Assignment 1

Erkin Kirdan - Manuel Rothenberg - Gabrielle Poerwawinata

1. CoolMUC3 warm-up

- Modules system to manage the user environment for different software, library or compiler versions
- Advantage: specifying paths for executable versions, library versions and locations no longer required
- **`module load <package>`** and **`module unload <package>`**

1. CoolMUC3 warm-up

- Interactive SLURM shell: users type commands to run their jobs
  - Request resources with the command `salloc`
  - Add parameters to specify how many nodes and how many tasks per node you need
  - After the resource was granted, run your program in this configuration with the command `mpiexec` right from the shell
- SLURM batch script: users prepare their job scripts such as myjob.cmd (similar to a bash script) then submit their jobs
  - Prepare a script which contains your configuration
  - Submit the job with the command `sbatch`

1. CoolMUC3 warm-up

- Hello World Program:

```
#include <stdio.h>
#include <unistd.h>
int main(int argc, char** argv){
    char hostname[1024];
    gethostname(hostname, 1024);
    printf("Hello World! (from: %s)\n", hostname);
    return 0;
}
```

1. CoolMUC3 warm-up

- Batch Script:

```
#!/bin/bash
#SBATCH -o
/home/hpc/t1221/di34mip/logs/a1_1_hello_world/%j_%N.txt
#SBATCH -D /home/hpc/t1221/di34mip/code/a1_1_hello_world
#SBATCH --clusters=mpp3
#SBATCH --nodes=2-2
#SBATCH --ntasks-per-node=1
#SBATCH --mail-user=manuel.rothenberg@tum.de
mpiexec ./hello_world.out
```

1. CoolMUC3 warm-up

- Result:

```
Hello World! (from: mpp3r03c02s01)
Hello World! (from: mpp3r03c02s02)
```

2. Auto-vectorisation

- Data structure alignment: adjustment of any data object in relation with other objects
    - Aim: fitting data into memory words so that they can be used efficiently without any alignment fault
    - Aim (in other words): starting at certain addresses to speed up memory access since misaligned memory accesses can incur large performance losses

2. Auto-vectorisation

- Non-contiguous memory access
  - In such a case, integers must be loaded separately using multiple instructions, which is considerably less efficient
- Data dependencies
  - Forms: read-after-write, write-after-read or write-after-write
  - Causes: data hazards which lead to miscalculation so in order to calculate correct result, vectorization is utilized less or not at all

2. Auto-vectorisation

- Language extensions assists compiler for vectorization
  - Preferring vector data types, intrinsic functions and especially putting pragmas to indicate options and blocks to be vectorized helps compiler to vectorize the code in a more appropriate way
  - Therefore, it is the developer's responsibility to think about vectorization beforehand designing and developing the program so that during the vectorization compiler could efficiently vectorize the code

2. Auto-vectorisation

- Loop optimization
    - Aim: increasing execution speed and reducing the overheads associated with loops so that cache performance and parallel processing capabilities can be improved
    - Some major examples are:
        - Fission (distribution)
        - Fusion (combining)
        - Tiling (blocking)
        - Splitting (peeling)