Weekly Work Report

Author Name

LeiChao

2021-08-[02, 08]

1 Gurobi Practise

1.1 Cash-Flow Matching Problem

Cash-flow mastching is a investment strategy which could be wilded by investor to manage their future available cash-flow. In a other word, CASH-Flow Matching is a structure investment portfolio.

1.2 Modeling Cash-Flow Problem

Assuming A cooporation's cash flow requirement showed in 1.

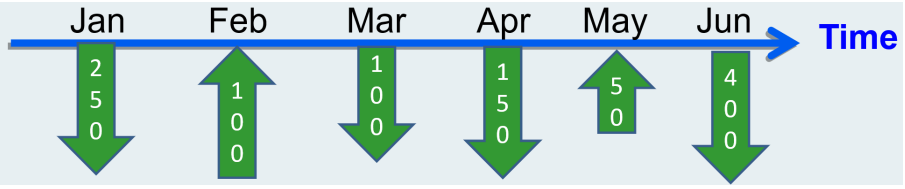

Figure 1: Cash-flow-requirement

A Corp has three kinds of invesment options.

option 1 Get a loan up to 150w which need to pay 1.5% per month interest from bank at the start of the month.

option 2 A corp could sell its 60-days commercial paper to the public, the total interest is 2%.

option 3 A invest the rest of the money of last month in a stable 0.5% interest per month.

1.3 Problem Analyse

Define parameter.

- X_i: money loaned from the bank at the begining of every month.$i = [1, 2, 3, 4, 5, 6]$
- Y_i: money from commercial paper. $i = [1, 2, 3, 4]$.
- Z_i: Rest money of each month. $i = [1, 2, 3, 4, 5, 6]$.

Objective function: maximize Z6.
Tips: A couldn't sell commercial paper in the real world, A can not loan from bank in 6-th month in order to bound the problem.

1.4 Using Gurobi to solve this LP problem

Coding file: coding.lp
Resulting file: resulting.sol

1.5 Conclude

I get some basic Cash-Flow Mathcing strategy through this experiemt, at the same time, I get familar with the modeling process of LP in **Gurobi**: Defining problem, Modeling, Solving, Testing, Executing.

2 CUDA GPU

2.1 GPU Thread structure

Content & Note.

2.2 Cuda stream Configuration

Just a notion, CUDA processing stream are seperated into default stream and self-defined stream. Default stream will halt all other self-defined which means Default stream are mutually exclusive with other self-defined stream. But self-defined streams can run simultaneously.

3 Solving mandelbrot set through multiple core using Pthread.

3.1 Problem background

We should caculate mandelbrot value for every single pixel in the image. In other word, Each pixel in the image corresponds to a value in the complex plane, and the brightness of each pixel is proportional to the computational cost of determining whether the value is contained in the Mandelbrot set—white pixels required the maximum (256) number of iterations, black ones only a few iterations, and gray pixels were somewhere in between.

The modified code is located in mandelbrot.cpp
The generated graph is located in serial ppm and multiple-thread ppm
The performance report is located in report.txt

3.2 understanding

Since multiple threads need cross-cores communication. So the speedup can not break into the upper ideal limit.

4 Vectorizing Code Using SIMD Intrinsics

4.1 Background

Originally, This experienment should craft an implementation using the SSE or AVX vector instrinsics that map to real vector instructions on moderns CPUs. But that's so complicated that cover the essence of the coding thought. To get the trade-off, I use the CMU-Simulation to finish the experienment.

SIMD: (Single Instruction Multiple Data) To illustrate this concept, A single CPU core is showed in 2. As we can see, there are two **Fetch/Decode** elements which can fetch and decode instruction from instruction stream seperately and simultaneously. On the right hand side located in the most important unit in the SIMD, 8 yellow cube called **executation unit**. Once the Fetch/Decode unit Got the instruction, 8 piceces of data are operated by these executation unit simultaneously.

