

README: ICS 432 project

GHZ_application_V0: THE ORIGINAL FILE

-no changes:

-time trials:

- | | |
|------------------|-------------------|
| 1) 91.87 seconds | 6) 88.41 seconds |
| 2) 89.51 seconds | 7) 88.72 seconds |
| 3) 89.00 seconds | 8) 87.82 seconds |
| 4) 88.85 seconds | 9) 89.49 seconds |
| 5) 88.76 seconds | 10) 88.95 seconds |

-average time: 89.138 seconds

GHZ_application_V1: -pg flag**-time trials:**

- | | |
|------------------|-------------------|
| 1) 97.47 seconds | 6) 94.96 seconds |
| 2) 94.65 seconds | 7) 95.49 seconds |
| 3) 98.38 seconds | 8) 95.47 seconds |
| 4) 95.19 seconds | 9) 94.23 seconds |
| 5) 94.53 seconds | 10) 95.54 seconds |

-average time: 95.591 seconds

-speed boost: no boost

-results:

As we can see with changing the flag to -pg the program actually loses speed. On average, this slowdown is by a dramatic 6 whole seconds. We see this trend because -pg is used to figure out what functions are being used to most. So while it is running the code normally, it is also computing which functions are the most expensive.



```
CFLAGS = -pg

default: app

run:
    ./app 0.18

app: main.cpp
    g++ $(CFLAGS) main.cpp -o app

clean:
    /bin/rm -f */*.o *.o app GHZ_output.txt
```

GHZ_application_V2: -Ofast flag

-added the -Ofast compiler flag

-time trials:

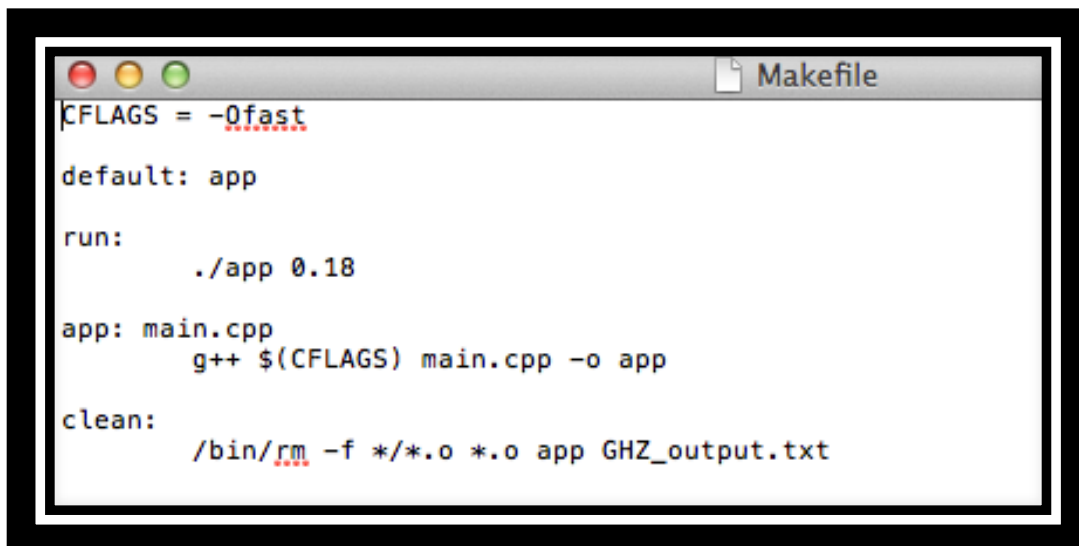
- | | |
|------------------|-------------------|
| 1) 20.62 seconds | 6) 20.10 seconds |
| 2) 20.30 seconds | 7) 20.28 seconds |
| 3) 20.08 seconds | 8) 20.18 seconds |
| 4) 20.15 seconds | 9) 20.12 seconds |
| 5) 20.09 seconds | 10) 20.15 seconds |

