Performance Tuning for High Performance Computing Applications

Daisuke Takahashi

University of Tsukuba, Japan

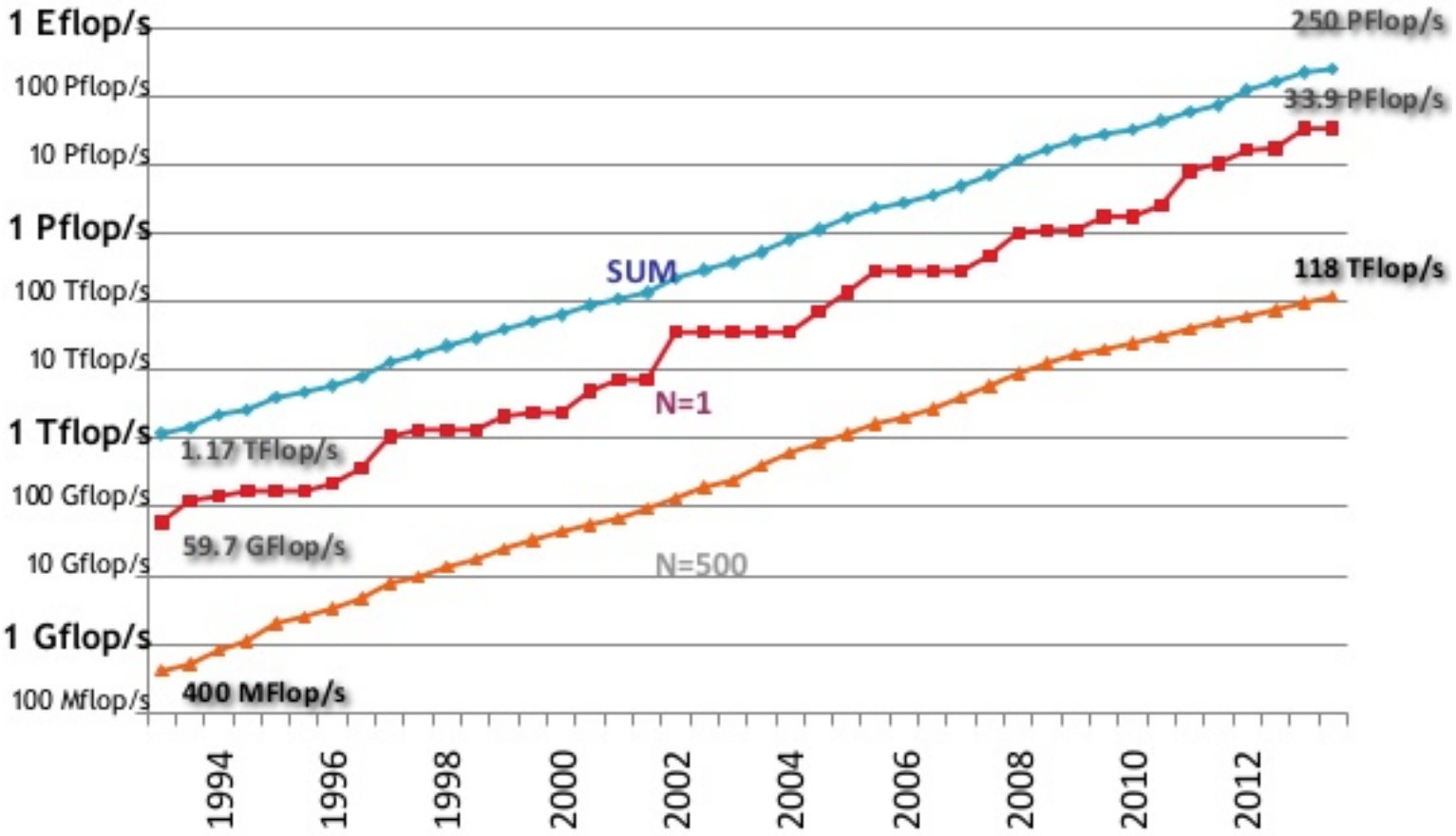
Outline

- Performance development of supercomputers
- HPC Challenge (HPCC) Benchmark Suite
- It’s all bandwidth
- Performance tuning
  - What is performance tuning?
  - Program optimization methods

