Exercise 4

Introduction to High-Performance Computing
WS 2019

Christian Terboven
contact@hpc.rwth-aachen.de

Exercise Tasks

1. Hello World
2. Exploiting Architectures: STREAM
3. First steps with Tasks
4. Reasoning about Work-Distribution
5. OpenMP Puzzles
6. Dry runs on various aspects

1. Hello World

Problem 1. Hello World

Go to the `hello` directory. Compile the `hello` code via '`make [debug|release]`' and execute the resulting executable via '`OMP_NUM_THREADS=procs make run`', where procs denotes the number of threads to be used.

a) Change the code that the thread number (*thread id*) and the total number of threads in the *team* are printed. Re-compile and execute the code in order to verify your changes.

C/C++: In order to print a decimal number, use the `%d` format specifier with `printf()`.

```
int i1 = value;
int i2 = other_value;
printf("Value of i1 is: %d, and i2 is: %d", i1, i2);
```

b) In which order did you expect the threads to print out the Hello World message? Did your expectations meet your observations? If not, is that wrong?

LIVE DEMO

2. Exploiting Architectures: STREAM

Problem 2. Exploiting Architectures: STREAM
The STREAM memory benchmark is a widely used instrument for memory bandwidth measurements. It tries to measure the bandwidth that is actually available to a program, which typically differs from what a vendor specifies as the maximum memory bandwidth.

On Linux, the `taskset` command can be used to restrict a process to a subset of all the cores in a system. The syntax that should be used in this exercise is as follows:

```
$ taskset -c cpu-list cmd [args]
```

where `cmd` denotes the program to be executed under the specified restriction. The program `lstopo` can be used to get a graphical representation of the machine architecture.

a) Go to the `stream` directory. Take a look at the source code and examine the operation that is done in order to measure the performance of the memory subsystem ($A = B + scalar \cdot C$). Parallelize this operation with OpenMP. Take care of producing correct time measurements.

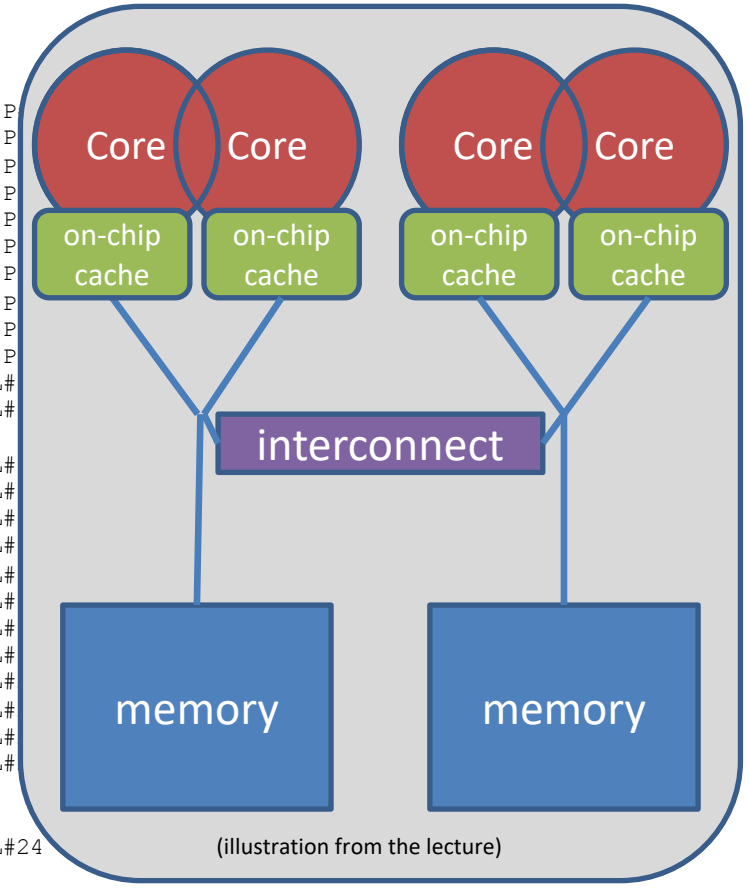
b) Compile the `stream` code via '`make release`' and execute the program on login-t (or the mpi backend) via '`OMP_NUM_THREADS=2 taskset -c binding ./stream.exe`', for each binding in the table below. With an array size of 40,000,000, each array is approximately 300 MiB in size. How do you explain the performance variations?
Hint: Consider NUMA effects.

