High Performance Computing (DD2356)

Assignment 1

Carried out by:

Akhil Yerrapragada (akhily@kth.se)

Gibson ()

EXERCISE-1:

Questions:

1. What is the performance in total execution time - do not consider data movement - according to our performance model on Beskow or your local computer for different sparse matrices nrows = 10^2, 10^4, 10^6 and 10^8? **Hint:** *Use **Time = nnz*2c** and calculate c from the given clock speed of the processor in use*.

   value of c for nrows =10^2 is nnz=460 and T =0.000001 => 0.000001/920 = 1.086956521e-9 sec

   value of c for nrows =10^4 is nnz=460 and T =0.000038 => 0.000038/920 = 4.1304347826e-8 sec

   value of c for nrows =10^6 is nnz=460 and T =0.004464 => 0.004464/920 = 4.8521739130e-6 sec

   value of c for nrows =10^8 is nnz=460 and T =0.442702 => 0.442702/920 = 4.8119782608e-4 sec

2. What is the measured performance in total execution time and floating-point operations per second running spvm.c for different sizes nrows = 10^2, 10^4, 10^6 and 10^8? **Note**: in spmv.c, we set up nrows by setting n = sqrt(nrows). **Hint:** *use nnz*2 and the total execution time to calculate the floating-point operations per second in spmv.*

   920/0.000001 = 9200,00,000
   920/0.000038 = 242,10,526.3157
   920/0.004464 = 2,06,093.1899
   920/0.442702 = 2,078.1473767

3. What is the main reason for the observed difference between the modeled value and the measured value?

   Algorithms are not transformed into performance. Need more insight in hardware

4. What are the <u>read bandwidth values</u> you measure running spvm.c for different sizes nrows = 10^2, 10^4, 10^6 and 10^8? **Hint:** *use* <u>nnz (sizeof(double) + sizeof(int)) + nrows (sizeof(double) + sizeof(int))</u> (slide 11 in here) *and the total execution time to calculate the bandwidth of spmv*

   460(12) + 10^2(12) = 6,720 B for 0.000001 sec

   for 1 sec 67200,00,000 B => 6720 Mb/sec

460(12) + 10^4(12) = 1,25,520 B for 0.000038 sec

for 1 sec 33031,57,894 B => 3303.157 Mb/sec

460(12) + 10^6(12) = 120,05,520 B for 0.004464 sec

for 1 sec 26894,08,602.150538 B => 2689.408 Mb/sec

460(12) + 10^8(12) = 12000,05,520 B for 0.442702 sec

for 1 sec 27106,39,482.089532 B => 2710.63 Mb/sec

5. What is the bandwidth you obtain by running the [STREAM benchmark] on your system? How does it compare to the bandwidth you measured in spvm? Compile the benchmark with:

Function	Best Rate MB/s	Avg time	Min time	Max time
Copy:	18628.4	0.008619	0.008589	0.008693
Scale:	11916.3	0.013492	0.013427	0.013547
Add:	13461.3	0.017887	0.017829	0.017963
Triad:	13559.4	0.017760	0.017700	0.017816

The difference is assumed to be in the ratio 1:3.5 comparing 5k and 18k

EXERCISE-2:

Questions:

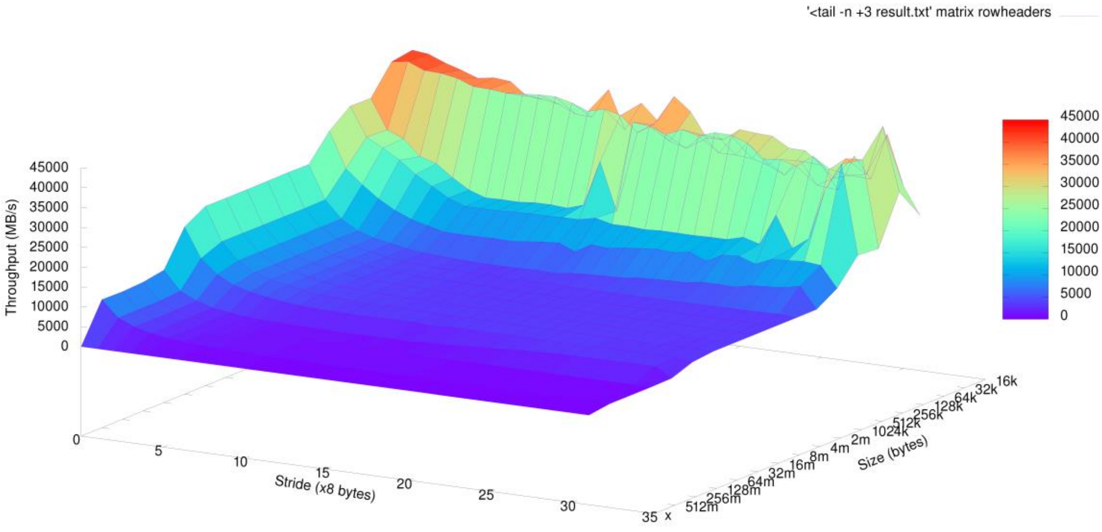
1. Report the name of the processor and the size of the L1, L2, and L3 of the processor you are benchmarking. You can check the specs of your processor online.

