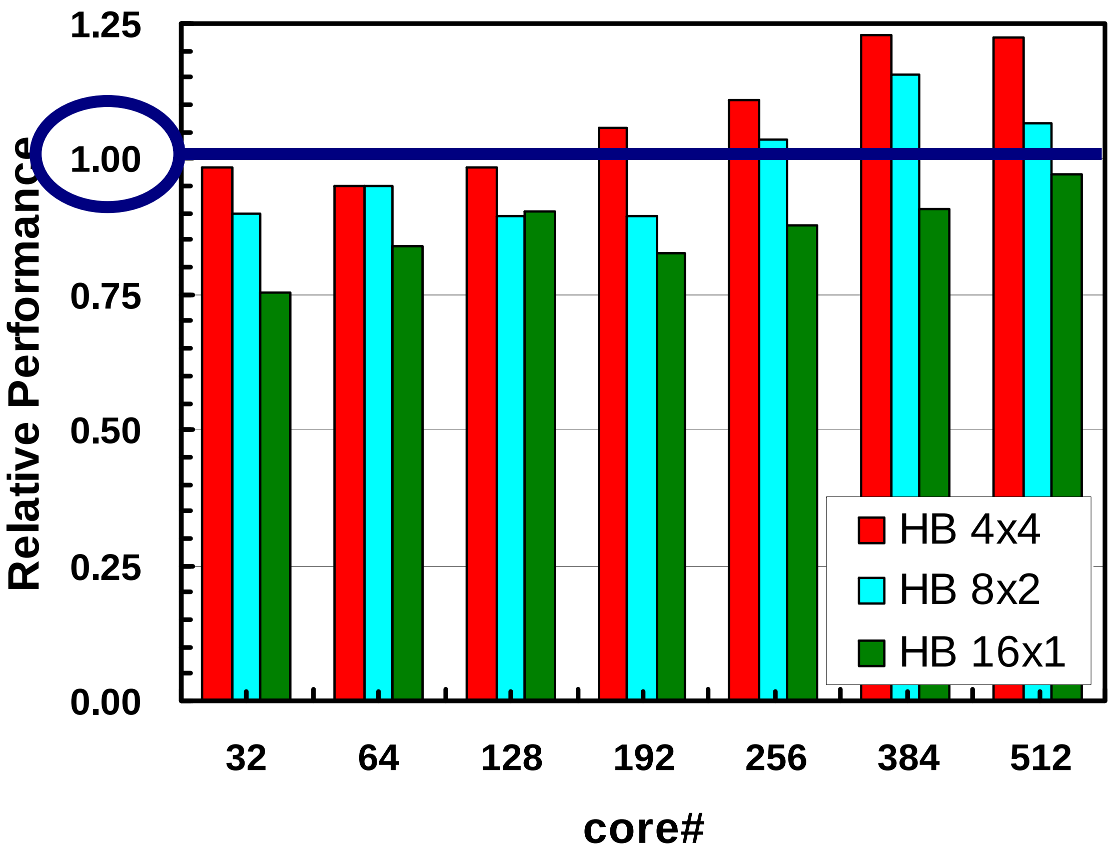
Performance of Flat MPI with 32 cores= 32.0



Relative Performance for Strong Scaling (Best Cases)

