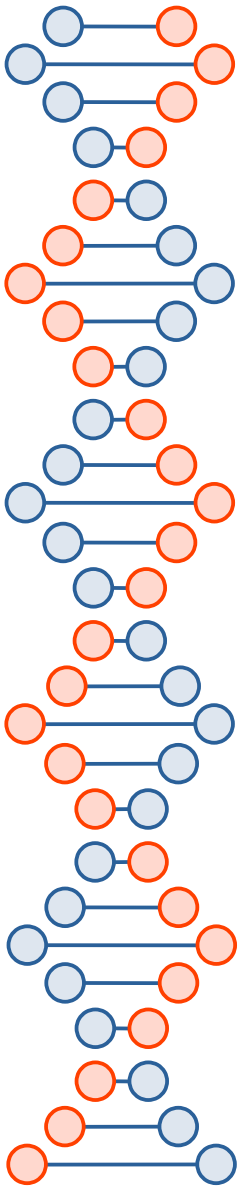
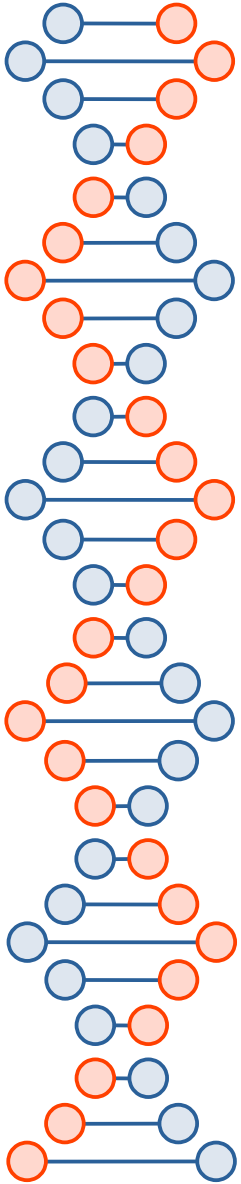


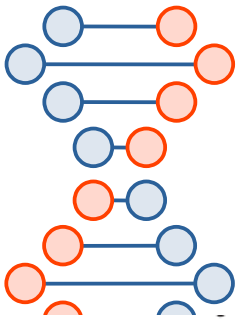
High Performance Computing

Ch. 3 Data Access Optimization

Computer Eng., KMITL
Assoc. Prof. Dr. Surin. K.

