

Scott R. Armstrong
sarmstr2
CS433
ILP Project

1. Project Description

I plan to employ and document static Instruction-Level Parallelism (ILP) approaches to improve performance of a brute-force prime number calculator, initially for ARM Instruction Set Architectures (ISA), and possibly also for the PowerPC ISA. The static ILP approaches to improve performance would include rewriting portions of the program using assembly language to employ different branching alternatives, loop unrolling, single-instruction multiple-loads, instruction reordering, and depending on the availability of registers, register renaming. Though the processing within the main loop isn't extremely complicated, I'll also look for opportunities to exploit data dependencies. This program will provide some interesting challenges in exploiting branch predictions because of the irregularity of prime numbers occurring within a sequential set of numbers.

I will use the remote eCos environment to modify the Assembly language instructions for this program.

During the process of improving the performance of this program, I will also identify any performance features that may have been realized by redesigning the higher-level code, and I will compare those performance enhancements with similar Assembly language modifications.

Since this program was initially written to be extremely taxing on a processor, I expect to be able to find some verifiable performance enhancements by using static ILP.

2. Program description

The program used for this project was written in 2003 using C++. It is based on an earlier version that I wrote in the 1990s using C to exercise some examples in the Kernighan and Ritchie book, *The C Programming Language*. This was an interesting program prior to the recent generations of processors, as it could demonstrate how quickly a computer processor could be brought to a crawl by a program. As a result, I intentionally made no attempts to improve the program's performance. The program uses division, modulo arithmetic, Boolean logic, and addition to check whether sequential integers between 2 and a user-supplied positive number satisfy the criteria for a Prime number.

3. Program listing

This C++ version was originally written using the Borland command line C++ compiler, imported into the Apple XCode IDE and recompiled for use on Apple computers.

```

// *****
// Scott R. Armstrong
// June, 2003
//
// No error checking is included to avoid errors caused by
// non-numeric or negative input when prompted for a value.
//
// Programming environment: Borland C++, Version 5.5, Win32
// *****

#include <iostream> // I/O preprocessor header file
#include <cstdlib> // for portability with EXIT_SUCCESS
#include <cmath> // for math functions
#include <iomanip> // for fixed**, setprecision()
#include <fstream> // for ifstream, ofstream

using namespace std;

int main()
{

// declaration of variables
    bool    prime, repeat;
    char    ch;
    int     x, y, z;

// function prototype
    void GetReply(bool&); // location reference

    repeat = true; // preset the boolean value
    while ( repeat )
    {
        cout << endl;
        cout << "Prime Number Calculator" << endl;
        cout << "=====" << endl;
        cout << "Enter a positive integer: ";

        cin >> z;

        for ( y = 2; y <= z; y++)
        {
            prime = true;
            x = 2;
            while ( (x <= y/2 ) && (prime == true) )
            {

//      cout << (y % x) << "\n";

```

```

        if ((y % x) == 0 )
        {
            prime = false;
        }
        else
        {
            x++;
        }
    }

    if (prime == true)
    {
        cout << y << " is a prime number.\n";
    }
    //     else
    //     {
    //         cout << y << "...is not a prime number";
    //     }
}

cout << endl;
cout << "Do you want to continue (Y or N)? ";
GetReply(repeat);
if ( !repeat )
    cout << endl << "Quitting the Prime Number Calculator"
        << endl;

} // end of while loop

return EXIT_SUCCESS;
}

// *****
// Function:  GetReply
//
// Parameters:  boolean reference
//
// This function reads from the standard input, confirms that
// the input is a 'Y', 'y', 'N', or 'n', and returns the
// reference to the boolean passed from main, true if yes,
// false if no.
//
// If the input is not in the set listed, an error message is
// displayed and the user is prompted again to input a valid response.
//
// *****

```

```

void GetReply(bool& returnBool)
{
    char charInput;
    bool loopRepeat;

    loopRepeat = true;
    while ( loopRepeat )
    {

// check for single character only or read to file??

// while not end of line, getchar to charInput...then the rest
// this reads all of the input to the end...last character wins

        cin >> charInput;
//        cin.ignore(80, '\n');    // to rid the evil extra return
//        cin.get(charInput);
        if (charInput == 'Y' || charInput == 'y')
        {
            loopRepeat = false;    // exit the while loop
            returnBool = true;    // for yes in main
        }
        else if (charInput == 'N' || charInput == 'n')
        {
            loopRepeat = false;    // exit the while loop
            returnBool = false;    // for no in main
        }
        else
            cout << endl << "Please enter only the letter Y or N. ";

        cin.ignore(80, '\n'); // ignore anything beyond the first
character
    } // end of while loop
}

```