Performance Development of Supercomputers

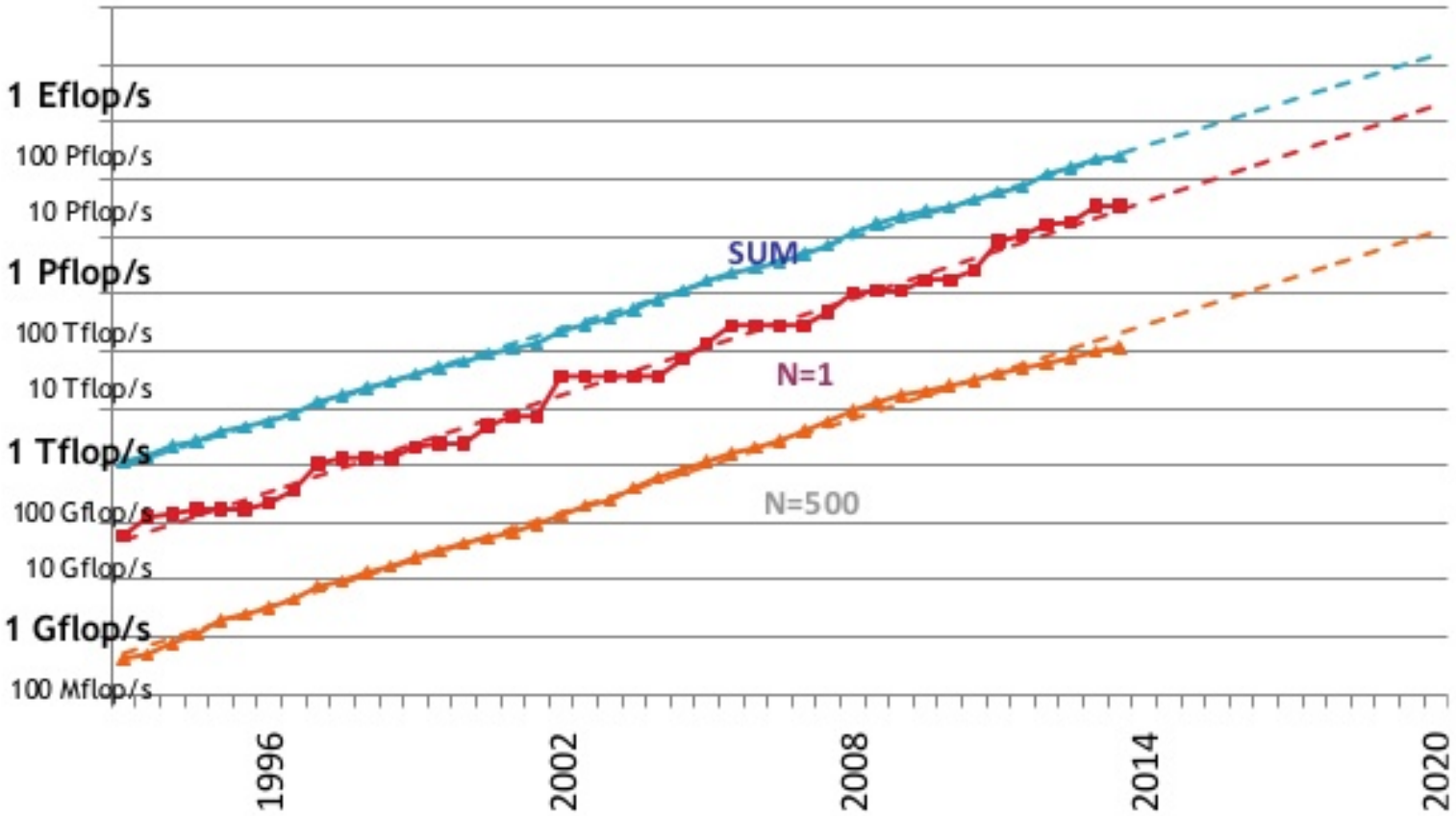
- November 2013 TOP500 Supercomputing Sites
  - Tianhe-2 (Intel Xeon E5-2692 12C 2.2GHz, Intel Xeon Phi 31S1P) : 33.862 PFlops (3,120,000 Cores)
  - Titan (Cray XK7, Opteron 6274 16C 2.2GHz, NVIDIA K20x) : 17.590 PFlops (560,640 Cores)
  - Sequoia (BlueGene/Q, Power BQC 16C 1.6GHz) : 17.173 PFlops (1,572,864 Cores)
  - K computer (SPARC64 VIIIfx 2.0GHz) : 10.510 PFlops (705,024 Cores)
- Recently, the number of cores keeps increasing.

Performance Development



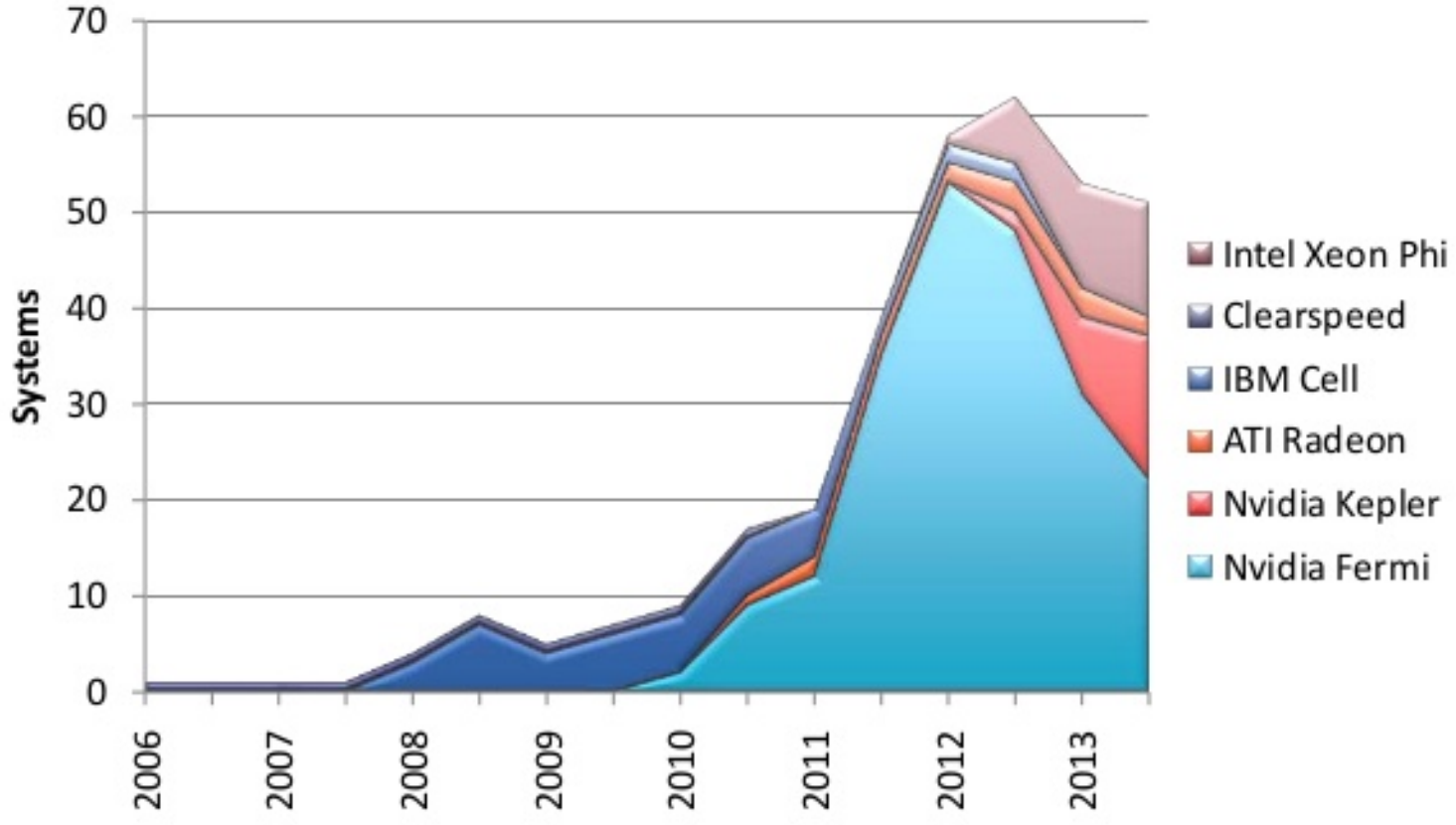
Source:http://www.slideshare.net/top500/top500-november-2013

