CS 211: High Performance Computing Project 1

Performance Optimization via Register Reuse

Due date: 11:59pm, Oct 3rd, 2016

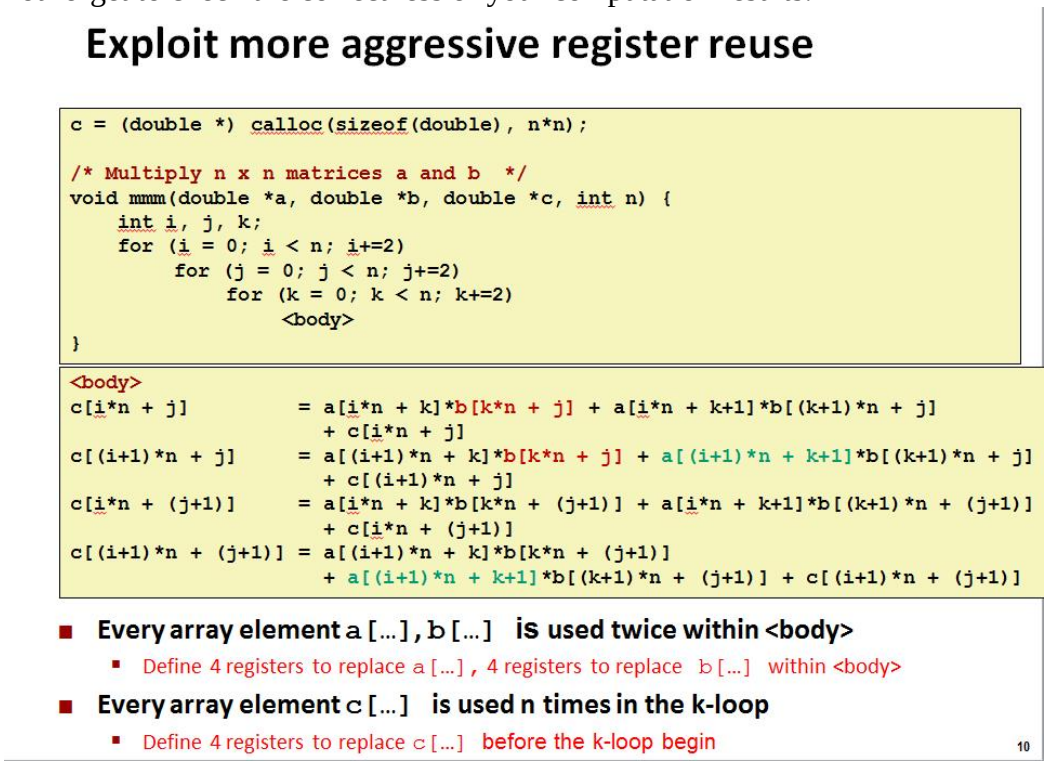
Note: You need to upload a pdf report for the project into the iLearn system. Please also upload all your source codes and a makefile as a tar file into iLearn system so that our TA can verify what you achieved in your report.

Part #1. (50 points) Assume your computer is able to complete 4 double floating-point operations per cycle when operands are in registers and it takes an additional delay of 100 cycles to access any operands that are not in registers. The clock frequency of your computer is 2 Ghz. How long it will take for your computer to finish the following algorithm ***dgemm0*** and ***dgemm1*** respectively for n= 1000? How much time is wasted on accessing operands that are not in registers? Implement the algorithm ***dgemm0*** and ***dgemm1*** and test them on TARDIS with n= 64, 128, 256, 512, 1024, 2048. Measure the time spend in the triple loop for each algorithm. Calculate the performance (in Gflops) of each algorithm. Performance is often defined as the number of floating-point operations performed per second. A performance of 1 Gflops means 1 billion of floating-point operations per second. You must use the system default compiler to compile your program. Your test matrices have to be 64 bit double floating point random numbers. Report the maximum difference of all matrix elements between the two results obtained from the two algorithms. This maximum difference can be used as a way to check the correctness of your implementation.

```
/*dgemm0: simple ijk version triple loop algorithm*/
for (i=0; i<n; i++)
    for (j=0; j<n; j++)
        for (k=0; k<n; k++)
            c[i*n+j]  += a[i*n+k] * b[k*n+j];

/*dgemm1: simple ijk version triple loop algorithm with register reuse*/
for (i=0; i<n; i++)
    for (j=0; j<n; j++) {
        register double r = c[i*n+j] ;
        for (k=0; k<n; k++)
            r += a[i*n+k] * b[k*n+j];
        c[i*n+j] = r;
    }
```

Part #2. (40 points) Let's use ***dgemm2*** to denote the algorithm in the following ppt slide from our class. Implement ***dgemm2*** and test it on TARDIS with n= 64, 128, 256, 512, 1024, 2048. Measure the time spend in the algorithm. Calculate the performance (in Gflops) of the algorithm. You must use the system default compiler to compile your program. Your test matrices have to be 64 bit double floating point random numbers. Do not forget to check the correctness of your computation results.



Exploit more aggressive register reuse

```
c = (double *) calloc(sizeof(double), n*n);

/* Multiply n x n matrices a and b  */
void mmm(double *a, double *b, double *c, int n) {
    int i, j, k;
    for (i = 0; i < n; i+=2)
        for (j = 0; j < n; j+=2)
            for (k = 0; k < n; k+=2)
                <body>
}
```

```
<body>
c[i*n + j]         = a[i*n + k]*b[k*n + j] + a[i*n + k+1]*b[(k+1)*n + j]
                     + c[i*n + j]
c[(i+1)*n + j]     = a[(i+1)*n + k]*b[k*n + j] + a[(i+1)*n + k+1]*b[(k+1)*n + j]
                     + c[(i+1)*n + j]
c[i*n + (j+1)]     = a[i*n + k]*b[k*n + (j+1)] + a[i*n + k+1]*b[(k+1)*n + (j+1)]
                     + c[i*n + (j+1)]
c[(i+1)*n + (j+1)] = a[(i+1)*n + k]*b[k*n + (j+1)]
                     + a[(i+1)*n + k+1]*b[(k+1)*n + (j+1)] + c[(i+1)*n + (j+1)]
```

- **Every array element `a[…], b[…]` is used twice within <body>**
  - Define 4 registers to replace `a[…]`, 4 registers to replace `b[…]` within <body>
- **Every array element `c[…]` is used n times in the k-loop**
  - Define 4 registers to replace `c[…]` before the k-loop begin

10

Part #3 (10 points). Assume you have 16 registers to use, please maximize the register reuse in your code (call this version code **dgemm3**) and compare your performance with ***dgemm0, dgemm1, and dgemm2***.