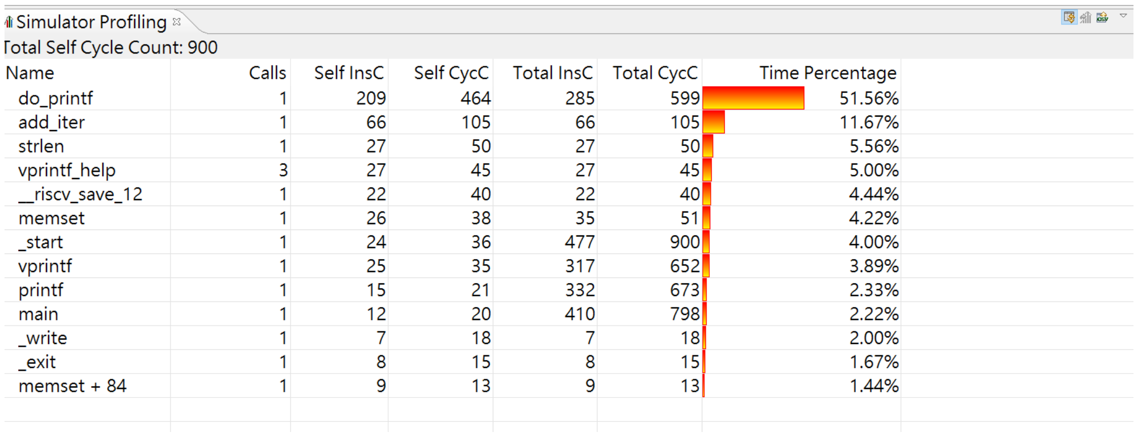
109062320 朱季葳 , discuss with 109062233, 109062306, 109062123,109062323

1.

(a) (6 points) Record the results for the two programs using CycC and InsC for “add_recur” function of “ADD_recursive” and for “add_iter” function of “ADD_iterative” . Based on your understanding of the characteristics of the two programs, compare the differences in their profiles.



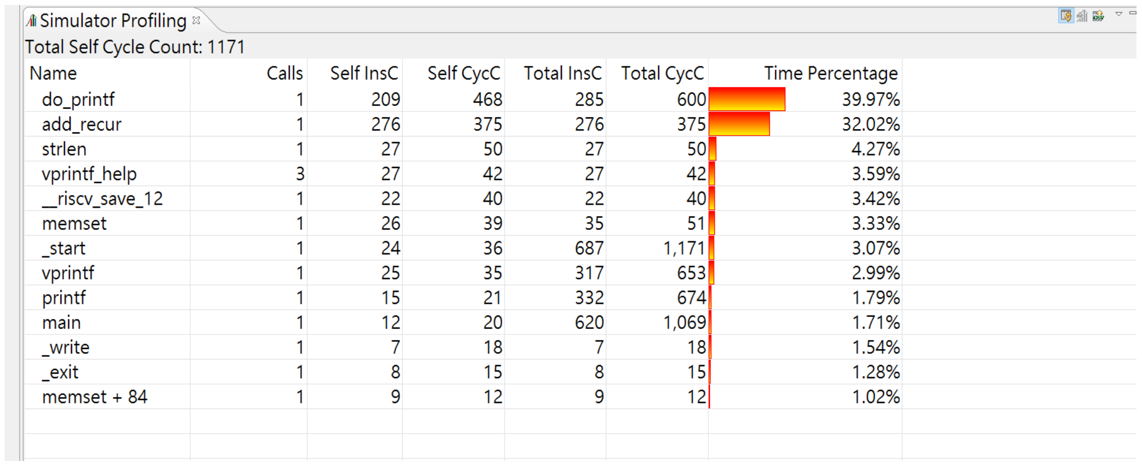
Simulator Profiling

Total Self Cycle Count: 900

Name	Calls	Self InsC	Self CycC	Total InsC	Total CycC	Time Percentage
do_printf	1	209	464	285	599	51.56%
add_iter	1	66	105	66	105	11.67%
strlen	1	27	50	27	50	5.56%
vprintf_help	3	27	45	27	45	5.00%
__riscv_save_12	1	22	40	22	40	4.44%
memset	1	26	38	35	51	4.22%
_start	1	24	36	477	900	4.00%
vprintf	1	25	35	317	652	3.89%
printf	1	15	21	332	673	2.33%
main	1	12	20	410	798	2.22%
_write	1	7	18	7	18	2.00%
_exit	1	8	15	8	15	1.67%
memset + 84	1	9	13	9	13	1.44%

<u>result of ADD_iterative</u>

add_iter :InsC 66 CycC 105



Simulator Profiling

Total Self Cycle Count: 1171

Name	Calls	Self InsC	Self CycC	Total InsC	Total CycC	Time Percentage
do_printf	1	209	468	285	600	39.97%
add_recur	1	276	375	276	375	32.02%
strlen	1	27	50	27	50	4.27%
vprintf_help	3	27	42	27	42	3.59%
__riscv_save_12	1	22	40	22	40	3.42%
memset	1	26	39	35	51	3.33%
_start	1	24	36	687	1,171	3.07%
vprintf	1	25	35	317	653	2.99%
printf	1	15	21	332	674	1.79%
main	1	12	20	620	1,069	1.71%
_write	1	7	18	7	18	1.54%
_exit	1	8	15	8	15	1.28%
memset + 84	1	9	12	9	12	1.02%

<u>result of ADD_recursive</u>

add_recur :InsC 276 CycC 375

By the above graph,we can see that the instruction count and clock cycle count of recursion are bigger than the ones of iteration

and the CPI of recursion is 375/276 = 1.358, the CPI of iteration is 105/66 = 1.59

As a result, we can see that the CPI of iteration is bigger

Furthermore, as we know about recursion, it calls it self until the terminal condition, and it is far more time-consuming than iteration in which just run a loop.