Projected Performance Development



Source:http://www.slideshare.net/top500/top500-november-2013

Accelerators



TOP500 SUPERCOMPUTER SITES

Source:http://www.slideshare.net/top500/top500-november-2013

Linpack Benchmark

- Developed by Jack Dongarra of the University of Tennessee.
- Benchmark test for evaluating floating-point processing performance
- Uses Gaussian elimination method to estimate the time required for solving simultaneous linear equations
- Also used for the “TOP500 Supercomputer” benchmark

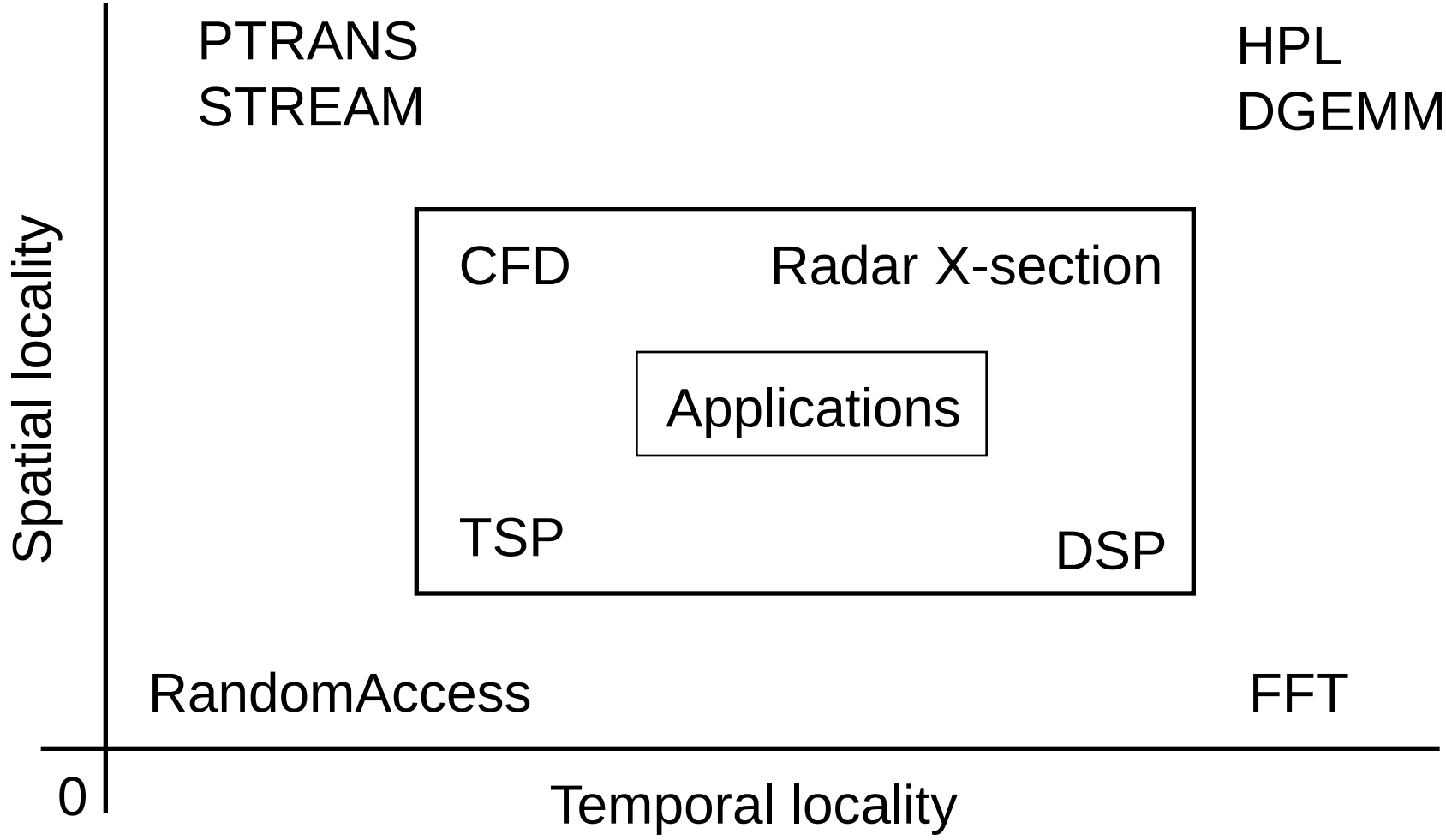
Overview of the HPC Challenge (HPCC) Benchmark Suite

- HPC Challenge (HPCC) is a suite of tests that examine the performance of HPC architectures using kernels.
- The suite provides benchmarks that bound the performance of many real applications as a function of memory access characteristics, e.g.,
  - Spatial locality
  - Temporal locality