32~512 cores

Normalized by BEST Flat MPI at each core#



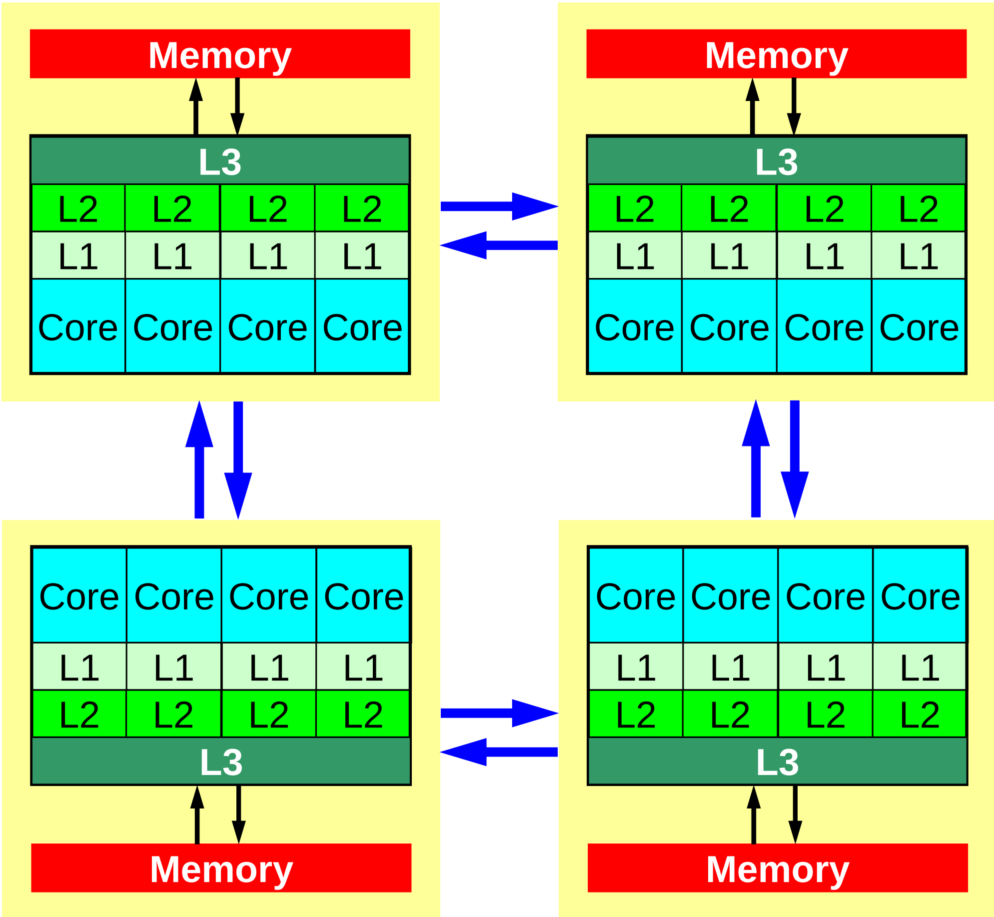
- Background
  - Why Hybrid ?
- Target Application
  - Overview
  - HID
  - Reordering
- Preliminary Results
- **Remarks**

Summary & Future Works

- HID for Ill-Conditioned Problems on T2K/Tokyo
  - Hybrid/Flat MPI, CM-RCM reordering
- Hybid 4x4 and Flat MPI are competitive
- Data locality and continuous memory access by (further re-ordering + F.T.) provide significant improvement on Hybrid 8x2/16x1.
- Performance of Hybrid is improved when,
  - many cores, smaller problem size/core (strong scaling)
- Future Works
  - Higher-order of Fill-ins: BILU(p)
  - Extension to Multigrid-type Solvers/Preconditioning
  - Considering Page-Size for Optimizaion
  - Sophisticated Models for Performance Prediction/Evaluation

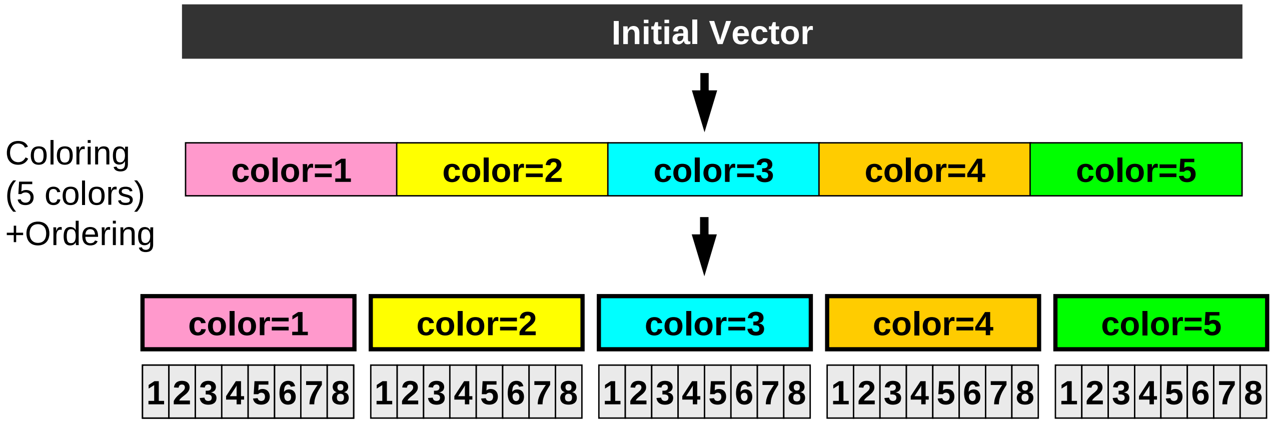
Summary & Future Works (cont.)

- Improvement of Flat MPI
  - Current “Flat MPI” is not really flat
  - Socket, Node, Node-to-Node
- Extension to GPGPU



GPGPU Community

- Coalesced Access (better one)



- Sequential Access