ECSE425 Computer Architecture and Organization Assignment 1 Solutions

Linda Wang

Fall 2008

Question 1: IC Cost Model (10%)

When $\alpha = 2$, the die area is $\frac{3}{2} \times 1\text{cm}^2 = 1.5\text{cm}^2$. Other parameters given in the question are: WaferDiameter $= 21\text{cm}^2$, DefectDensity $= 0.6$ per cm^2, and WaferYield $= 1$. The die yield is then

$$\text{DieYield} = 1 \times \left(1 + \frac{0.6 \times 1.5}{2}\right)^{-2} = 0.4756,$$

and the number of dies per wafer is

$$\text{DiesPerWafer} = \frac{\pi \times 10.5^2}{2.5} - \frac{\pi \times 21}{\sqrt{2 \times 1.5}} = 192.82.$$

So when $\alpha = 2$, the number of working chips we can get is $\lfloor 192.82 \times 0.4756 \rfloor = 91$.

When $\alpha = 4$, the die area is $\frac{3}{4} \times 1\text{cm}^2 = 0.75\text{cm}^2$. Calculations for the die yield and number of dies per wafer use the same formulas as before. We get

$$\text{DieYield} = 0.6528 \quad \text{and} \quad \text{DiesPerWafer} = 407.95.$$

So when $\alpha = 4$, the number of working chips we can get is $\lfloor 407.95 \times 0.6528 \rfloor = 266$.

Question 2: Amdahl's Law (15%)

Approach #1: Use CPU time equations. The equations for the CPU times before and after adding the enhancements are

$$\text{CPUtime}_{\text{old}} = \text{IC}_{\text{old}} \times \text{CPI}_{\text{old}} \times \text{CC}_{\text{old}},$$
$$\text{CPUtime}_{\text{new}} = \text{IC}_{\text{new}} \times \text{CPI}_{\text{new}} \times \text{CC}_{\text{new}}.$$

We see from the question that

$$\text{IC}_{\text{old}} = \text{IC}_{\text{new}} \quad \text{and} \quad \text{CC}_{\text{old}} = \text{CC}_{\text{new}},$$

so we only need to find the old and new CPI's in order to determine the speedup. The CPI's of each type of instruction are:

Type	Frequency	CPI_{old}	CPI_{new}
Load	0.25	$1+0.50\times 1=1.5$	$1+0.25\times 1=1.25$
Branch	0.10	2	$1+0.35\times 1=1.35$
Other	0.65	1	1

So the overall CPI's before and after enhancement are

$$\begin{aligned}\text{CPI}_{\text{old}} &= 1.5\times 0.25+2\times 0.10+1\times 0.65=1.225,\\ \text{CPI}_{\text{new}} &= 1.25\times 0.25+1.35\times 0.10+1\times 0.65=1.0975.\end{aligned}$$

The speedup is

$$\text{speedup}=\frac{\text{CPUtime}_{\text{old}}}{\text{CPUtime}_{\text{new}}}=\frac{1.225}{1.0975}=1.116.$$

Approach #2: Use Amdahl's Law. Let f_L denote the fraction of CPU time that load instructions take up before enhancement. Similarly, let f_B denote the fraction of CPU time that branch instructions take up before enhancement. We see from the previous calculations that

$$f_L=\frac{1.5\times 0.25}{1.225}=0.3061 \quad \text{and} \quad f_B=\frac{2\times 0.10}{1.225}=0.1633.$$

Also, the speedups of loads and branches after the enhancement are

$$\text{speedup}_L=\frac{1.5}{1.25}=1.2 \quad \text{and} \quad \text{speedup}_B=\frac{2}{1.35}=1.4815.$$

So using Amdahl's Law, the overall speedup is

$$\text{speedup}=\frac{1}{(1-f_L-f_B)+\frac{f_L}{\text{speedup}_L}+\frac{f_B}{\text{speedup}_B}}=1.116.$$

Question 3: CPI (15%)

(a) The CPI's for the instructions are

$$\begin{aligned}\text{CPI}_{\text{ALU}} &= 1+0.06\times 40=3.4,\\ \text{CPI}_{\text{load}} &= 2+0.06\times 40+0.10\times 40=8.4,\\ \text{CPI}_{\text{store}} &= 1+0.06\times 40+0.10\times 40=7.4,\\ \text{CPI}_{\text{branch}} &= 2+0.06\times 40=4.4.\end{aligned}$$

(b) The CPI for the overall machine is

$$\text{CPI}=0.4\times\text{CPI}_{\text{ALU}}+0.3\times\text{CPI}_{\text{load}}+0.1\times\text{CPI}_{\text{store}}+0.2\times\text{CPI}_{\text{branch}}=5.5.$$

(c) The ideal CPI without cache misses is

$$\text{CPI}_{\text{ideal}}=0.4\times 1+0.3\times 2+0.1\times 1+0.2\times 2=1.5.$$

So the machine without any cache misses is $\frac{5.5}{1.5}=3.67$ times faster.