   L1 - 64kb per core

   L2 - 256 kb per core

L3 - 24 mb shared

2. Create the memory mountain following the steps above.
   - Save the image of the displayed "memory mountain", and place the resulting image in your pdf file.



3. What region (array size, stride) gets the most **consistently** high performance? (Ignoring spikes in the graph that are noisy results...)
   - What is the read bandwidth reported?

     L1 Cache has high performance (stride 2 to 8, size 16k to 64k)

     16k to 64k

4. What region (array size, stride) gets the most **consistently** low performance? (Ignoring spikes in the graph that are noisy results...)
   - What is the read bandwidth reported?

     Main memory (stride 9 to 32, size 1024k to 512m)

     1024k to 512m

5. When you look at the graph for stride=1, you (should) see relatively high performance compared to stride=32. This is true even for large array sizes that are much larger than the L3 cache size.
   - How is this possible, when the array cannot possibly all fit into the cache? Your explanation should include a brief overview of hardware prefetching as it applies to caches.

     Loads and stores from caches are usually faster compared to main memory. The caches are usually of small size (hardware limitation) i.e L1-64kb, L2-256kb and L3- 24mb. Lower the size, faster the access time. If the size if the matrix does not fit in cache, compiler fails to manage spatial locality to it.

Therefore it will be fit into main memory. For example, time to access from main memory for transpose will be n^2(r+Lw). where L is elements per cache line.

6. What is temporal locality? What is spatial locality?

   Temporal locality - Reusing data is called temporal locality. In the below example we can say that there is no temporal locality because both the load and store are never used again.

   Example:

   ```
   loop1{
       loop2{
           a[i] = b[j];
       }
   }
   ```

   Spatial Locality: Using nearby data is called spatial locality. Assume the array being accessed is a[1,2,3,4,5,6,7]. If 1 is accessed in the array it is highly likely that other elements are accessed from the array. Therefore, all the elements are loaded into cache thereby increasing the performance.

7. Adjusting the total array size impacts temporal locality - why?
   - Will an increased array size increase or decrease temporal locality?

     If increased it will be increases. If decreased it will be decreased. If the array reused is adjusted, the temporal locality will adapt to these changes.

8. Adjusting the read *stride* impacts spatial locality - why?
   - Will an increased read stride increase or decrease spatial locality?

     If increased it will be increases. If decreased it will be decreased. If the array reused is adjusted, the spatial locality will adapt to these changes.

EXERCISE-3:

Questions:

Question: How do we switch the compiler environment on Beskow?

module swap PrgEnv-cray PrgEnv-gnu

1. Compiler the program in GNU environment with optimization flag `-O2` and execute it.

1. **Question**: what is the average running time?

   0.00000000 s

2. **Question**: Increase N and compile the code, what is the average running time now?

   0.00000000 s

3. Get the assembly instructions with

   Question: why is the execution time like that? Answer this question using the information you find in the assembly code.

   Since the arrays inside the loops are not used later in the code, the compiler ignores that line during execution. This is the problem with smart compilers when optimizing the code. This reflects in assembly instructions.

   O2 refers to optimization for code size and execution time. Which means this option increases both compilation time and the performance of the generated code. Therefore we can see improved time for execution of code. As per assembly level code, we cannot observer the loop iterations and floating point operations on a and b.

4. **Question**: What is the average execution time without `-O2` compilation flag?

   N = 5k

   1st observation - 0.00002408 s

   2nd observation - 0.00002503 s

   3rd observation - 0.00002789 s

   4th observation - 0.00002813 s

   5th observation - 0.00002313 s

   6th observation - 0.00002384 s

   Average value = 0.00002535 s

2. Check the tick (clock granularity) on Beskow or your local computer.

   Question: What is the clock granularity on Beskow ?

   The clock granularity is 9.54e-07 s

Question: What is the clock granularity when using the RDTSC timer?

The clock granularity is 8.96e-09 s

Question: What is the minimum and average run times? Run the tests multiple times to avoid interference.

