Introduction to Parallel Programming for Multicore/Manycore Clusters

Part IV: Introduction to Tuning

Kengo Nakajima
Information Technology Center
The University of Tokyo

- What is “tuning/optimization” ?
- Vector/Scalar Processors
- Example: Scalar Processors

What is Tuning/Optimization

Optimization of Performance

- Purpose
  - Reduction of computation time
  - Optimization

- How to tune/optimize codes
  - Applying new algorithms
  - Modification/optimization according to property/parameters of H/W
  - Tuning/optimization should not affect results themselves.
    - or you have to recognize that results may change due to tuning/optimization

How do we tune/optimize codes ?

- When do we apply tuning ?
  - It's difficult if you apply tuning to the codes with $O(10^4)$ lines ...
- You have to be careful to write "efficient" codes.
  - Several tips.
- Simple, Readable codes
  - codes with few bugs
- Using optimized libraries
- Good parallel program = good serial program
- Tuning/optimization – faster computation – efficient research ...

Some References (in Japanese)

- スカラープロセッサ
  - 寒川「RISC超高速化プログラミング技法」, 共立出版, 1995.
  - Dowd（久良知訳）「ハイ・パフォーマンス・コンピューティング-RISCワークステーションで最高のパフォーマンスを引き出すための方法」, トムソン, 1994.
  - Goedecker, Hoisie “Performance Optimization for Numerically Intensive Codes”, SIAM, 2001.
- 自動チューニング
  - 片桐「ソフトウェア自動チューニング」, 慧文社, 2004.
- ベクトルプロセッサ
  - 長島, 田中「スーパーコンピュータ」, オーム社, 1992.
  - 奥田, 中島「並列有限要素解析」, 培風館, 2004.

Tips for Tuning/Optimization

- Be careful about memory access patterns
- Avoid calling functions in innermost loops
  - Inline expansion of modern compilers might not work efficiently.
  - Avoid “if-clauses” in innermost loops.
- Avoid “too-multi-nested” loops
- Avoid many division operations, calling built-in-functions
- Avoid redundant operations
  - Storing in memory
  - Trade-off with memory capacity
- **Unfortunately, dependency on compilers and H/W is very significant !**
  - **Optimum options/directives through empirical studies**
  - **Today’s content is very general remedy.**

Example: Multi-Nested Loops

- Overhead for initialization of loop-counter occurs at every do-loop.
  - In the lower-left example (blue), innermost loop is reached 10^6 times. Therefore, 10^6 times initialization of loop-counter occurs.
  - In the lower-right example (yellow) with loop expansion, only one initialization of loop-counter occurs.

```
real*8 AMAT(3,1000000)
. . .
do j= 1, 1000000
  do i= 1, 3
    A(i,j)= A(i,j) + 1.0
  enddo
enddo
. . .
```

```
real*8 AMAT(3,1000000)
. . .
do j= 1, 1000000
  A(1,j)= A(1,j) + 1.0
  A(2,j)= A(2,j) + 1.0
  A(3,j)= A(3,j) + 1.0
enddo
. . .
```

Simple ways for measuring performance

- "time" command
- "timer" subroutines/functions
- Tools for "profiling"
  - Detection of "hot spots"
  - gprof (UNIX)
  - Tools for compilers/systems
    - pgprof: PGI
    - Vtune: Intel
    - Special Profiler: e.g. Fujitsu PRIMEHPC FX10

Files on Reedbush-U

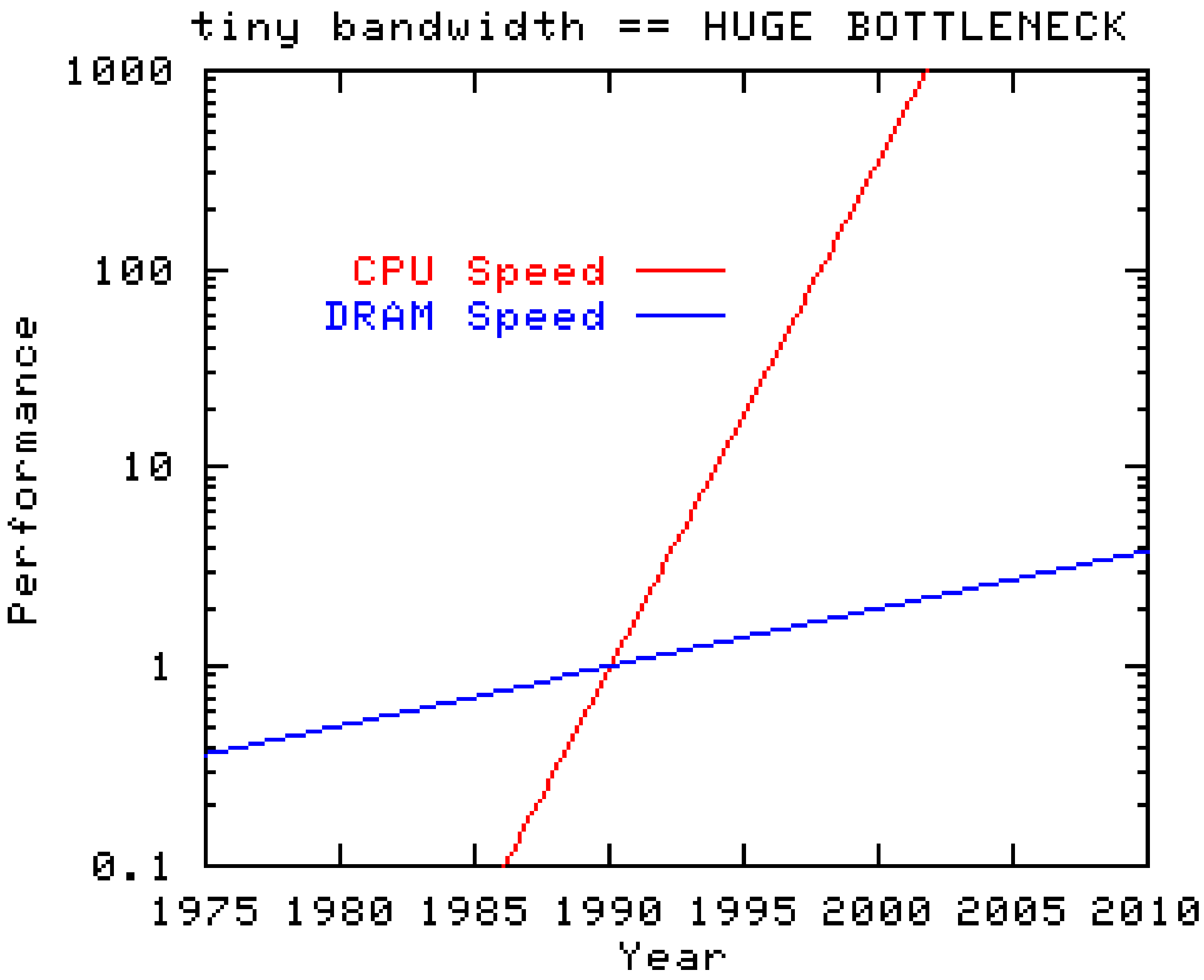
```
>$ cdw

>$ cp /lustre/gt00/z30088/omp/omp-f4.tar .

>$ tar xvf omp-f4.tar

>$ cd multicore/omp      <$O-omp>
>$ ls

dense run src20
```

- What is “tuning/optimization” ?
- **Vector/Scalar Processors**
- Example: Scalar Processors

