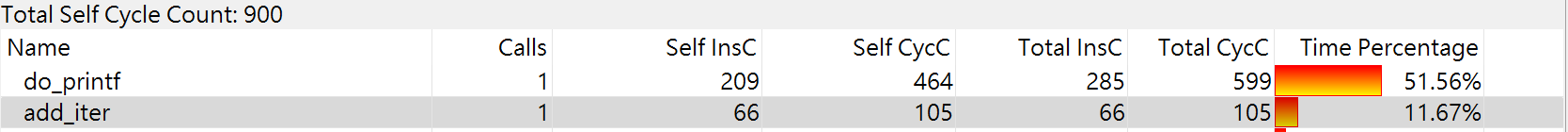
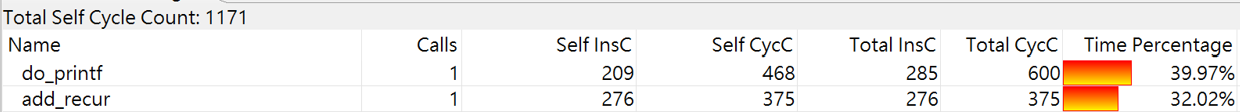
1.

(a)





CycC for add\_iter = 105

InsC for add\_iter = 66

CPU time for add\_iter = 1.5909\*66 = 104.9994

CycC for add\_recur = 375

InsC for add\_recur = 276

CPU time for add\_recur = 1.3587\*276 = 375.0012

It needs more InsC and CycC in add\_recur than in add\_iter.

Because recursion is slower than iteration due to the overhead of maintaining the stack. Recursion uses more memory than iteration. We need some dynamically-allocated memory to replace the build-up of stack-memory in the recursive form. And that is the main performance difference, that allocating stack-memory is a cheap constant-time operation , while allocating dynamic memory is a more expensive amortized constant operation.

(b)

In RISC, "LOAD" and "STORE" are independent instructions, but in CISC, "LOAD" and "STORE" incorporated in instructions. After a CISC-style "MULT" command is executed, the processor automatically erases the registers. In RISC, the operand will remain in the register until another value is loaded in its place.

a0 stores the return value for add\_iter function

(c)

Cycle per instruction = total cycle/instruction count

CPI for add\_recur = 375/276 = 1.3587..

CPI for add\_iter = 105/66 = 1.5909…

(d)

Clock rate => 10^9 cycles per second => 1 cycle per 10^-9 sec

Execution time = total cycle \* clock cycle time

execution time for “add\_recur” = 375\*10^-9 = 3.75\*10^-7

execution time for “add\_iter” = 105\*10^-9 = 1.05\*10^-7

(e)

Iter total cycle = 105

Other total cycle = 900-105 = 795

Communication cycles = 50

795+50 = 845 (cycles needs to run for other )

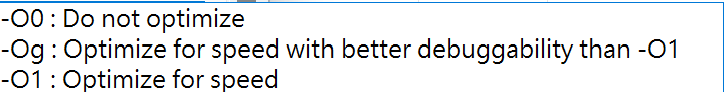
105/4 = 26.25 (cycles per core needs to run for add\_iter)

26.25\*10^-9 = 2.625 \* 10^-8 (time per core needs to run for add\_iter)

845\*10^-9 = 84.5\*10^-8

Total execution time = 2.625\*10^-8 + 84.5\*10^-8 = 8.7125\*10^-7

(f)

