GEMM in Multicore Arduinos

Liem Radita Tapaning Hesti

January 26, 2017

Background

This GEMM in Multicore Arduinos is inspired by lab 4 Parallel Computing homework. The third question was about how to make multicore Arduinos from Iteaduino and there’s follow up question about build it for real. I thought it will be fun since I want project that’s related to hardware and there’s a lot of thread about multicore Ardunios and interesting Kickstarter project named Fireduino, it made me curious how difficult it will be.

![Fireduino board](image)

Figure 1: Fireduino board. Fireduino is intelligence hardware open source platform, the core MCU is a dual Cortex-M3 kernel Soc, has the on-board wifi(AP6181) module.
Matrix multiplication used in this project because it’s like the basic of understanding parallel process. It’s simple enough to understand and to be implemented in Arduino machine.

Project

The project consists of two phases: building multicore Arduinos and deploying GEMM calculation in this multicore Arduinos

- **Building multicore Arduinos**

  This phase turns out not really difficult at all, it only needs 3 jumper wires to connect Arduino I2C pins (SDA & SCL pins) and ground. After that both Arduinos should be connected to power source. Arduino program will detect both Arduinos (in this case, I’m using Iteaduino from the university)

  Arduino has Wire library [2] already embedded there. There’s tutorial about how to set the Arduino as master and the other as slave. There’s also tutorial how to passing message from master to slave and slave to master.

  Using Wire library, this part is quite easy. The hard part is to connect both Arduinos using Assembly to have faster result. Some projects to replace Wire library are already available [3] and can be used but not working as expected (can’t figure it out how to send and receive message between slave and master) so this exploration is halted.

- **Deploying GEMM in Multicore Arduinos**

  To build GEMM in Arduinos, there are several researches made to find common method in parallelizing GEMM. There are :

  - Block stripe decomposition
  - Fox’s method
  - Cannon’s method

  All of them are basically splitting matrix data and spread it into multiple processes (can be in the form of thread, or to other nodes) and later collect and summarize the results. The differences is in how they split the data (one matrix or both matrix) and how they join the data (since the result will move around and need to be rearranged into correct result matrix) [4]

  Like every other parallelization process, it’s started by writing the serial algorithm. I used the common serial algorithm from the 4th lab homework and measure time spent to do the serial calculation (0.472 secs - including printing the end result matrix). The matrix itself is matrix 10 x 10 multiplied by another matrix 10 x 10
Figure 2: Two Arduinos connected with jumper wire (with orange color) in port SDA & SCL. It’s connected with USB cable to the computer to upload Arduino program.

```c
for (int rep = 0; rep < 10; rep++){
    for (int i = 0; i < n; i++){
        for (int j = 0; j < n; j++){
            for (int k = 0; k < n; k++){
                C[i+j*n] += A[i+k*n]*B[k+j*n];
            }
        }
    }
}
```

Figure 3: Serial implementation

Later, I’m trying to split the data from both matrix 10 x 10 to do block stripe decomposition into checkerboard form [b]. The matrix split into four calculation processes so each Arduinos will need to handle two calculations like illustrated below:
Figure 4: checkerboard decomposition into four parts, two calculated in slave arduino, two calculated in master Arduino

Data Distribution Process

Master Arduino’s tasks are to sending data (value that will be calculated) to slave, receiving result from slave, and summarize with result from master to get the final result. Slave Arduino’s tasks are to receive data from master, calculate two small matrix (matrix 5 x 5) and send calculation result to master.

In the Gitlab code, values that will be calculated in both Arduinos are already specified. The task of slave is to receive request about which part of data (in form of index number of matrix that need to be filled) needs to be sent to master and later sending calculation as requested.

Communication Process

I found problem when trying to send data back and forth. Listening process in slave and master is running in loop process since the device must keep listening for new request / data sent. There’s also possibility (and happened several times during running the program but I think it’s arbitrary) that master / slave didn’t receive any message (maybe because data not sent) so I let the process to be redundant until all requested data filled.

The end result calculation from distribution until summarization is 0.243 s (including printing end result matrix). It’s best to measure time for each process independently to figure it out why it’s only 2x faster compared to single core Arduino

Conclusion & Improvements
By using dual core Arduinos and Wire library, calculation can be two times faster. Personally, this experimentation is not really satisfying for me due to my limited knowledge about hardware and programming in Arduinos.

There are also several actions that can be added to further understand and to increase the performance:

- Using MatrixMath library [6] to run the matrix multiplication calculation. I’m curious whether it will make calculation faster even with the same data distribution method (using Wire to listen and request). I have some hunch that the calculation itself can be more optimized (one of the option is by using loop unrolling)

- Using protothread [7] to calculate processes in each Arduino. Currently calculating two parts of the matrix for each Arduinos is still in serial so it’s possible that it can be made ‘parallel’ by using protothreading. It also worth to compare speed result of running protothreading to calculate the matrix in single Arduino with in multicore.

References