Question 4: Tradeoffs (15%)

The equations for the CPU time before and after adding the enhancement are

$$\text{CPUtime}_{\text{old}} = \text{IC}_{\text{old}} \times \text{CPI}_{\text{old}} \times \text{CC}_{\text{old}},$$
$$\text{CPUtime}_{\text{new}} = \text{IC}_{\text{new}} \times \text{CPI}_{\text{new}} \times \text{CC}_{\text{new}}.$$

In order for the enhancement to be worthwhile, we must have $\text{CPUtime}_{\text{old}} > \text{CPUtime}_{\text{new}}$. We know the following from the question:

$$\text{CC}_{\text{new}} = x \times \text{CC}_{\text{old}} \quad \text{and} \quad \text{IC}_{\text{new}} = \text{IC}_{\text{old}}.$$

Also, from the previous question, we know that the CPI before enhancement is $\text{CPI}_{\text{old}} = 5.5$. The goal is to find the maximum value of x that would make the enhancement worthwhile.

After the enhancement, the new instructions replaced half of the branches (while the total instruction count remains the same). So the new instruction mix and the CPI for each instruction type are:

Type	Frequency	Clock Cycles	CPI
ALU ops	0.40	1	3.4
Loads	0.30	2	8.4
Stores	0.10	1	7.4
Branches	0.10	2	4.4
New Instr	0.10	1	$1 + 0.06 \times 40 = 3.4$

The new overall CPI is then

$$\text{CPI}_{\text{new}} = 0.40 \times 3.4 + 0.30 \times 8.4 + 0.10 \times 7.4 + 0.10 \times 4.4 + 0.10 \times 3.4 = 5.4.$$

In order to have improvement, we must have

$$\begin{aligned} \text{IC}_{\text{old}} \times \text{CPI}_{\text{old}} \times \text{CC}_{\text{old}} &> \text{IC}_{\text{old}} \times \text{CPI}_{\text{new}} \times x \times \text{CC}_{\text{old}} \\ 5.5 &> 5.4 \times x \\ x &< 1.019. \end{aligned}$$

Therefore the clock cycle time cannot increase more than 1.019 times.

Question 5: More Tradeoffs (15%)

(a) The question states that $\text{CPI}_{\text{ideal}} = 2$. The real CPI is

$$\text{CPI}_{\text{real}} = 2 + 0.40 \times 0.02 \times 30 + 0.025 \times 30 = 2.99.$$

The ideal and the real machines have the same IC and CC values. So the speedup of the ideal machine is simply $\frac{2.99}{2} = 1.495$.

(b) The CPI before enhancement is $\text{CPI}_{\text{old}} = 2.99$. Furthermore, we know that

$$\text{IC}_{\text{new}} = (1 - 0.30 \times 0.40)\text{IC}_{\text{old}} = 0.88\text{IC}_{\text{old}} \quad \text{and} \quad \text{CC}_{\text{new}} = 1.05\text{CC}_{\text{old}}.$$

The fraction of loads/stores after the enhancement is $\frac{0.70 \times 0.40}{0.88} = 0.318$. Therefore the CPI after enhancement is

$$\text{CPI}_{\text{new}} = 2 + 0.318 \times 0.02 \times 30 + 0.025 \times 30 = 2.94.$$

So the speedup after enhancement is

$$\frac{\text{CPUtime}_{\text{old}}}{\text{CPUtime}_{\text{new}}} = \frac{\text{IC}_{\text{old}} \times 2.99 \times \text{CC}_{\text{old}}}{0.88\text{IC}_{\text{old}} \times 2.94 \times 1.05\text{CC}_{\text{old}}} = 1.1.$$

Since this speedup is greater than one, the enhancement is worth being implemented.

Question 6: Performance (30%)

(a) Program listing

```
/*************************************************
 * ECSE425 Assignment 1, Question 6
 *
 * NAME: progA.c
 * BY:   Linda Wang
 * DATE: January 2004
 *************************************************/

#include<stdlib.h>
#include<time.h>
#include<math.h>

#define N  4000
#define PI 3.1416

double a[N][N];

int main ()
{
  int i, j;

  srand ((unsigned) time(NULL));

  /* fill a with values in range [-PI, +PI] */
  for (i=0; i<N; i++)
    for (j=0; j<N; j++)
      a[i][j] = (2.0*rand()/RAND_MAX - 1.0) * PI;

  /* replace elements in a by their cosines */
```

```
  for (i=0; i<N; i++)
    for (j=0; j<N; j++)
      a[i][j] = cos(a[i][j]);

  return 0;
}

/**************************************************
 * ECSE425 Assignment 1, Question 6
 *
 * NAME: progB.c
 * BY:   Linda Wang
 * DATE: January 2004
 **************************************************/

#include<stdlib.h>
#include<time.h>
#include<math.h>

#define N  500
#define PI 3.1416

double a[N][N], b[N][N], c[N][N];

int main ()
{
  double temp;
  int i, j, k;

  srand ((unsigned) time(NULL));

  /* fill a and b with values in range [-PI, +PI], */
  for (i=0; i<N; i++) {
    for (j=0; j<N; j++) {
      a[i][j] = (2.0*rand()/RAND_MAX - 1.0) * PI;
      b[i][j] = (2.0*rand()/RAND_MAX - 1.0) * PI;
    }
  }

  /* replace elements in a and by their cosines */
  for (i=0; i<N; i++) {
    for (j=0; j<N; j++) {
      a[i][j] = cos (a[i][j]);
      b[i][j] = cos (b[i][j]);
    }
  }
```

```
  /* take the transpose of b */
  for (i=0; i<N; i++) {
    for (j=0; j<(N-i); j++) {
      temp = b[i][j];
      b[i][j] = b[j][i];
      b[j][i] = temp;
    }
  }

  /* calculate c = ab */
  for (i=0; i<N; i++)
    for (j=0; j<N; j++)
      for (k=0; k<N; k++)
        c[i][j] += a[i][k] * b[k][j];

  return 0;
}
```