#Threads	Binding (no blanks in between)	SAXPYing [MiB/s]
2	0,1	
2	0,2	
2	0,12	
2	0,24	

2. Exploiting Architectures: STREAM

- **Cluster frontend: login18-1.hpc.itc.rwth-aachen.de**
- **`lstopo --no-io`**

```
Machine (383GB total)
  Package L#0 + L3 L#0 (33MB)
    NUMANode L#0 (P#0 95GB)
      L2 L#0 (1024KB) + L1d L#0 (32KB) + L1i L#0 (32KB) + Core L#0 + PU L#0 (P
      L2 L#1 (1024KB) + L1d L#1 (32KB) + L1i L#1 (32KB) + Core L#1 + PU L#1 (P
      L2 L#2 (1024KB) + L1d L#2 (32KB) + L1i L#2 (32KB) + Core L#2 + PU L#2 (P
      L2 L#3 (1024KB) + L1d L#3 (32KB) + L1i L#3 (32KB) + Core L#3 + PU L#3 (P
      L2 L#4 (1024KB) + L1d L#4 (32KB) + L1i L#4 (32KB) + Core L#4 + PU L#4 (P
      L2 L#5 (1024KB) + L1d L#5 (32KB) + L1i L#5 (32KB) + Core L#5 + PU L#5 (P
      L2 L#6 (1024KB) + L1d L#6 (32KB) + L1i L#6 (32KB) + Core L#6 + PU L#6 (P
      L2 L#7 (1024KB) + L1d L#7 (32KB) + L1i L#7 (32KB) + Core L#7 + PU L#7 (P
      L2 L#8 (1024KB) + L1d L#8 (32KB) + L1i L#8 (32KB) + Core L#8 + PU L#8 (P
      L2 L#9 (1024KB) + L1d L#9 (32KB) + L1i L#9 (32KB) + Core L#9 + PU L#9 (P
      L2 L#10 (1024KB) + L1d L#10 (32KB) + L1i L#10 (32KB) + Core L#10 + PU L#
      L2 L#11 (1024KB) + L1d L#11 (32KB) + L1i L#11 (32KB) + Core L#11 + PU L#
    NUMANode L#1 (P#1 96GB)
      L2 L#12 (1024KB) + L1d L#12 (32KB) + L1i L#12 (32KB) + Core L#12 + PU L#
      L2 L#13 (1024KB) + L1d L#13 (32KB) + L1i L#13 (32KB) + Core L#13 + PU L#
      L2 L#14 (1024KB) + L1d L#14 (32KB) + L1i L#14 (32KB) + Core L#14 + PU L#
      L2 L#15 (1024KB) + L1d L#15 (32KB) + L1i L#15 (32KB) + Core L#15 + PU L#
      L2 L#16 (1024KB) + L1d L#16 (32KB) + L1i L#16 (32KB) + Core L#16 + PU L#
      L2 L#17 (1024KB) + L1d L#17 (32KB) + L1i L#17 (32KB) + Core L#17 + PU L#
      L2 L#18 (1024KB) + L1d L#18 (32KB) + L1i L#18 (32KB) + Core L#18 + PU L#
      L2 L#19 (1024KB) + L1d L#19 (32KB) + L1i L#19 (32KB) + Core L#19 + PU L#
      L2 L#20 (1024KB) + L1d L#20 (32KB) + L1i L#20 (32KB) + Core L#20 + PU L#
      L2 L#21 (1024KB) + L1d L#21 (32KB) + L1i L#21 (32KB) + Core L#21 + PU L#
      L2 L#22 (1024KB) + L1d L#22 (32KB) + L1i L#22 (32KB) + Core L#22 + PU L#
      L2 L#23 (1024KB) + L1d L#23 (32KB) + L1i L#23 (32KB) + Core L#23 + PU L#
  Package L#1 + L3 L#1 (33MB)
    NUMANode L#2 (P#2 96GB)
      L2 L#24 (1024KB) + L1d L#24 (32KB) + L1i L#24 (32KB) + Core L#24 + PU L#24
[...]
```



(illustration from the lecture)

2. Exploiting Architectures: STREAM

- **login18-1**

# Threads	Binding	SAXPYing [MiB/s]	SAXPYing [MiB/s] (NUMA optimized)
2	0,1	16663.4	16568.9
2	0,2	16802.6	16567.8
2	0,12	16892.2	16605.8
2	0,24	14119.9	17300.5