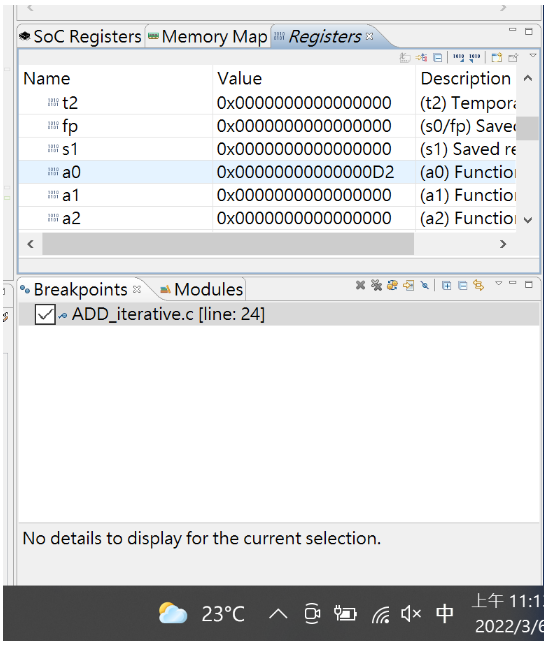
(b) (6 points) RISC-V has 32 general purpose registers, whereas X86 (a CISC architecture) only has 8. Why does RISC have more registers than CISC?

The primary goal of CISC architecture is to complete a task in as few lines of assembly code as possible while RISC architecture only use simple instructions that can be executed within 1 clock cycle.

As a result, RISC need more registers to store the instructions.

I take reference on https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/risccisc/

When executing "ADD_iterative", which register stores the return value for add_iter function? (You can select debug icon to view the contents of all registers. Note that you should set breakpoints at the right spots.)

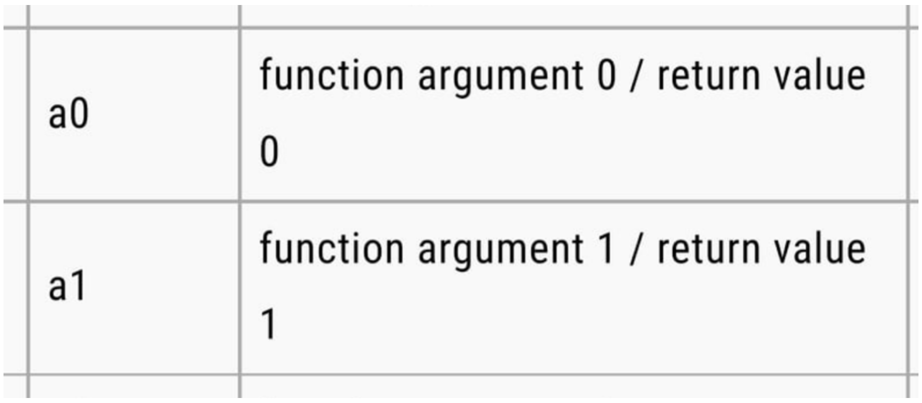


Name	Value	Description
t2	0x0000000000000000	(t2) Tempor
fp	0x0000000000000000	(s0/fp) Save
s1	0x0000000000000000	(s1) Saved re
a0	0x00000000000000D2	(a0) Functio
a1	0x0000000000000000	(a1) Functio
a2	0x0000000000000000	(a2) Functio

☑ ADD_iterative.c [line: 24]

No details to display for the current selection.

By the above image, after set breakpoint in the line after calling function,we can see the register "a0" store the return value for add_iter function



a0	function argument 0 / return value 0
a1	function argument 1 / return value 1

Besides, by the description of a0 and a1, we can see that a1 also store the return value

(c) (6 points) What are the average CPI for “add_recur” function in ADD_recursive.c and “add_iter” function in ADD_iterative.c, respectively?

CPI of recursion is 375/276 = 1.36, the CPI of iteration is 105/66 = 1.59

(d) (6 points) What are the CPU execution time for “add_recur” function in ADD_recursive.c and “add_iter” function in ADD_iterative.c, respectively, on a processor with a clock rate of 1GHz?