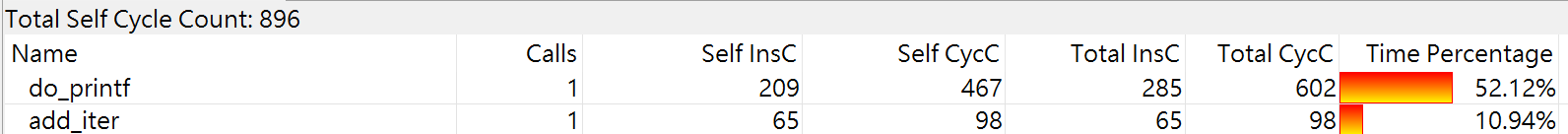


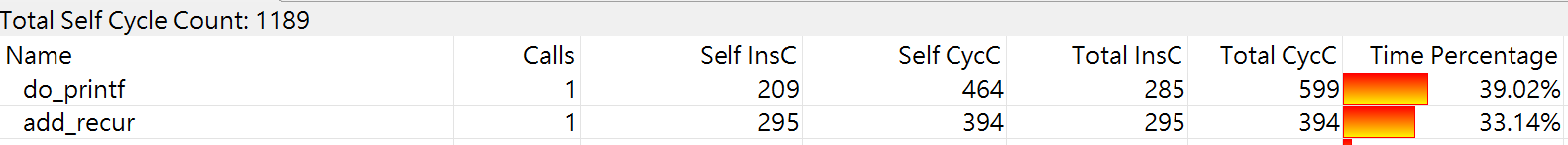
O0:不做任何優化

O1:不降低速度的前提下，盡量優化算法降低程式碼大小和執行程式碼的運行速度。

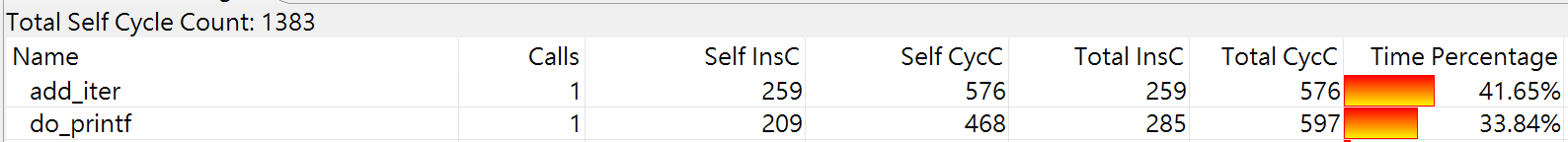
Og:提供本機與全域優化。

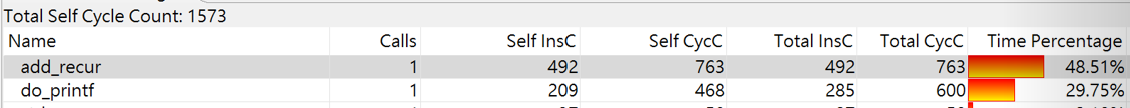
-O1





-O0





CycC for add\_iter-O1 = 98

InsC for add\_iter-O1 = 65

CPI for add\_iter-O1 = 98/65 = 1.5077

CycC for add\_iter-O0 = 576

InsC for add\_iter-O0 = 259

CPI for add\_iter-O0 = 576/259 = 2.2239

CPU time for add\_iter-O1 = 1.5077\*65 = 98.0005

CPU time for add\_iter-O0 = 2.2239\*259 = 575.9901

CycC for add\_recur-O1 = 394

InsC for add\_recur-O1 = 295

CPI for add\_recur-O1 = 394/295 = 1.3356

CycC for add\_recur-O0 = 763

InsC for add\_recur-O0 = 492

CPI for add\_recur-O0 = 763/492 = 1.5508

CPU time for add\_recur-O1 = 1.3356\*295 = 394.002

CPU time for add\_recur-O0 = 1.5508\*492 = 762.9936

Therefore, Performance: -O1>-O0

According to the result, the InsC for do\_print is similar in -O0 and -O1, and the CycC for do\_print is the same. On the other hand, the InsC, CycC and CPI are much larger in -O1 than in -O0. The main difference between different optimization levels is the CycC and InsC in add\_recur and add\_iter.

The CPU time reveals that both add\_recur and add\_iter have better performance in

-O1 than in -O0.

2.

(a)

(1) Hardware specification:

The CPU MHz:

Intel Core i3-4340(3600)>Intel Core i7-4770(3400)>Intel Core i5-4430(3000)

L3 cache:

Intel Core i7-4770(8MB I+D)> Intel Core i5-4430(6MB I+D)> Intel Core i3-4340(4MB I+D)

CPU(s) enabled:

Intel Core i3-4340:2 cores, 1 chip, 2 cores/chip, 2 threads/core

Intel Core i5-4430: 4 cores, 1 chip, 4 cores/chip

Intel Core i7-4770: 4 cores, 1 chip, 4 cores/chip, 2 threads/core

And the other specifications are same.

(2) Software specification:

Software specifications are same.

(b)