The Benchmark Tests

- The HPC Challenge benchmark consists at this time of 7 performance tests:
  - HPL (High Performance Linpack)
  - DGEMM (matrix-matrix multiplication)
  - STREAM (sustainable memory bandwidth)
  - PTRANS (A=A+B^T, parallel matrix transpose)
  - RandomAccess (integer updates to random memory locations)
  - FFT (complex 1-D discrete Fourier transform)
  - b_eff (MPI latency/bandwidth test)

Targeted Application Areas in the Memory Access Locality Space



HPCC Testing Scenarios

- Local (S-STREAM, S-RandomAccess, S-DGEMM, S-FFT)
  - Only single MPI process computes.
- Embarrassingly parallel (EP-STREAM, EP-RandomAccess, EP-DGEMM, EP-FFT)
  - All processes compute and do not communicate (explicitly).
- Global (G-HPL, G-PTRANS, G-RandomAccess, G-FFT)
  - All processes compute and communicate.
- Network only (RandomRing Bandwidth, etc.)

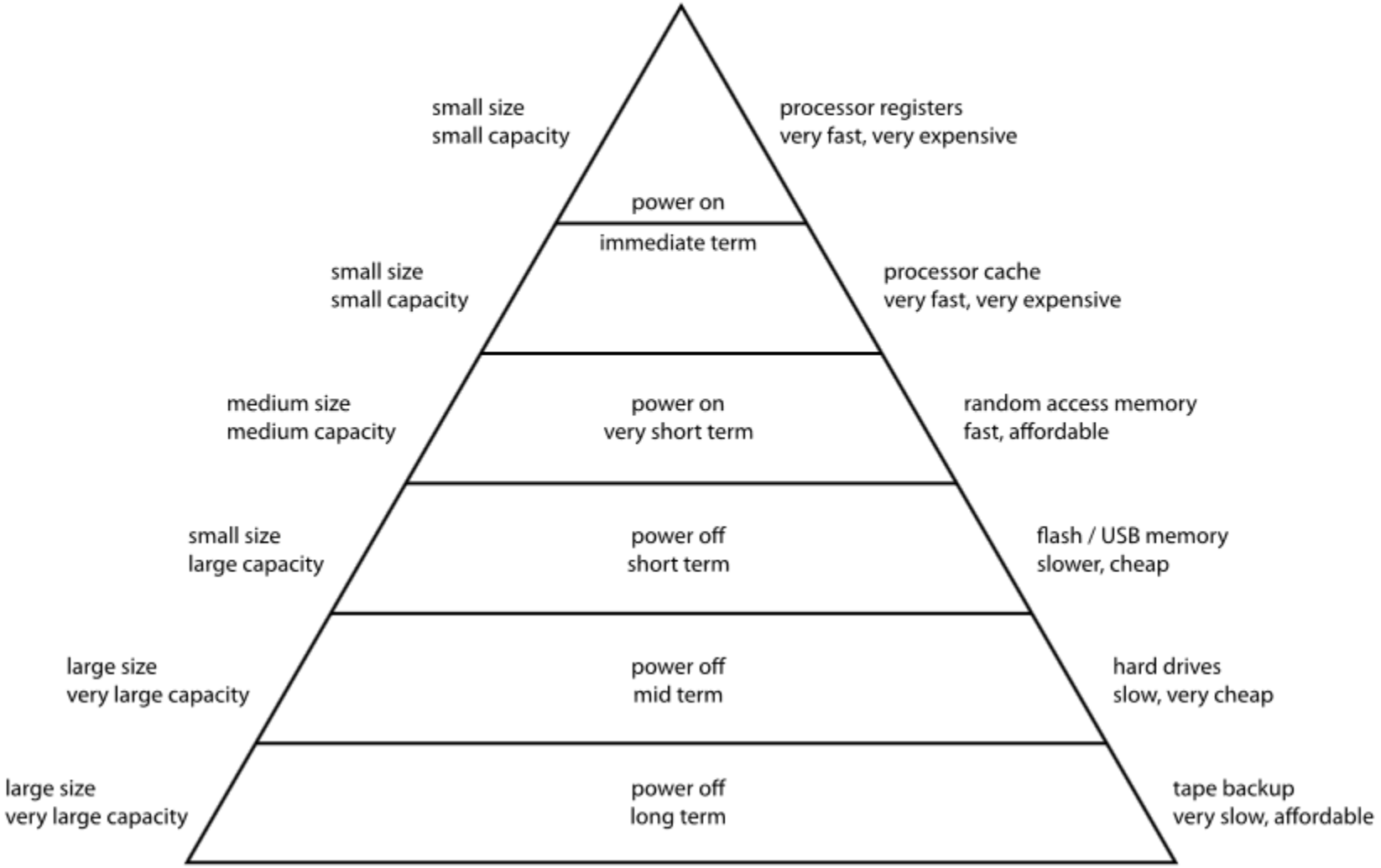
The winners of the 2013 HPC Challenge Class 1 Awards

- G-HPL: 9,796 TFlops
  - K computer (663,552 cores)
- G-RandomAccess: 2,021 GUPS
  - IBM Power 775 (63,648 cores)
- G-FFT: 205.9 TFlops
  - K computer (663,552 cores)
- EP-STREAM-Triad (system): 3,857 TB/s
  - K computer (663,552 cores)

Indicator of Capability for Supplying Data to the Processor

- In a computer system that performs scientific computations, the “capability for supplying data to the processor” is most important.
- Unless data is supplied to the arithmetic unit of the processor, computations cannot be performed.
- The computing performance of the processor is largely impacted by the data supply capacity.
- “Bandwidth” is used as an indicator of the data supply capability.