-average time: 20.207 seconds

-speed boost: around 4.411% boost from the original code

-results:

as we can see, the code get faster because we optimized the compiler by using the -Ofast command. This change was pretty much given from the lecture notes.



```
Makefile
CFLAGS = -Ofast

default: app

run:
    ./app 0.18

app: main.cpp
    g++ $(CFLAGS) main.cpp -o app

clean:
    /bin/rm -f */*.o *.o app GHZ_output.txt
```

GHZ_application_V3: library change + -Ofast flag

-changed the 'iostream' library to 'cstdio' library

-time trials:

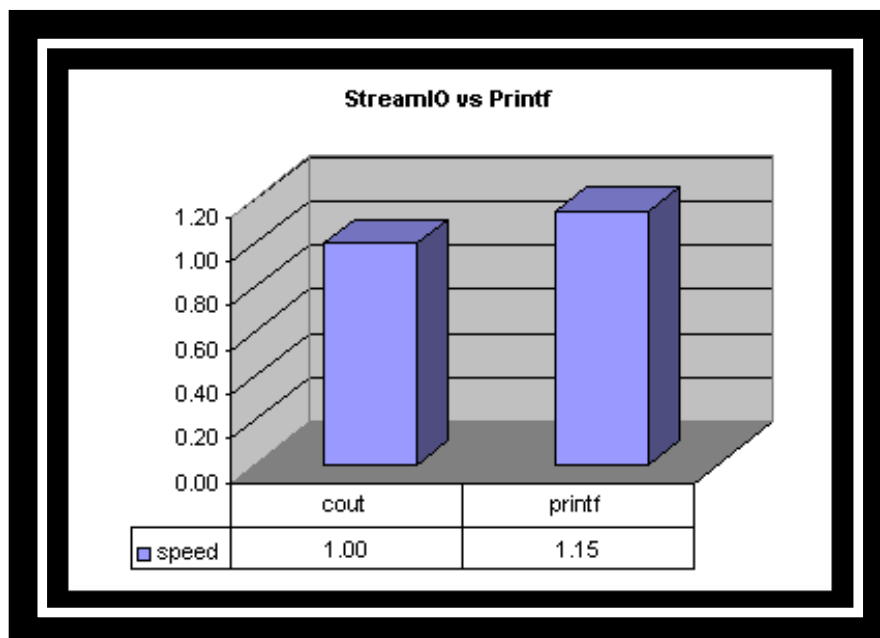
- | | |
|------------------|-------------------|
| 1) 20.70 seconds | 6) 19.86 seconds |
| 2) 20.04 seconds | 7) 19.82 seconds |
| 3) 19.87 seconds | 8) 19.95 seconds |
| 4) 19.79 seconds | 9) 19.81 seconds |
| 5) 19.84 seconds | 10) 19.84 seconds |

-average time: 19.952 seconds

-speed boost: around 4.467% boost from the original code

results:

The reason for the library switch is because although the C++ stream I/O is a very flexible and safe method of doing input and output and it allows you to define specific output formats for your own objects. The cstdio library's printf on the other hand, is not very safe. If you specify the wrong number of parameters, or give the wrong order, your crash. You can't define new output formats, either. But printf does have a few advantages to it; printf is easy to read, easy to use, and most important, fast. According to this picture below.



```

//outputs stats to file
void output_stats()
{
    FILE *GHZ_output;
    GHZ_output = fopen("GHZ_output.txt", "w");
    fprintf(GHZ_output, "column_number, radial distance, total stars, total
sterilizations- at least sterilized once\n");

    for (int i=1; i<=count_xy_glbl; ++i)
    {
        fprintf(GHZ_output, "%d,", i);
        fprintf(GHZ_output, "%.f,", stats[i]->distance);
        fprintf(GHZ_output, "%lld,", stats[i]->total_stars*360);
        fprintf(GHZ_output, "%lld\n", stats[i]->total_sterilizations*360);
        //fprintf(GHZ_output, "%d,", endl);
    }

    fprintf(GHZ_output, "The following results are multiplied by 360 such that they
apply to the entire Galaxy\n Number of TypeII: %llu", num_sn_glbl*360);
    fprintf(GHZ_output, " Number of TypeIa: %llu", num_sn_Ia_glbl*360);
    fprintf(GHZ_output, " Total Sterilizations SNII: %lu", total_sterilizations_II*
360);
    fprintf(GHZ_output, " Total Sterilizations Ia: %lu", total_sterilizations_Ia*360
);
    fprintf(GHZ_output, " Total stellar mass: %Lf", total_stellar_mass_glbl*360);
    fprintf(GHZ_output, "Number of SNII: %llu", num_SNII_specific*360);
    fprintf(GHZ_output, "\nNumber of SNIa: %llu", num_SNIa_specific*360);

    float slice_radial_distance=subsection_slice_radial_distance;
    float slice_h_z_distance=subsection_slice_h_z_distance;
    int z_cell_processed=0;

    fclose(GHZ_output);

    /* ofstream GHZ_output("GHZ_output.txt", ios::out);
    GHZ_output << "column_number, radial distance, total stars, total
sterilizations- at least sterilized once\n";