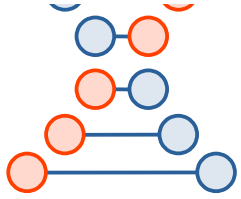


Ch.3 Data Access Optimization

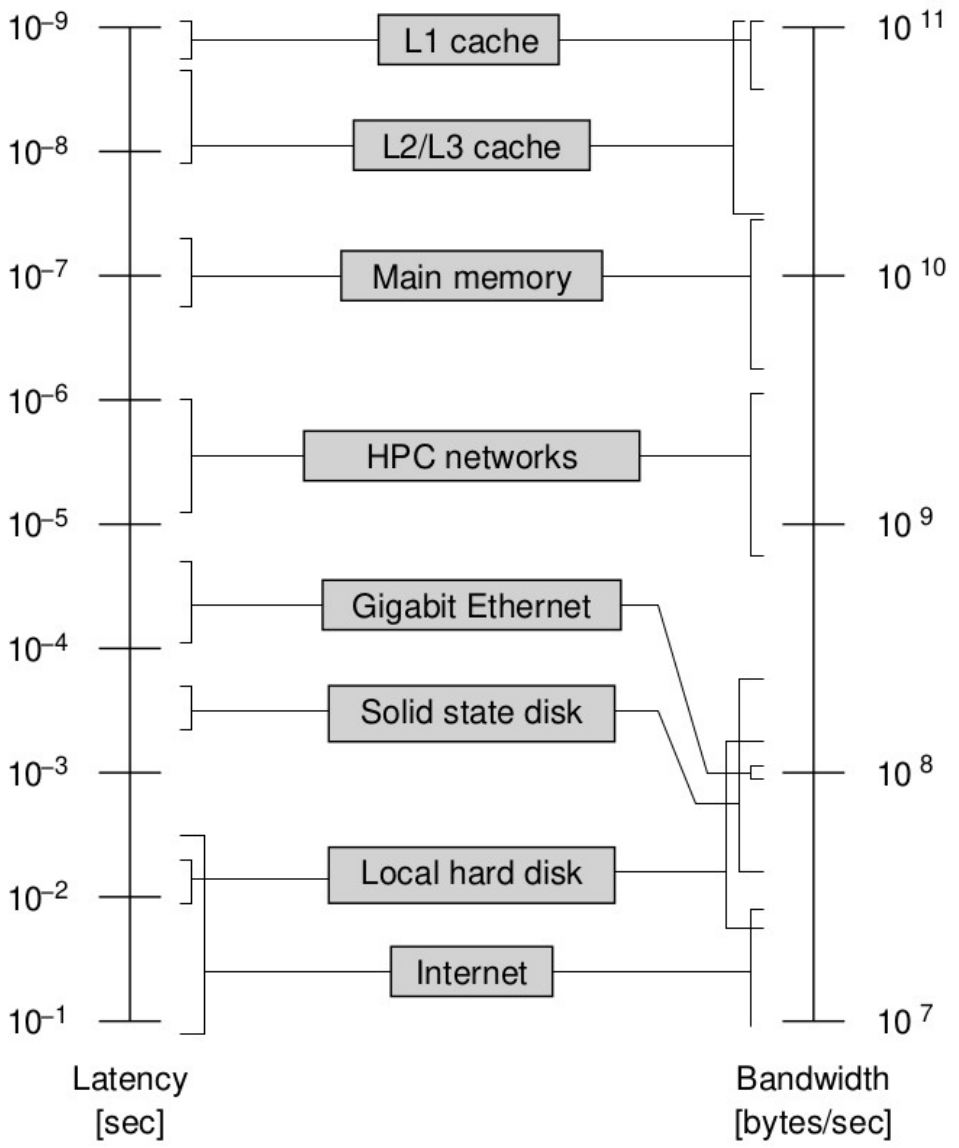


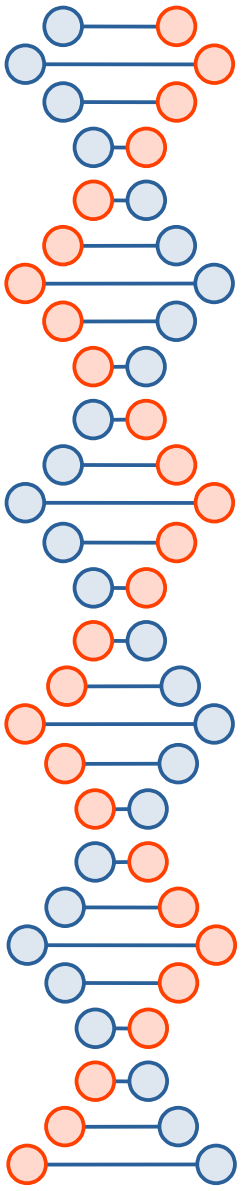
Ch.3 Data Access Optimization

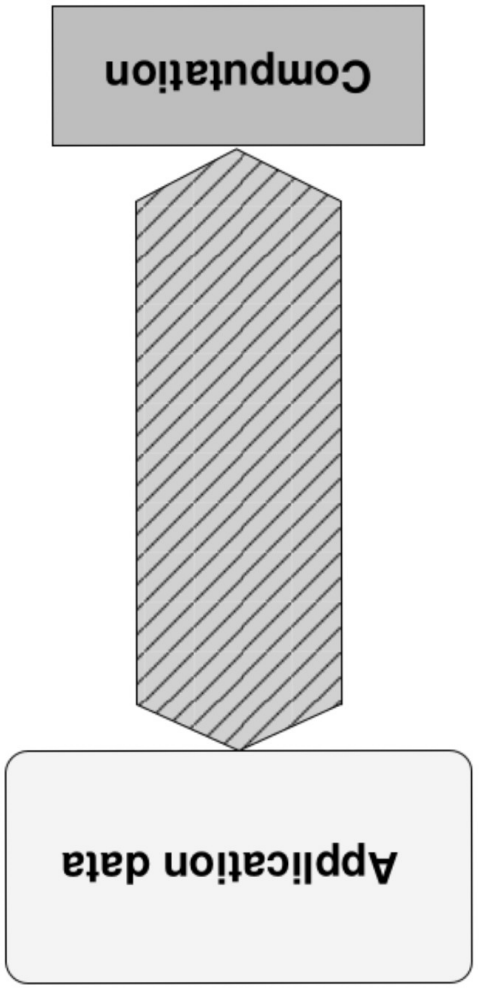
Figure 3.1 shows an overview of several data paths present in modern parallel computer systems, and typical ranges for their bandwidths and latencies. The functional units, which actually perform the computational work, sit at the top of this hierarchy. In terms of bandwidth, the slowest data paths are three to four orders of magnitude away, and eight in terms of latency. The deeper a data transfer must reach down through the different levels in order to obtain required operands for some calculation, the harder the impact on performance. Any optimization attempt should therefore first aim at reducing traffic over slow data paths, or, should this turn out to be infeasible, at least make data transfer as efficient as possible.

