CPEG 421/621: Compiler Design: Optimization of Dense Matrix Multiplication
Handout: Project Part 2
Issue Date: 23 November, 2010
Due Date: 8 December, 2010

1 Problem

The second part continues the line of work reported in the first part where a series of steps using static analysis and scheduling techniques were studied on the IBM Cyclops-64 (C64) many-core-on-a-chip architecture where they showed how to increase effectively the performance. An open question is: Can the performance and scalability be drastically improved again, and if so, how? The answer (to be detailed in this project) is that it can. However, much to our surprise, it cannot be achieved by using static techniques alone.

Our objective is to maximize the performance of the double precision dense matrix multiplication $A \times B = C$, with each matrix of size $m \times m$ (square matrices) using algorithms of running time $O(m^3)$ using the IBM Cyclops 64 (C64) many-core Architecture [1]. Operands of MM are in off-chip memory (DRAM). The pseudo-code for computing one block of C is shown on figure 1.

```
1 :  Initialize C_{i,j} to 0 on SRAM
2 :  Compute the block C_{i,j} = sum_{k=0}^{m/n - 1} A_{i,k} . B_{k,j} .  This can be subdivided in 2 subtasks:
2a:     Copy A_{i,k} and B_{k,j} from DRAM to SRAM.
2b:     Compute a partial result of C_{i,j} and accumulate.
3 :  Copy Back the block C_{i,j} calculated
```

Figure 1: Tasks for computing one block $C_{i,j} \in C$

Throughout the design process proposed, specific features of C64 the will be used to illustrate the advantages of a set of Compiler Optimizations.

2 Documents Required

This handout is also packed with the following documents:

1. Cyclops 64 Basic Instruction Set (sent by email).
2. Different Versions of code for Matrix multiplication on Cyclops 64. (sent by email).
3. Links to references to be downloaded from our repository. (at the end of the handout).

3 Procedure

3.1 Compiler, Simulator and Chip

- You need access to the CAPSL server atlantic.capsl.udel.edu. Contact the TA if you need assistance.
- Basics of the C64 compiler have been covered on part 1. Please follow the indications proposed there if you have any questions.
- To run programs on the real chip you will have the help of people of ET International. Ask the instructor about the procedure.

3.2 Optimized MM on SRAM

A fully optimized code for MM on C64 will be provided, this version includes a dynamic scheduling (DS) and all the other incremental compiler optimizations described in part 1 of the project.

1. **Dynamic Scheduling:** Look the folder mmDSOptimized. This MM version implements Dynamic Scheduling using tiles 6x6 [2].
   (a) Describe how DS works for MM using on-chip memory. Why can you expect to increase performance even for a regular application like MM?
   (b) Make a graph of performance vs number of threads (e.g. 1, 4, 9, ... , 144, 156) for the bigger matrix that fit on SRAM (size has to be multiple of 6 for a correct result). Compare with your optimized version that uses Static Scheduling (SS) from part 1 and comment your results.
   Note: You are able to modify your data prefetching submitted for part 1 in order to increase the performance of the SS version [3] using the code provided for calculating a tile.
   (c) Make a graph that shows the performance vs matrix size (e.g. 60, 66, ...) for SS and DS using 156 Threads. Comment your results and findings.

3.3 Optimized MM on DRAM

You will compare two versions of MM using off-chip memory. The first one will use barriers for data movement and computation tasks and the second one will use a runtime that schedules everything dynamically.

1. The code included on the folder mmNaiveDRAMtiming [2] follows the approach of figure 1. A new flag was introduced in the program. It is called -DPROFILE. It enables and disables the profiling function. The profile flag creates a file called profile.dump that includes the following information in order: Start Time, End Time, ThreadId, Address in Hexadecimal for the function (e.g. Copy), and functionID (a unique identifier for each function).

You can identify the name of the function based on the address using the *object dump* of C64 toolchain. You need to type:

cyclops64-linux-elf-objdump -S mm > out.dump

Where mm is the file generated by the compiler. This instruction will create the file out.dump where you can find the correspondence between the address and the function.