  - → Hardware threads 0 and {1,2} are on the **same socket**
  - → Hardware threads 0 and 12 are on the **same socket** on **different numa dom.**
  - → Hardware threads 0 and 24 are on **different sockets**

- **OpenMP Places**
  - → Use OpenMP Places / first touch for adequate thread binding / NUMA optimization (refer to lecture)

3. First steps with tasks

Problem 3. First steps with OpenMP tasks

The code below performs a traversal of a dynamic list and for each list element the process() function is called. The for-loop continues until `e->next` points to null. Such a loop could not be parallelized in OpenMP until the task directive was introduced, as the number of loop iterations (= list elements) could not be computed. Parallelize this code using the task concept of OpenMP. **State the scope of each variable explicitly.**

```
List l;
Element e;



for(e = l->first; e; e = e->next)
{

    process(e);

}
```

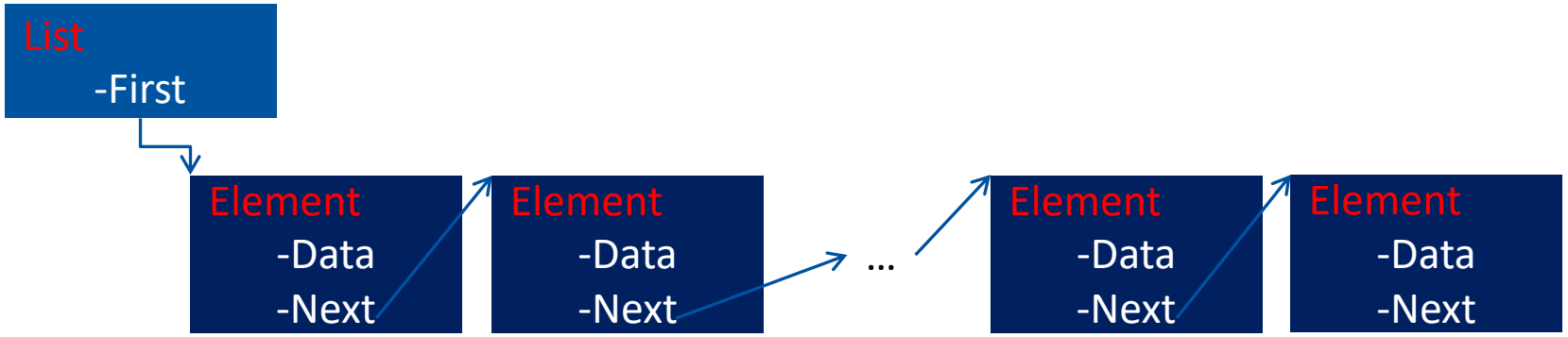
3. First steps with tasks

- **Traversal of dynamic list**

```
List l;
Element e;
for (e = l-> first ; e; e = e-> next)
{
        process (e);
}
```