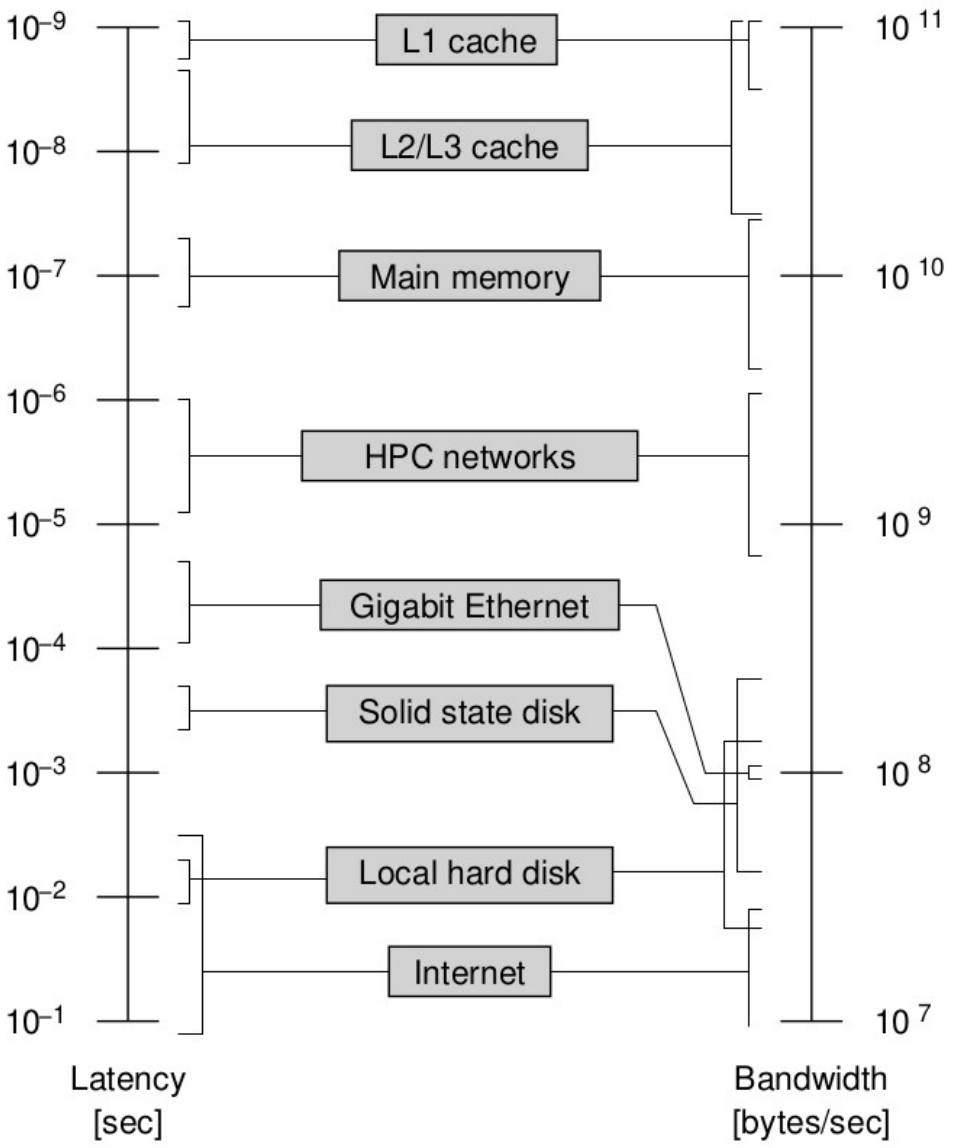


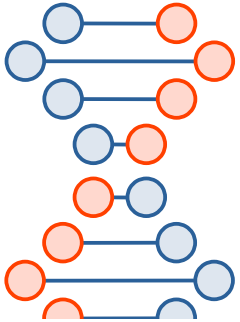
Latency vs Bandwidth






Computation
Application data


10^{-9}
10^{-8}
10^{-7}
10^{-6}
10^{-5}
10^{-4}
10^{-3}
10^{-2}
10^{-1}
L1 cache
L2/L3 cache
Main memory
HPC networks
Gigabit Ethernet
Solid state disk
Local hard disk
Internet
10^{11}
10^{10}
10^{9}
10^{8}
10^{7}
Latency [sec]
Bandwidth [bytes/sec]

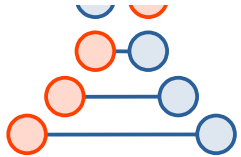


3.1 Balance analysis and lightspeed estimates

rules of thumb. The central concept to introduce here is *balance*. For example, the *machine balance* B_{m} of a processor chip is the ratio of possible memory bandwidth in GWords/sec to peak performance in GFlops/sec:

$$B_{\mathrm{m}} = \frac{\text{memory bandwidth [GWords/sec]}}{\text{peak performance [GFlops/sec]}} = \frac{b_{\max}}{P_{\max}} \tag{3.1}$$