execution time of recursion is 375/(1*10^9) = 0.000000375 = 0.375ms ~= 0.38ms

execution time of iteration is 105/(1*10^9) = 0.000000105 = 0.105ms ~= 0.11ms

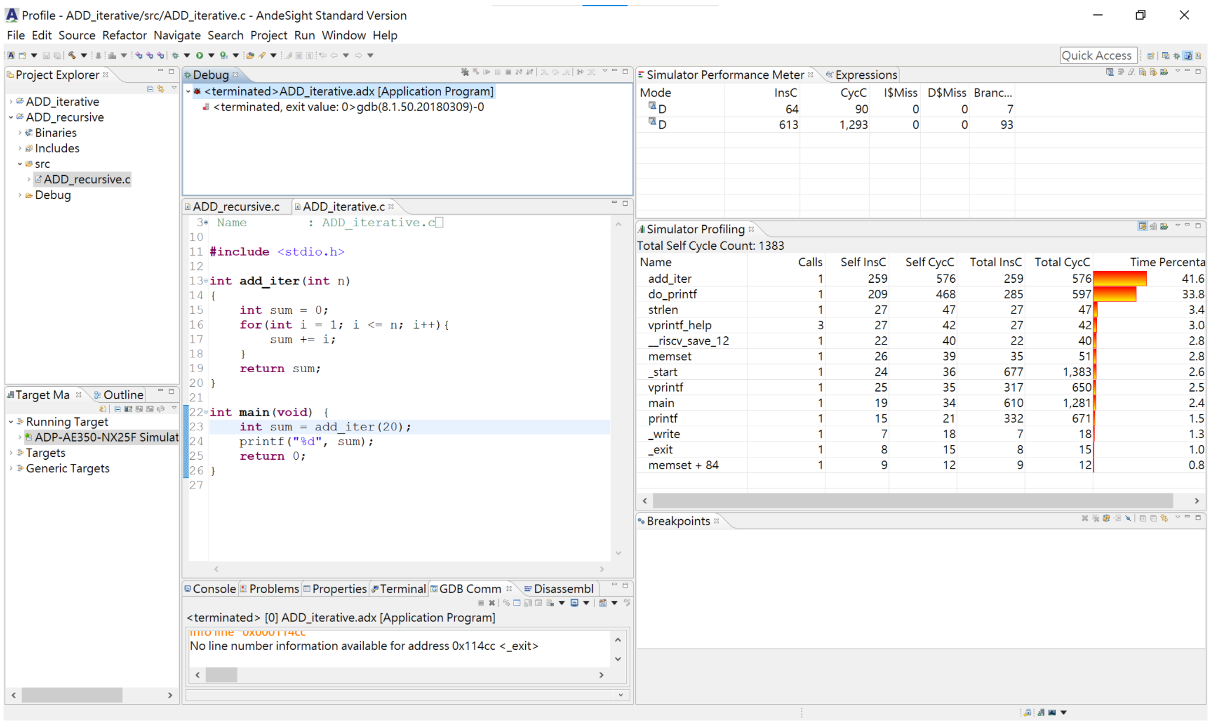
(e) (6 points) Assume we execute ADD_iterative.c in (d) but with a 4-core multiprocessor instead and also employ parallelization techniques on add_iter function to equally allocate computation to each core. But, the parallelization increases 50 communication cycles on the multiprocessor. What is the program execution time now? (Hint: Only add_iter can be parallelized.)

total cycle = 900

clock cycle count = 900 − 105 + 50 + 105/4 = 872

execution time = 872 / 10^9 = 0.872 ms ~= 0.87ms

(f) (20 points) Compiler will affect program performance. Compile ADD_recursive.c and ADD_iterative.c with two optimization levels, -O0 and -O1, respectively. Compare the performance of “add_recur” and “add_iter” function in ADD_recursive.c and ADD_iterative.c for different optimization levels (-O0 and -O1). You should report CycC, InsC and CPI and explain the differences in their profiles. (To change optimization level, see Fig. 1.)


Profile - ADD_iterative/src/ADD_iterative.c - AndeSight Standard Version

<terminated>ADD_iterative.adx [Application Program]
<terminated, exit value: 0>gdb(8.1.50.20180309)-0

```
 * Name         : ADD_iterative.c

#include <stdio.h>

int add_iter(int n)
{
    int sum = 0;
    for(int i = 1; i <= n; i++){
        sum += i;
    }
    return sum;
}

int main(void) {
    int sum = add_iter(20);
    printf("%d", sum);
    return 0;
}
```

