Assignment 2

Programming for Performance, Group SELF

Naman Singla 200619 nsingla20@iitk.ac.in

Question 3

Problem 3

[5+5+5+5+10 marks]

Consider the following code:

```
int i, j, t, k;
for (t = 0; t < 1024; t++) {
 for (i = t; i < 1024; i++) {
  for (j = t; j < i; j++) {
   for (k = 1; k < j; k++) {
    S(t, i, j, k);
   }
  }
 }
}
```

The data dependences for the loop are given to be (1,0,-1,1), (1,-1,0,1), and (0,1,0,-1).

(a) What are the valid permutations of the loop? Why?

(b) Which loops, if any, are valid to unroll and jam? Why?

(c) What tiling is valid, if any? Why?

(d) Which loops, if any, are parallel?

(e) Show code for the *tikj* form of the code. For this part, ignore the above dependences and assume *tikj* permutation is allowed.

Answer:

(a) **What are the valid permutations of the loop? Why?**
Permutation of the loops in a perfect nest is legal iff there are no "-" direction as the leftmost non–"0" direction in any direction vector. If any permuted vector is lexicographically negative, permutation is illegal.
Our distance vectors are (1,0,-1,1), (1,-1,0,1), and (0,1,0,-1). The given form is tijk:

Distance Vectors

	t	i	j	k
1.	1	0	-1	1
2.	1	-1	0	1
3.	0	1	0	-1

We have to permute in a way so that the vector doesn't become negative.

$(0,1,0,-1) \implies$ i should come before k $\implies$ **ik**
$(1,-1,0,1) \implies$ t should come before i $\implies$ **tik**
$(1,0,-1,1) \implies$ j can't be at first position
$\implies$ **tjik, tijk, tikj**

So, the only possible permutations are tjik, tijk and tikj.

(b) **Which loops, if any, are valid to unroll and jam? Why?**
Check for unrolling?
Complete unroll/jam of a loop is equivalent to a loop permutation that moves that loop innermost, without changing order of other loops. If such a loop permutation is valid, unroll/jam of the loop is valid.
Now, as shown in part (a), only possible permutation are tjik, tijk, tikj and we haven't changed relative order of t, i, k.
Hence, only loops available for unrolling are **j and k**(original).

(c) **What tiling is valid, if any? Why?**
Validity Condition for Loop Tiling?
A contiguous band of loops can be tiled if they are fully permutable. A band of loops is fully permutable if all permutations of the loops in that band are legal.
Clearly(possibles: tjik, tikj), this is not the case here. However, partial tiling is possible in the following pairs :

- i,j
- j,k

(d) **Which loops, if any, are parallel?**
If a loop carries no dependence with respect to any of the dependence vectors, it is said to be parallel. A loop doesn't have loop carried dependencies with respect to a single dependence vector if either that component is zero or a more significant position (to its left) in the dependence vector has a positive component.

- Loop t :
  NOT PARALLEL due to the 1 in its position in the dependence vectors (1,0,-1,1) and (1,-1,0,1).
- Loop i:
  NOT PARALLEL due to the 1 in its position in the dependence vector (0,1,0,-1).

- Loop j:
  PARALLEL since it has 0 in its position in these dependence vectors (1,-1,0,1) and (0,1,0,-1)
  NOTE: It has a -1 for (1,0,-1,1) but it has 1 in a position to its left (at loop t).
- Loop k:
  PARALLEL since it has 1 in dependence vectors (1,0,-1,1) and (1,-1,0,1) but there is 1 in a position to its left(at loop t) and it has -1 in (0,1,0,-1) there is a 1 in position to its left(at loop t)

(e) **Show code for the tikj form of the code. For this part, ignore the above dependences and assume tikj permutation is allowed.**
The following code will give the same permutations to function S() in tikj form:

```
int i, j, t, k;
for (t = 0; t < 1024; t++) {
    for (i = t; i < 1024; i++) {
        for (k = 1; k < i-1; k++) {
            for (j = max(t,k+1); j < i; j++) {
                S(t, i, j, k);
            }
        }
    }
}
```

Logic: k goes from 1 to $j-1$ and max $j = i-1$.