"Memory bandwidth" could also be substituted by the bandwidth to caches or even network bandwidths, although the metric is generally most useful for codes that are really memory-bound. Access latency is assumed to be hidden by techniques like



3.1 Balance analysis and lightspeed estimates

$$B_{\mathrm{m}} = \frac{\text{memory bandwidth [GWords/sec]}}{\text{peak performance [GFlops/sec]}} = \frac{b_{\max}}{P_{\max}} \qquad (3.1)$$

data path	**balance** [W/F]
cache	0.5–1.0
machine (memory)	0.03–0.5
interconnect (high speed)	0.001–0.02
interconnect (GBit ethernet)	0.0001–0.0007
disk (or disk subsystem)	0.0001–0.01

Table 3.1: Typical balance values for operations limited by different transfer paths. In case of network and disk connections, the peak performance of typical dual-socket compute nodes was taken as a basis.

3.1 Balance analysis and lightspeed estimates

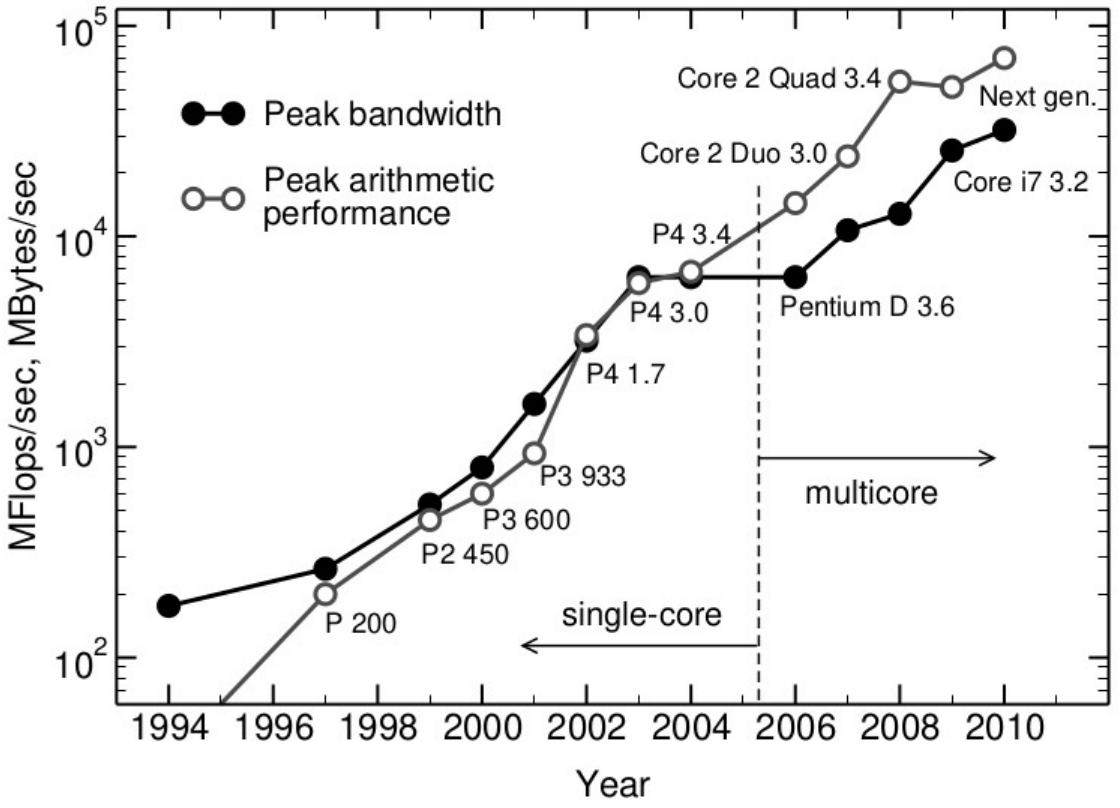


Figure 3.2: Progress of maximum arithmetic performance (open circles) and peak theoretical memory bandwidth (filled circles) for Intel processors since 1994. The fastest processor in terms of clock frequency is shown for each year. (Data collected by Jan Treibig.)

3.2 Storage order

Stride-N access

```
1 do i=1,N
2   do j=1,N
3     A(i,j) = i*j
4   enddo
5 enddo
```

Stride-1 access

```
for(i=0; i<N; ++i) {
  for(j=0; j<N; ++j) {
    a[i][j] = i*j;
  }
}
```

because C implements *row major order* (see Figure 3.3), whereas Fortran follows the so-called *column major order* (see Figure 3.4) for multidimensional arrays. Although mathematically insignificant, the distinction must be kept in mind when optimizing for data access: If an inner loop variable is used as an index to a multidimensional array, it should be the index that ensures stride-one access (i.e., the first in Fortran and the last in C). Section 3.4 will show what can be done if this is not easily possible.

