Preferred Networks Internship 2019 Coding Test (Performance Optimization Field)

Notes

- There is no restriction on the implementation language and libraries. Please specify versions of the languages and libraries you used in the report.
- The attached Python scripts are tested on Python 3.7.2, ply 3.11, numpy 1.16.2, and scipy 1.2.1.
- You may re-implement the interpreter because it is slow.
- Do not share or discuss any details of this coding task with anyone.
- Please tackle the task by yourself. Do not share or discuss this coding task with anyone including other applicants. Especially, do not upload your solution and/or problem description to public repository of GitHub during screening period. If we find any evidence of leakage, the applicant will be disqualified. If one applicant allows another applicant to copy answers, both applicants will be disqualified.
- We expect you to spend up to two days for this task. You can submit your work without solving all of the problems. Please do your best without neglecting your coursework.

Things to submit

- Reports for problems 1 and 2
- Source code for problems 1 and 2

Evaluation

It is desirable that your submission satisfies the following. (This is not mandatory. Your submission does not need to satisfy all of them if you are too busy.)

- The source code is readable for other programmers.
- There is an appropriate amount of unit tests and/or verification code to ensure the source code is correct.
- The submitted report is concise and easy to follow.

During the interview after the first screening process, we may ask some questions on your source code.

How to submit

- Create a zip archive with all of the submission materials and submit it on this form. Due date is Tuesday, May 7th, 2019, 12:00 JST.

Inquiry

- If you have any questions regarding this problem description, please contact us at intern2019-admin@preferred.jp. An updated version will be shared with all applicants if any changes are made. Note that we cannot comment on the approach or give hints to the solution.

Task description

Problem 1

Discrete Fourier Transform (DFT) is an operation calculating $x \in \mathbb{C}^N \mapsto y \in \mathbb{C}^N$, $y(l) = \sum_{k=0}^{N-1} x(k) \exp\left(-2\pi i \frac{kl}{N}\right)$ $(l = 0, \ldots, N-1)$. We define a simple language to describe a program to calculate the DFT as follows.

```
<statement> ::= <ident> = <expression>  (assignment to a scalar variable)
                        | <ident>[<int>] = <expression>  (assignment to an array element)
<expression> ::= <primitive> + <primitive> (summation)
                          | <primitive> - <primitive> (subtraction)
                          | <primitive> * <primitive> (multiplication)
                          | <primitive>
<primitive> ::= f(<int>, <int>)  (for input m and n, returns exp(-2πi m / n))
                      | <ident>  (read from a scalar variable)
                      | <ident>[<int>]  (read from an array element)
                      | <const>  (constant)
<ident> ::= [a-zA-Z_][a-zA-Z0-9_]*  (name of a variable)
<int> ::= -?\d+  (integer)
<const> ::= \([\djeE.+-]+\)  (complex valued constant, interpreted by complex() of
Python)
```

The attached interpreter.py is an interpreter of this simple language (requires Python packages ply, numpy, and scipy). When you invoke it with an input function -i {const,sin,sawtooth,chirp} and a dimension N, it initializes the input array x and the output array y with values of the specified function and zeroes respectively, interpretes and executes a program described in the simple language, and finally prints the contents of y and the error of y compared to the correct answer. Values of `<primitive>` and `<expression>` are complex. You cannot define arrays other than x and y. Instead you can define an arbitrary number of new variables.

The attached naive.py is a naive implementation of DFT in the simple language. When you invoke it with N, it prints an implementation to standard output.

Here is a usage example of the scripts.

```
$ python naive.py 2 > 2.txt
$ cat 2.txt
t = x[0] * f(0, 2)
y[0] = y[0] + t
t = x[1] * f(0, 2)
y[0] = y[0] + t
t = x[0] * f(0, 2)
y[1] = y[1] + t
t = x[1] * f(1, 2)
y[1] = y[1] + t
$ cat 2.txt | python interpreter.py -i const 2
(1+1j)
(5.551115123125783e-17-1.1102230246251565e-16j)
max_error: 1.2412670766236366e-16
$ cat 2.txt | python interpreter.py -i sin 2
```

```
(-1.8369701987210297e-16+0.8290816487408506j)
(2.2699482268387765e-16+0.12197486755430303j)
max_error: 4.3297802811774677e-17
```

We also define the standard cost of a program in the simple language as sum of 1 for each summation and subtraction, and 2 for each multiplication. The attached cost.py calculates the standard cost as follows.

```
$ cat 2.txt | python cost.py
12
```

You can reduce the computational amount of DFTs dramatically by utilizing $\exp(a+b) = \exp(a)\exp(b)$ and other relationships. Write a program to output DFT programs in the simple language optimized in the sense of standard cost for Ns in 2 ~ 256. Report how you optimized your implementation and the sum of standard cost of the programs for Ns in 2 ~ 256. The errors printed by interpreter.py (at the line of `max_error:`) should be less than 10^{-8} for all the initialization methods (const, sin, sawtooth, chirp) and for all the Ns.

Problem 2

The actual elapsed time on real-world computers is determined in a more complicated manner than the above definition of standard cost. Define another cost concept reflecting some effects occuring in real-world computers. Then write a program to output DFT programs optimized in the sense of your cost definition and report how you optimized your implementation as well as the sum of the cost of the programs for Ns in 2~256 using the cost you defined before in this problem. The error condition is also same as Problem 1. You may change the syntax of the simple language so that it can naturally express the effects you introduced.

Hint: For example, in the cost definition, you can consider a lower cost in multiplication with values of f() for specific inputs, or parallel execution of some instructions depending on (in)dependency among them, or register usage efficiency with your own definition of a register file structure and register allocation method.

End of coding test.