```

GHZ_application_V4: postfix to prefix + library change + -Ofast flag
-changed all the postfix operations to prefix

-time trials:

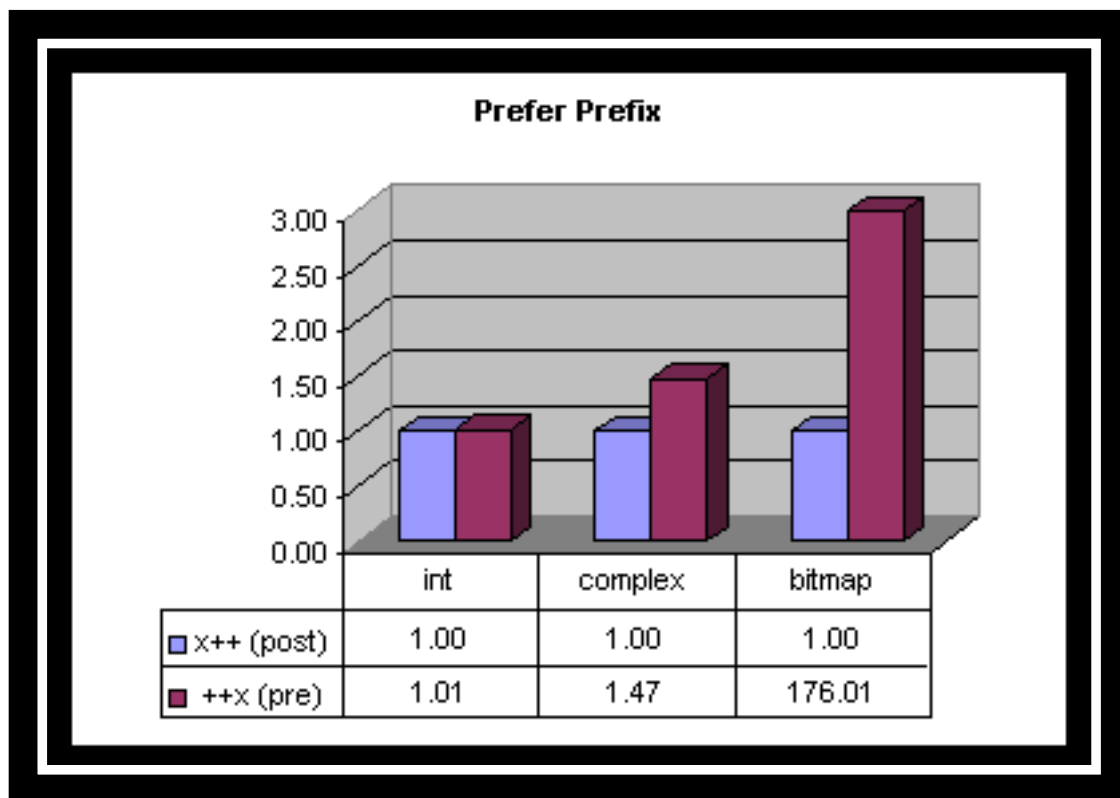
- | | |
|------------------|-------------------|
| 1) 20.04 seconds | 6) 19.83 seconds |
| 2) 19.81 seconds | 7) 19.80 seconds |
| 3) 19.91 seconds | 8) 19.83 seconds |
| 4) 19.97 seconds | 9) 19.79 seconds |
| 5) 19.87 seconds | 10) 19.84 seconds |

