MEMORY BANDWITH MEASUREMENT BETWEEN THE MAIN MEMORY AND CPU WITH PARALLEL PROGRAMMING

SYSTEM DESCRIPTION

Platform: **ASUS NOTEBOOK**

Windows Edition: **Windows 8.1 Pro**

Processor: **Intel(R) Core(Tm) i5-4200U CPU @ 1.60 GHz 2.30 GHz**

Installed Memory (RAM): ***8.00 (7.89 GB Usable)***

System Type: **64-bit Operating System, x64-based Processor**

Number of CPUs: **4**

Number of Cores: **2**

L1 Cache size: **128 KB**

L2 Cache size: **512 KB**

L3 Cache size: **3MB**

CODE C

```
typedef struct {
  double *array;
  int elems;
  double sum;
  int stride;
} threadInfoStep2;
void *sumThread( void *ud ) {
  threadInfo *ti = (threadInfo *)ud;
  for(i=0;i<ti->elems;i += ti->stride) {
    ti->sum += ti->array[i];
  }
}
void *step2( void *ud ) {
  threadInfoStep2 *ti = (threadInfoStep2 *)ud;
  double variance, std_deviation, sum = 0, sum1 = 0;
  for(i=0;i<ti->elems;i += ti->stride) {
   ti->sum += ti->array[i];
  }
  for (i = 0; i < ti->elems; i += ti->stride) {
      sum = sum + ti->array[i];
  }
  ti->avg = sum / (double)ti->elems;
  for (i = 0; i < ti->elems; i += ti->stride) {
      sum1 = sum1 + pow((ti->array[i] - ti->avg), 2);
  }
variance = sum1 / (double)ti->elems;
  ti->deviation = sqrt(variance);
}
int uniform_distribution(int rangeLow, int rangeHigh) {
    double myRand = rand()/(1.0 + RAND_MAX);
    int range = rangeHigh - rangeLow + 1;
    int myRand_scaled = (myRand * range) + rangeLow;
    return myRand_scaled;
}
```

CODE JAVA

```
class MemoryBandwidth {
	private static final int MINBYTES = (1 << 18);
	private static final int MAXBYTES = (1 << 27);
	private static final int MAXSTRIDE = 64;
	private static final int MAXELEMS = MAXBYTES * 8 / Integer.SIZE;
	private static int data[] = new int[MAXELEMS];
	private static volatile int sink;
	private static final int CORES = 4;
	private static SumThread[] st;
	public static void main(String[] args) {
		Compiler.disable();
		initData();
		for (int stride = 1; stride <= MAXSTRIDE; stride = stride * 2)
			System.out.print("s" + stride + "\t");
		System.out.print("\n");
		for (int size = MAXBYTES; size >= MINBYTES; size >>= 1) {
			if (size > (1 << 20))
				System.out.print(size / (1 << 20) + "m\t");
			else
				System.out.print(size / 1024 + "k\t");

			for (int stride = 1; stride <= MAXSTRIDE; stride = stride * 2) {
				System.out.print(run(size, stride) + "\t");
			}
			System.out.println();
		}
	}
	private static void testThread(int elems, int stride) {
		for (int i = 0; i < CORES; i++) {
			st[i].start();
		}
		/* wait for the threads to finish! */
		try {
			for (int i = 0; i < CORES; i++) {
				st[i].join();
			}
		} catch (InterruptedException e) {
			System.out.println("Interrupted");
		}
	}
	private static void prepareThread(int elems, int stride) {
		st = new SumThread[CORES];
		for (int i = 0; i < CORES; i++) {
			st[i] = new SumThread(i * (elems / CORES),
					(i * (elems / CORES) + (elems / CORES)), data);
		}
	}
	private static void test(int elems, int stride) {
		int i;
		int result = 0;
		sink = 0;
		for (i = 0; i < elems; i += stride) {
			result += data[i];
		}
		sink = result;
	}
	private static double run(int size, int stride) {
		int elems = size * 8 / Integer.SIZE;
		prepareThread(elems, stride);
		testThread(elems, stride);
		prepareThread(elems, stride);
```

RESULT

USING SINGLE THREAD

Clock frequency is approx. 2290.4 MHz (Measured from Program)
Memory Speed (MB/sec)
S= Stride
Size = Data size