Scalar/Vector Processors

- Scalar Processors
  - Gap between clock rate and memory bandwidth.
    - getting better, but multi/many-core architectures appear
  - Low Peak-Performance Ratio
    - Ex.: IBM Power3/Power4, 5-8% in FEM applications
- "Traditional" Vector Processors
  - High Peak-Performance Ratio
    - Ex.: Earth Simulator, 35% in FEM applications
  - requires …
    - very special tuning for vector processors
    - sufficiently long loop (problem size)
  - Appropriate for rather simpler problems
  - GPU, Intel Xeon Phi

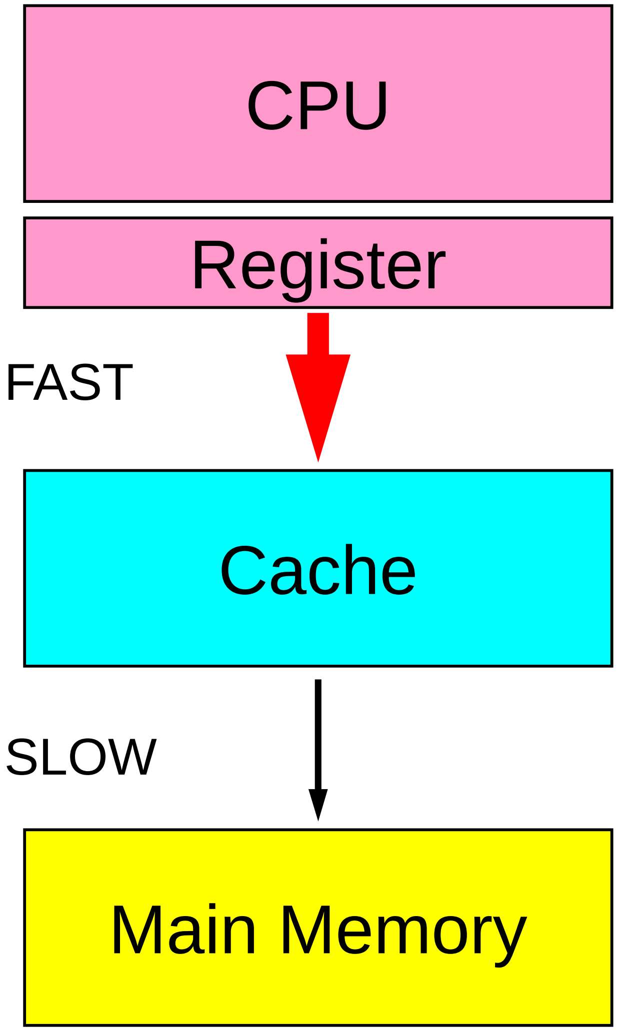
Gap between performance of CPU and Memory



http://www.streambench.org/

Scalar Processors

CPU-Cache: Hierarchical Structure

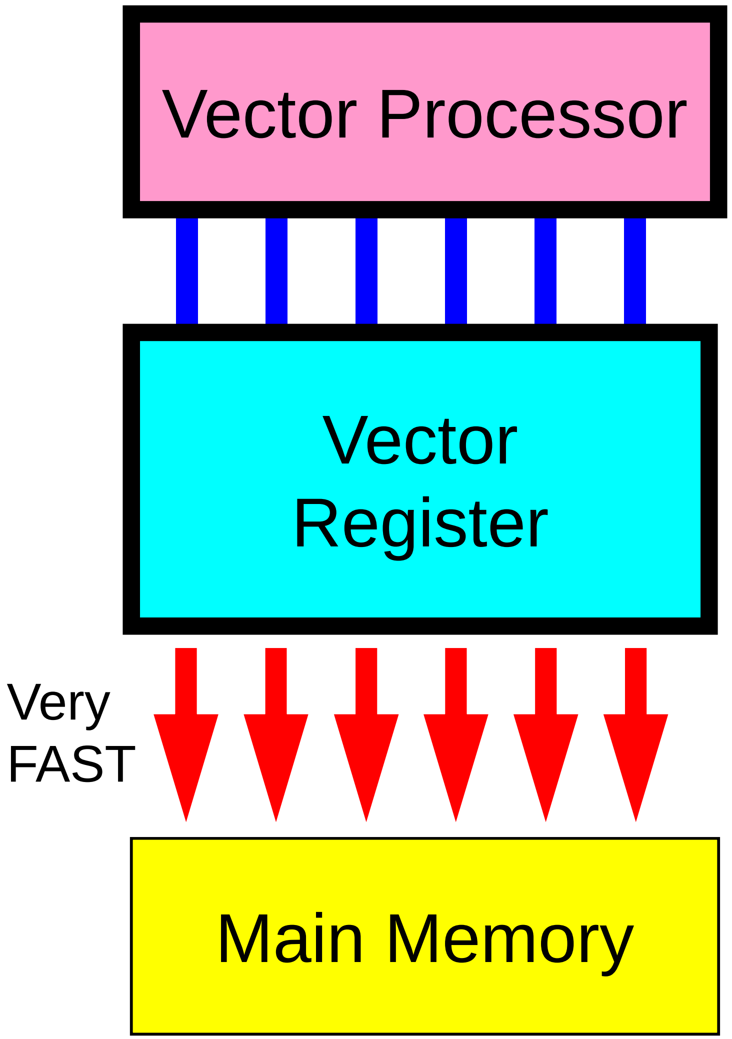


Small (MB)
Expensive
Large (with $O(10^8)$-$O(10^9)$ transistors)
Instruction/Data Cache
Hierarchy: L1, L2, L3

Large (GB)
Cheap

"Traditional" Vector Processors

Vector Registers/Fast Memory



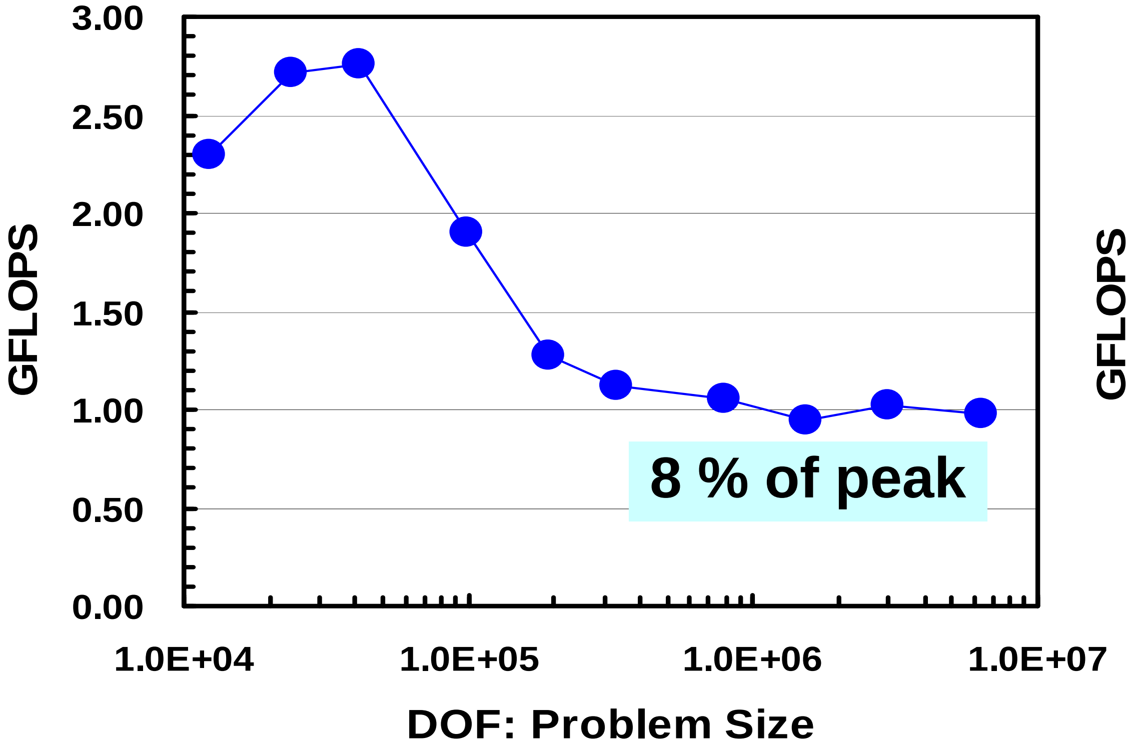
- Parallel operations for simple do-loops.
- Good for large-scale simple problems

