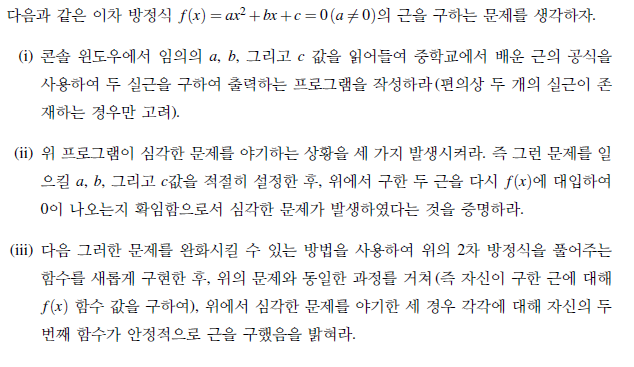
『고급 프로그래밍 기법』

4주차: 코드 최적화 기법/부동 소수점 연산에 대한 소개

고급소프트웨어실습 3분반

20151591 이지현

# 숙제 2

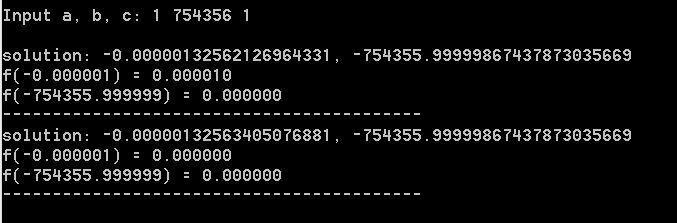


(ⅱ)

1. 에서 비슷한 숫자끼리의 뺄셈이 일어나는 경우
2. 분모에 매우 작은 숫자가 들어가는 경우, 즉 a가 매우 작을 경우
3. b>>|ac| 인 경우

(ⅲ)

|  |
| --- |
| void solution2(double a, double b, double c)  {  if (b\*b - 4 \* a\*c < 0) {  printf("해가 없다.\n");  }  else if (b\*b - 4 \* a\*c == 0) {  sol1 = (-1 \* 4 \* a \* c) / (b + sqrt(b\*b - 4 \* a\*c));  sol1 = sol1 / (2 \* a);  sol2 = (-1 \* b - sqrt(b\*b - 4 \* a\*c)) / (2 \* a);  printf("중근: %.20lf, %.20lf\n", sol1, sol2);  equation(a, b, c);  }  else {  sol1 = (-1 \* 4 \* a \* c) / (b + sqrt(b\*b - 4 \* a\*c));  sol1 = sol1 / (2 \* a);  sol2 = (-1 \* b - sqrt(b\*b - 4 \* a\*c)) / (2 \* a);  printf("solution: %.20lf, %.20lf\n", sol1, sol2);  equation(a, b, c);  }  } |



위의 결과 값은 고치기 이전의 함수의 결과 값이고, 아래의 결과 값은 고치고 난 후의 결과값이다.

# 숙제3 3

|  |
| --- |
| Loop interchange |
| void loop\_interchange() {  float Array\_1[TWO\_23][TWO\_23];  int i, j;  float run\_time;  printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  printf("\* Loop interchange \*\n");  printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  srand(time(NULL));  for (int i = 0; i < TWO\_23; i++) {  for (int j = 0; j < TWO\_23; j++) {  Array\_1[i][j] = 0;  }  }  printf("\nThe problem is to add one array of %d \* %d elements...\n\n", TWO\_23, TWO\_23);  ///////////////////////////////////  CHECK\_TIME\_START;  for (int j = 0; j < TWO\_23; j++) {  for (int i = 0; i < TWO\_23; i++) {  Array\_1[i][j] = i+j;  }  }  CHECK\_TIME\_END(run\_time);  printf("The runtime using four separate loops is %.3f(ms).\n", run\_time \* 1000);  printf("The sum is %e.\n\n", Array\_1[TWO\_23-1][TWO\_23 - 1]);  ///////////////////////////////////  //////////////////////////////////  CHECK\_TIME\_START;  for (int i = 0; i < TWO\_23; i++) {  for (int j = 0; j < TWO\_23; j++) {  Array\_1[i][j] = i + j;  }  }  CHECK\_TIME\_END(run\_time);  printf("The runtime using four separate loops is %.3f(ms).\n", run\_time \* 1000);  printf("The sum is %e.\n\n", Array\_1[TWO\_23 - 1][TWO\_23 - 1]);  ///////////////////////////////////  printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");  } |
| C:\Users\leehu\Desktop\loop interchange.JPG |
| Loop interchange 의 주요 목적은 array element에 액세스할 때 CPU 캐시를 이용하는 것이다. 프로세서가 처음으로 array element에 액세스하면 메모리에서 캐시로 전체 데이터 블록을 검색한다. 이 블록에는 첫 번째 블록 이후 더 많은 연속 요소가 있을 수 있으므로, 다음 array element 액세스 시 캐쉬에서 바로 가져올 수 있다. 따라서 메인 메모리에서 가져오는 것보다 속도가 빠르다. 캐시 누락은 루프 내에서 인접하게 액세스되는 array element가 다른 캐시 블록에서 온 경우 발생하며, loop interchange가 이를 방지할 수 있다. Loop interchange 의 효율성은 기본 하드웨어와 컴파일러가 사용하는 array 모델에 따라 달라지며, 캐시 모델에 따라 고려되어야 한다. |