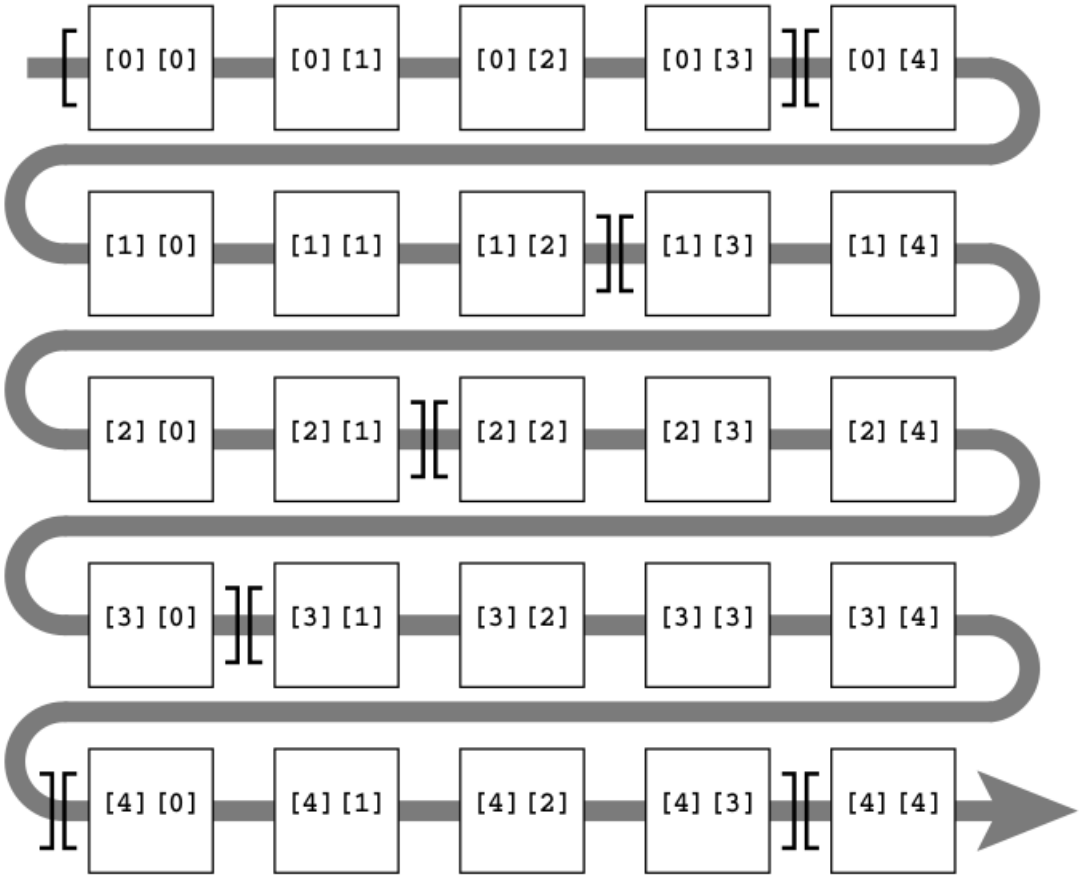


Figure 3.3: Row major order matrix storage scheme, as used by the C programming language. Matrix rows are stored consecutively in memory. Cache lines are assumed to hold four matrix elements and are indicated by brackets.