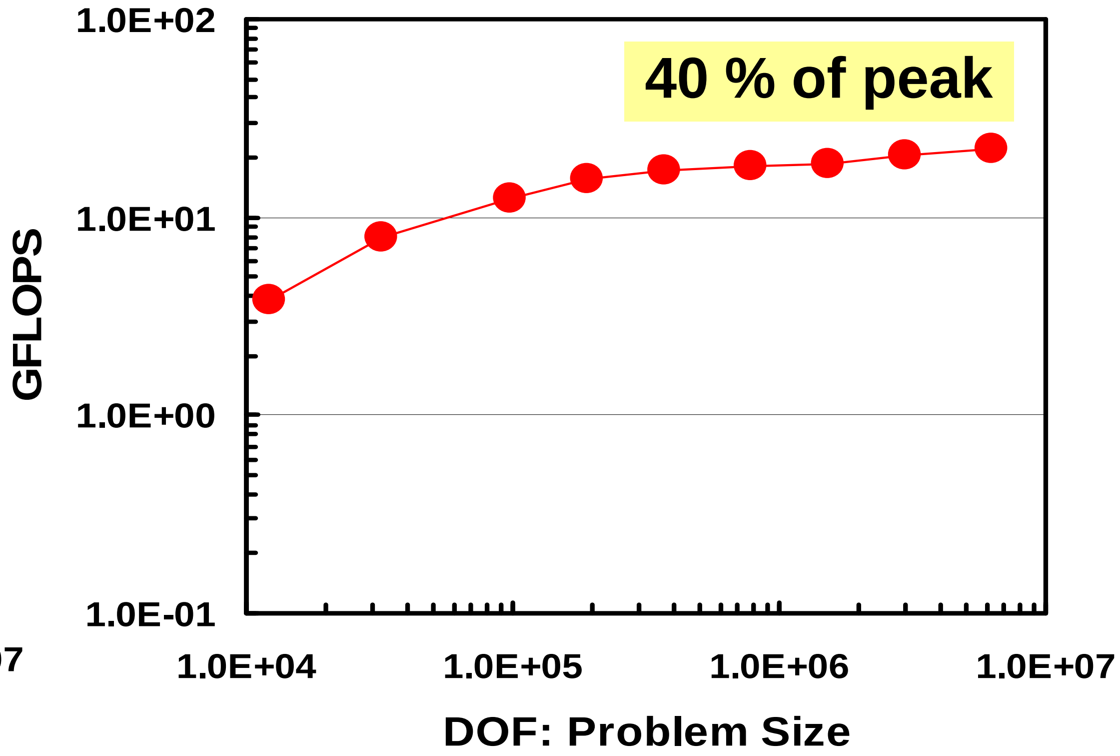
```
do i= 1, N
  A(i)= B(i) + C(i)
enddo
```

