1 CoolMUC3 warm-up (1P)

Consult the documentation of the Linux-Cluster [4] and inform yourself about using the CoolMUC3 cluster.

- What is the module system and how do you use it?
- How can you execute programs on the cluster's compute nodes? Describe the interactive mode and the batch mode.
- Write a simple "hello world" program which prints the host-name of the current machine. Compile the program for the Knights Landing processor. Write a batch script which runs your program on 2 different nodes of the CoolMUC3 cluster. (In parallel, do not just execute the batch script twice.) Inspect the logs and check that indeed two different host-names were printed.

  Include the source code, batch script, and log in your submission.

2 Auto-vectorisation (2P)

Please read Chapter 32 "Automatic Vectorization" of the **Intel C++ Compiler User and Reference Guides** [3] and answer following questions:

- Why is it important to align data structures?
- Which kinds of obstacles can prevent vectorisation?
- Is there a way to assist the compiler through language extensions? If yes, please give details.
- Which loop optimisations are performed by the compiler in order to vectorise and pipeline loops?

3 Vectorisation across equal-shaped problems (3P)

Application `gauss.c` provides an implementation of Gaussian-elimination without pivot-search but including backwards-substitution. It solves a system of linear equations with rank $\mathtt{n}$ directly. The system is given in form $\mathtt{Ax} = \mathtt{b}$, with $\mathtt{A}$ as coefficient-Matrix, $\mathtt{b}$ as right hand side, and $\mathtt{x}$ as solution.

- Sketch the given algorithm by using a figure of your choice.
- We want to solve a system of rank $\mathtt{n} = 3$ for 2000 different right hand sides. However, solving a single linear system of equations with rank 3 cannot be implemented efficiently using vectorisation. Please explain why.

- Change `gauss.c` accordingly in order to have a perfectly vectorised solver for systems with different right-hand sides. Consider the use of an optimal data-structure and measure GFLOPS-rates for the original code and your optimised implementation. How large are your improvements?
- Generate an optimisation report using compiler flags. Which loops where vectorised by the compiler?

4 The perfect DGEMM micro-kernel (4P)

DGEMM is an abbreviation for GEneral Matrix Multiplication in Double precision. Given matrices $A \in \mathbb{R}^{M \times K}$, $B \in \mathbb{R}^{K \times N}$, and $C \in \mathbb{R}^{M \times N}$, this operation shall implement

$$C_{mn} = \sum_{k=0}^{K-1} A_{mk} \cdot B_{kn} \,, \qquad m = 0, \ldots, M-1, \quad j = 0, \ldots, N-1.$$

A reference implementation is given in `dgemm.c`. Your task is to optimise DGEMM for the Knights Landing architecture. Describe your optimisation strategy in your report.

Try to reach at least 75 % of peak performance (31.2 GFLOPS).

Rules:

- You must only use a single KNL core.
- You have to set $K = 128$. The parameters $M, N \in \mathbb{N}$ may be chosen freely.
- Your code must match the reference code up to machine precision.
- You may use any combination of the following techniques: Auto-vectorisation, `#pragma`-directives, intrinsics, inline assembly.
- If you achieve 38.5 GFLOPS ($\approx 93\,\%$ peak) we will award you with a crate of beer. (In the unlikely case that multiple teams achieve this you have to share the crate.)

  The award can only be granted if you can explain the necessary optimisations.

Hints:

- Read the literature, especially [5]. (You will need to look at it anyway for the next assignment.)
- Use Intel C++ compiler 17.
- Experiment with smaller values of K, e.g. $K = 64$.
- Compile your application with `-S -fsource-asm`. This will generate an assembly listing which allows you inspect the code generated by the compiler. You should especially check if C is cached in registers.

- Do not listen to the compiler. E.g. the compiler might print "`remark: unroll pragma will be ignored`", which is a lie.
- Advanced: When the offset between columns of B is larger than 1024, the instruction size of a fused-multiply-add (FMA) increases from 7 to 10 bytes and FMAs cannot be fetched in a sustained manner. (Check for something like `vfmadd231pd 1536(%rdi){1to8}, %zmm20, %zmm17` in the assembly listing.)

  Think of a way to use spare general purpose registers to fix this issue.

Have fun!

Deliverables

The following deliverables have to be handed in no later than 08:00 AM, Monday, 6 November 2017. If there is no submission until this deadline, the exercise sheet is graded with 0 points. Small files (<1 MB in total) can be send as an attachment directly to *uphoff AT in.tum.de* and *chaulio.ferreira AT tum.de*. Larger files have to be uploaded at a place of your choice, e.g. `https://gitlab.lrz.de/`, `http://home.in.tum.de/`, `https://syncandshare.lrz.de`. In either case inform us about the final state of your solution via e-mail.

- A short report which describes your work and answers all questions in this assignment.
- All of your code.
- Slides for the presentation during the next meeting.
- Output of all runs. Figures (e.g. scaling graphs) if applicable.
- Documentation how to build and use your code.

Literature

[1] GEMM: From pure C to SSE optimized micro kernels. `http://apfel.mathematik.uni-ulm.de/~lehn/sghpc/gemm/index.html`. Accessed: 2016-10-21.

[2] How to optimize GEMM. `https://github.com/flame/how-to-optimize-gemm/wiki`. Accessed: 2016-10-21.

[3] Intel C++ compiler 17.0 user and reference guide. `https://software.intel.com/en-us/intel-cplusplus-compiler-17.0-user-and-reference-guide-pdf`. Accessed: 2017-10-18.

[4] Linux-Cluster. `https://www.lrz.de/services/compute/linux-cluster/`. Accessed: 2017-10-19.

[5] Kazushige Goto and Robert A. van de Geijn. Anatomy of high-performance matrix multiplication. *ACM Trans. Math. Softw.*, 34(3):12:1–12:25, May 2008.