Simulator Performance Meter

Mode	InsC	CycC	I$Miss	D$Miss	Branc...
D	64	90	0	0	7
D	613	1,293	0	0	93

Simulator Profiling

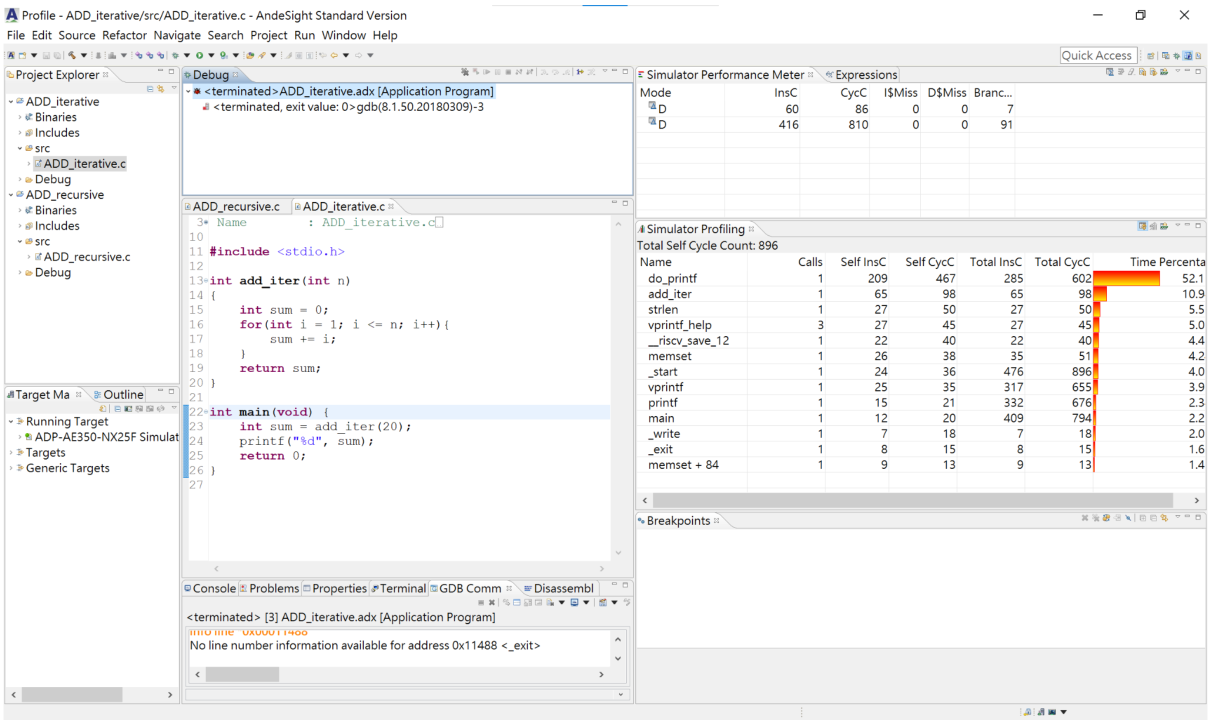
Total Self Cycle Count: 1383

Name	Calls	Self InsC	Self CycC	Total InsC	Total CycC	Time Percenta
add_iter	1	259	576	259	576	41.6
do_printf	1	209	468	285	597	33.8
strlen	1	27	47	27	47	3.4
vprintf_help	3	27	42	27	42	3.0
__riscv_save_12	1	22	40	22	40	2.8
memset	1	26	39	35	51	2.8
_start	1	24	36	677	1,383	2.6
vprintf	1	25	35	317	650	2.5
main	1	19	34	610	1,281	2.4
printf	1	15	21	332	671	1.5
_write	1	7	18	7	18	1.3
_exit	1	8	15	8	15	1.0
memset + 84	1	9	12	9	12	0.8

```
<terminated> [0] ADD_iterative.adx [Application Program]
No line number information available for address 0x114cc <_exit>
```

result of iteration with -O0

CycC: 576, InsC: 259, CPI: 576/259 = 2.2239 ~= 2.22



Profile - ADD_iterative/src/ADD_iterative.c - AndeSight Standard Version

```
<terminated>ADD_iterative.adx [Application Program]
  <terminated, exit value: 0>gdb(8.1.50.20180309)-3
```

```c
 * Name        : ADD_iterative.c

#include <stdio.h>

int add_iter(int n)
{
    int sum = 0;
    for(int i = 1; i <= n; i++){
        sum += i;
    }
    return sum;
}

int main(void) {
    int sum = add_iter(20);
    printf("%d", sum);
    return 0;
}
```

Simulator Performance Meter

Mode	InsC	CycC	I$Miss	D$Miss	Branc...
D	60	86	0	0	7
D	416	810	0	0	91

Simulator Profiling

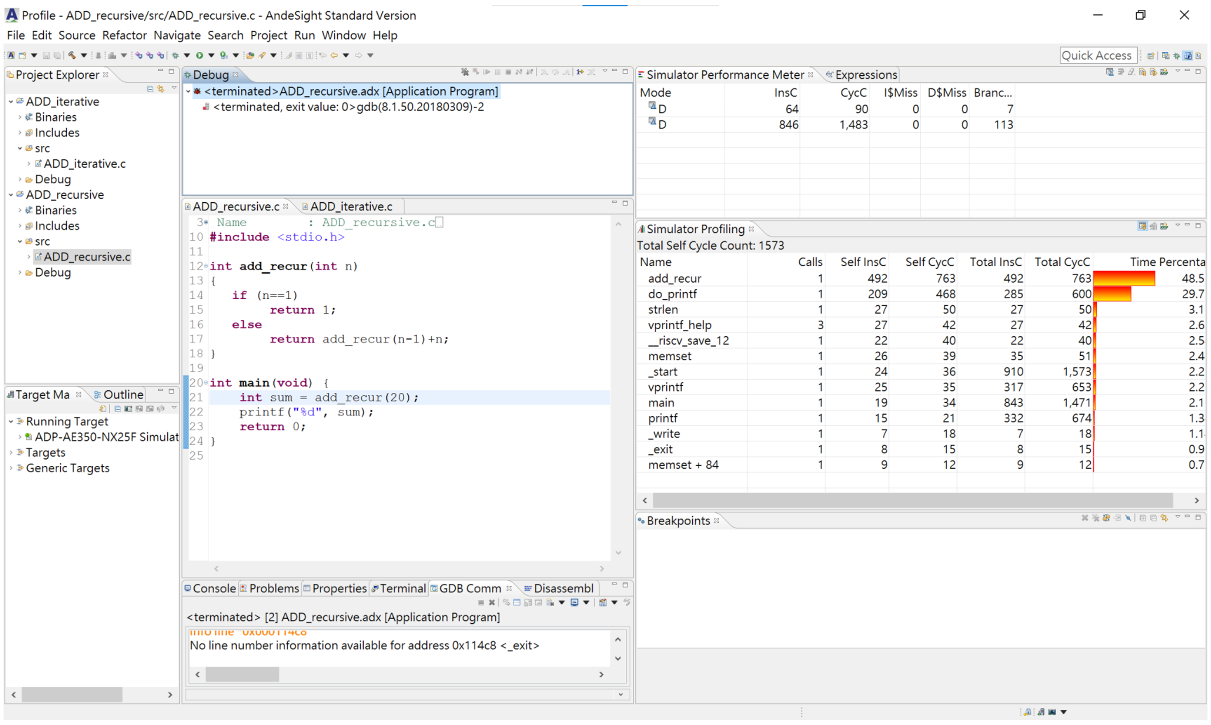
Total Self Cycle Count: 896

Name	Calls	Self InsC	Self CycC	Total InsC	Total CycC	Time Percenta
do_printf	1	209	467	285	602	52.1
add_iter	1	65	98	65	98	10.9
strlen	1	27	50	27	50	5.5
vprintf_help	3	27	45	27	45	5.0
__riscv_save_12	1	22	40	22	40	4.4
memset	1	26	38	35	51	4.2
_start	1	24	36	476	896	4.0
vprintf	1	25	35	317	655	3.9
printf	1	15	21	332	676	2.3
main	1	12	20	409	794	2.2
_write	1	7	18	7	18	2.0
_exit	1	8	15	8	15	1.6
memset + 84	1	9	13	9	13	1.4

```
<terminated> [3] ADD_iterative.adx [Application Program]
info line *0x00011488
No line number information available for address 0x11488 <_exit>
```

result of iteration with -O1

CycC: 98, InsC: 65, CPI: 98/65 = 1.5076 ~= 1.51



Profile - ADD_recursive/src/ADD_recursive.c - AndeSight Standard Version

```
<terminated>ADD_recursive.adx [Application Program]
  <terminated, exit value: 0>gdb(8.1.50.20180309)-2
```

```c
 * Name        : ADD_recursive.c
#include <stdio.h>

int add_recur(int n)
{
   if (n==1)
        return 1;
   else
        return add_recur(n-1)+n;
}

int main(void) {
    int sum = add_recur(20);
    printf("%d", sum);
    return 0;
}
```