3. First steps with tasks

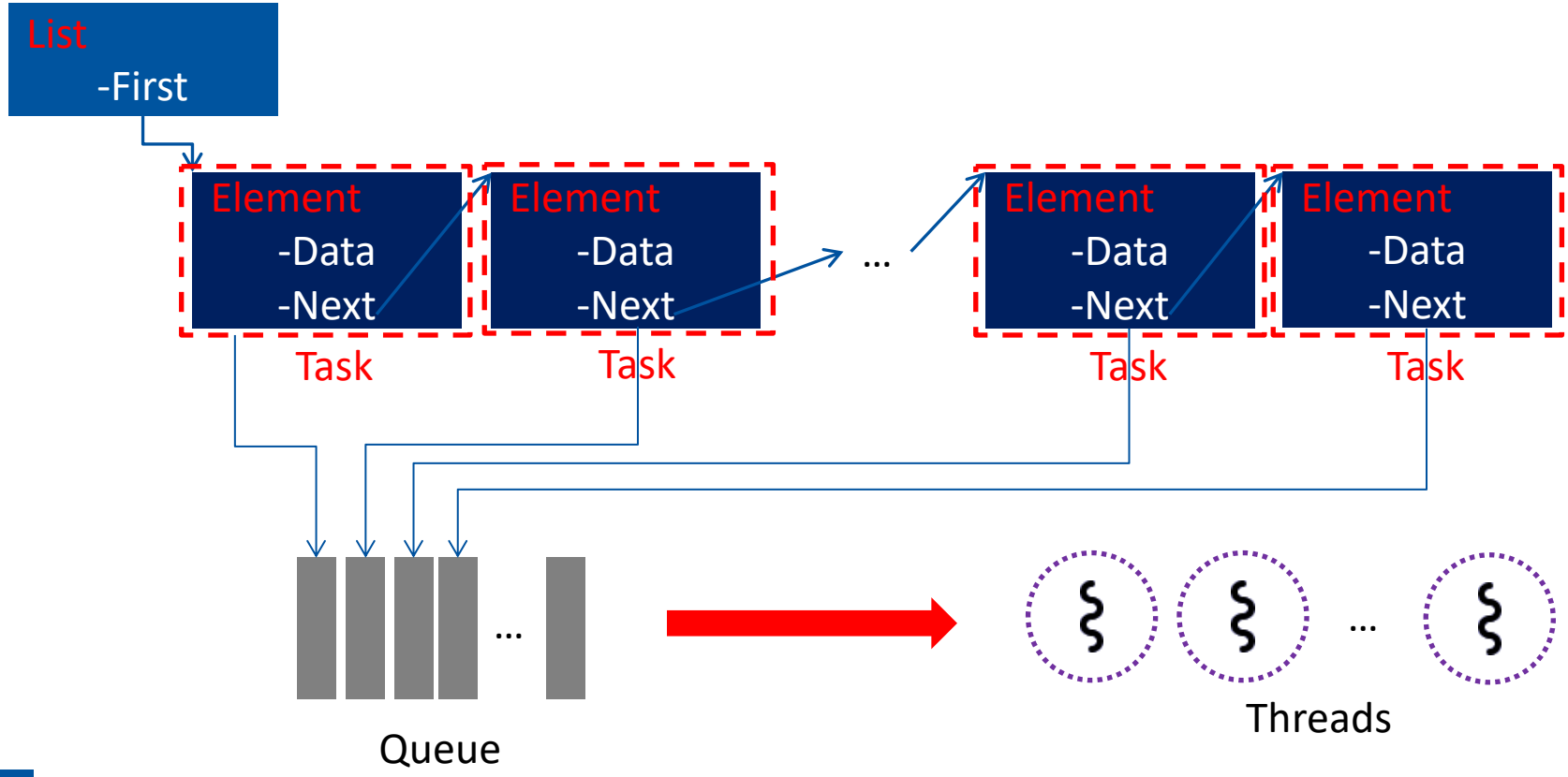
- **Traversal of dynamic list**



- **How can this be parallelized?**

3. First steps with tasks

- **Traversal of dynamic list**
  - → Number of elements unknown before traversal
  - → Tasks are created and subsequently processed by threads



3. First steps with tasks

- How to parallelize it?

```
List l;
Element e;
#pragma omp parallel shared(l,e)
{
        #pragma omp single shared(l,e)
        {
                for (e = l-> first ; e; e = e-> next )
                {
                        #pragma omp task firstprivate(e)
                        {
                                process (e);
                        }
                }
        }
}
```

4. Reasoning about Work-Distribution

Problem 4. Reasoning about Work-Distribution

Go to the `for` directory. Compile the `for` code via '`make [debug|release]`' and execute the resulting executable via '`OMP_NUM_THREADS=procs make run`', where *procs* denotes the number of threads to be used.

a) Examine the code and think about where to put the parallelization directive(s).

b) Measure the speedup and the efficiency of the parallelize code. How good does the code scale and which scaling did you expect?

#Threads	Runtime [sec]	Speedup	Efficiency
1			

LIVE DEMO

5. OpenMP Puzzles

Problem 5. OpenMP Puzzles

Try to parallelize the following loops by inserting the missing OpenMP directives. If a loop cannot be parallelized, state reasons why you think so.

a) Insert missing OpenMP directives to parallelize the given loop.

```
double A[N] = { ... }, B[N] = { ... }, D[N];
const double c = ...;
const double x = ...;
double y;




for (int i = 0; i < N; i++ )
{


    y = sqrt(A[i]);


    D[i] = y + A[i] / (x*x);
}
```

5. OpenMP Puzzles

- **Puzzle 1:**

```
#pragma omp parallel for private(y)
for (int i = 0; i < N; i++ )
{
        y = sqrt (A[i]);
        D[i] = y + A[i] / (x*x);
}
```