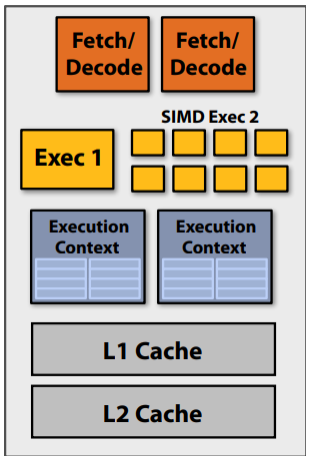

Figure 2: Single CPU Core

MASK: How about the conditional executation? SIMD just operate 8 piceces simultaneously, we should specify or said, **SETTING MASK** to specify which elements we want to operate. The example showed in figure 3.

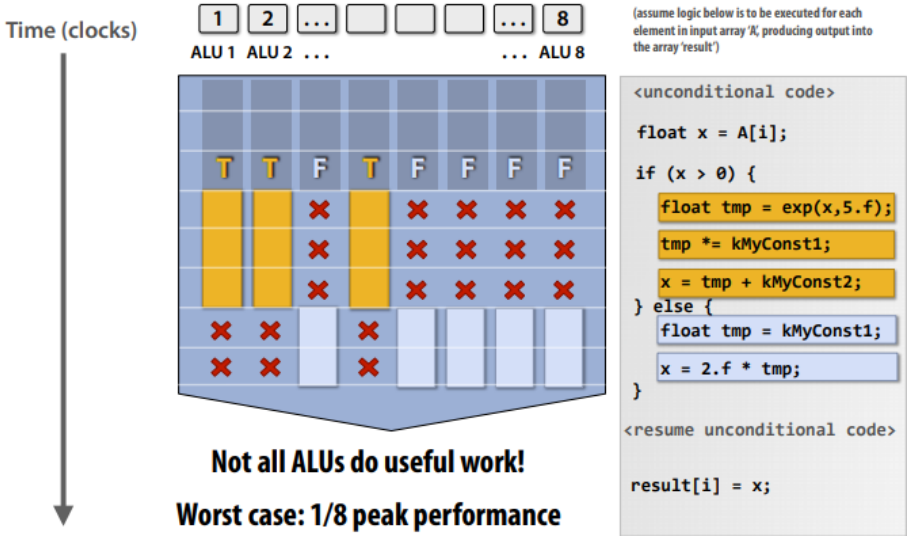

Figure 3: SIMD MASK

4.2 Implementation

Based on the set of instructions provided by CMU which located in CMU418intrin.h and CMU418intrin.cpp. My implementation of vector-adding in SIMD is located in functions::clampedExpVector.

4.3 performance And Question Report

Performance & Question Report is located in GitHub report.

5 ISPC Experienment

5.1 Background

ISPC–(Implicit Simple program multiple data Programing Compiler) is a compiler made by INTEL provide us an efficient way to implement SIMD prgrams running in the single/multiple cores.

5.2 Experienment Content

The purpose is to get familar with ISPC by implement a program to caculate mandelbrot set. In ISPC program, **We assign data overall instead of assigning every data to each thread/execution unit explicitly** using keyword **forall**.

5.3 Implementation

The Ispc code is located in mandelbrot.iscp.

5.4 Result Report

Result Report is located in report.txt

5.4.1 Determine how many tasks to creat And Cope with Bound area

A: using script to change the parameter of threadCount, Drawing graph based on these different threadCount to get the optimize point. Dealing with the situation that the rows in the image is not divisible by the number of tasks.

A: EASY!!! check out mandelbrot.ispc::mandelbrot_ispc_task, it has already copying with the fontier of last segment, so what we need to do in mandelbrot.ispc::mandelbrot_ispc_withtasks is only determine whether adding or not a thread which used to caculate the extra part of data.

5.4.2 the difference bwtween foreach and launch

In terms of my view from now on, foreach is running just like the serial function. the difference bwtween serial and foreach is their running CPU, the first is common cpu, the latter is vector cpu. The reason that vector cpu could sppedup is that the cooprating unit in vector cpu is VECTOR. LAUNCH is more like creating multiple threads running on different cores. so it's at the same level of multiple threads. From simplicity, foreach is still a single thread, LAUNCH is already in the level of multiple threads.