|  |
| --- |
| Loop inversion |
| void loop\_inversion() {  float Array\_1[TWO\_23];  int i=0;  float run\_time;  printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  printf("\* Loop inversion \*\n");  printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  srand(time(NULL));  for (int i = 0; i < TWO\_23; i++) {  Array\_1[i] = 0;  }  i = 0;  printf("\nThe problem is to add one array of %d elements...\n\n", TWO\_23, TWO\_23);  ///////////////////////////////////  CHECK\_TIME\_START;  while (i < TWO\_23) {  Array\_1[i] = i;  i++;  }  CHECK\_TIME\_END(run\_time);  printf("The runtime using four separate loops is %.3f(ms).\n", run\_time \* 1000);  printf("The sum is %e.\n\n", Array\_1[TWO\_23 - 1]);  ///////////////////////////////////  for (int i = 0; i < TWO\_23; i++) {  Array\_1[i] = 0;  }  i = 0;  //////////////////////////////////  CHECK\_TIME\_START;  if (i < TWO\_23) {  do {  Array\_1[i] = i;  i++;  } while (i < TWO\_23);  }  CHECK\_TIME\_END(run\_time);  printf("The runtime using four separate loops is %.3f(ms).\n", run\_time \* 1000);  printf("The sum is %e.\n\n", Array\_1[TWO\_23 - 1]);  ///////////////////////////////////  printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");  } |
|  |
| Loop inversion은 while 문을 do while 문으로 바꿔 block을 더해줌으로써 컴파일러 최적화와 루프 변형을 해준다. 이는 파이프라이닝으로 인한 퍼포먼스 향상을 가져올 수 있다. |

|  |
| --- |
| Loop-invariant code motion |
| void loop\_invariant\_code\_motion() {  float Array\_1[TWO\_23];  int i = 0;  float run\_time;  float x, y, z;  float t1;  printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  printf("\* Loop-invariant code motion \*\n");  printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  srand(time(NULL));  x =0;  y = rand() - RAND\_MAX / 2.0f;  z = rand() - RAND\_MAX / 2.0f;  for (int i = 0; i < TWO\_23; i++) {  Array\_1[i] = 0;  }  i = 0;  printf("\nThe problem is to add one array of %d elements...\n\n", TWO\_23, TWO\_23);  ///////////////////////////////////  CHECK\_TIME\_START;  for (i = 0; i < TWO\_23; ++i) {  x = y + z;  Array\_1[i] = 6 \* i + x \* x;  }  CHECK\_TIME\_END(run\_time);  printf("The runtime using four separate loops is %.3f(ms).\n", run\_time \* 1000);  printf("The sum is %e.\n\n", Array\_1[TWO\_23 - 1]);  ///////////////////////////////////  for (int i = 0; i < TWO\_23; i++) {  Array\_1[i] = 0;  }  i = 0;  x = 0;  //////////////////////////////////  CHECK\_TIME\_START;  x = y + z;  t1 = x \* x;  for (i = 0; i < TWO\_23; ++i) {  Array\_1[i] = 6 \* i + t1;  }  CHECK\_TIME\_END(run\_time);  printf("The runtime using four separate loops is %.3f(ms).\n", run\_time \* 1000);  printf("The sum is %e.\n\n", Array\_1[TWO\_23 - 1]);  ///////////////////////////////////  printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");  } |
|  |
| Loop-invariant code motion는 덜 실행되어 속도 향상에 도움을 준다. 또한, 변수들을 레지스터에 저장하여 매번 계산하거나 메모리에 접근하지 않아도 된다. |

