# 第五单元:

## 单元练习目的

学生在了解计算机不同类型的存储器特性后，掌握计算机对于数据的安排和操作原理和具体执行方式。

### 练习1

调试以下代码，观察在执行的过程中的反汇编代码和代码在内存中的布局方式。

sum = 0;

for (i = 0; i < MAX; i++)

sum += array[i];

### 练习2

建立工程调用以下排序函数，观察排序后的结果在内存中的具体放置方式。同时了解CPU不同级别的缓存大小和组织方式对排序操作的性能影响。

/\* Assumes n is a power of two \*/

void merge\_sort (int \* data, int n) {

int half = n >> 1;

if (n == 1) return;

binary\_sort(data, half);

binary\_sort(data + half, half);

merge(data, data + half, half);

}

### 练习3

比较以下两段代码的性能，了解如何通过改进代码达到对于CPU缓存的充分利用从而提高程序性能的。

代码段1：

int data[MAXSIZE];

for (r = 0; r < repeat; r++) {

for (i = 0; i < N; i++) {

dummy = data[i];

}

}

代码段2：

int data[MAXSIZE];

for (r = 0; r < repeat; r++) {

for (i = 0; i < N; i += stride) {

dummy = data[i];

}

}

### 练习4

对比以下代码段的性能表现，了解cache line大小对于程序性能的影响。

参考知识: Pentium Dual Core的 L1 Date(数据）和L2 cache的cache line长度为64个字节。

代码段1A:

for (i = 0; i < MAX; i++) {

<Complicated operation on A[i]>

<Complicated operation on B[i]>

<Complicated operation on C[i]>

}

代码段1B:

for (i = 0; i < MAX; i++) {

<Complicated operation on A[i]>

}

for (i = 0; i < MAX; i++) {

<Complicated operation on B[i]>

}

for (i = 0; i < MAX; i++) {

<Complicated operation on C[i]>

}

代码段2A:

int a[N];

<other stuff...>

int b[N];

<other stuff...>

int c[N];

for (i = 0; i < N; i++) {

c[i] = a[i] + b[i];

}

代码段2B:

#define CACHELINESIZE <Cache line size of system>

#define COFFSET ((2 \* CACHELINESIZE) / sizeof(int))

int a[N];

<other stuff...>

int b[N];

<other stuff...>

int

int c[N + COFFSET];

for (i = 0; i < N; i++) {

c[i + COFFSET] = a[i] + b[i];

}

代码段3A:

int data[M][N];

for (i = 0 ; i < N; i++) {

for (j = 0; j < M; j++) {

sum += data[j][i];

}

}

代码段3B:

int data[M][N];

for (j = 0; j < M; j++) {

for (i = 0 ; i < N; i++) {

sum += data[j][i];

}

}

### 练习5

调试以下代码，了解Virtual Memory里面的Page size对于程序性能的影响。

// stack.cpp -- stack timing example

#include "precise.h"

#include "stdio.h"

#define DEPTH 1000

double call\_time[DEPTH]; // time to call at this depth

// recurse -- recursively call up to DEPTH, recording

// the time for each recursion into call\_time[]

void recurse(int depth)

{

call\_time[depth] = precise\_stop();

depth++;

if (depth < DEPTH) {

precise\_start();

recurse(depth);

}

}

void main()

{

precise\_start();

recurse(0);

FILE \*rf;

if (fopen\_s(&rf, "results.txt", "w") != 0) {

printf("results.txt could not be opened\n");

return;

}

for (int i = 0; i < DEPTH; i++) {

fprintf(rf, "%g\n", call\_time[i]);

}

fclose(rf);

}