5. OpenMP Puzzles

b) Can you parallelize this loop — if yes, how? if not, why?

```
double A[N] = { ... }, B[N] = { ... }, D[N];
const double c = ...;
const double x = ...;
double y;



for (int i = 1; i < N; i++ )
{

    A[i] = B[i] - A[i - 1];
}
```

5. OpenMP Puzzles

- **Puzzle 2:**

  → Can you parallelize this loop?

```
for (int i = 1; i < N; i++ )
{
        A[i] = B[i] - A[i - 1];
}
```

Recap: Data dependencies

- **Dependencies within loops may prevent efficient software pipelining**
  - → Software pipelining: interleaving of loop iterations to meet latency requirements (done by the compiler)

No Dependency

```
//FORTRAN
DO I=1,N
  A(I)=A(I)*c
END DO


//C
for(i=0;i<N;++i)
  A[i]=A[i]*c;
```

Real Dependency

```
//FORTRAN
DO I=2,N
  A(I)=A(I-1)*c
END DO


//C
for(i=1;i<N;++i)
  A[i]=A[i-1]*c;
```

Pseudo Dependency

```
//FORTRAN
DO I=1,N-1
  A(I)=A(I+1)*c
END DO


//C
for(i=0;i<N-1;++i)
  A[i]=A[i+1]*c;
```

5. OpenMP Puzzles

- **Puzzle 2:**

  → Can you parallelize this loop?

```
for (int i = 1; i < N; i++ )
{
        A[i] = B[i] - A[i - 1];
}
```

I	A[i]
1	B[1] – A[0]
2	B[2] – A[1] = B[2] – B[1] + A[0]
3	B[3] – A[2] = B[3] – B[2] + B[1] – A[0]
4	B[4] – A[3] = B[4] – B[3] + B[2] – B[1] + A[0]
5	…

5. OpenMP Puzzles

- **Puzzle 2:**

  → Can you parallelize this loop?

```
for (int i = 1; i < N; i++ )
{
        A[i] = B[i] - A[i - 1];
}
```

No – There is a real dependency inside the loop which can't be resolved.

6. Dry runs on various aspects

Problem 6. Dry runs on various aspects

The code snippet below implements a matrix times vector (MxV) operation, where a is a vector of $\mathbb{R}^m$, B is a matrix of $\mathbb{R}^{m \times n}$ and c is a vector of $\mathbb{R}^n$: $a = B \cdot c$.

```
1  void mxv_row(int m, int n, double *A, double *B, double *C)
2  {
3      int i, j;
4
5
6
7      for (i = 0; i < m; i++)
8      {
9          A[i] = 0.0;
10
11
12
13         for (j = 0; j < n; j++)
14         {
15             A[i] += B[i * n + j] * C[j];
16         }
17     }
18 }
```

a) Parallelize the `for` loop in line 07 by providing the appropriate OpenMP pragmas. Which variables have to be private and which variables have to be shared?

b) Which schedule would you propose for this parallelization? Explain your answer and briefly list the pros and cons for each of the three OpenMP worksharing schedules (namely *static* *dynamic* and *guided*) for this specific case.

c) Would it be possible to parallelize the `for` loop in line 13? If your answer is yes, provide the appropriate line in OpenMP and explain what scaling you would expect. If your answer is no, explain why you think it is not possible.

6. Dry runs on various aspects

a) **Parallelize the for loop in line 07 by providing the appropriate OpenMP pragmas. Which variables have to be private and which variables have to be shared?**

