University of Zurich^UZH

Department of Physics **Scientific Programming with Python**

Hardware Speedup Exercises

September 6, 2017

Exercise 1: Optimizing arithmetic expressions

1. Use script `poly.py` to check how much time it takes to evaluate the next polynomial:
   `y = .25*x**3 + .75*x**2 - 1.5*x - 2`
   with x in the range [-1, 1], and with 10 millions points.

   - Set the 'what' parameter to "numexpr" and take note of the speed-up versus the "numpy" case. Why do you think the speed-up is so large?

2. The expression below:
   `y = ((.25*x + .75)*x - 1.5)*x - 2`
   represents the same polynomial than the original one, but with some interesting side-effects in efficiency. Repeat this computation for numpy and numexpr and get your own conclusions.

   - Why do you think numpy is doing much more efficiently with this new expression?
   - Why the speed-up in numexpr is not so high in comparison?
   - Why numexpr continues to be faster than numpy?

3. The C program `poly.c` does the same computation than above, but in pure C. Compile it like this:
   `gcc -O3 -o poly poly.c -lm`
   and execute it.

   - Why do you think it is more efficient than the above approaches?

Exercise 2: Evaluating transcendental functions

4. Activate the evaluation of the `sin(x)**2+cos(x)**2` expression in `poly.py`, a function that includes transcendental functions and run the script.

   - Why the difference in time between NumPy and Numexpr is so small?

5. In `poly.c`, comment out expression 1) (around line 51) and uncomment expression 3) (the transcendental function).

   - Do this pure C approaches go faster than the Python-based ones?
   - What would be needed to accelerate the computations?

Exercise 3: Using Numba

The goal of Numba is to compile arbitrarily complex Python code on-the-flight and executing it for you. It is fast, although one should take in account the compile times.

6. Edit `poly-numba.py` and look at how numba works.

   - Run several expressions and determine which method is faster. What is the compilation time for numba and how it compares with the execution time?
   - Raise the amount of data points to 100 millions. What happens?

Exercise 4: Parallelism

7. Be sure that you are on a multi-processor machine and activate the:
   `y = ((.25*x + .75)*x - 1.5)*x - 2`
   expression in `poly-mp.py`. Repeat the computation for both numpy and numexpr for a different number of processes (numpy) or threads (numexpr)
   (pass the desired number as a parameter to the script).

   - How does the efficiency scale?
   - Why do you think it scales that way?
   - How is the performance compared with the pure C computation?

8. With the previous examples, compute the expression:
   `y = x`
   That is, do a simple copy of the 'x' vector. What is the performance that you are seeing?

   - How does it evolve when using different threads? Why it scales very similarly than the polynomial evaluation?
   - Could you have a guess at the memory bandwidth of this machine?