Computer Memory Hierarchy



Source: Wikipedia

Memory Hierarchy (1/2)

- Memory hierarchy is designed based on the assumed locality of patterns of access to the memory area.
- Different types of locality:
  - Temporal locality
    - Property whereby the accessing of a certain address reoccurs within a relatively short time interval
  - Spatial locality
    - Property whereby data accessed within a certain time interval is distributed among relatively nearby addresses

Memory Hierarchy (2/2)

- These tendencies often apply to business computations and other non-numeric computations, but are not generally applicable to numeric computation programs.
- Especially in large-scale scientific computations, there is often no temporal locality for data references.
- This is a major reason why vector-type supercomputers are advantageous for scientific computations.

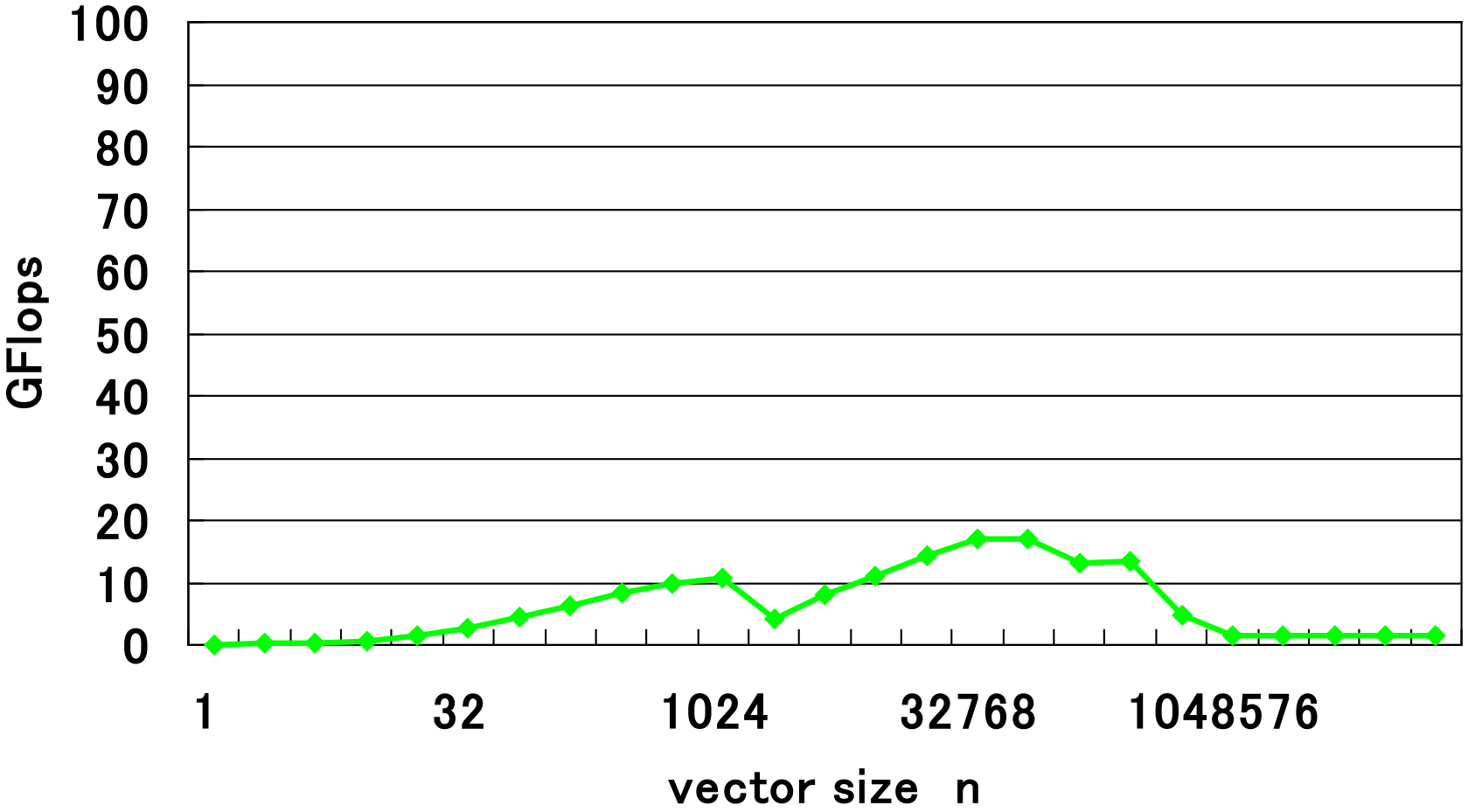
Concept of Byte/Flop

- The amount of memory access needed when performing a single floating-point operation is defined in byte/flop.

```
void daxpy(int n, double a, double *x, double *y)
{
    int i;
    for (i = 0; i < n; i++)
        y[i] += a * x[i];
}
```

- With daxpy, double-precision real-number data must be loaded/stored three times (24 bytes total) in order to perform two double-precision floating-point operations per single iteration.
  - In this case, 24Byte/2Flop = 12Byte/Flop.
- The smaller the Byte/Flop value is better.