Question 4

Problem 4

[50+30 marks]

Part (i) Perform loop transformations to improve the performance of the attached C code (`prob4-v0.c`) for sequential execution on one core (i.e., no multithreading). You may use any transformation we have studied, e.g., loop permutation and loop tiling, but no array padding or layout transformation of arrays is allowed.

Explain the reason for the poor performance of the provided reference code, your proposed optimizations (with justifications) to improve performance, and the improvements achieved. Some transformations will lead to speedup, some will not. Include all the transformations that you tried, even if they did not work. Finally, summarize the set of transformations (e.g., xx times unrolling + yy permutation) that gave you the best performance. Report your speedup using the GNU gcc compiler. You can compile the reference code with `gcc -O3 -std=c17 -D_POSIX_C_SOURCE=199309L prob4-v0.c -o prob4-v0.out`.

Use a workstation in the KD lab for your performance evaluation. A good start is to check the processor architecture and cache parameters for your workstation to better understand the impact on performance. Include the system description (e.g., levels of private caches, L1/L2 cache size, processor frequency) in your report. Also include the name of the workstation in your report. The TAs will reuse the same system to reproduce the performance results.

Part (ii) This part is open ended. You are free to apply any valid code optimization trick (e.g., LICM, function inlining, and changing function prototypes) on the version from Part (i) for improved performance.

Answer:
Part(i)

```
s1 = floor((dd2 - dd1) / dd3);
s2 = floor((dd5 - dd4) / dd6);
...
s10 = floor((dd29 - dd28) / dd30);

for (int r1 = 0; r1 < s1; ++r1) {
  x1 = dd1 + r1 * dd3;

  for (int r2 = 0; r2 < s2; ++r2) {
    x2 = dd4 + r2 * dd6;
          ...
                  for (int r10 = 0; r10 < s10; ++r10) {
                    x10 = dd28 + r10 * dd30;

                    q1 = fabs(c11 * x1 + c12 * x2 + ... + c110 * x10 - d1);
```

Total time = 421.382426 seconds

(a) Making it a perfect loop nest won't help as the inner statement gets executed more times.

Lets try to minimize the the number of variables. Loops index variables r_n depend on s_n for upper limits and inner statements use only x_n that is calculated from r_n. We can note that s_n has same dd_n as multiplied by r_n to obtain x_n. So, we can completely remove s_n by incrementing r_n by dd_n instead of 1. Upper and lower limit can be same as used to calculate s_n.
After this we can also see that $x_n = r_n$ now. Hence, replace x with r. And what we got is a perfect nest with reduce number of variables.

```
for (r1 = dd1; r1 < dd2; r1+=dd3) {
    for (r2 = dd4; r2 < dd5; r2+=dd6) {
            ....
                    for (r10 = dd28; r10 < dd29; r10+=dd30) {

                      q1 = fabs(c11 * r1 + c12 * r2 + ... + c110 * r10 -
   d1);
```

Total time = 361.630364 seconds, Speedup : 1.16x (200619-prob4-v1-1.c)
NOTE : I will take above as reference now

(b) Since there is not data dependence in inner-most loop, we can permute the loops in any manner. I tried reversing the order of r_n by time taken increased significantly

```
for (r10 = dd28; r10 < dd29; r10+=dd30)
    for (r9 = dd25; r9 < dd26; r9+=dd27)
      for (r8 = dd22; r8 < dd23; r8+=dd24)
        for (r7 = dd19; r7 < dd20; r7+=dd21)
          for (r6 = dd16; r6 < dd17; r6+=dd18)
            for (r5 = dd13; r5 < dd14; r5+=dd15)
              for (r4 = dd10; r4 < dd11; r4+=dd12)
                for (r3 = dd7; r3 < dd8; r3+=dd9)
                  for (r2 = dd4; r2 < dd5; r2+=dd6)
                    for (r1 = dd1; r1 < dd2; r1+=dd3)

```

Total time = 556.030280 seconds (200619-prob4-v1-2.c)

(c) I tried unrolling innermost r10 loop but it doesn't show any improvement in performance. May be due to the jump inserted using conditional statement and my index variables are not integers.
Total time = 413.894837 seconds (200619-prob4-v1-unrol.c)

