CS3210 – Parallel Computing (AY 2012/2013 Sem 1)

Assignment 1 – Cache Profiling and Performance Optimizations (15 marks)

Individual Submission due on 5 October 2012

This assignment, covering lectures 1 to 4, is designed to enhance your understanding of the important of cache performance and hands-on in exploiting SIMD parallelism. First, you will analyze cache organization using the computers in the Parallel and Distributed Computing Lab. Solution can be developed using your own personal notebook or PC but the final results submitted must be collected using the computers in the lab. Secondly, use the cache organization to optimize the cache performance of a matrix multiplication program given by us. The goal of your optimization is to speed up its execution time and compared it with a baseline provided by us. You will also explore SIMD program execution using Intel SIMD instructions. Marks will be awarded based on the improvement in execution time.

[*Optional Performance Challenge:* Besides the above, you are encouraged to explore other forms of optimization and bonus marks will be awarded. This is an opportunity to score more than full marks.]

1. **Cache Hierarchy Profiling (6 marks)**

   Write a C program to determine four aspects of the cache organization for both types of computers in the Parallel and Distributed Computing Lab:
   a. levels of cache memories for processor core *(2 marks)*
   b. size of each level of cache *(1 mark)*
   c. type of write operation supported such as write-back or write-through *(1 mark)*
   d. size of L1 data cache line *(2 mark)*
   Compare your answers with the processor specification when appropriate.

   [Hint: You are not allowed to read these values from the system configuration files (i.e. /sys filesystem) or by decoding CPU/BIOS codes. You can however use whatever information you wish to optimize/reduce the design space of your program .]

2. **Optimization of Matrix Multiplication Performance (6 marks)**

   mm-seq.c is a C++ program that computes the product of two square matrices with single precision *float* data types. The program executes the O(n^3) matrix multiplication algorithm, using three for loops:

```
for (int i = 0; i < n ; i++)
for (int j = 0; j < n; j++)
     for (int k = 0; k < n; k++)
          result[i][j]  += a[i][k] * b[k][j];
```

Figure 1: Matrix Multiplication Program

   You task is to optimize its execution as much as possible using one of the two computers in the lab (select any one). We suggest two optimizations:

   a. **Reduce Cache Misses:** Analyze the pattern of memory accesses and infer the number of cache accesses and cache misses performed in each iteration of the program. Based on what you have learned about the cache hierarchy, modify the program to reduce the total number of cache misses. [Advice: This optimization is the most important. Failure to implement it will make other optimizations much less effective]

b. **SIMD Execution:** Use SIMD programming to execute several arithmetic operations in parallel per cycle. The Intel processor cores in the lab computer support SIMD execution using a set of special SIMD instructions called SSE (Streaming SIMD Extensions). SSE instructions can perform four arithmetic operations in the same cycle. You can use special functions provided by GCC, called intrinsic functions to generate the SIMD instructions. In the references section, you are provided with a link to the documentation on these intrinsic functions.

Example
simd-intel.c is a sample program that computes the dot product of two vectors of floats using both normal instructions and SIMD instructions from SSE version 3 instruction set. The SSE3 instructions operate on 128-bit of data in parallel. Thus, these instructions execute 4 floating point operations in the same cycle.

Below is an excerpt of the code that shows the dot product with non-SIMD and with SIMD instructions.

```
float norm(float *a, float * b, int size) {
     int i;
     float prod = 0;
     float sa, sb, sc;
     for (i = 0; i < size; i++) {
          sa = a[i];
          sb = b[i];
          sc = sa * sb;
          prod = prod + sc;
     }
     return prod;
}
```

Figure 2: Dot Product of Vectors *a* and *b* using **Normal Instructions**

```
float norm_simd(float *a, float* b, int size) {
     int i;
     __m128 vprod = _mm_setzero_ps();
     __m128 va, vb, vc;
     float prod[4];
     for (i = 0; i < size; i += 4) {
          va = _mm_load_ps(&a[i]);
          vb = _mm_load_ps(&b[i]);
          vc = _mm_mul_ps(va, vb);
          vprod = _mm_add_ps(vprod, vc);
     }
     _mm_store_ps(prod, vprod);
```

```
        return prod[0] + prod[1] + prod[2] + prod[3];
    }
```

Figure 3: Dot Product of Vectors *a* and *b* using **SSE3 SIMD Instructions**

Read the source files carefully before proceeding to implement your own SIMD code. In particular, pay attention to the memory alignment issue – see how the memory is allocated with *memalign* instead of the usual *malloc*. This is because the SSE instructions require that the data be aligned to 16 bytes memory zones. For more details, we recommend the Intel manual linked in the references.