Simulator Performance Meter

Mode	InsC	CycC	I$Miss	D$Miss	Branc...
D	64	90	0	0	7
D	846	1,483	0	0	113

Simulator Profiling

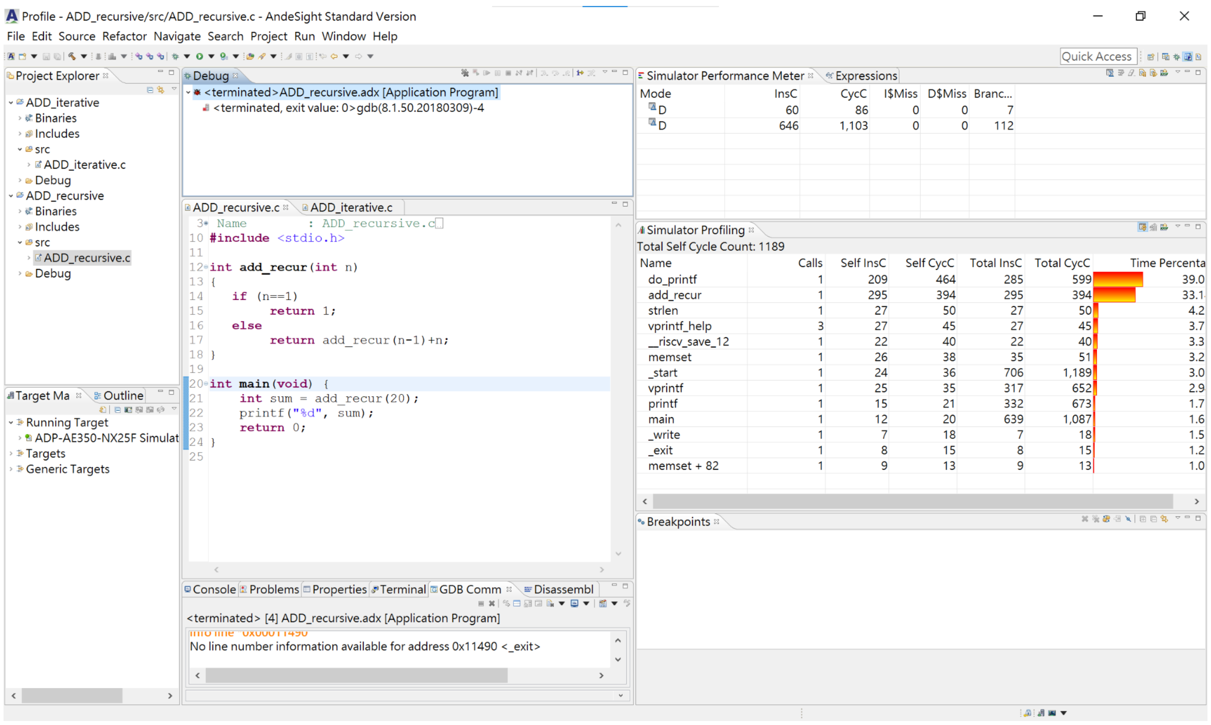
Total Self Cycle Count: 1573

Name	Calls	Self InsC	Self CycC	Total InsC	Total CycC	Time Percenta
add_recur	1	492	763	492	763	48.5
do_printf	1	209	468	285	600	29.7
strlen	1	27	50	27	50	3.1
vprintf_help	3	27	42	27	42	2.6
__riscv_save_12	1	22	40	22	40	2.5
memset	1	26	39	35	51	2.4
_start	1	24	36	910	1,573	2.2
vprintf	1	25	35	317	653	2.2
main	1	19	34	843	1,471	2.1
printf	1	15	21	332	674	1.3
_write	1	7	18	7	18	1.1
_exit	1	8	15	8	15	0.9
memset + 84	1	9	12	9	12	0.7

```
<terminated> [2] ADD_recursive.adx [Application Program]
info line *0x000114c8
No line number information available for address 0x114c8 <_exit>
```

result of recursion with -O0

CycC: 763, InsC: 492, CPI: 763/492 = 1.5508 ~= 1.55


Profile - ADD_recursive/src/ADD_recursive.c - AndeSight Standard Version

<terminated>ADD_recursive.adx [Application Program]
<terminated, exit value: 0>gdb(8.1.50.20180309)-4

```c
 Name          : ADD_recursive.c
#include <stdio.h>

int add_recur(int n)
{
   if (n==1)
       return 1;
   else
       return add_recur(n-1)+n;
}

int main(void) {
    int sum = add_recur(20);
    printf("%d", sum);
    return 0;
}
```

Mode	InsC	CycC	I$Miss	D$Miss	Branc...
D	60	86	0	0	7
D	646	1,103	0	0	112

Total Self Cycle Count: 1189

Name	Calls	Self InsC	Self CycC	Total InsC	Total CycC	Time Percenta
do_printf	1	209	464	285	599	39.0
add_recur	1	295	394	295	394	33.1
strlen	1	27	50	27	50	4.2
vprintf_help	3	27	45	27	45	3.7
__riscv_save_12	1	22	40	22	40	3.3
memset	1	26	38	35	51	3.2
_start	1	24	36	706	1,189	3.0
vprintf	1	25	35	317	652	2.9
printf	1	15	21	332	673	1.7
main	1	12	20	639	1,087	1.6
_write	1	7	18	7	18	1.5
_exit	1	8	15	8	15	1.2
memset + 82	1	9	13	9	13	1.0

```
<terminated> [4] ADD_recursive.adx [Application Program]
No line number information available for address 0x11490 <_exit>
```

result of recursion with -O1

CycC: 394, InsC: 295, CPI: 394/295 = 1.3355 ~= 1.34

By the above images, we can see that O1 has less CycC, InsC and CPI than O0.

