Program Profiling & Optimization

건축공학과/소프트웨어학과 2019314517 이재빈

시스템프로그램/SWE2001_42

1. 프로그램 용도 및 실행 방법

함수를 통해 리스트를 활용하는 것이 주된 목적입니다. 리스트에 Index를 통해 접근할 수도 있지만, 포인터를 이용하여 접근하는 것도 가능하기 때문에 어떻게 접근하는지 연습하고 알아봤습니다. 또한 함수 안에 함수를 사용하며, Function call을 연습하고, 초기 리스트에 저장된 int값을 ascii코드를 이용하여 char 변환하는 등 ascii 코드의 활용에도 초점이 맞춰져있습니다. 리스트에 접근 한 후 버블 정렬, 초기 10개의 값을 비교하여 리스트 선정 등을 구현했습니다.

리눅스에서 gcc -Og -pg 파일명 -o prog 입력 후 ./prog file.text 입력하면 터미널에 결과 출력됩니다.

2. 프로그램 과정 설명

프로그램은 첨부한 사진의 함수을 활용했습니다. 각 함수의 역할을 간략히 주석으로 첨부했으며, 상세한 내용은 최적화 과정을 진행하며 설명하겠습니다.

버블 정렬과 같은 경우는 정렬할 배열의 길이에 따라서 시간이 달라지기 때문에, 입력 값은 모두 고정해두었습니다. 입력 값을 고정해야 최적화 전 후를 비교할 수 있기 때문입니다.

```c
void randlist(int (*arr)); //List를 배열 요소 값으로 난수로 채우기
void randpointlist(int *arr); //List를 포인터로 접근하여 난수로 채우기
void calculatecheck(int (*arr)); //List에 Index로 접근하여 계산 결과 출력하기. 배열 할당 유무, 접근 확인.
int selectsumlist(int (*arr), int(*arr1)); //두 리스트의 첫 20개의 값을 더한 값이 큰 리스트를 선택.
int selectduplist(int (*arr), int(*arr1)); //두 리스트의 첫 10개의 값을 곱한 값이 큰 리스트를 선택.
int selectpick(int (*arr), int(*arr1)); // 위 두 함수의 결과가 동률이면 이 함수를 실행해서 최종 리스트를 선택.
void asciitobig(char (*arr)); // 아스키코드를 통해 소문자인 배열을 대문자로 변환.(아스키코드 활용)
void asciitosmall(char (*arr)); // 아스키코드를 통해 대문자인 배열을 소문자로 변환.(아스키코드 활용)
void calculateascii(char (*arr)); // 아스키코드가 정수로 계산도 가능한지 확인.
void changeascii(int (*arr), char(*arr1), int length); // 선택된 리스트를 아스키코드를 통해 char 리스트에 넣기.
void countab(char *arr); // 생성된 리스트에서 a,b의 개수가 몇 개인지 출력해주는 함수.
int mincount(char *arr); // 리스트에서 생성된 요소들 중 가장 적게 생성된 요소 출력 함수.
int maxcount(char *arr); // 리스트에서 생성된 요소들 중 가장 많이 생성된 요소 출력 함수.
void summinmax(char *arr); // 리스트에서 최대, 최소 카운트 된 요소를 호출하여 아스키코드 10진수를 더한 값을 출력.(함수에서 함수 호출 확인)
void bubblesort(char *arr,int a); // a에서 끝까지 구간에서 아스키코드를 통해 bubble sort를 수행.
void functioncall(int (*arr),char(*arr2)); // main이 아닌 함수 call을 통한 실행.
```

```
Calculate Success : 8307
TO Big : ABCDABCD
TO Small : abcdabcd
Sum by Ascii : 196
a : 1010 b : 997 Other : 37992
max count character : P
min count character : E
Max+Min 149(function call)
sorted part array : AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
```

출력 형태는 다음과 같으며, 모두 printf를 통해 출력했습니다.

처음에 randlist,randpointlist,selectsumlist,selectduplist,selectpick 함수를 사용합니다. 이 함수들을 실행하면 2개의 리스트가 생성되고, 각 값에 접근하여 조건에 맞는 리스트 하나를 선택하게 됩니다.

Calculate Success는 calculatecheck 함수에서 list에 값이 잘 할당됐는지, 접근이 잘 되는지 확인한 출력입니다.