NO cache

Modern vector processors have cache
GPU, Xeon Phi: short vectors

Typical Behaviors

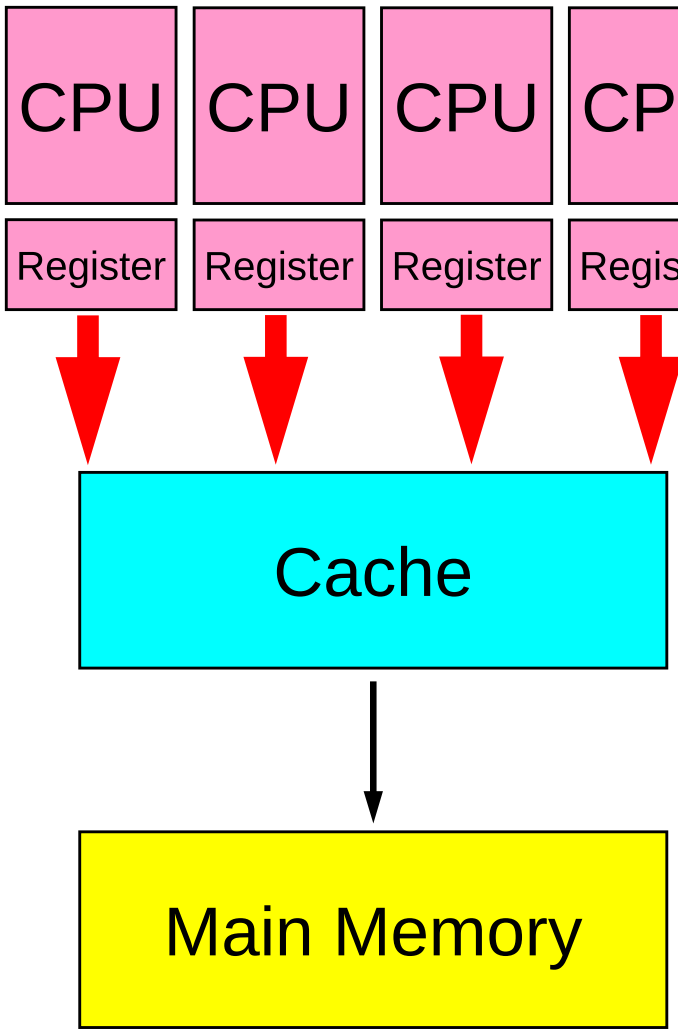


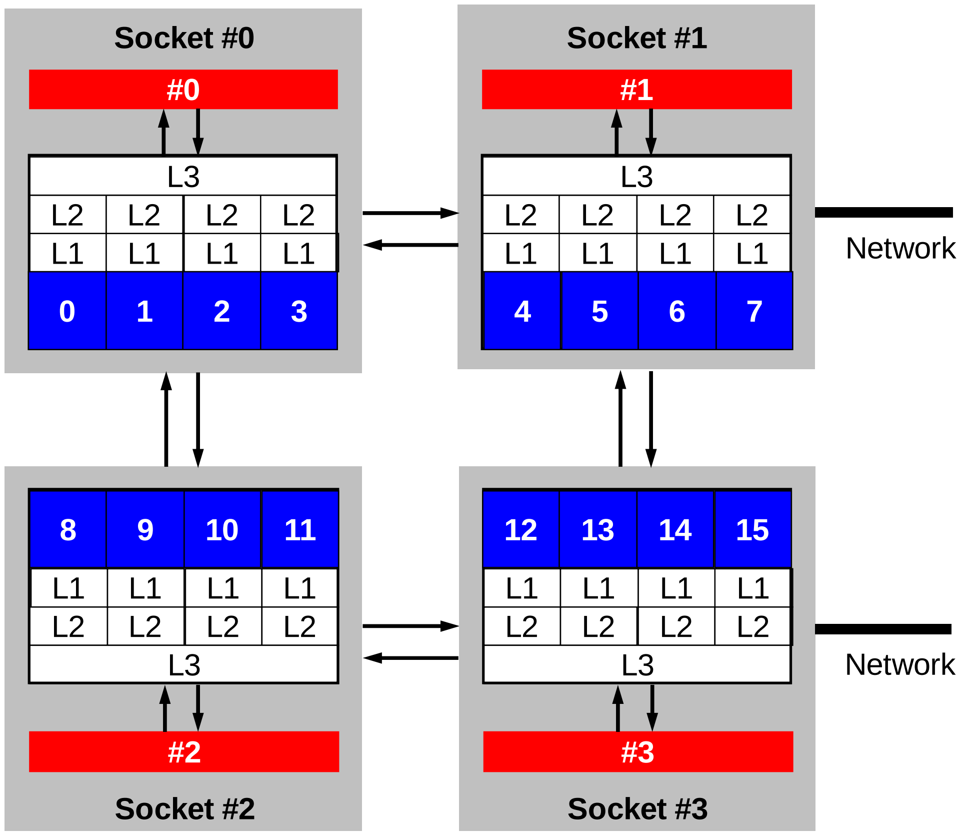


IBM-SP3:
Higher performance for small problems, effect of cache

Earth Simulator:
Higher performance for large-scale problems with longer loops

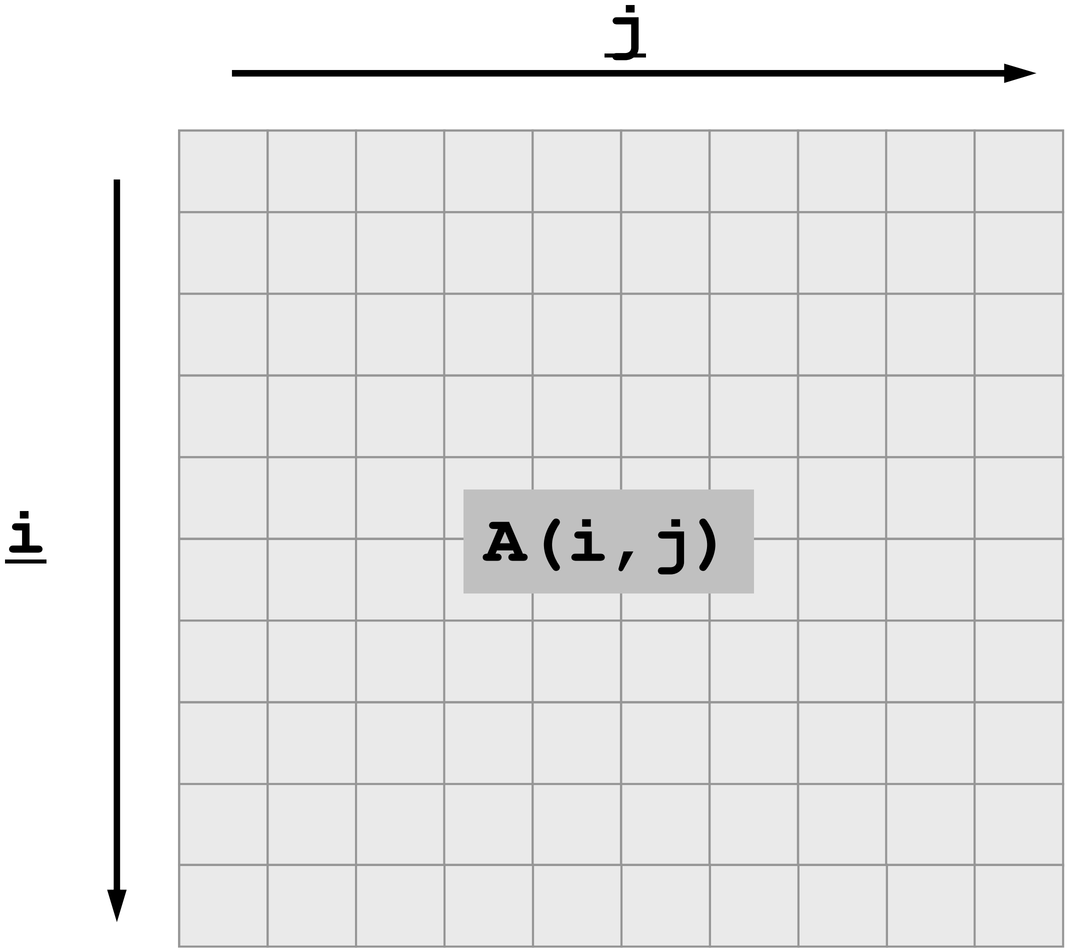
Multicores/Manycores
Multiple cores share memory/cache
-> Saturation of Memory





How to get opt. performance by Tuning ?

- = Optimization of memory access



How to get opt. performance by Tuning ? (cont.)

- Vector Processors
  - Long loops
- Scalar Processors
  - Utilization of cache, small chunks of data
- Common Issues
  - Continuous memory access
  - Localization
  - Changing sequence of computations might provide change of results.

- What is “tuning/optimization” ?
- Vector/Scalar Processors
- **Example: Scalar Processors**

Typical Methods of Tuning for Scalar Processors

- Loop Unrolling
  - loop overhead
  - loading/storing

- Blocking
  - Cache miss

BLAS: Basic Linear Algebra Subprograms

- Library API for fundamental operations of vectors and (dense) matrices)

- Level 1: Vectors: dot products, DAXPY
- Level 2: Matrix x Vector
- Level 3: Matrix x Matrix

- LINPACK
  - DGEMM: Level 3 BLAS

Scalar

Loop Unrolling

reduction of loading/storing (1/4)

mpiifort -O3 -xCORE-AVX2 -align array32byte t2.f

<$O-omp/dense>/t2.f

- Ratio of computation increases

```
N= 10000

do j= 1, N
  do i= 1, N
    A(i)= A(i) + B(i)*C(i,j)
  enddo
enddo

do j= 1, N-1, 2
  do i= 1, N
    A(i)= A(i) + B(i)*C(i,j)
    A(i)= A(i) + B(i)*C(i,j+1)
  enddo
enddo

do j= 1, N-3, 4
  do i= 1, N
    A(i)= A(i) + B(i)*C(i,j)
    A(i)= A(i) + B(i)*C(i,j+1)
    A(i)= A(i) + B(i)*C(i,j+2)
    A(i)= A(i) + B(i)*C(i,j+3)
  enddo
enddo
```

```
$> cdw
$> cd multicore/omp/dense
$> mpiifort -O3 -xCORE-AVX2 -align array32byte t2.f

<modify "go1.sh">
$> qsub go1.sh (1 process)

    1.029940E-01
    9.935689E-02
    9.015894E-02
```

go1.sh for a single core

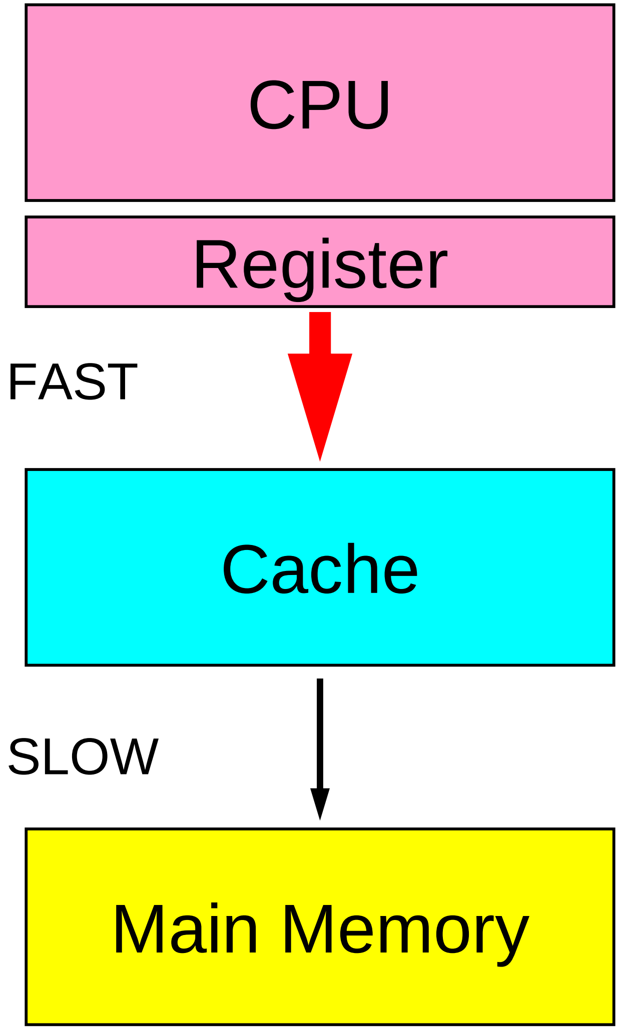
```
#PBS -q u-lecture7
#PBS -N tuning
#PBS -l select=1:mpiprocs=1
#PBS -Wgroup_list=gt27
#PBS -l walltime=00:05:00
#PBS -e err
#PBS -o test.lst

cd $PBS_O_WORKDIR
. /etc/profile.d/modules.sh

mpirun ./impimap.sh ./a.out
```