Size	s1	s2	s4	s8	s16	s32	s64
64m	6778.6	6614.7	3996.0	1836.5	1240.0	955.0	988.2
32m	6779.1	6601.2	4006.0	1838.7	1245.8	973.2	1016.9
16m	6763.9	6628.5	4096.6	1877.1	1282.7	1047.6	1101.3
8m	6682.6	6617.6	4399.9	1998.0	1465.4	1246.2	1316.5
4m	6749.1	6590.8	5102.2	2752.5	1663.4	1504.4	1974.7
2m	6865.1	6673.7	6271.1	4021.6	3680.8	3591.9	4056.6
1024k	6903.2	6902.2	6855.4	4117.8	3787.2	3681.8	4172.2
512k	6636.4	6634.2	6581.8	4126.9	3830.5	3727.2	4188.1
256k	6900.2	6896.3	6882.2	6434.9	6382.9	6658.1	6386.2
128k	6899.6	6896.7	6890.4	6817.8	6732.2	6564.9	6399.2
64k	6895.8	6887.9	6866.5	6744.3	6578.7	6296.2	6171.9
32k	6886.0	6866.5	6842.7	6783.3	6653.4	6478.8	6171.9
16k	6882.8	6860.2	6798.0	6710.5	6478.8	6171.9	5637.8
8k	6860.2	6798.0	6710.5	6478.8	6171.9	5637.8	12215.2
4k	6798.0	6710.5	6478.8	6171.9	11275.6	4806.0	6107.6
2k	6710.5	6478.8	6171.9	10109.2	9772.2	6980.1	4071.7

Table 1: Single Thread in C

Size	s1	s2	s4	s8	s16	s32	s64	
128m	4239.0	4571.0	3088.0	1898.0	859.0	609.0	430.0	
64m	5147.0	4729.0	3104.0	1784.0	916.0	522.0	416.0	
32m	4907.0	5304.0	3043.0	1817.0	926.0	566.0	440.0	
16m	5138.0	5279.0	3449.0	1915.0	890.0	589.0	377.0	
8m	5962.0	5249.0	3489.0	1544.0	916.0	661.0	579.0	
4m	5090.0	4981.0	3518.0	2269.0	1154.0	916.0	809.0	
2m	4733.0	5190.0	3912.0	2166.0	1680.0	1191.0	1310.0	
1024k	6168.0	5890.0	4681.0	2674.0	1680.0	1638.0	1638.0	
512k	6241.0	6096.0	3360.0	3276.0	1724.0	1820.0	2048.0	
256k	6393.0	3971.0	5041.0	4096.0	3276.0	2730.0	2048.0	

Table 2: Single Thread in JAVA

USING MULTITHREAD

Clock frequency is approx. 2295.0 MHz

Memory Mountain (MB/sec)

Number of Threads: 2

Size	s1	s2	s4	s8	s16	s32	s64

64m	11223.3	9127.9	4667.8	2290.6	1309.2	954.7	1218.2
32m	10769.0	8167.2	4248.8	2189.4	1238.2	908.7	1075.5
16m	9890.2	7599.7	4002.3	2026.9	1154.2	897.9	906.3
8m	8660.4	6442.3	3090.0	1548.7	865.4	781.6	629.7
4m	7481.0	5346.2	2853.3	1431.2	1180.4	699.1	401.6
2m	6665.9	4553.4	2674.2	1348.4	757.7	442.9	234.7
1024k	4276.4	2639.9	1604.0	840.9	479.5	241.8	123.3
512k	2807.7	1641.7	922.1	473.2	235.8	116.0	61.8
256k	1572.4	944.6	475.8	244.9	121.7	60.3	30.8
128k	958.7	502.9	239.6	121.3	59.8	28.6	14.5
64k	492.2	195.5	119.3	61.2	29.8	15.5	6.5
32k	237.4	118.5	61.2	27.9	13.1	6.9	3.5
16k	122.5	55.2	27.9	15.2	6.9	3.7	1.7
8k	60.1	27.8	13.7	7.8	3.5	1.9	0.9
4k	31.0	15.3	7.7	3.8	2.0	0.9	0.5
2k	15.1	7.7	3.7	1.9	0.9	0.5	0.2

Table 3: Multithread with 2 threads in C

Size	s1	s2	s4	s8	s16	s32	s64	
128m	16446.0	8354.0	4110.0	2078.0	1010.0	528.0	259.0	
64m	15502.0	7564.0	3911.0	1931.0	927.0	471.0	246.0	
32m	14807.0	7093.0	3706.0	1760.0	901.0	463.0	235.0	
16m	13662.0	6904.0	3449.0	1706.0	851.0	413.0	212.0	
8m	8867.0	5584.0	2766.0	1232.0	717.0	333.0	179.0	
4m	8192.0	4306.0	1885.0	1063.0	543.0	263.0	132.0	
2m	5295.0	2781.0	1524.0	742.0	335.0	177.0	92.0	
1024k	3297.0	1489.0	832.0	416.0	196.0	97.0	42.0	
512k	1941.0	981.0	464.0	237.0	112.0	58.0	27.0	
256k	897.0	445.0	243.0	120.0	63.0	31.0	16.0	