N=5k

1st observation - 0.00002694 s

2nd observation - 0.00002599 s

3rd observation - 0.00002789 s

4th observation - 0.00003099 s

5th observation - 0.00002480 s

6th observation - 0.00002503 s

EXERCISE-4:

Questions:

EVENT NAME	MSIZE = 64 Naive
Elapsed time (seconds)	______0.000808_________
Instructions per cycle	_______6.52________
L1 cache miss ratio	_______N/A______
L1 cache miss rate PTI	_______N/A________
LLC cache miss ratio	_______0.08245________
LLC cache miss rate PTI	_______82.455________

EVENT NAME	MSIZE = 1000 Optimised
Elapsed time (seconds)	______1.601681_______
Instructions per cycle	____0.82_________
L1 cache miss ratio	___ 3.1071e-5________
L1 cache miss rate PTI	_____31.0750_______
LLC cache miss ratio	_____0.52136_______
LLC cache miss rate PTI	______521.364______

EVENT NAME	MSIZE = 1000 Naïve
Elapsed time (seconds)	____7.403300_________
Instructions per cycle	____1.36_________
L1 cache miss ratio	______0.105632_______
L1 cache miss rate PTI	_____105.63278________
LLC cache miss ratio	__0.00202__________
LLC cache miss rate PTI	____2.02581_________

EVENT NAME	MSIZE = 64 Optimized
Elapsed time (seconds)	______0.000276________
Instructions per cycle	__0.21____________
L1 cache miss ratio	____N/A______
L1 cache miss rate PTI	___N/A______
LLC cache miss ratio	____0.37844__________
LLC cache miss rate PTI	_______378.443________

Question: What are the factors that impact the most the performance of the matrix multiply operation for different matrix sizes and implementations (naive vs optimized)?

The smaller matrices are usually faster than larger matrices. With size of matrix, memory operations computation time increases which in turn increases time. To improve performance of larger matrices cache blocking (strip mining) is a good idea. However. For smaller matrices this is not good since it has lot of overhead like loops etc.

O2 refers to optimization for code size and execution time. Which means this option increases both compilation time and the performance of the generated code. Therefore, we can see improved time for execution of code. As per assembly level code, we cannot observer the loop iterations and floating-point operations on a and b.

EXERCISE-5:

Questions:

EVENT NAME	**N=2049**
Elapsed time (seconds)	___4.03*10^-2____
Bandwidth/Rate (from the code and not PERF)	_______ ___2.03e+03 Mb/sec___________
Instructions per cycle	____0.14__________
L1 cache miss ratio	_____1.17803_________
L1 cache miss rate PTI	___1178.03___________
LLC cache miss ratio	____0.2980_________
LLC cache miss rate PTI	___298.078___________

EVENT NAME	**N=2048**
Elapsed time (seconds)	____4.21*10^-2____
Bandwidth/Rate (from the code and not PERF)	______ ____7.69e+02 Mb/sec__________
Instructions per cycle	___0.11___________
L1 cache miss ratio	_____1.1671_________
L1 cache miss rate PTI	____1166.17__________
LLC cache miss ratio	___0.1249___________

LLC cache miss rate PTI	___124.987__________

EVENT NAME	N=128
Elapsed time (seconds)	_____2.09*10^-5_____ ______
Bandwidth/Rate (from the code and not PERF)	___6.76e+02 Mb/sec___________
Instructions per cycle	___2.62___________
L1 cache miss ratio	_____N/A__________
L1 cache miss rate PTI	_____N/A__________
LLC cache miss ratio	___0.1239___________
LLC cache miss rate PTI	_____123.954__________

EVENT NAME	N=64
Elapsed time (seconds)	_____2.38*10^-6__________
Bandwidth/Rate (from the code and not PERF)	__ 1.49e+04 Mb/sec___________
Instructions per cycle	___1.67___________
L1 cache miss ratio	_____N/A__________
L1 cache miss rate PTI	_____N/A__________
LLC cache miss ratio	___0.3461___________
LLC cache miss rate PTI	_____346.119__________

- What are the factors that impact the most the performance of the transpose operation for different matrix sizes and implementations?

  When compiler fails to manage spatial locality, it effects the performance. For larger matrices, there is a larger memory requirement, therefore it requires space in main memory since it wont fit in caches. However, some techniques such as strip-mining, loop unrolling sounds reasonably will to enhance the performance of large matrices.

- Which code transformations can be used in the code for the matrix transpose to improve the re-usage of cache?

<u>EXERCISE-6:</u>

<u>Questions</u>:

- Find out how to request your compiler, e.g. gcc, Cray compiler … apply vectorization by checking on-line resources. For some systems and compilers, vectorization is the default with certain optimization flags.
  - Find out which optimation flag for your compiler includes vectorization.

    gcc -O2 -ftree-vectorize transpose.c -O2 -o transpose.out

    Works for Optimizations 1 to 5

- Find out how you can get a report from the compiler about its success at vectorization.
  - In particular, find out which compiler flag enables a vectorization report for your compiler.

    gcc -O2 -ftree-vectorize -fopt-info-vec=result.txt transpose.c -O2 -o transpose.out

- Read your compiler's documentation to find out what special directives or command-line options can affect vectorization

- Obtain the vectorization report for the [matrix-matrix multiply code]() for MSIZE = 1000.
  - Which lines of the code are not vectorized if any, and in case why the compiler is not vectorizing them?