Marks:

- 4 marks are awarded if you reduce the execution time by at least 6 times.
- 6 marks are awarded if you reduce the execution time by at least 12 times.

The matrices size must be at least 1024x1024 and at most 4096x4096. Any comparison of execution time must be performed for the same problem size on both baseline and optimized version.

You can implement other optimizations, as long as they are supported by GCC and the processors in the lab. However, you may not use multi-threading, multi-processes or any optimized numerical libraries.

Bonus:

- 1 bonus mark if you reduce the execution time by at least 18 times
- 2 bonus marks if you reduce the execution time by at least 24 times

Deliverables and Special Considerations (3 marks)

You must hand in two C/C++ source files – the first for the cache hierarchy profiling and the second for the optimized version of the matrix multiplication. In your submission, remove program code such as printing statements that you used for debugging your programs. You must either provide a makefile or detailed compilation instructions in a README file. The source files must be properly commented and indexed, and must contain readible and well-modularized code. There is a penalty of 1 mark if these criteria are not met.

The optimized matrix multiplication must produce correct results. You must include in your program a method to verify the results, by comparing the baseline versus the optimized results. The easiest way to verify is to multiply the same two matrices using both versions and compare the outputs. The absolute difference between the matrices must be less than 0.000001. Failure to include the verification code is penalized with 1 mark.

You must also prepare a report in PDF format that documents the following:

1. The strategy for determining the four aspects of the cache hierarchy. Please document this part as thoroughly as possible.
2. The values of the levels of cache, size of each level, type of write operation and size of cache line. The values must be determined using reproducible experiments.
3. The strategy for the optimization of the matrix multiplication
4. Any special consideration or implementation detail that you consider non-trivial.

5. Details on how to reproduce your experiments, including **the problem size used for both baseline and optimized matrix multiplication.**

There is a penalty of 1 mark for not including these aspects.

Instructions for Compilation of Programs

To compile the baseline matrix multiplication program, you can use the makefile provided by us, or use the following command:

```
$ gcc -O3 mm-seq.c -o mm-seq-baseline -lrt
```

For your optimized version, we suggest the following optimization flags:

- -O3 – to direct the compiler to optimize as much as possible;
- -fstrict-aliasing – to allow the compiler to assume that pointers used in the program will point to non-overlapping memory zones;
- -msse3 – to allow the compiler to generate SIMD instructions for SSE version 3.
- -lrt – if you want to measure execution times with nanosecond resolution.

You may use other compilation flags, as you see fit.

Individual Submission and Deadline

The code and the PDF must be archived in a zip file with named after your student ID. The archive must be uploaded to IVLE in the workbin folder “Assigment1” by **5 October 2012 23:59**. There is a penalty of 1 mark per day for late submissions.

References

1. **GCC Options That Control Optimization** - http://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

2. **MMX, SSE, and SSE2 Intrinsics** - http://msdn.microsoft.com/en-us/library/y0dh78ez(v=vs.80).aspx – here you can find a through list of all SIMD instructions supported by the processors, their syntax and method of using it. The documentation is intended for Microsoft compiler under Windows, but also works with GCC under Linux.

3. **Intel x64 and x86 Architectures Optimization Reference Manual** - http://www.intel.com/Assets/en_US/PDF/manual/248966.pdf – see chapter 4.3.1.2 but other chapters might be relevant as well.

Programs (see course webpage – A01-program-code.)

1. mm-seq.c multiples two square matrices using three for loops.
2. simd-intel.c computes the dot products of two vectors using both normal instructions and Intel SIMD instructions and prints the different in execution time.