(d) We don't have arrays that's why most optimizations are not implementable. Index of loops are used as variables in calculations rather than being used as index of some array. We can take advantage of this by scalar expansion. We can store some precomputed calculation at each step. Continuing from (a) , We can store the value of q_n of parent loop and then do computation in outer loops.

```
for (r1 = dd1; r1 < dd2; r1+=dd3) {
    qb[1][1] = c11 * r1 - d1;
    qb[1][2] = c21 * r1 - d2;
    ...
    qb[1][10] = c101 * r1 - d10;
    for (r2 = dd4; r2 < dd5; r2+=dd6) {
      qb[2][1] = qb[1][1] + c12 * r2;
      qb[2][2] = qb[1][2] + c22 * r2;
      ...
      qb[2][10] = qb[1][10] + c102 * r2;

        ....
                    for (r10 = dd28; r10 < dd29; r10+=dd30) {
                      q1 = fabs(qb[10][1]);
                      q2 = fabs(qb[10][2]);
                      ...
                      q10 = fabs(qb[10][10]);
```

Total time = 166.152974 seconds, Speedup : 2.5x (200619-prob4-v1.c)
10 Scalar expansion + LICM

(e) Apart from this I also tried to maximize the cache hits. But those programs had less or no impact on performance. (200619-prob4-v1-3.c) (200619-prob4-v1-3-1.c)

- **Reason for poor performance**
  The main reason for poor performance is non-utilisation of cache model. In this code memory is being accesses are vary far from each other (no spatial locality). There are also some unnecessary variables which can be removed.

- **Specification of system used**

```
Static hostname: csews54
       Icon name: computer-desktop
         Chassis: desktop
      Machine ID: 456294d50a3447caa9bac60dbf33dd8f
         Boot ID: c1ae26eab99845bcbe0ac98accc50c4f
Operating System: Ubuntu 22.04.1 LTS
          Kernel: Linux 5.15.0-50-generic
    Architecture: x86-64
 Hardware Vendor: Lenovo
  Hardware Model: ThinkCentre M920t
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         39 bits physical, 48 bits virtual
  Byte Order:            Little Endian
CPU(s):                  12
  On-line CPU(s) list:   0-11
Vendor ID:               GenuineIntel
  Model name:            Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
    CPU family:          6
    Model:               158
```

```
    Thread(s) per core:  2
    Core(s) per socket:  6
    Socket(s):           1
    Stepping:            10
    CPU max MHz:         4600.0000
    CPU min MHz:         800.0000
    BogoMIPS:            6399.96
Virtualization features:
  Virtualization:        VT-x
Caches (sum of all):
  L1d:                   192 KiB (6 instances)
  L1i:                   192 KiB (6 instances)
  L2:                    1.5 MiB (6 instances)
  L3:                    12 MiB (1 instance)
NUMA:
  NUMA node(s):          1
  NUMA node0 CPU(s):     0-11
Vulnerabilities:
  Itlb multihit:         KVM: Mitigation: VMX disabled
  L1tf:                  Mitigation; PTE Inversion; VMX conditional cache
    flushes, SMT vulnerable
  Mds:                   Mitigation; Clear CPU buffers; SMT vulnerable
  Meltdown:              Mitigation; PTI
  Mmio stale data:       Mitigation; Clear CPU buffers; SMT vulnerable
  Retbleed:              Mitigation; IBRS
  Spec store bypass:     Mitigation; Speculative Store Bypass disabled
   via prctl and seccomp
  Spectre v1:            Mitigation; usercopy/swapgs barriers and __user
   pointer sanitization
  Spectre v2:            Mitigation; IBRS, IBPB conditional, RSB filling,
    PBRSB-eIBRS Not affected
  Srbds:                 Mitigation; Microcode
  Tsx async abort:       Mitigation; TSX disabled
```

Part(ii)

Continuing from (d) in Part(i),

Lets convert c_n to an 2D array to reduce the number of parameters in function and more control over structure. Then we will swap this 2D array. So that we get our access pattern in row-major.

Total time = 169.982355 seconds, Speedup : 2.5x (200619-prob4-v2.c)