# Technische Universität München
# Course IN2075: Microprocessors

## Exercise 2.1 : Caches

```
void Exercise21()
{
    for (int i = 0; i < 1024 * 4096; i++)
    {
        ByteArray[i] = char(rand());
    }


    for (int i = 1; i <= 100; i++)
    {
        Stride(i);
    }

    printf("ENDE Exercise 1 \n");
}
```

- Method „Stride“ accesses the ByteArray char by char

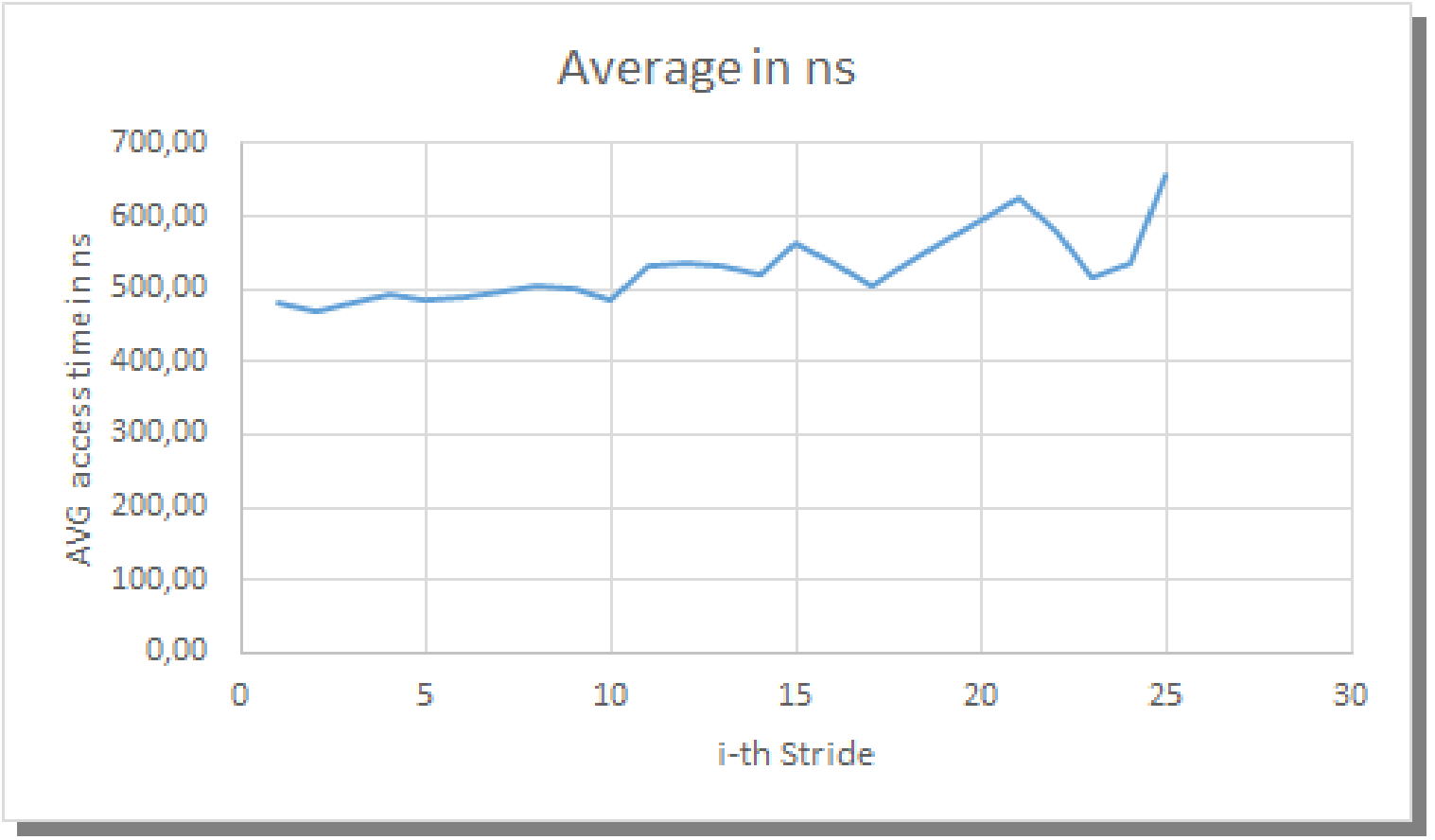
→ Adds char-Variables to Vector „Charly“

