Hardware Aware Scientifc Computing (WS 2023) **Exercise 1**

Prof. Dr. Peter Bastian, Santiago Ospina De Los Ríos Submission date: 2023-11-06

IWR, Universität Heidelberg

Exercise 1 *Pointer Chasing*

You can find the code for this exercise in `pointer_chasing.cc` in the `hasc-code` git repository.

a) Read through the code and try to understand what is going on. Try to answer the following questions:

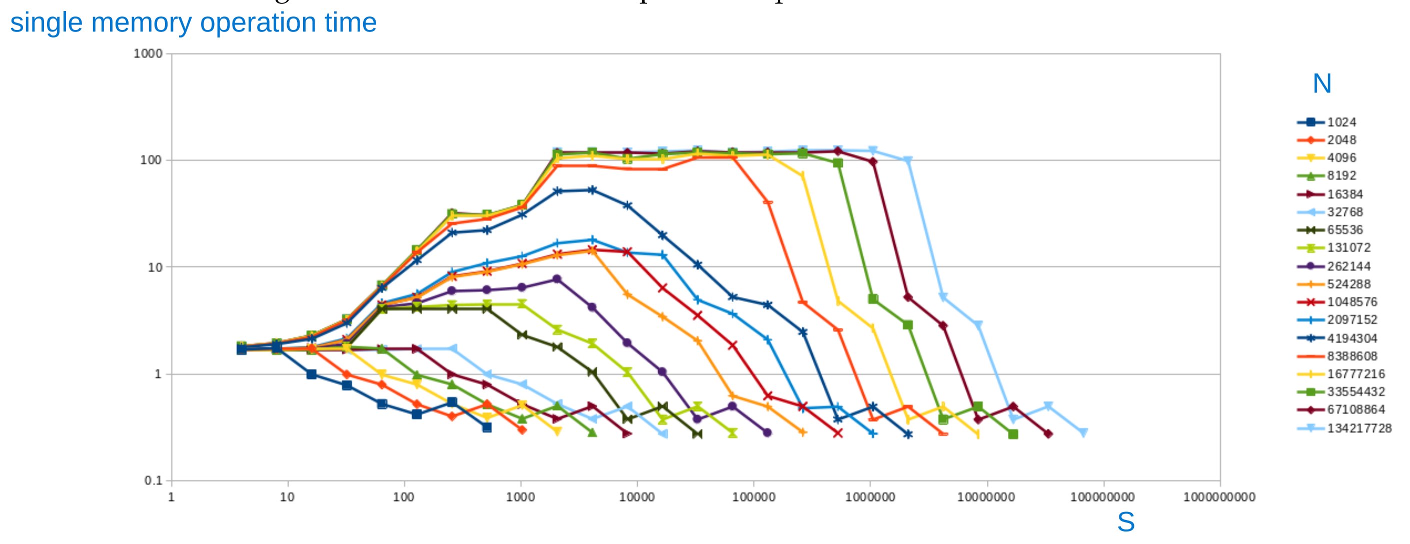
- What happens in the main experiment?
- What is the purpose of the empty experiment and why do we measure its time?
- Why do we include the loop `for (int k=0; k<s; k++)`?

b) Try to run the program on your machine. Running this (with a smaller max size `N`) should produce something like this:

```
N, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384
1024, 1.73572, 1.78134, 0.91246, 0.625306, 0.474395, 0.437216, 0.473088, 0.718708
2048, 1.7323, 1.74601, 1.7695, 0.880862, 0.602514, 0.4341, 0.415038, 0.447265, 0.681217
4096, 1.73704, 1.7295, 1.74095, 1.77269, 0.902484, 0.62297, 0.432622, 0.406638, 0.439876, 0.679652
8192, 1.7197, 1.71312, 1.73604, 1.74524, 1.76319, 0.877842, 0.606251, 0.432656, 0.406274, 0.43031, 0.670469
16384, 1.72204, 1.70714, 1.71682, 1.7237, 1.75665, 1.77592, 0.896538, 0.608088, 0.414412, 0.399741, 0.421001, 0.659733
```

Fix all problems you encounter when you try to create reasonable numbers[2]. You can also try different compilers and see if it makes a difference.

c) Produce a log-log plot that plots the time[3] over the stride for all the different array sizes[4]. Using the original max size `N` this should produce a plot like this:



For different hardware the plot will look different but it should show similar behavior.

d) Now comes the main part of the exercise. Try to find a relation between your plot and the following specifications of your CPU: L1 cache size, L2 cache size, L3 cache size and clock rate. Can you explain what happens at the data points, that take the longest time.

(3 Points)

[2]We are aware that there are problems for certain compilers, so fixing this is part of the exercise ;)
[3]More precisely: Time per memory access, which is exactly what the program reports
[4]If you have a big L3 cache you should increase the max size further to get a plateau at the top

Exercise 2 *Peak Performance*

If we want to find out if we have a good implementation we often measure times. This is a good approach for comparing different implementations on the same machine. If you have two implementations A and B that do the same thing and A is ten times faster we are of course happy and use A. Unfortunately A could still be a very bad implementation and B was just even worse.

For this reason it can make sense to look at additional metrics. Two other metrics one can look at are the operation throughput (GFlops/s) or the transferred memory (GB/s). Comparing these numbers to the peak performance of your CPU or the memory bandwidth can give insight, if our code performs well or not.

a) Find out what the peak performance of your processor is. Try to find your peak performance if you use only a single core and for the case that you use all cores. You can try to find these numbers in data sheets of the CPU-vendor or by finding and running benchmark problems.

b) Try to understand where these numbers come from and try to calculate/approximate them by a simple computation using your processor specifications (like clockrate, SIMD width, ...).

(2 Points)

Exercise 3 *Vectorized Numerical Integration*

In this exercise we will implement the midpoint rule for numerical approximation of integrals. We will compare the runtimes of a unvectorized and a vectorized implementation.

The midpoint rule approximates $\int_a^b f(x)\,\mathrm{d}x$ by partitioning the domain $[a, b]$ into n equidistant intervals $[x_i, x_{i+1}]$. On each interval we choose the midpoint value $\bar{x}_i := (x_i + x_{i+1})/2$ to make a quadrature evaluation of the function. Then, the approximation follows

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{b-a}{n} \sum_{i=0}^{n-1} f(\bar{x}_i). \tag{1}$$

a) Implement the midpoint rule for a parameter[5] n and approximate $\int_0^1 f(x)\,\mathrm{d}x$ for the two functions $f(x) = x^3 - 2x^2 + 3x - 1$ and $f(x) = \sum_{i=0}^{15} x^i$.

b) Implement a vectorized version of your functions.

d) Measure the runtimes for different numbers of intervals and compare your implementations.

e) Try to estimate/approximate the number of operations your program does per time (GFlops/s). Why might this be tricky? pow, horizontal sum

(5 Points)

[5]This may not need to be a hard requirement, in other words, you are allowed to partition $[a, b]$ with slightly more intervals