Objectives and learning goals

In this example we learn how to vectorise and parallelise regions using SIMD and OpenMP pragmas

• To enable the compiler to generate diagnostic information
• Understand the vectorisation performance
• Understand vectorisation reports
• To control memory allocation on MIC

1. Lab 1

• Compile the program on the host without modifying the original code.
• Use the –no-vec flag to turn off the vectorisation:
  o $mpicc –no-vec –qopenmp –mmic nBody.c –o nbody.exe
• run the program with: $micnativeloadex ./nbody.exe and record execution time.
• Add the vector report flags: -qopt-report –qopt-report-phase:vec
• Display the optimisation report file “nbody.opttrpt” and try to understand the vectorised regions.
• Remove the –no-vec and –qopt-report flags and repeat the execution step above to record the execution time in the end. Check the performance results.
• Display the source code and switch on the parallelisations lines.
• Compile the program only with:
  o $mpicc –qopenmp –mmic nBody.c –o nbody.exe
  o and repeat the execution line above.
• Check the performance results.

What about the performance.
2. Lab 2

- Display nboby.c code and replace the LRZ WORK FOR YOU comments with SIMD and OpenMP calls.
- Display the Makefile
- Add the **vector report flags**: `-qopt-report -qopt-report-phase:vec`
- Compile the program: `make`
- Display the output reports and try to understand the vectorised regions.
- Display the Makefile, remove the vector report flags and compile again
- Run: `make run`
- Check the performance results
- Set the following environment:
  - `export MIC_KMP_AFFINITY=explicit,granularity=fine,proclist=[1-236:1]`
  - `export KMP_AFFINITY=granularity=fine,compact,1,0`
- What about the performance.

Try now to understand the performance numbers observed for the host and native execution.