Scalar

Loop Unrolling

reduction of loading/storing (2/4)

- Load : Memory-Cache-Register
- Store: Register-Cache-Memory
- Fewer loading/storing, better performance



Scalar

Loop Unrolling

reduction of loading/storing (3/4)

```
do j= 1, N
  do i= 1, N
    A(i)= A(i) + B(i)*C(i,j)
   Store   Load   Load Load
  enddo
enddo
```

- Loading/Storing for A(i), B(i), C(i,j) occurs in each loop.
- 1*S, 3*L: 2*C

Loop Unrolling

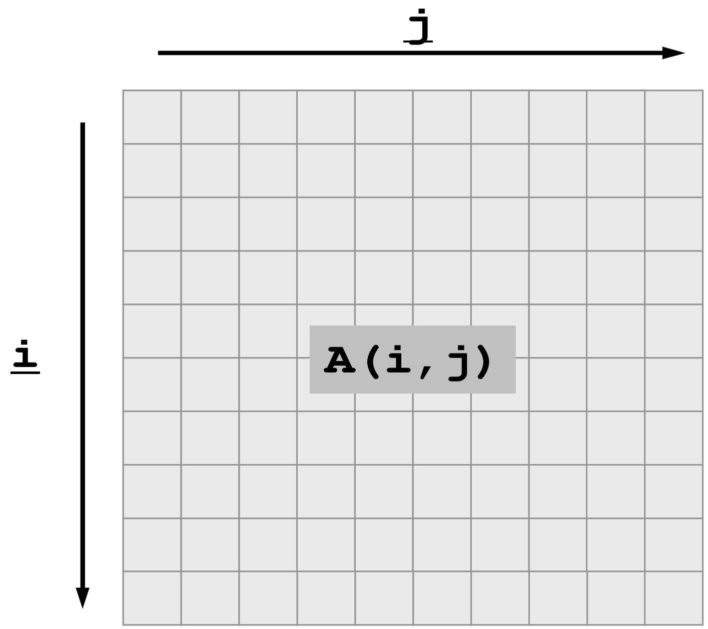
reduction of loading/storing (4/4)

```
do j= 1, N-3, 4
  do i= 1, N
    A(i)= A(i) + B(i)*C(i,j)
          Load   Load Load
    A(i)= A(i) + B(i)*C(i,j+1) Load
    A(i)= A(i) + B(i)*C(i,j+2) Load
    A(i)= A(i) + B(i)*C(i,j+3) Load
    Store
  enddo
enddo
```

- Values of arrays are kept on register during each loop. Storing occurs only at the end of the loop.
- Ratio of memory access (loading/storing) to computation can be reduced (1*S, 6*L: 8*C)
- Be careful about sequence of computations.

Loop Exchanging (1/2)

```
TYPE-A
do i= 1, N
  do j= 1, N
    A(i,j)= A(i,j) + B(i,j)
  enddo
enddo

TYPE-B
do j= 1, N
  do i= 1, N
    A(i,j)= A(i,j) + B(i,j)
  enddo
enddo
```



- In Fortran, component of A(i,j) is aligned in the following way: A(1,1), A(2,1), A(3,1),…, A(N,1), A(1,2), A(2,2),…, A(1,N), A(2,N),…, A(N,N)
  - In C: A[0][0], A[0][1], A[0][2], …, A[N-1][0], A[N-1][1],…,A[N-1][N-1]
- Access must be according to this alignment for higher performance.

Scalar

Loop Exchanging (2/2)

mpiifort -O3 -xCORE-AVX2 -align array32byte 2d-1.f

<$O-omp>/dense/2d-1.f

```
TYPE-A
do i= 1, N
  do j= 1, N
    A(i,j)= A(i,j) + B(i,j)
  enddo
enddo

TYPE-B
do j= 1, N
  do i= 1, N
    A(i,j)= A(i,j) + B(i,j)
  enddo
enddo
```

```
TYPE-A
for (j=0; j<N; j++){
  for (i=0; i<N; i++){
    A[i][j]= A[i][j] + B[i][j];
  }
}

TYPE-B
for (i=0; i<N; i++){
  for (j=0; j<N; j++){
    A[i][j]= A[i][j] + B[i][j];
  }
}
```

```
$> cdw
$> cd multicore/omp/dense
$> mpiifort …
$> qsub go1.sh
### N ###     500
WORSE     2.307892E-04
BETTER    1.819134E-04
### N ###    1000
WORSE     1.492023E-03
BETTER    7.221699E-04
### N ###    1500
WORSE     5.147934E-03
BETTER    1.756191E-03
### N ###    2000
WORSE     1.727295E-02
BETTER    5.671978E-03
### N ###    2500
WORSE     3.727102E-02
BETTER    1.051402E-02
### N ###    3000
WORSE     7.017899E-02
BETTER    1.530194E-02
### N ###    3500
WORSE     8.954906E-02
BETTER    2.009606E-02
### N ###    4000
WORSE     1.272562E-01
BETTER    2.680802E-02
```

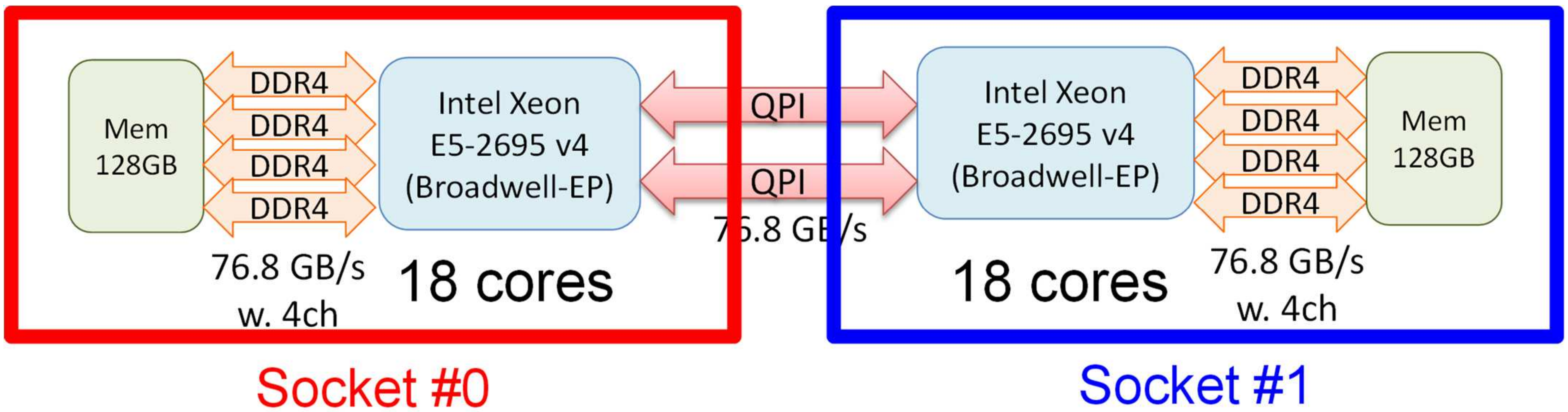
Blocking for Cache Miss (1/7)

```
do i= 1, NN
  do j= 1, NN
    A(j,i)= A(j,i) + B(i,j)
  enddo
enddo
```

- Consider this situation.