To Big과 To Small, Sum By Ascii는 asciitobig, asciitosmall, calculateascii 함수를 통해 {'a','b','c','d','A','B','C','D'} 배열을 숫자를 이용하여 대문자 소문자 변환 및 계산을 구현했습니다.

5,6,7,8번째 줄은 각각

countab: a,b,그 외의 숫자 카운트

mincount, maxcount: 가장 작게 표현된 문자와 가장 많이 표현된 문자

summinmax : 가장 적게 표현된 문자와 가장 많이 표현된 문자의 아스키코드 합입니다.

sorted part array는 버블 정렬을 이용하여 일부를 정렬하고, 정렬됐는지 확인하는 출력입니다.

3. 최적화

최초 코드 Gprof

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 93.51    182.46   182.46        4    45.61    45.61  bubblesort
  2.22    186.79     4.34        4     1.08     1.08  countab
  2.20    191.09     4.30        4     1.07     1.07  mincount
  2.20    195.37     4.29        4     1.07     1.07  maxcount
  0.00    195.37     0.00        4     0.00     0.00  changeascii
  0.00    195.37     0.00        4     0.00     0.00  randlist
  0.00    195.37     0.00        4     0.00     2.15  summinmax
  0.00    195.37     0.00        1     0.00     0.00  asciitobig
  0.00    195.37     0.00        1     0.00     0.00  asciitosmall
  0.00    195.37     0.00        1     0.00     0.00  calculateascii
  0.00    195.37     0.00        1     0.00     0.00  calculatecheck
  0.00    195.37     0.00        1     0.00   146.53  functioncall
  0.00    195.37     0.00        1     0.00     0.00  randpointlist
  0.00    195.37     0.00        1     0.00     0.00  selectduplist
  0.00    195.37     0.00        1     0.00     0.00  selectsumlist
```

Gprof tool을 이용하여 분석한 결과입니다.

Bubblesort 함수가 시간의 93.51%를 차지하고 있는 모습입니다.

Function call 횟수가 4번으로 가장 많긴하지만, 평균적인 시간도 45.61초로 가장 많은 시간을 차지합니다.

그래서 평균 시간이 높은 bubblesort, countab, mincount, maxcount 최적화를 진행하겠습니다.

1. Bubblesort 최적화

```
void bubblesort(char *arr,int a){
  for (int j = 0; j < strlen(arr) - 1; j++) {
    for (int i = a; i < strlen(arr) - j - 1; i++) {
      if (arr[i] > arr[i + 1]) {
        int temp = arr[i];
        arr[i] = arr[i + 1];
        arr[i + 1] = temp;
      }
    }
  }
  printf("sorted part array : ");
    for(int i=a;i<strlen(arr);i++){
        printf("%c",arr[i]);
    }
    printf("\n");
};
```

1. **Code motion**

For 문 안에 strlen(arr)라는 함수가 for문이 한번 loop될 때 마다 호출되는 것을 알 수 있습니다.

Strlen도 while문을 사용한 함수이기 때문에, for문 안에 strlen함수를 사용하게 되면 for문과 while 문이 여러 번 실행됩니다. 총 3개의 strlen(arr)가 쓰이고 있는데, 항상 값이 일정하기 때문에(cod motion) 이를 하나의 변수에 저장하여 사용하는 것이 도움이 됩니다.

2. Loop unrolling

sort하는 부분은 Loop unrolling을 하기에는 if문을 추가해야하고, 변수를 더 추가해야해서 적절하지 않다고 생각했습니다. 하지만 아래 printf부분은 2개씩 프린트를 하며 look unrolling이 가능하다고 생각했습니다.