|  |
| --- |
| Loop splitting |
| void loop\_splitting() {  float Array\_1[TWO\_23], Array\_2[TWO\_23];  int i = 0, j;  float run\_time;  printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  printf("\* Loop splitting \*\n");  printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  srand(time(NULL));  for (int i = 0; i < TWO\_23; i++) {  Array\_1[i] = 0;  Array\_2[i] = rand() - RAND\_MAX / 2.0f;  }  i = 0;  printf("\nThe problem is to add one array of %d elements...\n\n", TWO\_23);  ///////////////////////////////////  CHECK\_TIME\_START;  j = TWO\_23 - 1;  for (i = 0; i < TWO\_23; ++i) {  Array\_1[i] = Array\_2[i] + Array\_2[j];  j = i;  }  CHECK\_TIME\_END(run\_time);  printf("The runtime using four separate loops is %.3f(ms).\n", run\_time \* 1000);  printf("The sum is %e.\n\n", Array\_1[TWO\_23 - 1]);  ///////////////////////////////////  for (int i = 0; i < TWO\_23; i++) {  Array\_1[i] = 0;  }  i = 0;  //////////////////////////////////  CHECK\_TIME\_START;  Array\_1[0] = Array\_2[0] + Array\_2[TWO\_23-1];  for (i = 1; i < TWO\_23; ++i) {  Array\_1[i] = Array\_2[i] + Array\_2[i-1];  }  CHECK\_TIME\_END(run\_time);  printf("The runtime using four separate loops is %.3f(ms).\n", run\_time \* 1000);  printf("The sum is %e.\n\n", Array\_1[TWO\_23 - 1]);  ///////////////////////////////////  printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");  } |
|  |
| Loop splitting은 컴파일러 최적화 기술로 루프를 단순화하거나 의존성을 제거하기 위해 인덱스범위의 연속된 다른 부분을 반복하는 하나의 multiple loop에서 시도한다. |

|  |
| --- |
| Loop tiling |
| void loop\_tiling() {  float Array\_1[TWO\_23][TWO\_23], Array\_2[TWO\_23], Array\_3[TWO\_23];  int i = 0, j, x, y;  float run\_time;  printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  printf("\* Loop tiling \*\n");  printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  srand(time(NULL));  for (int i = 0; i < TWO\_23; i++) {  Array\_2[i] = rand() - RAND\_MAX / 2.0f;  for (j = 0; j < TWO\_23; j++) {  Array\_1[i][j] = rand() - RAND\_MAX / 2.0f;  }  }  printf("\nThe problem is to add one array of %d elements...\n\n", TWO\_23);  ///////////////////////////////////  CHECK\_TIME\_START;  for (i = 0; i < TWO\_23; i++) {  Array\_3[i] = 0;  for (j = 0; j < TWO\_23; j++) {  Array\_3[i] = Array\_3[i] + Array\_1[i][j] \* Array\_2[j];  }  }  CHECK\_TIME\_END(run\_time);  printf("The runtime using a multiple loop is %.3f(ms).\n", run\_time \* 1000);  printf("The sum is %e.\n\n", Array\_3[TWO\_23 - 1]);  ///////////////////////////////////  //////////////////////////////////  CHECK\_TIME\_START;  for (i = 0; i < TWO\_23; i += 2) {  Array\_3[i] = 0;  Array\_3[i + 1] = 0;  for (j = 0; j < TWO\_23; j += 2) {  for (x = i; x < min(i + 2, TWO\_23); x++) {  for (y = j; y < min(j + 2, TWO\_23); y++) {  Array\_3[x] = Array\_3[x] + Array\_1[x][y] \* Array\_2[y];  }  }  }  }  CHECK\_TIME\_END(run\_time);  printf("The runtime using a multiple loops is %.3f(ms).\n", run\_time \* 1000);  printf("The sum is %e.\n\n", Array\_3[TWO\_23 - 1]);  ///////////////////////////////////  printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");  } |
|  |
| Loop tiling은 loop transformation의 집합의 지역성 최적화와 병렬화 또는 loop overhead를 줄이기 위해 사용된다. 특히, 메모리 접근 지연이나 선형 대수 알고리즘에서 캐시 재사용으로 인한 대역폭의 필요로 인해 사용된다. |