Performance of DAXPY (Intel Xeon E3-1230 3.2GHz 8MB L3 cache, Intel MKL 10.3)



PC and Vector-type Supercomputer Memory Bandwidth

- Intel Xeon E5-2697 v2 (Ivy Bridge-EP 2.7GHz, 4 x DDR3-1866, 2 sockets/node)
  - The theoretical peak performance of each node is 21.6GFlops × 12 cores × 2 sockets=518.4GFlops
  - Memory bandwidth up to 119.4GB/s
  - Byte/Flop value is 119.4/518.4 ≒ 0.23
- NEC SX-ACE (4 cores/node)
  - The theoretical peak performance of each node is 69GFlops × 4 cores=276GFlops
  - Memory bandwidth up to 256GB/s
  - Byte/Flop value is 256/276 ≒ 0.93

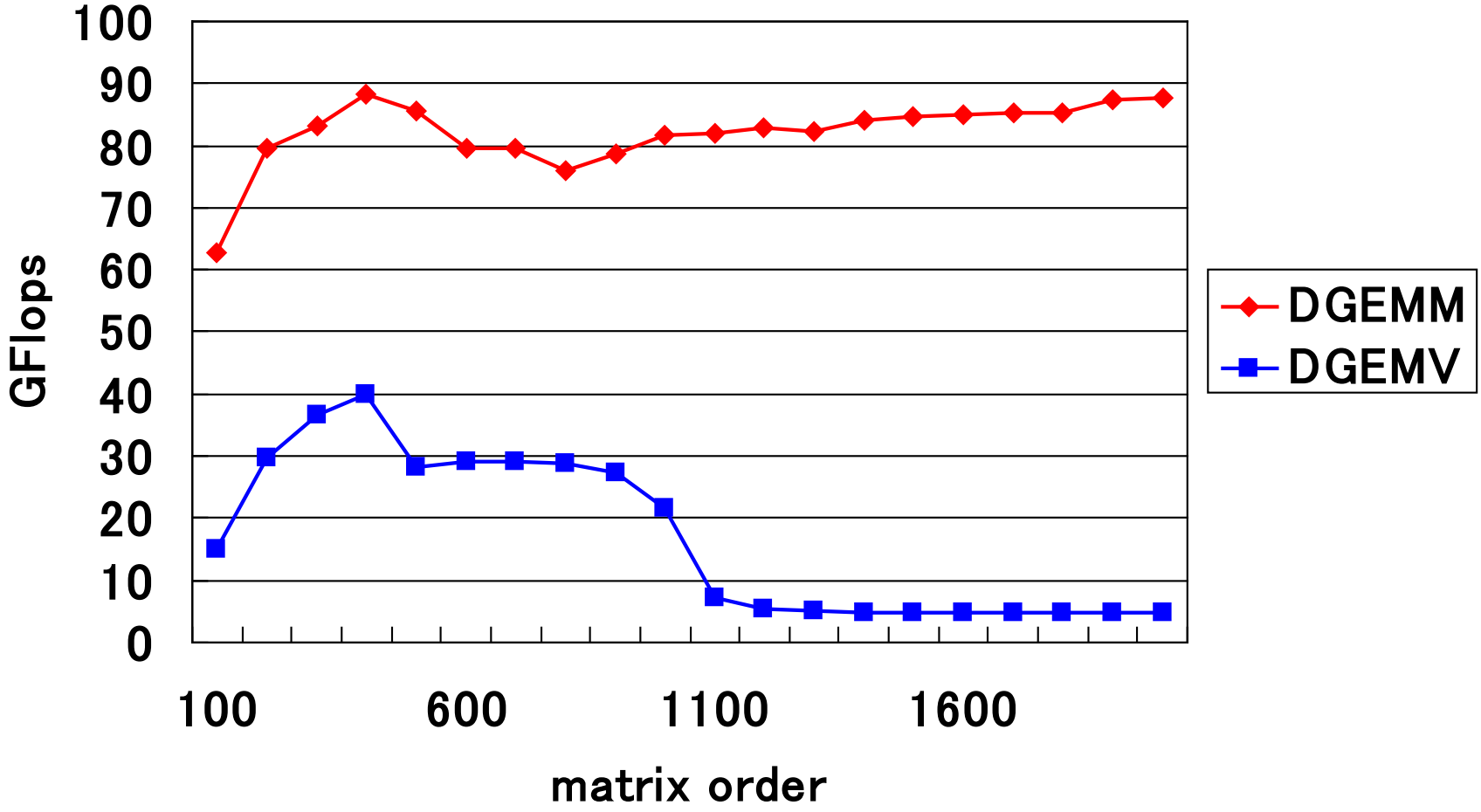
Comparison of Theoretical Performance in DAXPY

- Intel Xeon E5-2697 v2
  - Theoretical peak performance of each node: 518.4GFlops
  - In the case where the working set exceeds the cache capacity, the memory bandwidth (119.4GB/s) is rate-limiting and so the limit is (119.4GB/s)/(12Byte/Flop)≒9.95GFlops
  - Only approximately 1.9% of theoretical peak performance!
- NEC SX-ACE
  - Theoretical peak performance of each node: 276GFlops
  - The memory bandwidth (256GB/s) is rate-limiting, and so the limit is (256GB/s)/(12Byte/Flop)≒21.3GFlops
  - Approximately 7.7% of theoretical peak performance.

Arithmetic Operations in BLAS

BLAS	Loads + Stores	Operations	Ratio $n = m = k$
Level 1 DAXPY $y = y + \alpha x$	$3n$	$2n$	$3:2$
Level 2 DGEMV $y = \beta y + \alpha Ax$	$mn + n + 2m$	$2mn$	$1:2$
Level 3 DGEMM $C = \beta C + \alpha AB$	$2mn + mk + kn$	$2mnk$	$2:n$

Performances of DGEMV and DGEMM (Intel Xeon E3-1230 3.2GHz 8MB L3 cache, Intel MKL 10.3)



Significance of Performance Tuning

- In the case of calculations whose runtime lasts for several months or longer, optimization may result in a reduction of runtime on the order of a month.
- As in the case of numeric libraries, if a program is used by many people, tuning will have sufficient value.
- If tuning results in a 30% improvement in performance, for example, the net result is the same as using a machine having 30% higher performance.

Optimization Policy

- If available, use a vendor-supplied high-speed library as much as possible.
  - BLAS, LAPACK, etc.
- The optimization capability of recent compilers is extremely high.
- Optimization that can be performed by the compiler must not be performed on the user side.
  - Requires extra effort
  - Results in a program that is complicated and may contain bugs
- Overestimates the optimizing capability of compilers
  - Humans are dedicated to improving algorithms.
  - Unless otherwise unavoidable, do not use an assembler.