average time: 19.869 seconds

speed boost: 4.486% boost

results:

the code gets around .02 seconds faster with this change. There are two types of incrementing prefix (++x) and postfix (x++). Correctly implementing the postfix case requires saving the original value of the object. In other words, that means creating a temporary object. The recommendation to make the code faster is that we want to avoid this case. In the case of ints the speedup isn't enormous like complex and bitmap variables, but the speedup is still there: as shown below.



```
for (int i = 1; i<column; ++i)
{
    total=total+column_array[i]->num_stars;
}
```

```
if (sn_type==1)
{
    ++star1[j].sterilized_count;
    ++total_sterilizations_II;
}
else if (sn_type==2)
{
    ++star1[j].sterilized_count;
    ++total_sterilizations_Ia;
}
```

GHZ_application_V5: removing function calls in for loops + postfix to prefix + library change + -Ofast flag

-removed the function calls in the for loops in the function process_SN()

-time trials:

- | | |
|-----------------|------------------|
| 1) 7.58 seconds | 6) 7.64 seconds |
| 2) 7.62 seconds | 7) 7.74 seconds |
| 3) 7.54 seconds | 8) 7.77 seconds |
| 4) 7.55 seconds | 9) 7.80 seconds |
| 5) 7.70 seconds | 10) 7.67 seconds |

average time: 7.661 seconds

speed boost: 11.635% boost

results:

There is a significant speed boost when doing this change because the for loop does a function call each time it is used. Function call overhead is relatively high, especially when each time because space must be allocated for that function. By removing the function call and turning it into a temp variable we remove the mass overhead from calling a function to an atomic assignment to the temp variable.

```
long int temp2 =get_star_range_upper_cell_left(star2[l].cell);  
//for (long int m=get_star_range_lower_cell_left(star2[l].cell);  
    m<=get_star_range_upper_cell_left(star2[l].cell); ++m)  
    for (long int m=get_star_range_lower_cell_left(star2[l].cell); m<=temp2; ++m)  
    {
```

GHZ_application_V6: check last column of in the running_total function +
removing function calls in for loops + postfix to prefix + library change + -Ofast flag
-changed the running_total() function on the if statement check

-time trials:

- | | |
|-----------------|------------------|
| 1) 7.45 seconds | 6) 7.57 seconds |
| 2) 7.66 seconds | 7) 7.68 seconds |
| 3) 7.71 seconds | 8) 7.54 seconds |
| 4) 7.48 seconds | 9) 7.66 seconds |
| 5) 7.56 seconds | 10) 7.53 seconds |

average time: 7.584 seconds

speed boost: 11.753% boost

results:

THIS CHANGE WAS NOT MY IDEA AND WAS SUGGESTED BY A CLASSMATE.

this change I was a little skeptical about. It was not of my own idea and research, but a classmate (Collon) of mine suggested to try it out. There is a slight boost from the previous version, bringing a speed boost of about .1 seconds. The reason for this slight boost deals with how the for loop was checking for the column. When it was the last column it would still go into the loop and do the calculations even though we already had the data to return without using the loop. To fix this Collon proposed that we do a check before we enter the for loop and return the data right then and there to avoid the problem previously mentioned.

```
//returns the total number of stars before the column
long int running_total(int column)
{
    if (column == last_column)
    {
        return (old_running_total);
    }

    long int total=0;

    for (int i = 1; i<column; ++i)
    {
        total=total+column_array[i]->num_stars;
    }

    last_column = column;
    old_running_total=total;
    return (total);
}
```