- Average Time per Operation measured using *GetTickCount()*

```
void Stride(int i)
{

    long int before = GetTickCount();

    for (int x = 0; x < sizeof(ByteArray); x++)
    {
        if (x % i == 0 && x != 0)
        {
            Charly.push_back(ByteArray[x]);
        }

    }

    long int after = GetTickCount();

    long TimeEffortTick = after - before;

    double TimeEffortSec = TimeEffortTick / 1000.0f;

    double AVGReadOperationTime = TimeEffortSec / (sizeof(ByteArray)/i);

    printf("I: %i Zeit: %f AverageReadOperation %e \n",i, TimeEffortSec, AVGReadOperationTime);

    while (Charly.size() > 0)
    {
        char temp = Charly.back();
        Charly.pop_back();
    }
}
```

Observation:

→ Increasing the Stride lowers the overall time but increases the average time used per access on a variable




Reason:

→ A higer stride value means that the process has to ignore more variables. That way it has to start more Compare-operations before it can work with a value.

# Exercise 2.2 : Cache Sizes

```
void Exercise22()
{
    printf("Begin Exercise 2.2 \n");
    unsigned long N = 1000000;
    int counter = 0;
    int stride = 4160;
    char MemArray[100000];
    char* MemBlocks[100000];
    //Fill with a predefined char
    for (int i = 0; i < sizeof(MemArray) / sizeof(*MemArray); i++)
    {
        MemArray[i] = 'c';
    }
    //Given RandomPointer-Formula
    for (int i = 0; i < sizeof(MemArray) / sizeof(*MemArray); i++)
    {
        MemBlocks[i] = &MemArray[(i + stride) % sizeof(MemArray)];
    }
    long int before = GetTickCount();

    for (int x = 1; x <= 10; x++)
    {
        for (int i = 0; i < sizeof(MemBlocks) / sizeof(*MemBlocks); i++)
        {
            char temp = *MemBlocks[i];
        }
    }
    long int after = GetTickCount();
    long TimeEffortTick = after - before;
    double TimeEffortSec = TimeEffortTick / 1000.0f;
    printf("Time: %f \n",TimeEffortSec);
    printf("End Exercise 2.2\n");
}
```

Idea:

- Create an array with predefined char-Values

- Use the given formula to fill pointer-array:

$$x \rightarrow (x+stride) \% N$$

- Access each char using Pointer-Chasing and measure the time using *GetTickCount()*

Problem:

In our case the time-measurement always returned 0.0000000.
(It worked fine when putting a printf-order in the for-loop)

# Exercise 2.3 : likwid

```
CPU type:       Intel Core SandyBridge EP processor
*************************************************************
Hardware Thread Topology
*************************************************************
Sockets:        1
Cores per socket:       2
Threads per core:       1
-------------------------------------------------------------
HWThread        Thread          Core            Socket
0               0               0               0
1               0               1               0
-------------------------------------------------------------
Socket 0: ( 0 1 )
-------------------------------------------------------------

*************************************************************
Cache Topology
*************************************************************
Level:  1
Size:   32 kB
Cache groups:   ( 0 ) ( 1 )
-------------------------------------------------------------
Level:  1
Size:   32 kB
Cache groups:   ( 0 ) ( 1 )
-------------------------------------------------------------
Level:  2
Size:   6 MB
Cache groups:   ( 0 ) ( 1 )
-------------------------------------------------------------

*************************************************************
NUMA Topology
*************************************************************
NUMA domains: 1
-------------------------------------------------------------
Domain 0:
Processors:  0 1
Relative distance to nodes:  10
Memory: 1293.47 MB free of total 3009.16 MB
-------------------------------------------------------------
```

```
Level:  1
Size:   32 kB
Type:   Data cache
Associativity:  8
Number of sets: 64
Cache line size:        64
Inclusive cache
Shared among 1 threads
Cache groups:   ( 0 ) ( 1 )
-------------------------------------------------------------
Level:  1
Size:   32 kB
Type:   Data cache
Associativity:  8
Number of sets: 64
Cache line size:        64
Inclusive cache
Shared among 1 threads
Cache groups:   ( 0 ) ( 1 )
-------------------------------------------------------------
Level:  2
Size:   6 MB
Type:   Data cache
Associativity:  24
Number of sets: 4096
Cache line size:        64
Inclusive cache
Shared among 1 threads
Cache groups:   ( 0 ) ( 1 )
-------------------------------------------------------------

*************************************************************
NUMA Topology
*************************************************************
NUMA domains: 1
-------------------------------------------------------------
Domain 0:
Processors:  0 1
Relative distance to nodes:  10
Memory: 1329.22 MB free of total 3009.16 MB
```

## Interpretation:

- Not possible at this moment because exercise 2.2 did not work out as expected