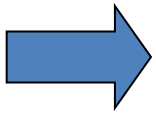
Optimization Information of Fortran Compiler on K computer

```
(line-no.)(nest)(optimize)
      80                !$OMP DO
      81    1   p             DO 70 II=1,NX,NBLK
      82    2   p               DO 30 JJ=1,NY,NBLK
                        <<< Loop-information Start >>>
                        <<<  [OPTIMIZATION]
                        <<<    PREFETCH       : 2
                        <<<      A: 2
                        <<< Loop-information  End >>>
      83    3   p                 DO 20 I=II,MIN0(II+NBLK-1,NX)
      84    3                !OCL SIMD(ALIGNED)
                        <<< Loop-information Start >>>
                        <<<  [OPTIMIZATION]
                        <<<    SIMD
                        <<<    SOFTWARE PIPELINING
                        <<< Loop-information  End >>>
      85    4   p   8v              DO 10 J=JJ,MIN0(JJ+NBLK-1,NY)
      86    4   p   8v                B(J,I-II+1)=A(I,J)
      87    4   p   8v       10       CONTINUE
```

Loop Unrolling (1/2)

- Loop unrolling expands a loop in order to do the following:
  - Reduce loop overhead
  - Perform register blocking
- If expanded too much, register shortages or instruction cache misses may occur, and so care is needed.

```
double A[N], B[N], C;
for (i = 0; i < N; i++) {
  A[i] += B[i] * C;
}
```

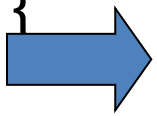


```
double A[N], B[N], C;
for (i = 0; i < N; i += 4) {
  A[i] += B[i] * C;
  A[i+1] += B[i+1] * C;
  A[i+2] += B[i+2] * C;
  A[i+3] += B[i+3] * C;
}
```

Loop Unrolling (2/2)

```
double A[N][N], B[N][N],
          C[N][N], s;
for (j = 0; j < N; k++) {
   for (i = 0; i < N; j++) {
      s = 0.0;
      for (k = 0; k < N; k++) {
         s += A[i][k] * B[j][k];
      }
      C[j][i] = s;
   }
}
```

Matrix multiplication



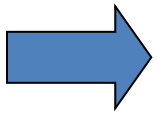
```
double A[N][N], B[N][N],
          C[N][N], s0, s1;
for (j = 0; j < N; j += 2)
   for (i = 0; i < N; i++) {
      s0 = 0.0;  s1 = 0.0;
      for (k = 0; k < N; k++) {
         s0 += A[j][k] * B[j][k];
         s1 += A[j+1][k] * B[j+1][k];
      }
      C[j][i] = s0;
      C[j+1][i] = s1;
   }
```

Optimized matrix multiplication

Loop Interchange

- Loop interchange is a technique mainly for reducing the adverse effects of large-stride memory accesses.
- In some cases, the compiler judges the necessity and performs loop interchanges.

```
double A[N][N], B[N][N], C;
for (j = 0; j < N; j++) {
  for (k = 0; k < N; k++) {
    A[k][j] += B[k][j] * C;
  }
}
```



```
double A[N][N], B[N][N], C;
for (k = 0; k < N; k++) {
  for (j = 0; j < N; j++) {
    A[k][j] += B[k][j] * C;
  }
}
```

Before loop interchange

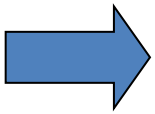
After loop interchange

Padding

- Effective in cases where multiple arrays have been mapped to the same cache location and thrashing occurs
  - Especially in the case of an array having a size that is a power of two
- It is recommended to change the defined sizes of two-dimensional arrays.
- In some instances, this can be handled by specifying the compile options.

```
double A[N][N], B[N][N];
for (k = 0; k< N; k++) {
  for (j = 0; j < N; j++) {
    A[j][k] = B[k][j];
  }
}
```

Before padding



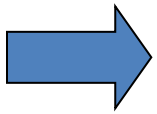
```
double A[N][N+1], B[N][N+1];
for (k = 0; k < N; k++) {
  for (j = 0; j < N; j++) {
      A[j][k] = B[k][j];
  }
}
```

After padding

Cache Blocking (1/2)

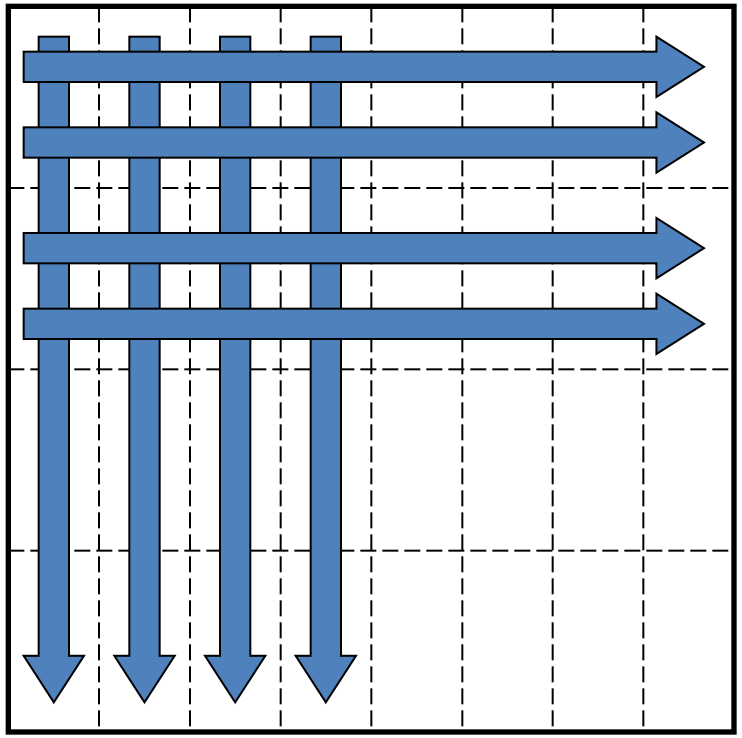
- Effective method for optimizing memory accesses
- Cache misses are reduced as much as possible.

```
double A[N][N], B[N][N], C;
for (i = 0; i < N; i++) {
  for (j = 0; j < N; j++) {
    A[i][j] += B[j][i] * C;
  }
}
```

