Holden Sandlar

Advanced Computer Architecture

0301-810

Assignment 4

**Problem 4.9**

a. Arithmetic intensity is defined as the ratio of floating-point operations per byte of memory accessed.

Looking at the given loop we can see that there are 6 floating point operations performed per 6 memory references thus yielding an arithmetic intensity of 6/6 = 1

d. We know that there are m = 6 chimes from part c.

We also know that for a vector length of n, we require m\*n clock cycles

Vector length in this case is 64

It is also important to consider the start-up overhead of each unit.

4 mult’s – 4 \* 8 = 32 cycles overhead

2 add/sub’s – 2 \* 5 = 10 cycles overhead

1 load – 1 \* 15 \* 6 = 90 cycles overhead

**Now, total cycles = 6\*64 + 32 + 10 + 90 = 516 clock cycles**

**516 clock cycles / 128 results = 4.03125 cycles/result**

**Problem 4.13**

GPU Architecture with 10 SIMD Processors

SIMD width = 32

SIMD # of lanes = 8/CPU for single-precision and load/store instructions 🡪 each non-diverged SIMD instruction can produce 32 results every 4 cycles.

Assume kernel with 80% of threads active

Assume 70% of all SIMD instructions executed are single-precision arithmetic

Assume 20% of all SIMD instructions execute are load/store

Assume average SIMD instruction issue rate of 0.85

Assume GPU has clock speed of 1.5 GHz

1. Compute the throughput in GFLOP/sec.
2. 1. Increasing number of single precision lanes to 16
   2. Increasing the number of SIMD processors to 15
   3. Adding cache that will reduce memory latency by 40% -- increase instruction issue rate to 0.95

**Problem 4.14**

a.

for(I = 0; i<100; i++)

{

A[i] = B[2\*i+4];

B[4\*i+5] = A[i];

}

Does the loop have loop-carried dependencies?

**No, the outcome of each statement does not depend on previously executed iterations.**

**To further prove this we can do the following:**

**2i + 4 = 4i + 5 🡪 i only has 1 solution = -0.5 which is outside the bounds of the loop, thus proving no dependence.**

1. For(I = 0; i<100;i++)

{

A[i] = A[i] \* B[i]; /\* S1 \*/

B[i] = A[i] + c; /\* S2 \*/

A[i] = C[i] \* c ; /\* S3 \*/

C[i] = D[i] \* A[i]; /\* S4 \*/

}

Following dependencies exist:

* True dependency from S1 to S2 and from S3 to S4 because of A[i]
* Anti-dependency from S1 to S2 because of B[i]
* Anti-dependency from S2 to S3 because of A[i]
* Anti-dependency from S3 to S4 because of C[i]
* Output dependency from S1 to S3 because of A[i]

Eliminate the dependencies by renaming:

For(int i = 0; i<100; i++)

{

T[i] = T[i] \* G[i]; /\* A renamed to T and B renamed to G to remove output dependency and anti- dependency \*/

B[i] = T[i] + c; /\* A renamed to T \*/

A[i] = J[i] \* c; /\* C renamed to J \*/

C[i] = D[i] \* A[i];

}

True dependencies still exist from S1 to S2 on T[i] and from S3 to S4 on A[i]!

1. Consider the following loop:

For(int i=0;i<100;i++)

{

A[i] = A[i] + B[i]; /\* S1 \*/

B[i+1] = C[i] + D[i]; /\* S2 \*/

}

Are there dependencies between S1 and S2?

**No**

Is this loop parallel?

This is Example on page 317

Directly from text: “S1 uses the value assigned in the previous iteration by statement S2, so there is a loop-carried dependence between S2 and S1. Despite this loop-carried dependence, this loop can be made parallel. This dependence is not circular; neither statement depends on itself, and although S1 depends on S2, S2 does not depend on S1. A loop is parallel if it can be written without a cycle in the dependences, since the absence of a cycle means that the dependences give a partial ordering on the statements.

Although there are no circular dependences in the above loop, it must be transformed to conform to the partial ordering and expose the parallelism. Two observations are critical to this transformation.

1. There is no dependence from S1 to S2. If there were, then there would be a cycle in the dependences and the loop would not be parallel. Since this other dependence is absent, interchanging the two statements will not affect the execution of S2.
2. On the first iteration of the loop, statement S2 depends on the value of B[0] computed prior to initiating the loop.

These two observations allow us to replace the loop above with the following code sequence:

A[0] = A[0] + B[0];

For( i=0;i<99;i++)

{

B[i+1] = C[i] + D[i];

A[i+1] = A[i+1] + B[i+1];

}

B[100] = C[99] + D[99];

The dependence between the two statements is no longer loop carried, so that iterations of the loop may be overlapped, provided the statements in each iteration are kept in order.”