**1. 464.h264ref:**

**一張含有 桌 的圖片

自動產生的描述**

I3 reference i5 test:(323/373) = 0.8660

I3 reference i7 test:(323/323) = 1.0000

I5 reference i3 test:(373/323) = 1.1548

I5 reference i7 test:(373/323) = 1.1548

I7 reference I3 test:(323/323)=1.0000

I7 reference I5 test:(323/373)=0.8660

**2. 471.omnetpp:**

**一張含有 桌 的圖片

自動產生的描述**

I3 reference i5 test:( 257/251) =1.0239

I3 reference i7 test:(257/223)= 1.1525

I5 reference i3 test:(251/257) = 0.9767

I5 reference i7 test:(251/223) = 1.1256

I7 reference I3 test:(223/257)=0.8677

I7 reference I5 test:(223/251)=0.8884

**3. 473.astar:**

**一張含有 桌 的圖片

自動產生的描述**

I3 reference i5 test:(233/247) = 0.9433

I3 reference i7 test:( 233/200) = 1.1650

I5 reference i3 test:(247/233) = 1.0601

I5 reference i7 test:(247/200) = 1.2350

I7 reference I3 test:(200/233)=0.8584

I7 reference I5 test:(200/247)=0.8097

**4.Arithmetic mean:**

一張含有 桌 的圖片

自動產生的描述

I3 reference i5 test:(323/373+257/251+233/247)/3 = 0.9444

I3 reference i7 test:(323/323+257/223+233/200)/3 = 1.1058

I5 reference i3 test:(373/323+251/257+247/233)/3 = 1.0638

I5 reference i7 test:(373/323+251/223+247/200)/3 = 1.1718

I7 reference I3 test:(323/323+223/257+200/233)/3=0.9087

I7 reference I5 test:(323/373+223/251+200/247)/3=0.8547

(c)

I3 reference i5 test:(323/373\*257/251\*233/247)^1/3 = 0.9422

I3 reference i7 test:(323/323\*257/223\*233/200)^1/3 = 1.1032

I5 reference i3 test:(373/323\*251/257\*247/233) ^1/3 = 1.0614

I5 reference i7 test:(373/323\*251/223\*247/200) ^1/3 = 1.1709

I7 reference I3 test:(323/323\*223/257\*200/233) ^1/3=0.9065

I7 reference I5 test:(323/373\*223/251\*200/247) ^1/3=0.8541

一張含有 桌 的圖片

自動產生的描述

(d)

I7 has the best performance(shortest execution time), and I5 has the worst performance both in arithmetic and geometric mean. Compared with SPECint\_base2006 ratio, i7’s SPECint\_base2006 is the largest, and i5’s SPECint\_base2006 is the lowest, and this result is proportional to the performance of the chips.

Furthermore, their ratio are all bigger than 1, so i3, i5 and i7 are all faster than the SPECint\_base2006.

一張含有 桌 的圖片

自動產生的描述

一張含有 桌 的圖片

自動產生的描述

一張含有 桌 的圖片

自動產生的描述

3.

(a)

Cycle time = 1/(4\*10^-9) = 2.5\*10^-10

Old Time for fp = CPI\*InsC\*cycle time = 2\* 90\*10^6 \* 2.5\*10^-10 = 4.5\*10^-2

Old time for other = (110+500+50)\*10^6 \*2.5\*10^-10 = 1.65\*10^-1

We want the total time become = (16.5+4.5)\*10^-2/2 = 1.05\*10^-1

But old time for other > the total time we want, so it is impossible to make the program run two times faster by only improving the CPI for FP.

(b)

Old total time = (180+110+500+50)\*10^6 \* 2.5\*10^-10 = 2.1\*10^-1

New CPI for FP = 2\*0.69 = 1.38

New CPI for INT = 1\*0.69 = 0.69

New CPI for L/S = 5\*0.23 = 1.15

New CPI for branch = 2\*0.23 = 0.46

New total time = (90\*1.38+110\*0.69+100\*1.15+25\*0.46)\*10^6 \* 2.5\*10^-10

= 326.6\*10^6\*2.5\*10^-10 = 8.165\*10^-2

New Total Time/ Old total time = 8.165\*10^-2 / 2.1\*10^-1 = 0.3888

2.1\*10^-1 – 8.165\*10^-2 = 0.12835

The New execution time is 38.88 percent of the old execution time.

The new execution time improved, and it was 1.2835\*10^-1 seconds faster.

4.

(a)

90 = 1/2\*C\*1.25^2\*5\*10^9

Average capacity load is = 90\*2/1.25^2/5/10^9

= 2.304\*10^-8

(b)

the percentage of the total dissipated power comprised by dynamic power

= 90/(60+90) = 3/5

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

= 2/3

(c)

New dissipated power = (60+90)\*0.65 =97.5

Leakage current = 60/1.25 = 48

(1/2 \* 2.304\*10^-8 \* V^2 \* 5\*10^9 )+ 48\* V= 97.5

=> 57.6\*V^2 + 48\*V = 97.5

=> V = 0.9495

Voltage should be reduced to 0.9495 V

5.

(a)

Die area = wafer area/ die per wafer = 12.5\*12.5\*3.14/120 = 4.0885

1/(1+(0.02\*4.0885/2))^2 = 1/1.08344= 0.9230

(b)

Cost per wafer/(die per wafer\*yield) = 15/(120\*0.9230) = 0.1354

(c)

New Die area = new wafer area/ new die per wafer= Old wafer area/ 1.2\*old die per wafer = 3.4071

defects per area = 0.02\*1.35/(cm^2) = 0.027

yield = 1/(1+(0.027\*3.4071/2))^2 = 1/1.0941073 = 0.9140

New Die area = 3.4071

New yield = 0.9140