I take reference on the official website: https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

O0 compiler is the original one which doesn’t have any optimization,while O1 compiler optimiz e compilations that take more time and more memory for a large function.

Besides, O1 compiler will try to reduce code size and execution time without performing the opti mizations that take lots of compilation time.

As for the original setting --> -Og

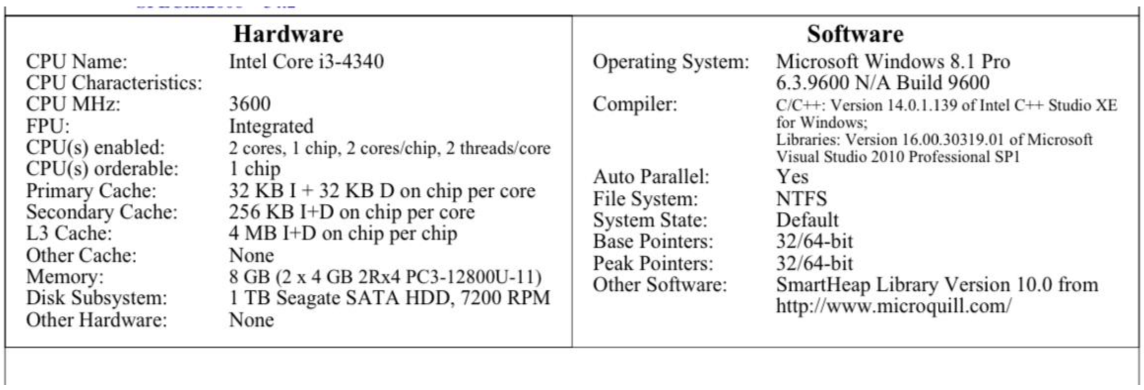
Optimize debugging experience. -Og should be the optimization level of choice for the standard ed it-compile-debug cycle, offering a reasonable level of optimization while maintaining fast compila tion and a good debugging experience.

It is a better choice than -O0for producing debuggable code because some compiler passes that coll ect debug information are disabled at -O0

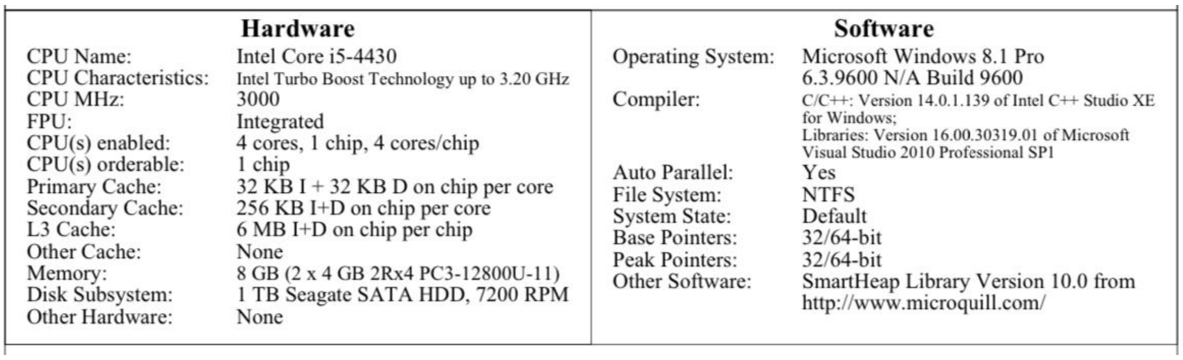
2. (20 points) Go to SPEC Benchmarks website and search for the benchmarks of the following systems: • Intel DH87MC Motherboard (Intel Core i3-4340) • Intel DH87MC Motherboard (Intel Core i5-4430) • Intel DH87MC Motherboard (Intel Core i7-4770)

We want to compare the performance of the three computer systems using different methods to normalize the performance.

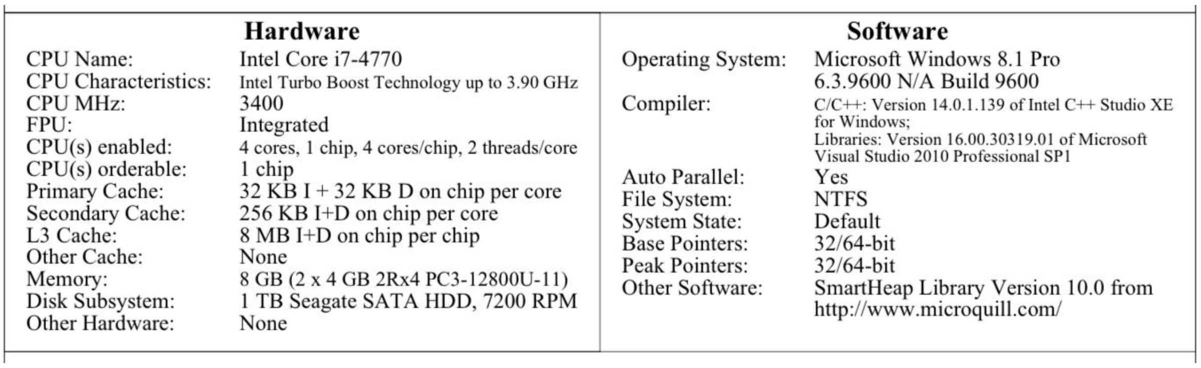
(a) (5 points) Compare the hardware and software specifications of these three motherboards and identify their differences.