```c
void bubblesort(char *arr,int a){
    int leng=strlen(arr);
  for (int j = 0; j < leng - 1; j++) {
    for (int i = a; i < leng - j - 1; i++) {
      if (arr[i] > arr[i + 1]) {
        int temp = arr[i];
        arr[i] = arr[i + 1];
        arr[i + 1] = temp;
      }
    }
  }
  printf("sorted part array : ");
    for(int i=a;i<leng;i+=2){
        printf("%c",arr[i]);
        printf("%c",arr[i+1]);
    }
    printf("\n");
};
```

최적화 코드입니다.

3. bubblesort 최적화 결론

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 44.32      4.29     4.29        4     1.07     1.07  maxcount
 44.32      8.57     4.29        4     1.07     1.07  mincount
 11.18      9.65     1.08        1     1.08     1.08  countab
  0.31      9.68     0.03        4     0.01     0.01  bubblesort
  0.00      9.68     0.00        4     0.00     0.00  changeascii
  0.00      9.68     0.00        4     0.00     0.00  randlist
  0.00      9.68     0.00        4     0.00     2.14  summinmax
  0.00      9.68     0.00        1     0.00     0.00  asciitobig
  0.00      9.68     0.00        1     0.00     0.00  asciitosmall
  0.00      9.68     0.00        1     0.00     0.00  calculateascii
  0.00      9.68     0.00        1     0.00     0.00  calculatecheck
  0.00      9.68     0.00        1     0.00     6.45  functioncall
  0.00      9.68     0.00        1     0.00     0.00  randpointlist
  0.00      9.68     0.00        1     0.00     0.00  selectduplist
  0.00      9.68     0.00        1     0.00     0.00  selectpick
  0.00      9.68     0.00        1     0.00     0.00  selectsumlist
```

Code motion을 했을 때 182.46초에서 0.03초까지 줄었는데, 이를 통해 code motion이 시간을 상당히 줄였습니다. code motion을 통한 최적화를 하지 않은 것이 주된 요인임을 알 수 있습니다.

2. Countab 최적화

```
void countab(char *arr,int *a,int *b,int *c){

    for(int i=0;i<strlen(arr);i++){
        if(arr[i]=='a'){
            *a+=1;
        }
        else if(arr[i]=='b'){
            *b+=1;
        }
        else{
            *c+=1;
        }
    }

};
```

1. **Reducing memory references**

함수에서 for문이 실행될 때 마다, *a, *b, *c 중 하나에 무조건 접근하게 되어있습니다. 그래서 메모리 접근을 줄이고, 레지스터 변수를 이용해서 루프를 돌린 후 메모리에 한번 씩만 접근하여 값을 저장했습니다.

2. **Code motion**

Bubblesort와 마찬가지로 strlen(arr)가 계속 호출됩니다. Strlen(arr)는 항상 같은 값이기 때문에 변수로 설정해서 한번만 호출합니다.

```
void countab(char *arr,int *a,int *b,int *c){
    int leng=strlen(arr);
    int cnt1=0;
    int cnt2=0;
    int cnt3=0;
    for(int i=0;i<leng;i++){
        if(arr[i]=='a'){
            cnt1+=1;
        }
        else if(arr[i]=='b'){
            cnt2+=1;
        }
        else{
            cnt3+=1;
        }
    }
    *a=cnt1;
    *b=cnt2;
    *c=cnt3;
};
```

최적화 코드입니다.

3. Countab 최적화 결론

Bubblesort와 다르게 메모리 접근(Reducint memory references)에 관한 최적화를 했고, 동일하게 Code Motion 최적화를 진행했습니다. 1.08초에서 0.00초로 시간이 줄었습니다. countab함수 역시 code motion 최적화 진행 시에 시간이 굉장히 많이 줄었습니다.

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 50.24      4.35     4.35        4     1.09     1.09  maxcount
 49.66      8.64     4.30        4     1.07     1.07  mincount
  0.23      8.66     0.02        4     0.01     0.01  bubblesort
  0.00      8.66     0.00        4     0.00     0.00  changeascii
  0.00      8.66     0.00        4     0.00     0.00  randlist
  0.00      8.66     0.00        4     0.00     2.16  summinmax
  0.00      8.66     0.00        1     0.00     0.00  asciitobig
  0.00      8.66     0.00        1     0.00     0.00  asciitosmall
  0.00      8.66     0.00        1     0.00     0.00  calculateascii
  0.00      8.66     0.00        1     0.00     0.00  calculatecheck
  0.00      8.66     0.00        1     0.00     0.00  countab
  0.00      8.66     0.00        1     0.00     6.50  functioncall
  0.00      8.66     0.00        1     0.00     0.00  randpointlist
  0.00      8.66     0.00        1     0.00     0.00  selectduplist
  0.00      8.66     0.00        1     0.00     0.00  selectpick
  0.00      8.66     0.00        1     0.00     0.00  selectsumlist
```

3. Mincount, Maxcount 최적화