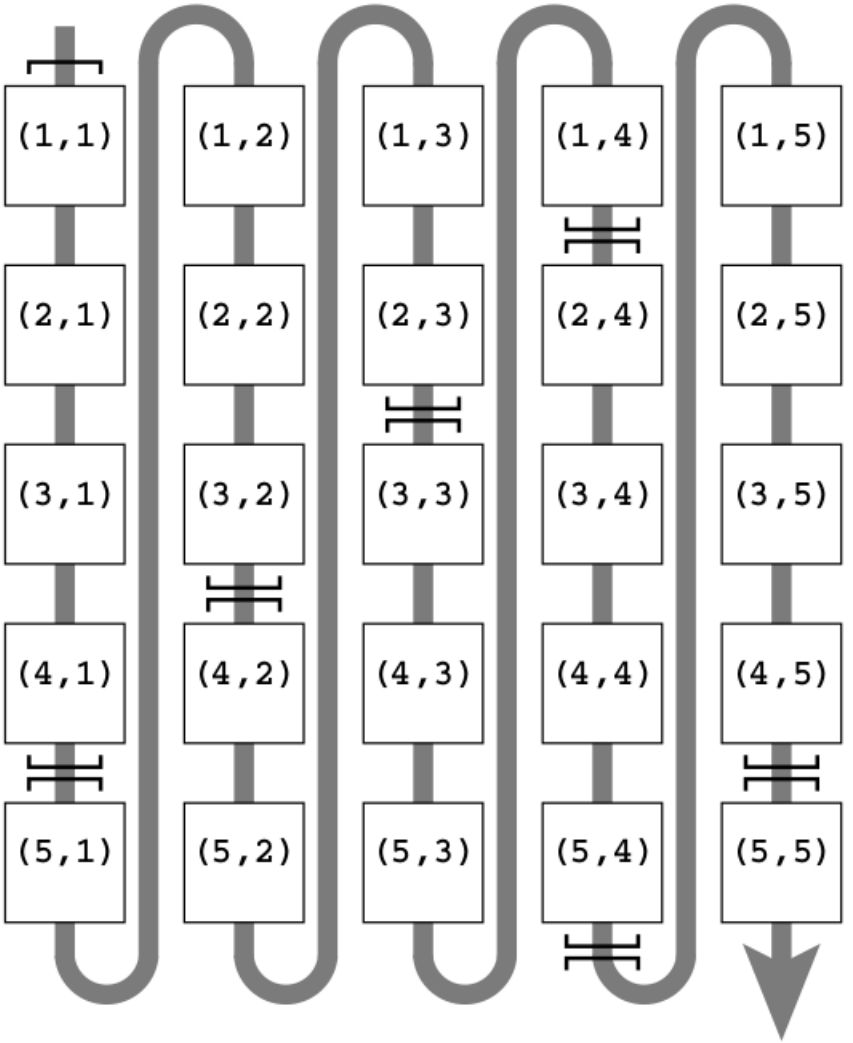
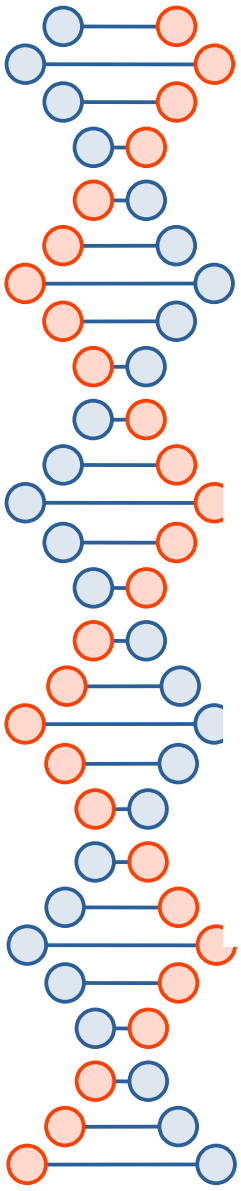


Figure 3.4: Column major order matrix storage scheme, as used by the Fortran programming language. Matrix columns are stored consecutively in memory. Cache lines are assumed to hold four matrix elements and are indicated by brackets.



3.4 Case study: Dense matrix transpose

For the following example we assume column major order as implemented in Fortran. Calculating the transpose of a dense matrix, $A = B^{\mathrm{T}}$, involves strided memory access to A or B, depending on how the loops are ordered. The most unfavorable way of doing the transpose is shown here:

```
1 do i=1,N
2   do j=1,N
3     A(i,j) = B(j,i)
4   enddo
5 enddo
```