(b) Tabulated execution times

The programs were run under Cygwin release version 1.5.5-1 with gcc compiler version 3.2.

Table of CPU times in sec (N=4000 for progA, N=500 for progB)

	progA_noopt		progB_noopt		progA_opt		progB_opt	
Run #	user	system	user	system	user	system	user	system
1	3.615	0.080	3.414	0.030	3.505	0.080	2.183	0.030
2	3.615	0.080	3.414	0.030	3.525	0.070	2.173	0.020
3	3.605	0.110	3.394	0.050	3.505	0.080	2.183	0.010
4	3.625	0.070	3.414	0.040	3.495	0.100	2.173	0.020
5	3.665	0.050	3.404	0.020	3.515	0.070	2.163	0.020
6	3.675	0.070	3.484	0.020	3.484	0.080	2.203	0.030
7	3.585	0.090	3.414	0.020	3.535	0.050	2.193	0.020
8	3.655	0.080	3.424	0.020	3.505	0.070	2.153	0.020
9	3.565	0.110	3.434	0.030	3.494	0.070	2.173	0.030
10	3.645	0.080	3.424	0.040	3.505	0.080	2.153	0.030
mean	3.625	0.082	3.422	0.030	3.507	0.075	2.175	0.023
s.d.	0.013	0.008	0.022	0.007	0.008	0.009	0.015	0.007

Optimization did not improve system CPU time nearly as much as it improved user CPU time. This is because system CPU time is spent in the operating system, so it is not as dependent on complier optimization.

(c) Performance comparison using weighted arithmetic means

On a machine *without* optimizating complier available, the weighted arithmetic mean of programs A and B's user CPU times is

$$w_A \times 3.625 + w_B \times 3.422,$$

where

$$w_A = \frac{1}{1+\frac{3.625}{3.422}} = 0.486 \quad \text{and} \quad w_B = \frac{1}{1+\frac{3.422}{3.625}} = 0.514.$$

So the weighted arithmetic mean is 3.520 sec. (The formula for calculating the weights is on p. 37 of the textbook.)

On a machine *with* optimizating complier available, the weighted arithmetic mean of programs A and B's user CPU times is

$$w_A \times 3.507 + w_B \times 2.175,$$

where

$$w_A = \frac{1}{1+\frac{3.507}{2.175}} = 0.383 \quad \text{and} \quad w_B = \frac{1}{1+\frac{2.175}{3.507}} = 0.617.$$

So the weighted arithmetic mean is 2.685 sec.

Comparing the two weighted arithmetic means, we can see that the speedup in using an optimizing compiler is $\frac{3.520}{2.685} = 1.31$.

(c) Performance comparison using MFLOPS ratings

To compute MFLOPS ratings, first we need to estimate the number of floating point operations in both programs (note that we are only counting *floating point* operations here, so integer operations such as `i++` are *not* included):

Number of floating point operations

	progA (N = 4000)	progB (N = 500)
add/sub	16,000,000	125,500,000
mult/div	48,000,000	126,500,000
trig	16,000,000	500,000
total (native)	80,000,000	252,500,000
total (normalized)	336,000,000	635,500,000

The native MFLOPS ratings are

$$\text{MFLOPS}_{\text{native, noopt}} = \frac{80000000 + 252500000}{3.625 + 3.422} \times 10^{-6} = 47.18,$$
$$\text{MFLOPS}_{\text{native, opt}} = \frac{80000000 + 252500000}{3.507 + 2.175} \times 10^{-6} = 58.52.$$

The normalized MFLOPS ratings are

$$\text{MFLOPS}_{\text{normalized, noopt}} = \frac{336000000 + 635500000}{3.625 + 3.422} \times 10^{-6} = 137.86,$$
$$\text{MFLOPS}_{\text{normalized, opt}} = \frac{336000000 + 635500000}{3.507 + 2.175} \times 10^{-6} = 170.98.$$