Intel Xeon E5-2695 v4 (Broadwell-EP)

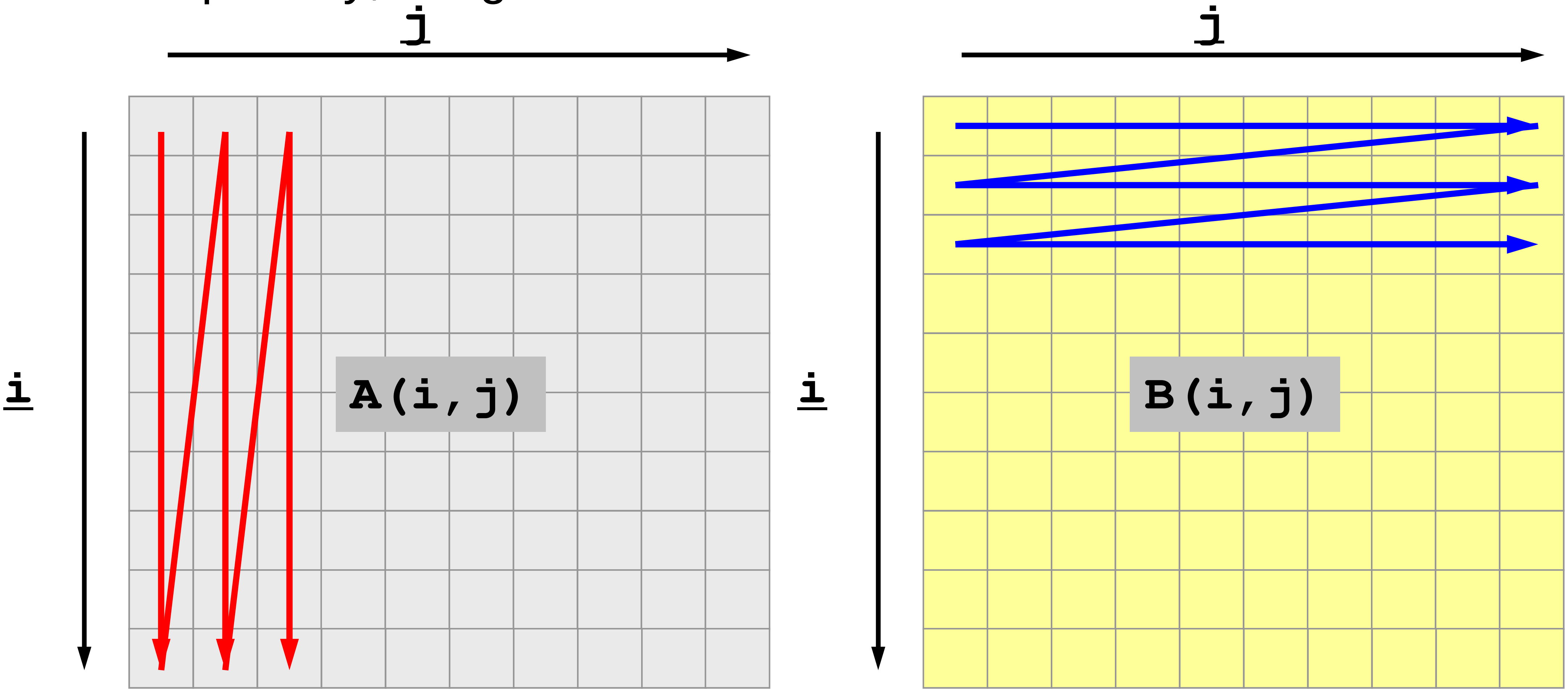
	Size	X-way set associative	Cache Line
L1 Data	32 KB/core	8-way	64 B
L1 Instruction	32 KB/core	8-way	64 B
L2	256 KB/core	8-way	64 B
L3	45 MB/socket	20-way	64 B



Scalar

Blocking for Cache Miss (2/7)

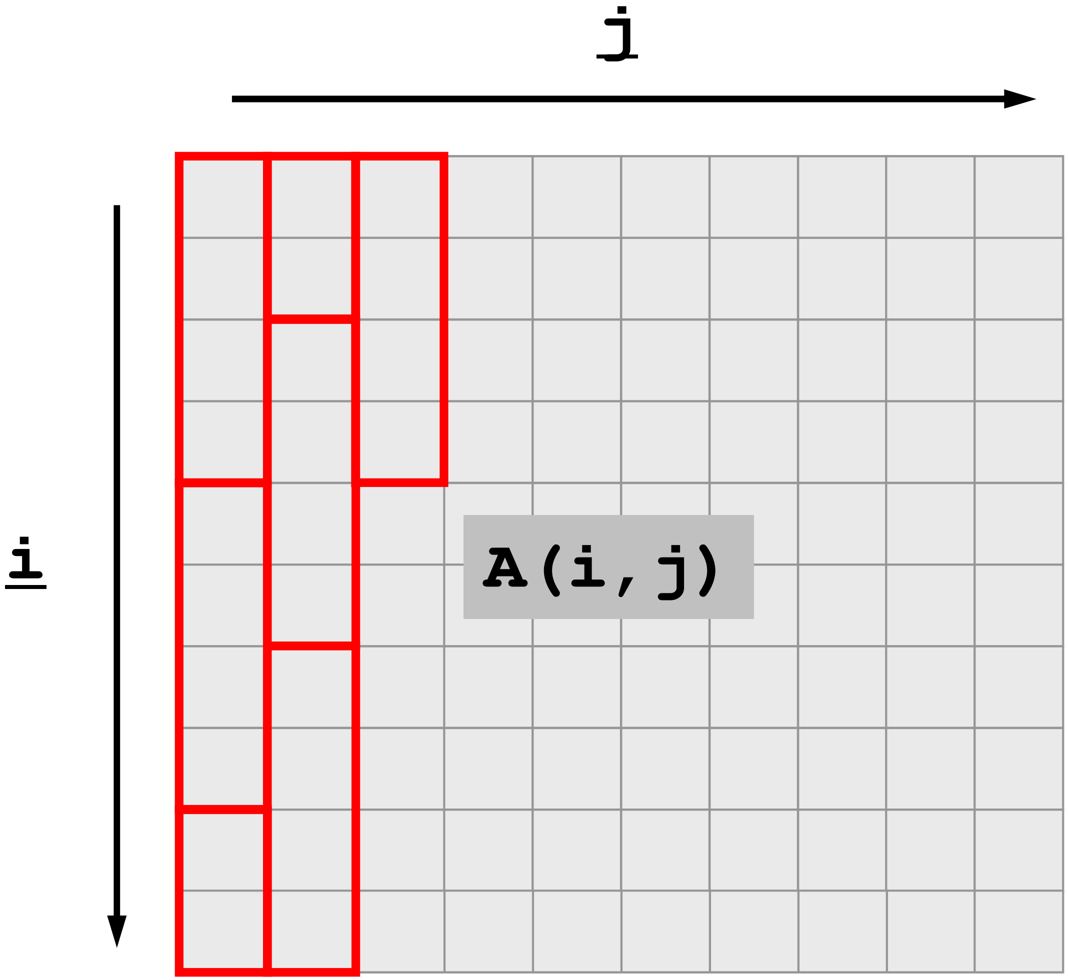
- Direction of optimum memory access for “A” is different from that of “B”. Especially, not good for “B”.



