Vectorisation Practicals

These practicals will use the optimisation reports from the Intel compiler to tell us about how the compiler is vectorising things. This is done by using the compiler flag `-qopt-report=5`. For every C file this will produce a text file called .optrpt. This will have vectorisation reports of every loop in that C file.

Auto-vectorisation

Part 1

This example explores the auto-vectorisor and the optimisation reporting.

1. Compile the code without changing it and read the optimisation report for the function vadd. Has it vectorised?
2. Add the restrict keyword to the function arguments of type `float *` in vadd. Recompile. Does the optrpt change?
3. Use aligned mallocs (`_mm_malloc`). Recompile. Does the optrpt change?
4. Drop a hint to the compiler that the arrays are aligned. How does the optrpt look now?

Part 2

In this example you will betray the compiler's trust in you.

1. Compile the code and look at the optrpt for worker.cpp. Is it vectorised?
2. Run the code as is and note the answer.
3. Add the compiler hint `#pragma ivdep` above the loop in worker.cpp. Recompile and check the optrpt. Has it vectorised?
4. Re-run the code. Has the result changed since adding the pragma?
5. Re-compile with the flag `-no-vec`. What is the result now?
6. What was the effect of `#pragma ivdep` ?

Alignment

Part 1

In this example you need to align a multi-dimensional array.

1. Align the 2D array so that all of its rows start on a 64 byte boundary.

2. Tell the compiler to generate aligned vector code in the main loop using OpenMP SIMD pragmas.
3. Offload the whole code.
4. Run the offload code using the batch script provided : `$ msub offload_job.sh`
5. Does alignment affect performance?
6. What happens if you tell the compiler something is aligned when it isn't?

Part 2 * Optional Challenge*

1. Using only `malloc` allocate an array that is 64 byte aligned.
2. Free the array using `free` .

Explicit Vectorisation

Part 1

Re-try the experiments in Part-1 of auto-vectorisation using OpenMP SIMD pragmas.

Part 2

In this exercise we will experiment with vectorised functions. There is a loop that calls the function `my_simple_add` .

1. In v1. Compile this code and look at the optrpt. Did it vectorise? Why?
2. Add `__declspec(noinline)` above the definition of `my_simple_add` and try again. What happens now?
3. In v2. Compile this code and look in main.optrpt. Does this vectorise.
4. Use OpenMP SIMD to vectorise the `my_simple_add` function. Then vectorise the loop. Compile and check the optrpt. What is the performance like compared to the others?

Part 3

In this practical we look at a larger example that evolves a diffusion PDE. This code already ready for offload.

1. Compile the code and check the optrpt. Is it vectorised?
2. Using the job script supplied, submit the job using `msub offload_job.sh` . Wait for the result to come back. Note the time.
3. Vectorise this code using OpenMP SIMD pragmas.
4. How much speed-up do you achieve?