Hardware		Software	
CPU Name:	Intel Core i3-4340	Operating System:	Microsoft Windows 8.1 Pro 6.3.9600 N/A Build 9600
CPU Characteristics:		Compiler:	C/C++: Version 14.0.1.139 of Intel C++ Studio XE for Windows; Libraries: Version 16.00.30319.01 of Microsoft Visual Studio 2010 Professional SP1
CPU MHz:	3600	Auto Parallel:	Yes
FPU:	Integrated	File System:	NTFS
CPU(s) enabled:	2 cores, 1 chip, 2 cores/chip, 2 threads/core	System State:	Default
CPU(s) orderable:	1 chip	Base Pointers:	32/64-bit
Primary Cache:	32 KB I + 32 KB D on chip per core	Peak Pointers:	32/64-bit
Secondary Cache:	256 KB I+D on chip per core	Other Software:	SmartHeap Library Version 10.0 from http://www.microquill.com/
L3 Cache:	4 MB I+D on chip per chip		
Other Cache:	None		
Memory:	8 GB (2 x 4 GB 2Rx4 PC3-12800U-11)		
Disk Subsystem:	1 TB Seagate SATA HDD, 7200 RPM		
Other Hardware:	None		



Hardware		Software	
CPU Name:	Intel Core i5-4430	Operating System:	Microsoft Windows 8.1 Pro 6.3.9600 N/A Build 9600
CPU Characteristics:	Intel Turbo Boost Technology up to 3.20 GHz	Compiler:	C/C++: Version 14.0.1.139 of Intel C++ Studio XE for Windows; Libraries: Version 16.00.30319.01 of Microsoft Visual Studio 2010 Professional SP1
CPU MHz:	3000	Auto Parallel:	Yes
FPU:	Integrated	File System:	NTFS
CPU(s) enabled:	4 cores, 1 chip, 4 cores/chip	System State:	Default
CPU(s) orderable:	1 chip	Base Pointers:	32/64-bit
Primary Cache:	32 KB I + 32 KB D on chip per core	Peak Pointers:	32/64-bit
Secondary Cache:	256 KB I+D on chip per core	Other Software:	SmartHeap Library Version 10.0 from http://www.microquill.com/
L3 Cache:	6 MB I+D on chip per chip		
Other Cache:	None		
Memory:	8 GB (2 x 4 GB 2Rx4 PC3-12800U-11)		
Disk Subsystem:	1 TB Seagate SATA HDD, 7200 RPM		
Other Hardware:	None		



Hardware		Software	
CPU Name:	Intel Core i7-4770	Operating System:	Microsoft Windows 8.1 Pro 6.3.9600 N/A Build 9600
CPU Characteristics:	Intel Turbo Boost Technology up to 3.90 GHz	Compiler:	C/C++: Version 14.0.1.139 of Intel C++ Studio XE for Windows; Libraries: Version 16.00.30319.01 of Microsoft Visual Studio 2010 Professional SP1
CPU MHz:	3400	Auto Parallel:	Yes
FPU:	Integrated	File System:	NTFS
CPU(s) enabled:	4 cores, 1 chip, 4 cores/chip, 2 threads/core	System State:	Default
CPU(s) orderable:	1 chip	Base Pointers:	32/64-bit
Primary Cache:	32 KB I + 32 KB D on chip per core	Peak Pointers:	32/64-bit
Secondary Cache:	256 KB I+D on chip per core	Other Software:	SmartHeap Library Version 10.0 from http://www.microquill.com/
L3 Cache:	8 MB I+D on chip per chip		
Other Cache:	None		
Memory:	8 GB (2 x 4 GB 2Rx4 PC3-12800U-11)		
Disk Subsystem:	1 TB Seagate SATA HDD, 7200 RPM		
Other Hardware:	None		

Firstly, we compare the differences in software.We can see that there is no difference in software among these three motherboards.

As for the differences in hardware,there are four differences:

1.Turbo-Boost, i3 doesn’ t has turbo-boost enabled,while i5 and i7 have;besides,i7’ s speed is fa ster than i5.

2.cores, i3 has 4 cores,i5 has at least 4 cores while i7 has at least 6 cores

3.cache memory, the storage of cache memory in i7 > i5 > i3

4.the running frequency, the running frequency in i3 > i7 > i5

(b) (5 points) Consider the three benchmarks: 464.h264ref, 471.omnetpp, and 473.astar in the resu lts table.

Please use the first column of “seconds” (check Fig. 4) to calculate the relative performance of the three computer systems based on the three benchmarks.

Fill out the following table, which uses each of the three computers as the reference for compariso n.

Summarize the performance results with the arithmetic mean of the performance ratios of the three benchmark programs. Please show the calculation procedure.

464.h264ref

	performance ratio		
reference	intel core 13-4340	intel core i5-4430	intel core i7-4770
intel core i3-4340	1	323/373=0.87	323/323=1
intel core i5-4430	373/323=1.16	1	373/323=1.16
intel core i7-4770	323/323=1	323/373=0.87	1

471.omnetpp

	performance ratio		
reference	intel core 13-4340	intel core i5-4430	intel core i7-4770
intel core i3-4340	1	257/251=1.02	257/223=1.15
intel core i5-4430	251/257=0.98	1	251/223=1.13
intel core i7-4770	223/257=0.87	223/251=0.89	1

473.astar

	performance ratio		
reference	intel core 13-4340	intel core i5-4430	intel core i7-4770
intel core i3-4340	1	233/247=0.94	233/200=1.17
intel core i5-4430	247/233=1.06	1	247/200=1.24
intel core i7-4770	200/233=0.86	200/247=0.81	1