Scalar

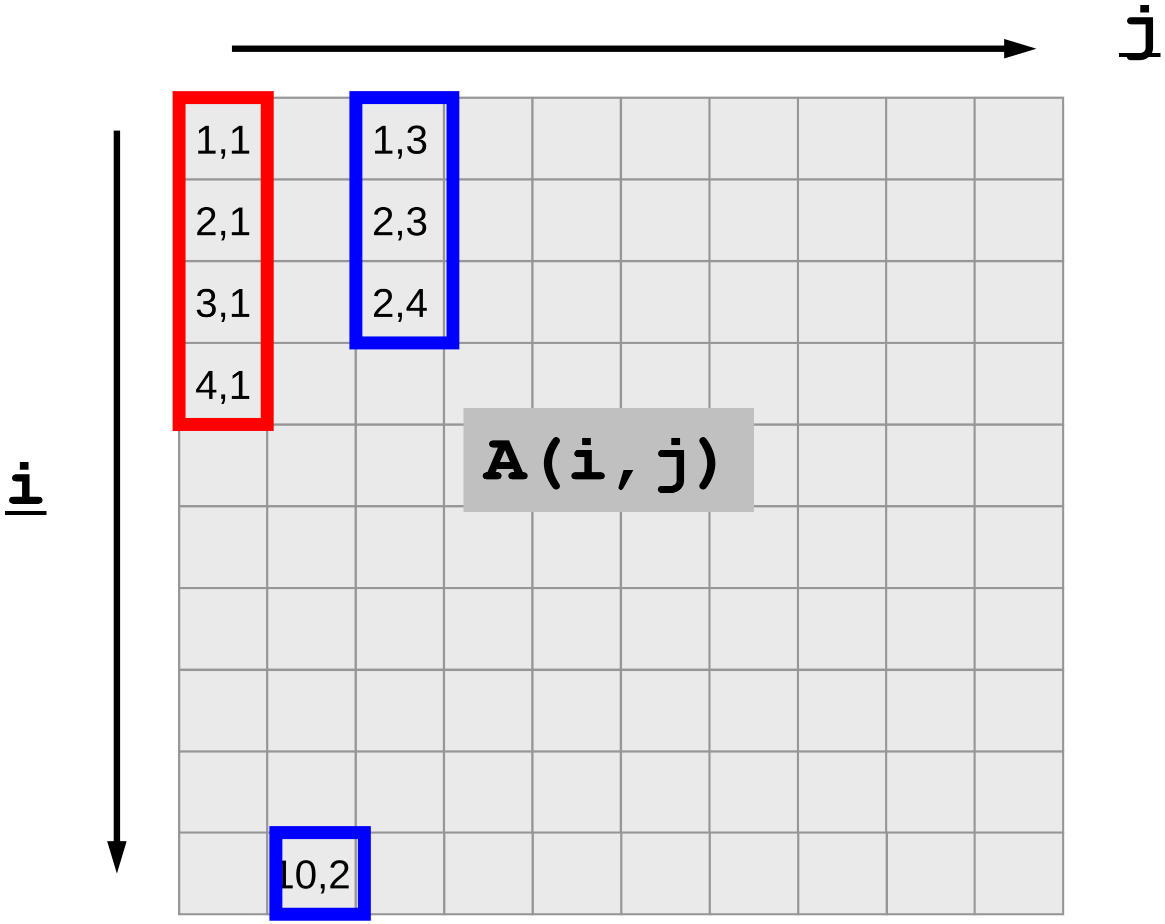
Blocking for Cache Miss (3/7)

- If the size of cache-line is 4-word, data on array is sent to cache from main memory in the following way:



Blocking for Cache Miss (4/7)

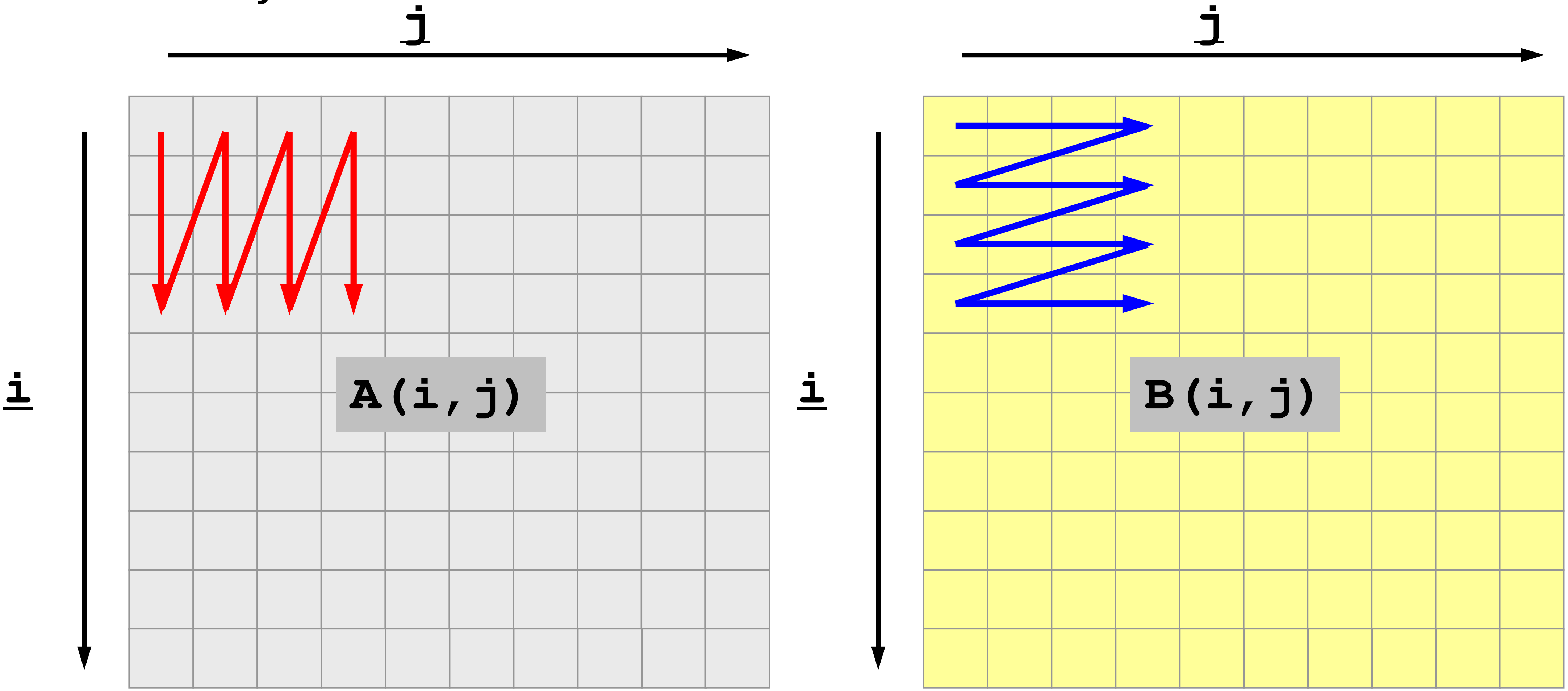
- Therefore, if A(1,1) is touched, A(1,1), A(2,1), A(3,1), A(4,1) are on cache. If A(10,2) is touched A(10,2), A(1,3), A(2,3), A(3,3) are on cache.