- matrix_multiply.c:19:31: note: not vectorized: no grouped stores in basic block.
- matrix_multiply.c:26:3: note: not vectorized: loop contains function calls or data references that cannot be analyzed
- matrix_multiply.c:27:5: note: not vectorized: loop contains function calls or data references that cannot be analyzed
- matrix_multiply.c:29:33: note: not vectorized: not enough data-refs in basic block.
- matrix_multiply.c:27:5: note: not vectorized: no grouped stores in basic block.
- matrix_multiply.c:26:3: note: not vectorized: not enough data-refs in basic block.
- matrix_multiply.c:33:1: note: not vectorized: not enough data-refs in basic block.
- matrix_multiply.c:39:3: note: not vectorized: multiple nested loops.
- matrix_multiply.c:41:7: note: not vectorized: no vectype for stmt: vect__3.41_23 = MEM[(double *)vectp_matrix_b.39_8];
- scalar_type: vector(2) double
- matrix_multiply.c:41:7: note: not vectorized: no grouped stores in basic block.
- matrix_multiply.c:41:7: note: not vectorized: no vectype for stmt: MEM[(double *)vectp_matrix_r.45_52] = vect__5.43_48;
- scalar_type: vector(2) double
- matrix_multiply.c:41:7: note: not vectorized: not enough data-refs in basic block.
- matrix_multiply.c:41:7: note: not vectorized: no vectype for stmt: vect_matrix_r_I_I_lsm.35_19 = MEM[(double *)vectp_matrix_r.33_33];
- matrix_multiply.c:41:7: note: not vectorized: not enough data-refs in basic block.
- matrix_multiply.c:39:3: note: not vectorized: not enough data-refs in basic block.
- matrix_multiply.c:39:3: note: not vectorized: not enough data-refs in basic block.
- matrix_multiply.c:46:1: note: not vectorized: not enough data-refs in basic block.
- matrix_multiply.c:41:7: note: not vectorized: no grouped stores in basic block.
- matrix_multiply.c:41:7: note: not vectorized: no vectype for stmt: MEM[(double *)vectp_matrix_r.127_153] = vect__27.125_149;
- scalar_type: vector(2) double
- matrix_multiply.c:41:7: note: not vectorized: not enough data-refs in basic block.
- matrix_multiply.c:41:7: note: not vectorized: no vectype for stmt: vect_matrix_r_I_I_lsm.117_34 = MEM[(double *)vectp_matrix_r.115_46];
- scalar_type: vector(2) double
- matrix_multiply.c:41:7: note: not vectorized: not enough data-refs in basic block.
- matrix_multiply.c:39:3: note: not vectorized: not enough data-refs in basic block.

- matrix_multiply.c:39:3: note: not vectorized: not enough data-refs in basic block.
- matrix_multiply.c:60:5: note: not vectorized: no grouped stores in basic block.
- matrix_multiply.c:41:7: note: not vectorized: no vectype for stmt: vect__36.109_58 = MEM[(double *)vectp_matrix_b.107_60];
- scalar_type: vector(2) double
- matrix_multiply.c:41:7: note: not vectorized: no grouped stores in basic block.
- matrix_multiply.c:41:7: note: not vectorized: no vectype for stmt: MEM[(double *)vectp_matrix_r.113_137] = vect__38.111_4;
- scalar_type: vector(2) double
- matrix_multiply.c:41:7: note: not vectorized: not enough data-refs in basic block.
- matrix_multiply.c:41:7: note: not vectorized: no vectype for stmt: vect_matrix_r_I_I_lsm.103_79 = MEM[(double *)vectp_matrix_r.101_86];
- scalar_type: vector(2) double
- matrix_multiply.c:41:7: note: not vectorized: not enough data-refs in basic block.
- matrix_multiply.c:39:3: note: not vectorized: not enough data-refs in basic block.
- matrix_multiply.c:39:3: note: not vectorized: not enough data-refs in basic block.
- matrix_multiply.c:71:3: note: not vectorized: not enough data-refs in basic block.
- matrix_multiply.c:60:5: note: not vectorized: no grouped stores in basic block.
- matrix_multiply.c:53:5: note: not vectorized: not enough data-refs in basic block.
- matrix_multiply.c:52:3: note: not vectorized: not enough data-refs in basic block.
- matrix_multiply.c:79:3: note: not vectorized: not enough data-refs in basic block.
- matrix_multiply.c:79:3: note: not vectorized: not enough data-refs in basic block.
-