3.4 Case study: Dense matrix transpose

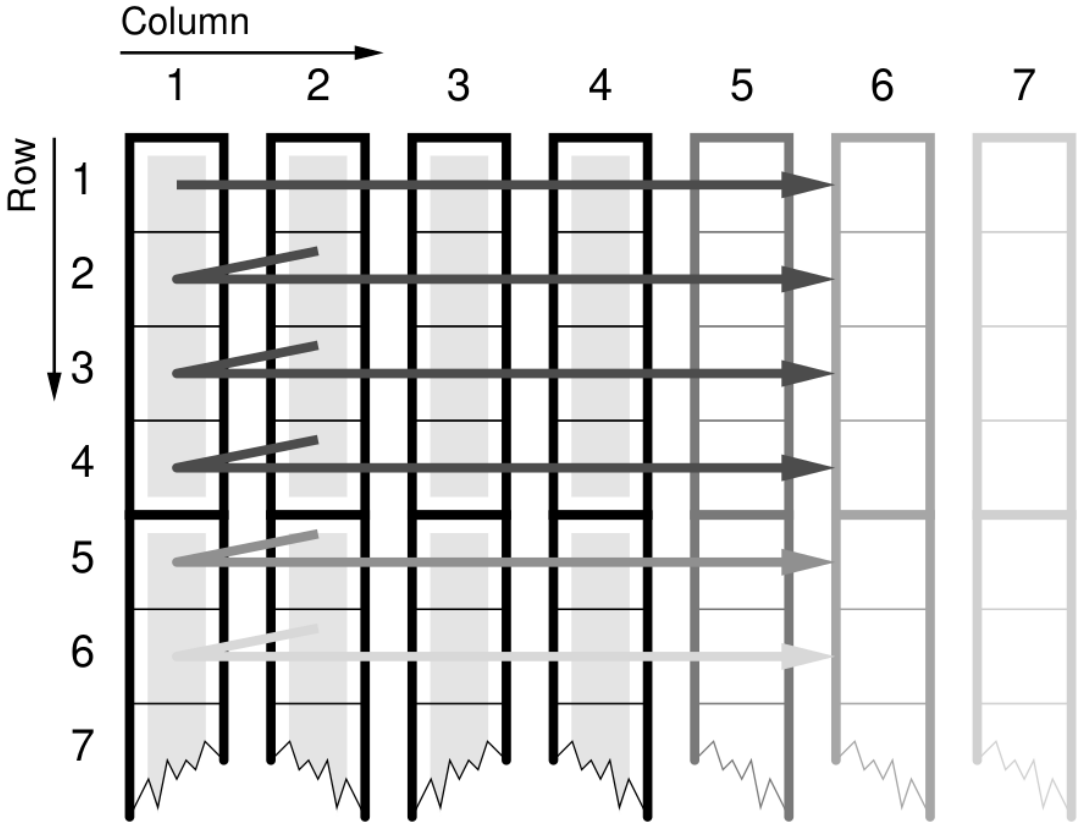
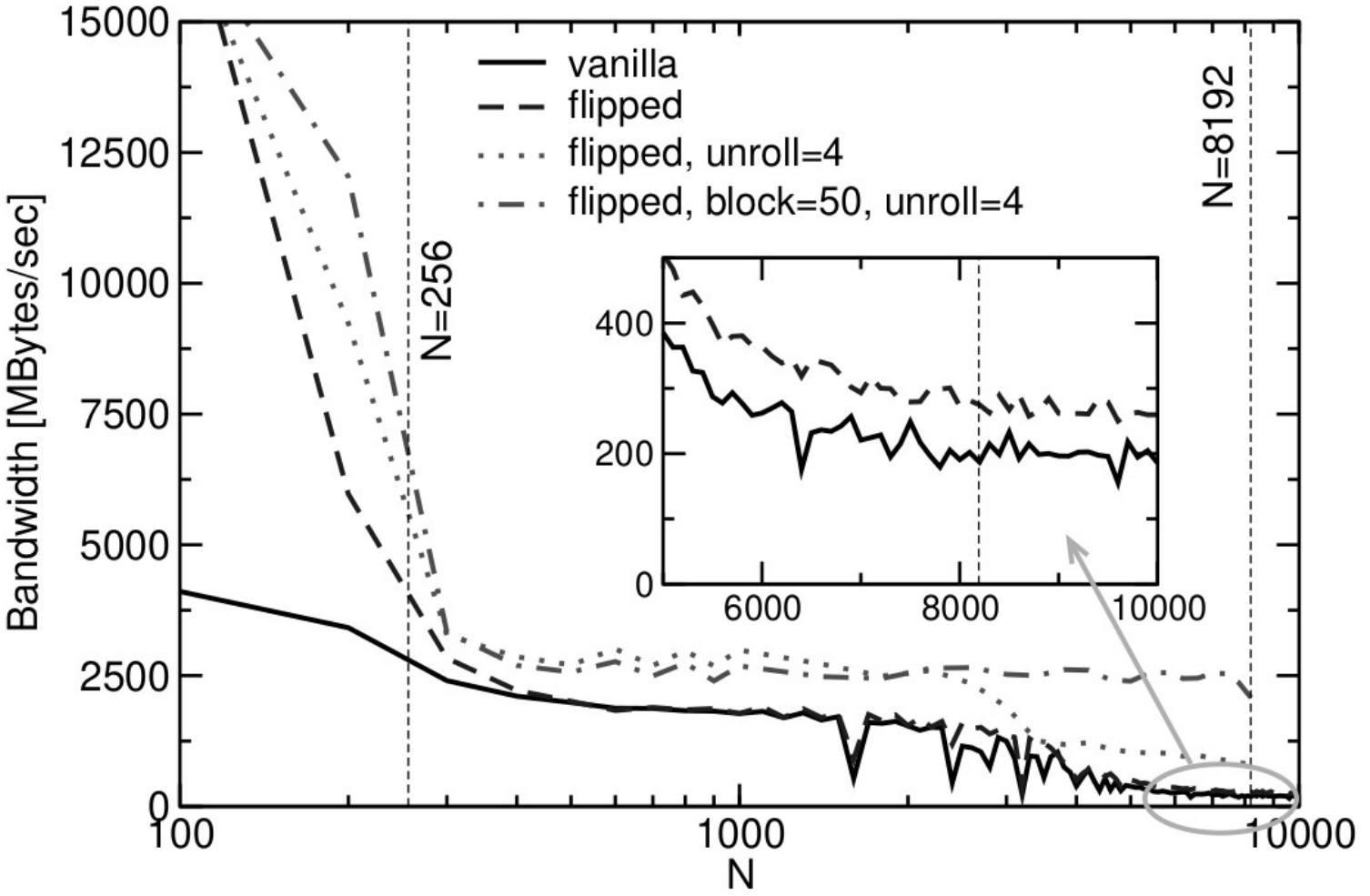
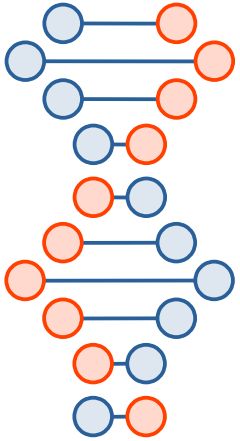