Scalar

Blocking for Cache Miss (5/7)

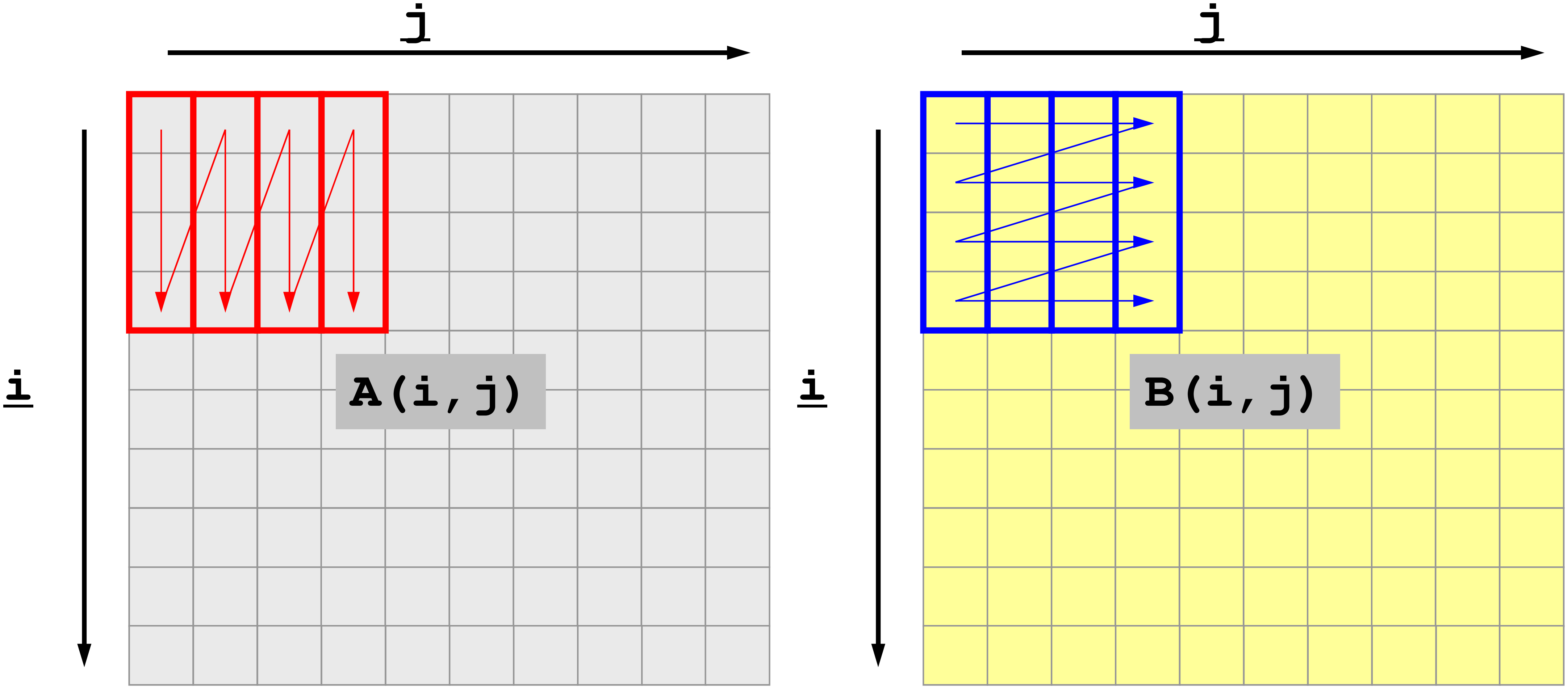
- Therefore, following block-wise access pattern utilizes cache efficiently.



Scalar

Blocking for Cache Miss (6/7)

- □, □ are on cache.



Blocking for Cache Miss (7/7)

mpiifort -O3 -xCORE-AVX2 -align array32byte 2d-2.f

- 2x2 block

<$O-omp>/dense/2d-2.f

```
do i= 1, NN
  do j= 1, NN
    A(j,i)= A(j,i) + B(i,j)
  enddo
enddo


do i= 1, NN-1, 2
  do j= 1, NN-1, 2
    A(j  ,i  )= A(j  ,i  ) + B(i  ,j  )
    A(j+1,i  )= A(j+1,i  ) + B(i  ,j+1)
    A(j  ,i+1)= A(j  ,i+1) + B(i+1,j  )
    A(j+1,i+1)= A(j+1,i+1) + B(i+1,j+1)
  enddo
enddo

do i= 1, NN-1, 2
  do j= 1, NN/2, 2
    A(j  ,i  )= A(j  ,i  ) + B(i  ,j  )
    A(j+1,i  )= A(j+1,i  ) + B(i  ,j+1)
    A(j  ,i+1)= A(j  ,i+1) + B(i+1,j  )
    A(j+1,i+1)= A(j+1,i+1) + B(i+1,j+1)
  enddo
enddo

do i= 1, NN-1, 2
  do j= NN/2+1, NN-1, 2
    A(j  ,i  )= A(j  ,i  ) + B(i  ,j  )
    A(j+1,i  )= A(j+1,i  ) + B(i  ,j+1)
    A(j  ,i+1)= A(j  ,i+1) + B(i+1,j  )
    A(j+1,i+1)= A(j+1,i+1) + B(i+1,j+1)
  enddo
enddo
```

Loop-Fission is also effective for reduction of cache/TLB miss's.

```
$> cdw
$> cd multicore/omp/dense
$> mpiifort …
$> qsub go1.sh

### N ###      500
BASIC     3.120899E-04
2x2       2.901554E-04
2x2-b     2.548695E-04
### N ###     1000
BASIC     1.405001E-03
2x2       1.138926E-03
2x2-b     1.201868E-03

...

### N ###     4000
BASIC     1.250241E-01
2x2       7.714486E-02
2x2-b     6.398797E-02
### N ###     4500
BASIC     1.862700E-01
2x2       1.056359E-01
2x2-b     8.264184E-02
### N ###     5000
BASIC     2.493391E-01
2x2       1.350009E-01
2x2-b     1.066360E-01
```

Summary: Tuning

- Scalar Processor
- Dense Matrices: BLAS

- Optimization of operations for sparse matrices (which appear in this class) is much more difficult (still research topics)
  - Basic idea is same as that for dense matrices.
  - Optimum memory access.

Scalar

Sparse/Dense Matrices

```
do i= 1, N
  Y(i)= D(i)*X(i)
  do k= index(i-1)+1, index(i)
    Y(i)= Y(i) + AMAT(k)*X(item(k))
  enddo
enddo
```

```
do j= 1, N
  do i= 1, N
    Y(j)= Y(j) + A(i,j)*X(i)
  enddo
enddo
```

- "X" in RHS
  - Dense: continuous on memory, easy to utilize cache
  - Sparse: continuity is not assured, difficult to utilize cache
    - more "memory-bound"
- Effective method for sparse matrices: Blocking
  - Reordering: provides "block" features
  - Utilizing physical features of matrices: multiple DOF on each element/node.

Scalar

Summary: Tuning (cont.)

- Sparse Matrices
  - Strategy for tuning may depend on alignment of data components.
  - Program may change according to alignment of data components.
- Dense Matrices
  - Structured, Regularity
  - Performance mainly depends on machine parameters & mat. size.
    - Effect of options of compilers
    - Automatic tuning (AT) is applicable.
  - Libraries
    - ATLAS (Automatic Tuning)
      - http://math-atlas.sourceforge.net/
    - GoToBLAS (Manual Tuning)
      - Kazushige Goto (Microsoft)