Table 4: Multithread with 2 threads in JAVA

Clock frequency is approx. 2290.9 MHz

Memory Mountain (MB/sec)

Number of Threads: 4

Size	s1	s2	s4	s8	s16	s32	s64
64m	16058.9	8964.8	4526.3	2263.8	1231.1	932.4	1315.8
32m	14476.6	8263.3	4303.2	2157.4	1144.4	886.8	1056.3
16m	12723.3	7464.6	3866.6	1948.3	1103.3	842.1	707.7
8m	10804.8	6572.9	3453.8	1730.7	1090.5	723.5	432.9
4m	8722.7	5072.9	2763.6	1380.0	777.0	421.6	248.4
2m	5236.1	3285.6	1623.1	875.6	469.2	249.9	121.1
1024k	2928.0	1514.4	860.9	441.1	205.1	110.4	55.0
512k	1654.7	880.4	460.9	224.4	107.5	57.4	27.5
256k	906.8	450.0	223.0	114.8	64.7	27.1	13.1
128k	399.6	224.5	119.3	51.0	29.8	16.1	7.7
64k	263.1	130.8	65.5	27.0	14.5	6.9	3.6
32k	120.2	62.0	25.7	13.8	7.1	3.6	1.8

16k	51.5	27.5	14.1	6.8	3.5	1.8	0.9
8k	28.7	13.1	6.8	3.6	1.5	0.8	0.4
4k	14.1	6.8	3.5	1.6	0.8	0.5	0.2
2k	7.0	3.6	1.8	0.8	0.4	0.3	0.1

Table 5: Multithread with 4 threads in C

Size	s1	s2	s4	s8	s16	s32	s64	
128m	19146.0	13780.0	7090.0	3403.0	1710.0	875.0	400.0	
64m	21824.0	11902.0	3698.0	2858.0	1543.0	757.0	400.0	
32m	21959.0	10645.0	4328.0	2606.0	1351.0	661.0	236.0	
16m	14979.0	6961.0	3865.0	1775.0	1020.0	472.0	110.0	
8m	9586.0	5777.0	1941.0	1075.0	502.0	248.0	107.0	
4m	5660.0	3241.0	1829.0	800.0	388.0	175.0	91.0	
2m	2668.0	1889.0	989.0	438.0	12.0	94.0	47.0	
1024k	1989.0	896.0	300.0	194.0	104.0	51.0	31.0	
512k	1026.0	422.0	191.0	104.0	54.0	34.0	16.0	
256k	407.0	240.0	96.0	58.0	27.0	15.0	8.0	

Table 6: Multithread with 4 threads in JAVA

MEAN AND STANDARD DEVIATION

Clock frequency is approx. 2294.5 MHz

Number of Elements processed per second with 2 Threads

Size	s1	s2	s4	s8	s16	s32	s64
64m	11436.8	8996.5	4873.7	2431.6	1377.6	1063.5	1514.4
32m	11575.5	9431.5	4741.9	2381.6	1347.6	1023.1	1353.8
16m	11154.0	9001.6	4574.5	2349.4	1306.3	979.1	1162.5
8m	10519.2	8263.4	4158.4	2125.6	1372.9	1054.5	1028.0
4m	9690.6	6972.7	3922.3	2018.0	1487.2	1317.0	931.7
2m	8503.9	6868.2	5158.5	2826.2	1812.7	1032.4	658.7
1024k	6736.1	5863.9	3633.5	1879.7	949.5	513.7	259.7
512k	5763.7	3843.6	2122.8	1226.0	556.9	348.7	181.0
256k	3725.7	2007.3	1369.1	730.4	259.0	178.5	66.0
128k	2146.9	1135.5	693.3	351.3	171.7	84.0	43.8
64k	1334.7	669.5	283.8	179.2	87.6	47.7	21.0
32k	726.3	370.3	130.7	65.9	32.8	15.2	12.2
16k	370.3	172.4	89.4	45.0	23.5	12.0	6.2
8k	179.7	83.6	32.4	21.2	12.0	6.1	2.7
4k	89.9	32.3	16.2	11.5	4.2	3.0	1.3
2k	32.0	24.1	8.1	4.1	2.0	1.3	0.7

Table 7: Mean and Standard deviation using 2 Threads in C

EXPLANATION

I have used C Programming language and JAVA for the same problem.

2014280162

Step1. Used PThread to count the memory bandwidth as Parallel Computing in C. The number of thread is set to the number of physical cores in your computer first which is 2. Then same test is done with 4 threads and the bandwidth speed increased.

Step2.

I have traversed the array thrice to calculate the sum, average and standard deviation. So, data size is multiplied with 3 to calculate the bandwidth.

REFERENCE

- Introduction to Computer Systems, Carnegie Mellon University