Figure 3.7: Cache line traversal for vanilla matrix transpose (strided store stream, column major order). If the leading matrix dimension is a multiple of the cache line size, each column starts on a line boundary.

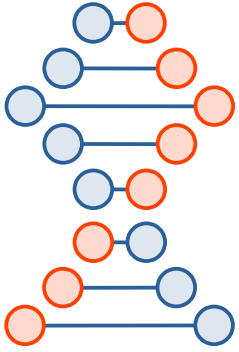
3.4 Case study:





3.4 Case study: Dense matrix transpose

Figure 3.8: Performance (effective bandwidth) for different implementations of the dense matrix transpose on a modern microprocessor with 1 MByte of L2 cache. The $N = 256$ and $N = 8192$ lines indicate the positions where the matrices fully fit into the cache and where N cache lines fit into the cache, respectively. (Intel Xeon/Nocona 3.2 GHz.)



3.4 Case study: Dense matrix transpose

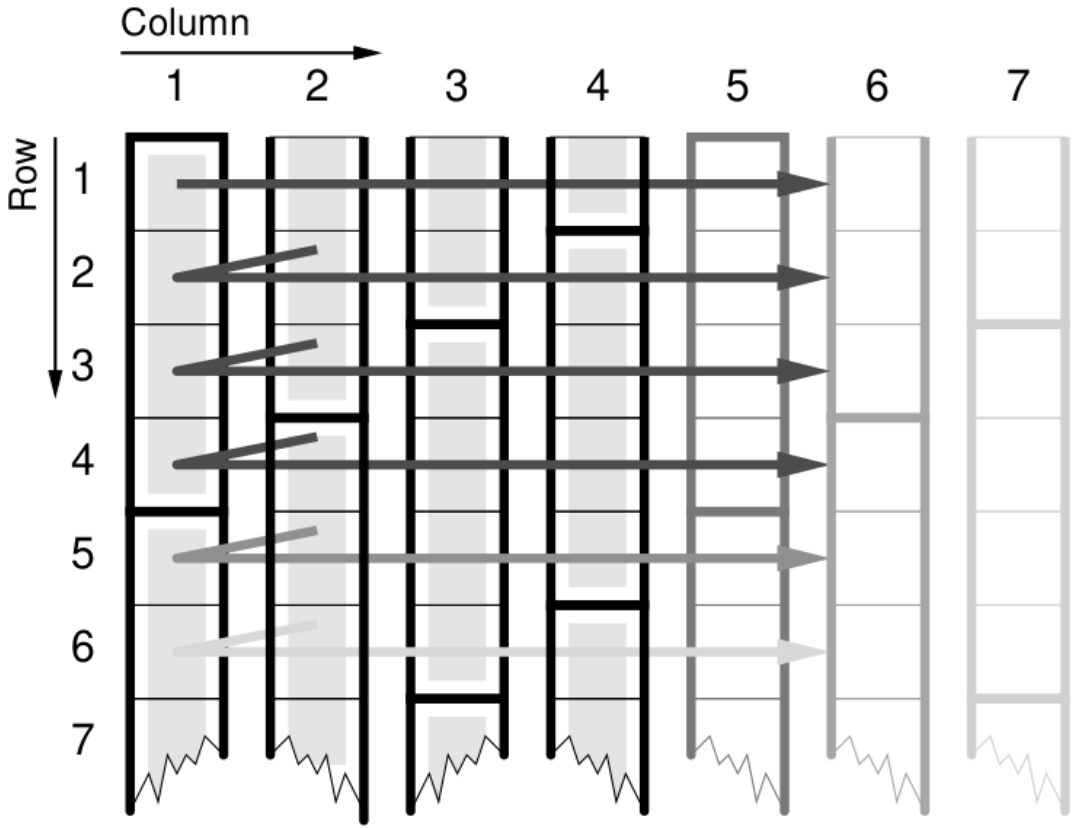


Figure 3.10: Cache line traversal for padded matrix transpose. Padding may increase effective cache size by alleviating associativity conflicts.

3.4 Case study: Dense matrix transpose