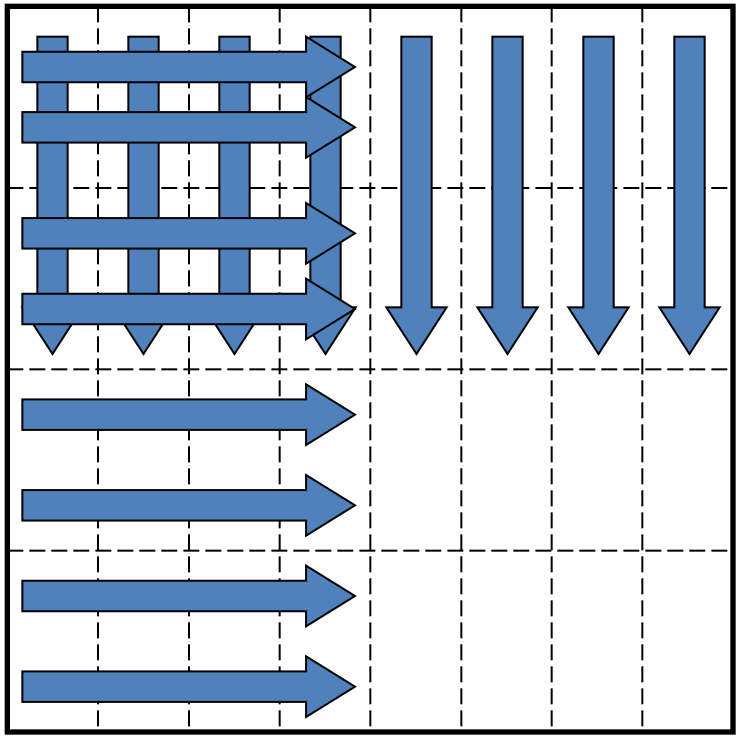


```
double A[N][N], B[N][N], C;
for (ii = 0; ii < N; ii += 4) {
  for (jj = 0; jj < N; jj += 4) {
    for (i = ii; i < ii + 4; i++) {
      for (j = jj; j < jj + 4; j++) {
        A[i][j] += B[j][i] * C;
      }
    }
  }
}
```

Cache Blocking (2/2)



Memory access pattern without blocking



Memory access pattern with blocking

Use of Fused Multiply-Add (FMA) Instructions

- Today, floating-point multiplication is as fast as floating-point addition on the latest processors.
- Moreover, many processors have a fused multiply-add instruction.

$$d = \pm a \pm b \times c$$

  where a, b, c and d are floating-point registers.
- An addition, a multiplication, or a fused multiply-add each requires one machine cycle on many processors that have fused multiply-add instructions.

Goedecker's Method for Fused Multiply-Add Instructions

- Goedecker's method consists of repeated transformations of the expression:

$$ax + by = a(x + (b/a))y$$

$$\text{where} \quad a \neq 0$$

- Applying repeated transformations of the above equation to a conventional radix-2 FFT, a radix-2 FFT algorithm suitable for fused multiply-add instructions can be obtained.

Conventional Radix-2 FFT Kernel

```
SUBROUTINE FFT(AR,AI,BR,BI,CS,SN,M,L)
 DIMENSION AR(M,2,*),AI(M,2,*),BR(M,L,*),BI(M,L,*),CS(M,*),SN(M,*)
 DO J=1,L
     WR=CS(1,J)
     WI=SN(1,J)
     DO I=1,M
        TR=WR*AR(I,2,J)-WI*AI(I,2,J)
        TI=WR*AI(I,2,J)+WI*AR(I,2,J)
        BR(I,J,1)=AR(I,1,J)+TR
        BI(I,J,1)=AI(I,1,J)+TI
        BR(I,J,2)=AR(I,1,J)-TR
        BI(I,J,2)=AI(I,1,J)-TI
     END DO
 END DO
 RETURN
 END
```

- 2 Multiplications
- 4 Additions
- 2 FMAs

In total, 8 cycles are needed.

Radix-2 FFT Kernel Suitable for Fused Multiply-Add Instructions

```
SUBROUTINE FFT_FMA(AR,AI,BR,BI,CS,SN,M,L)
DIMENSION AR(M,2,*),AI(M,2,*),BR(M,L,*),BI(M,L),CS(M,*),SN(M,*)
DO J=1,L
    WR=CS(1,J)
    WIWR=SN(1,J)/WR
    DO I=1,M
       TR=AR(I,2,J)-WIWR*AI(I,2,J)
       TI=AI(I,2,J)+WIWR*AR(I,2,J)
       BR(I,J,1)=AR(I,1,J)+TR*WR
       BI(I,J,1)=AI(I,1,J)+TI*WR
       BR(I,J,2)=AR(I,1,J)-TR*WR
       BI(I,J,2)=AI(I,1,J)-TI*WR
    END DO
END DO
RETURN
END
```

- 6 FMAs

In total, 6 cycles are needed.

Challenges

- To abstract the hardware configuration for application developers is much more important.
- Domain specific-language is one of the solutions.
  - SPIRAL, etc.
- The another way is developing numerical libraries to exploit the system performance.
- How to reduce the cost of performance tuning?
  - Automatic tuning
  - Automated code generation