```
int mincount(char *arr){
    int a[40]={0};
    for(int i=0;i<strlen(arr);i++){
        for(int j=65;j<105;j++){
            if(arr[i]==j){
                a[j-65]+=1;
            }
        }
    }
    int count=0;
    int min=0;
    for(int i=0;i<sizeof(a)/4-1;i++){
        if(a[min]>a[i+1]){
            min=i+1;
        }
    }
    printf("min count character : %c\n",min+65);
    return min+65;
}
```

1. **Code motion**

Maxcount와 mincount 역시 strlen(arr)와 sizeof(a)함수가 변하지 않는 값임에도 반복해서 call하고 있습니다. 그래서 변수를 설정하여 한 번만 call하도록 설정했습니다.

```
int mincount(char *arr){
    int leng1=strlen(arr);
    int a[40]={0};
    int leng2=sizeof(a);
    for(int i=0;i<leng1;i++){
        for(int j=65;j<105;j++){
            if(arr[i]==j){
                a[j-65]+=1;
            }
        }
    }
    int count=0;
    int min=0;
    for(int i=0;i<leng2/4-1;i++){
        if(a[min]>a[i+1]){
            min=i+1;
        }
    }
    printf("min count character : %c\n",min+65);
    return min+65;
}
```

최적화 코드입니다.

2. Mincount, maxcount 최적화 결론

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 66.76      0.04     0.04        4    10.01    10.01  bubblesort
 33.38      0.06     0.02        4     5.01     5.01  mincount
  0.00      0.06     0.00        4     0.00     0.00  changeascii
  0.00      0.06     0.00        4     0.00     0.00  maxcount
  0.00      0.06     0.00        4     0.00     0.00  randlist
  0.00      0.06     0.00        4     0.00     5.01  summinmax
  0.00      0.06     0.00        1     0.00     0.00  asciitobig
  0.00      0.06     0.00        1     0.00     0.00  asciitosmall
  0.00      0.06     0.00        1     0.00     0.00  calculateascii
  0.00      0.06     0.00        1     0.00     0.00  calculatecheck
  0.00      0.06     0.00        1     0.00     0.00  countab
  0.00      0.06     0.00        1     0.00    45.06  functioncall
  0.00      0.06     0.00        1     0.00     0.00  randpointlist
  0.00      0.06     0.00        1     0.00     0.00  selectduplist
  0.00      0.06     0.00        1     0.00     0.00  selectpick
  0.00      0.06     0.00        1     0.00     0.00  selectsumlist
```

각각 1.07초에서 0.00초로 최적화가 잘 되었습니다.

4. Randlist 최적화

```
void randlist(int (*arr)){
    for(int i=0;i<sizeof(*arr)*10000;i++){
        int a=rand()%40+65;
        arr[i]=a;
    }
};
```

Randlist함수는 시간을 많이 차지하지 않지만, loop unrolling를 통해 최적화가 가능합니다. 왜냐하면 배열의 값을 하나씩 설정하는 것이 아니라 두 개씩 설정하게 되면 loop의 횟수를 줄여서 처리할 수 있기 때문입니다.

```
void randlist(int (*arr)){
    int leng=sizeof(*arr);
    for(int i=0;i<leng*10000;i+=2){
        int a=rand()%40+65;
        int b=rand()%40+65;
        arr[i]=a;
        arr[i+1]=b;
    }
};
```

최종 결론

총 5개의 함수에 대해서 12회의 최적화를 거쳤습니다. 그리고 최적화 방법은 Code motion 7회, Loop unrolling 2회, Reducing memory references 3회를 이용하였습니다. 195.37초에서 0.06초로 실행 시간을 줄였습니다. 가장 영향을 많이 끼친 요소는 Code Motion입니다. 특히 Bubblesort에서 Code Motion에 의해 실행시간을 굉장히 많이 줄였습니다. 물론 Bubblesort 같은 경우는 N값에 따라 실행시간이 달라지지만, 이는 알고리즘을 바꿔서 실행시간을 줄일 수 있습니다. 제 코드가 function을 여러 번 호출하지 않고, 메모리 접근이 많지 않아서 그런지 Loop unrolling과 Reducing memory references는 Code motion에 비해 효과적이지 않았습니다. 하지만 메모리 접근이 매우 많은 경우에는 Reducing memory references는 필수라고 생각합니다.