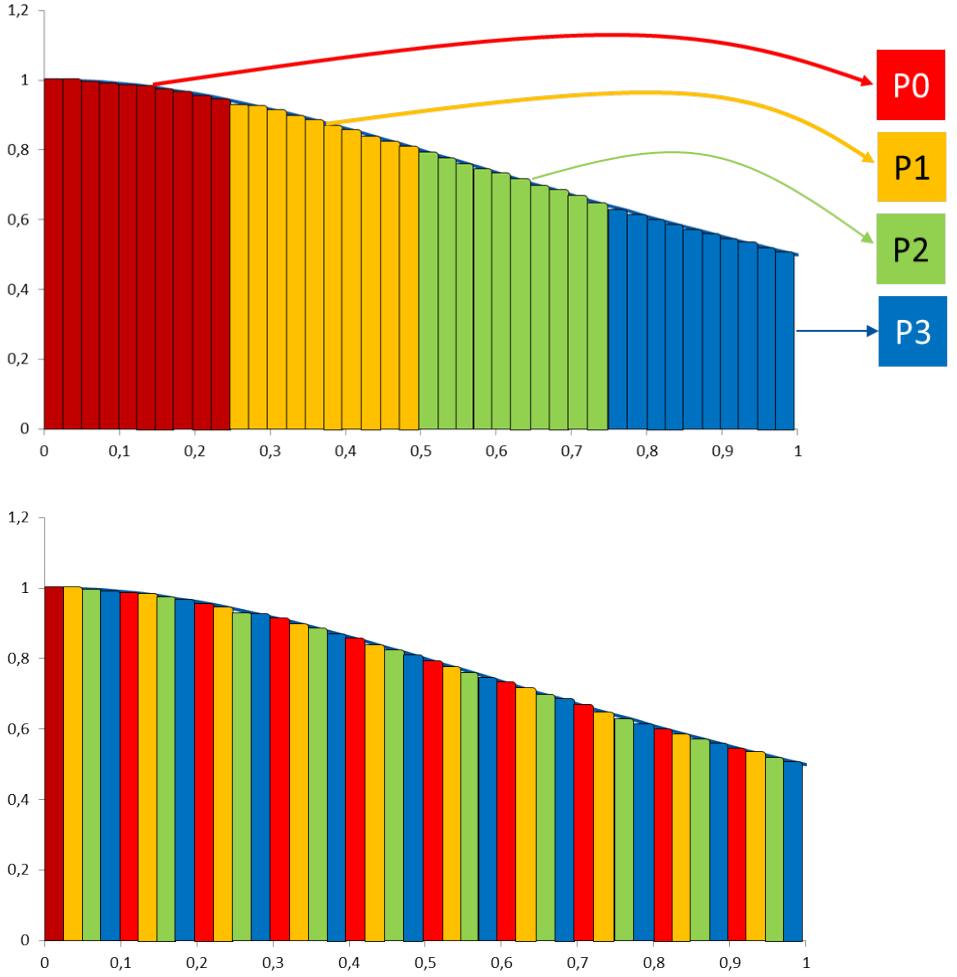
```
 1    void mxv_row ( int m, int n, double *A, double *B, double *C)
 2    {
 3        int i, j;
 4
 5
 6        #pragma omp parallel for shared(A,B,C, m, n) private(i,j)
 7        for (i = 0; i < m; i++)
 8        {
 9            A[i] = 0.0;
10
11
12
13            for (j = 0; j < n; j++)
14            {
15                A[i] += B[i * n + j] * C[j];
16            }
17        }
18    }
```

6. Dry runs on various aspects

b. **Which schedule would you propose for this parallelization? Explain your answer and briefly list the pros and cons for each of the three OpenMP worksharing schedules (namely *static, dynamic* and *guided*) for this specific case.**

6. Dry runs on various aspects

- **Static Schedule**
  - → `schedule(static [, chunk])`
  - → Decomposition depending on chunksize
  - → Equal parts of size ‘chunksize’ distributed in round-robin fashion
- **Pros?**
  - → No/low runtime overhead
- **Cons?**
  - → No dynamic workload balancing



6. Dry runs on various aspects

- **Dynamic schedule**
  - → `schedule(dynamic [, chunk])`
  - → Iteration space divided into blocks of chunk size
  - → Threads request a new block after finishing the previous one
  - → Default chunk size is 1
- **Pros ?**
  - → Workload distribution
- **Cons?**
  - → Runtime Overhead
  - → Chunk size essential for performance
  - → No NUMA optimizations possible

6. Dry runs on various aspects

- **Guided schedule**
  - → Similar to dynamic: Threads request chunks as they require
  - → Chunk size decreases over time until it reaches 1
- **Pros & Cons?**
  - → As for dynamic

6. Dry runs on various aspects

b. Which schedule would you propose for this parallelization? Explain your answer and briefly list the pros and cons for each of the three OpenMP worksharing schedules (namely *static, dynamic* and *guided*) for this specific case.

Static, because work is evenly distributed among the threads if the MxV operation is parallelized among the matrix rows.

6. Dry runs on various aspects

c. **Would it be possible to parallelize the for loop in line 13? If your answer is yes, provide the appropriate line in OpenMP and explain what scaling you would expect. If your answer is no, explain why you think it is not possible.**

It is possible with an OpenMP reduction. For performance reason one should in general try to parallelize the most outer loop