Introduction to High-Performance Computing, Winter 2019

Exercise 2: Memory Access, Performance Analysis and Code Balance

Problem 1. Norm calculation of a matrix
Let $A = (a_{ij})_{i,j=1,\dots,n} \in \mathbb{R}^{n \times n}$ be a real matrix. The norms $\|\cdot\|_1$ and $\|\cdot\|_\infty$ are defined by:

$$\|A\|_1 := \max_{j=1,\dots,n} \sum_{i=1}^{n} |a_{ij}| \qquad (\text{“maximum column sum”})$$

$$\|A\|_\infty := \max_{i=1,\dots,n} \sum_{j=1}^{n} |a_{ij}| \qquad (\text{“maximum row sum”}).$$

This exercise may be done with pen and paper or using the C++ code template in `exercise2.tar.gz`

a) Design an algorithm to compute the norm $\|A\|_\infty$ by passing through the memory consecutively.

b) Calculate the cache-miss-ratio ($\#cachemiss/\#memoryaccess$) and the time $T^{(1)}(n)$ of that algorithm in terms of T_A, T_M, T_C, l, and c where

T_A	time for an arithmetic operation
T_M	time for accessing the main memory
T_C	time for accessing the cache
l	size of a cache line (in elements)
c	size of the cache (in elements).

Hint: Auxiliary variables are stored in registers. Assume that the computation of $|\cdot|$ and the comparison of two real numbers both take time T_A. Assume for all calculations that the matrix doesn't fit to the cache ($c \ll n \cdot n$) and the cache is initially cold (empty).

c) By switching the order of the two `for` loops in a), the algorithm now computes $\|A\|_1$. What time $T^{(2)}(n)$ does this algorithm take? Calculate the cache-miss-ratio.

Hint: Consider the cases $n > c$, $n \cdot l > c$, and $n \cdot l < c$.

d) Design an algorithm to compute $\|A\|_1$ with consecutive memory access and calculate its runtime $T^{(3)}(n)$.

Hint: Use an auxiliary array for the column sums.

e) What is the worst case scenario regarding cache-miss-ratio for the algorithm of d) when we assume a direct-mapped cache?

f) Calculate the speed-up of the algorithm of d) with respect to the one of c).

Hint: $Speedup = \frac{T^{(2)}(n)}{T^{(3)}(n)}$

(optional) Develop a cache-blocking version of the algorithm of d) and formulate its runtime $T^{(4)}(n)$ (cf. chapter 4 slide 38 f.).

Assume that the following relations hold for an imaginary processor type:

$$\begin{aligned} T_M &= 180\,T_A \\ T_C &= 35\,T_A \\ l &= 8. \end{aligned}$$

Problem 2. Performance analysis tools
If you decided to solve Problem 1 by pen and paper, use the provided binaries in the `binary` directory in `exercise2.tar.gz`. Check your execution rights if you have problems executing them (`chmod u+x ./*.exe`). Otherwise use your own code.

Problem 2.1. Profiling with gprof
You may run `make gprof` **or** execute `binary/norm.gprof.exe` and inspect the result with `gprof binary/norm.gprof.exe | less`
What is the time for the various algorithms according to gprof?
Try the target gprof2: `make gprof2` **or** execute `binary/norm.gprof2.exe` and inspect the result with `gprof binary/norm.gprof2.exe | less`
What is the difference of the outputs (You may have a look at the gprof targets in the Makefile)? What is the calculation time for the algorithms?

Problem 2.2. Hardware Performance Counter
Modern processor architectures have special-purpose registers to store the counts of hardware-related activities. These so called hardware performance counters can be used for low-level performance analysis or tuning. For accessing the hardware performance counters you can use the tool likwid[1] which is installed on the RWTH Compute Cluster.
Note 1: For this task you can use your own implementation of a), c), and d) from the exercise *norm calculation of a matrix* and build them with the Intel compiler (`make single` in the provided template) or use the provided binaries (norm_max.exe, norm_1_col.exe, and norm_1_row.exe).
Note 2: Use the special tuning node `login18-t.hpc.itc.rwth-aachen.de` for your measurements. Make sure that you are the only user on the system (e.g., with the `w` command) to obtain reliable results.

1. Examine the different programs with hardware counters using `likwid-perfctr`:
   ```
   module load likwid
   likwid-perfctr -C <processor id> -g <performance group> norm_max.exe
   ```
   Use the performance group `MEM_DP` for your investigations.

2. Analysis of the results. Likwid derives different metric from the counted events. Answer the following questions:

[1] https://github.com/RRZE-HPC/likwid

- Let be $n = 2^{15} = 32768$. Determine the read date volume D_{read} for the computation of $\|A\|_\infty$? What is reported by likwid? Why do these values differ?
- Which floating point performance (in `MFLOP/s`) is reported by likwid? Why is it not as high as reported by the norm application?
- Which memory bandwidth is reported by likwid? How can you assess whether the value is reasonable?

Problem 3. Balance Metric
The balance metric is applied for performance modeling.

Problem 3.1. Machine Balance
Your reference machine is a 2-socket Intel Skylake architecture with the following characteristics:

- 48 cores in total
- 2.1 GHz clock frequency
- 2 fused multiply add (FMA) units per core (each unit: 1 ADD & 1 MULT per cycle)
- FMA units operate on AVX-512 registers (512 bits)
- Cache sizes: L1 $\widehat{=}$ 32 KiB, L2 $\widehat{=}$ 1 MiB, L3 $\widehat{=}$ 33 MiB
- Sustainable main memory bandwidth gained by Stream benchmark (cf. exercise 1)

Determine the machine balance B_m of this architecture (double-precision operations). With respect to the *norm calculation of a matrix*, which performance limits could be adapted?

Problem 3.2. Code Balance
Determine the code balance B_c of the $\|A\|_\infty$ computation.

Hint: Think about which elements are located in registers, caches and main memory. Assume that the access to main memory is the slowest data path. Now neglect the floating-point comparison (i.e., do not count it as floating-point operation). Furthermore, neglect the (statistical) sign flipping in the abs-function.

Problem 3.3. Lightspeed
Determine the (relative) lightspeed l of the $\|A\|_\infty$ computation on the given architecture. How do you interpret this value? In general, how can a lightspeed value be improved by the application developer?
Now, compute the lightspeed P for absolute performance in GFlop/s. You can take this value to evaluate the real-world performance. The real-world performance can be measured with any performance analysis tool. You have already used the likwid tool for that. Compare the measured performance [GFlop/s] (from the previously-done likwid *MEM_DP* run) with the computed theoretical performance. How close is the code to what it can reach at maximum? Why can there be a difference between theoretical and experimentally-gathered results?

Discussion of this exercise on November 12, 2019.