Arithmetic mean

	performance ratio		
reference	intel core 13-4340	intel core i5-4430	intel core i7-4770
intel core i3-4340	1	(1.02+0.87+0.94)/3 =0.94	(1.15+1+1.17)/3 =1.11
intel core i5-4430	(0.98+1.16+1.06)/3 =1.07	1	(1.13+1.16+1.24)/3 =1.18
intel core i7-4770	(0.87+1+0.86)/3 =0.91	(0.89+0.87+0.81)/3 =0.86	1

(c) (5 points) Repeat (b) with the geometric mean of the performance ratios of the three benchmar
k programs.

	performance ratio		
reference	intel core 13-4340	intel core i5-4430	intel core i7-4770
intel core i3-4340	1	(1.02*0.87*0.94)^(1/3) =0.94	(1.15*1*1.17) ^(1/3) =1.10
intel core i5-4430	(0.98*1.16*1.06)^(1/3) =1.06	1	(1.13*1.16*1.24) ^(1/3) =1.18
intel core i7-4770	(0.87*1*0.86) ^(1/3) =0.91	(0.89*0.87*0.81) ^(1/3) =0.86	1

(d) (5 points) Which observations can you make from (b) and (c)? How are these results compared with SPECint_base2006 ratio?

We can see that there is little difference between geometric mean and arithmetic mean;what’ s m ore,it shows that the performance of i7 > i3 > i5.

But the result of arithmetic mean looks more changable than geometrc mean.

After comparing with the result of SPECint_base2006,we can also see that the performance of

i7 > i3 > i5 （The SPECint_base2006 is i3 : 52.3 i5 : 48.3 i7 : 59.6 ）

3. (10 points) Assume a program requires the execution of 90×10^6 FP instructions, 110×10^6 INT instructions, 100×10^6 L/S instructions, and 25×10^6 branch instructions. The CPI for each type of instruction is 2, 1, 5, and 2, respectively. Assume that the processor has a 4GHz clock rate.

1. (a) (5 points) By how much must we improve the CPI of FP instructions if we want the pr ogram to run two times faster? Please show the calculation procedure.

   clock cycles = (90 x 10^6 x 2) + (110 x 10^6 x 1) + (100 x 10^6 x 5) + (25 x 10^6 x 2)

   = (180+110+500+50)x10^6 = 840 x 10^6

   run two times faster-->clock cycles/2 (clock rate is fixed) = 420 x 10^6

   new CPI of FP instruction = (420 – 110 – 500 – 50) x 10^6 / (90 x 10^6)

   = -240/ 90 --->it is impossible!

2. (b) (5 points) By how much is the execution time of the program improved if the CPI of I NT and FP instructions is reduced by 31% and the CPI of L/S and Branch is reduced by 7 7%? Please show the calculation procedure.

   clock cycles = {0.69 x [(90 x 10^6 x 2) + (110 x 10^6 x 1)]}

   + {0.23 x [(100 x 10^6 x 5) + (25 x 10^6 x 2)]}

   = (290x0.69 + 550x0.23)x10^6 = 326.6 x 10^6

   because clock rate is fixed: 326.6/840 = 0.388 ~= 39% improved

   so the improvement is 61%

4. (10 points) Processor P1 has a clock rate of 5GHz and a voltage of 1.25V. Assume that, on ave rage, it consumes 60W of static power and 90W of dynamic power. Assume that the dynamic power and static power are respectively calculated by the following equations:

   dynamic power = 1/2 × capacitive load × volatge^2 × clock frequency

   static power = voltage × leakage current

   (a) (2 points) Find the average capacitive load.

   capative load = 2 x dynamic power / (voltage^2 x clock frequency)

   P1:capative load = 2 x 90 / (1.25^2 x 5*10^9)

   = 2.30 x 10^(-8)

   (b) (2 points) Find the percentage of the total dissipated power comprised by d ynamic power and the ratio of static power to dynamic power.

   percentage of total dissipated power comprised by dynamic power 90 / (90 + 60) = 0.6

   ratio of static power to dynamic power = 60/90 = 0.67

   (c) (6 points) If the total dissipated power is to be reduced by 35%, how much should the voltage be reduced to maintain the same leakage current?

   abbriviate Static power,dynamic power,voltage,capative load,

   clock frequency,leakage current to S,D,V,C,F,L

   (Snew + Dnew) / (Sold + Dold) = 0.65, Vnew = (2 x Dnew / (C x F))^(1/2), Dnew = 0.65 x (Sold + Dold) – Snew

   Snew = Vnew x L ,Sold = Vold x L -->Snew/Vnew = Sold/Vold

   L = 60 / 1.25 = 48

   Dnew = 0.65 x (60+90) – Vnew x 48 = 97.5 – Vnew x 48

   Vnew = [(2 x 97.5 – Vnew x 96)/(2.3 x 10^(-8) x 5 x 10^(9))]^(1 /2) = 0.95

Vold = 60 / 48 = 1.25

Vnew/Vold = 0.95/1.25 = 0.76, the voltage reduced by 24%

5. (10 points) Assume that a 25 cm diameter wafer has a cost of 15, contains 120 dies, and has 0.02 defects/cm^2.

   (a) (3 points) Find the yield for this wafer using the equation on page 28 of the textbook (or page 28 of the lecture notes of Chapter 1).

   Die area = pi x 12.5^2 / 120 = 4.09

   Yield = 1/((1+Defects per area x Die area/2))^2

   = 1/((1 + 0.02 x 4.09/2))^2

   = 0.92

   (b) (3 points) Find the cost per die for this wafer.

   15/(120 x 0.92) = 0.14

   (c) (4 points) If the number of dies per wafer is increased by 20% and the defects per area unit increases by 35%, find the die area and yield.

   number of dies per wafer = 120*1.2 = 144

   die area = pi x 12.5^2 / 144 = 3.41

   Yield = 1/((1 + 0.02 x 1.35 x 3.41/2))^2 = 0.91