Using the profile.dump file, you need to create a graph ThreadID vs Time that shows the type of task each thread is doing (e.g. Copy, ProcessBlock, etc), including the idle time. Figure 2 shows an example:

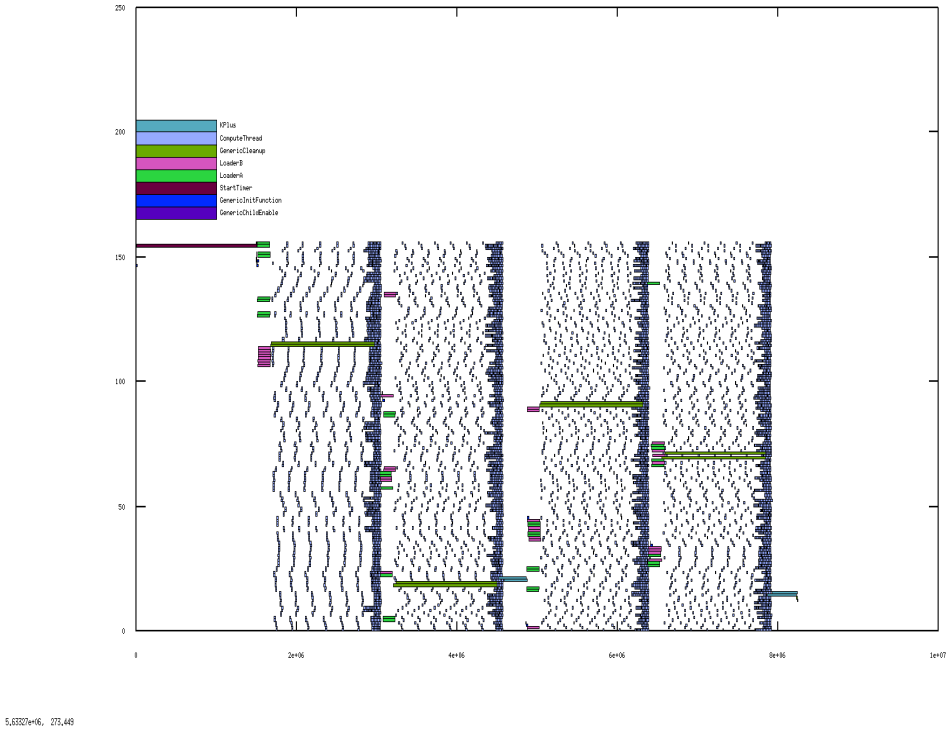

Figure 2: Example of profiling. The blank spaces mean the tread is idle

(a) Use different number of threads and multiple sizes of matrices and blocks, compare the results between them. Describe the disadvantages of Static Scheduling (SS) and under which conditions they affect more.

(b) Measure the overhead of the profiling (e.g. how the performance is affected by the profiling functions) for each case.

(c) Compare for at least one case the differences between results on simulator and real chip.

Note: The size of blocks (matrices on SRAM) has to be multiple of 6 and the size of matrices (on DRAM) has to be multiple of block size.

2. The folder mm-ds contains several binaries for mm using a runtime that schedule dynamically and efficiently the different tasks of MM. This mm also uses double buffering on SRAM. There are versions with and without profiling. The relevant information included in each column is: ThreadID, Address in Decimal for the function (e.g. Copy), Initial Time, End Time, Idle Time.

   Note: In this case the program is written on a single program multiple data (spmd) style. Do not forget to use the flag –spmd instead of –spsd (used on the programs of previous parts) on the C64 simulator.

   (a) What are the advantages of Double buffering? How triple or quad buffering can modify the performance? Explain briefly.

   (b) Create a profiling for different sizes, similar to the ones created for SS. Compare and conclude.

   (c) What is the overhead of the profiling?

   (d) Based on data of Performance vs Matrix/Block Size. Which algorithm is more scalable, SS or DS?

   (e) Compare carefully for at least one case the differences between results on simulator and real chip using DS. What could be the reasons of the differences?

3.4 What to hand in

- A report with all the experiences, answer of questions and analysis made for part of the project.
- The programs designed (or modified) with their corresponding makefile.

Team Work and Project Management

You need to establish a team methodology as in a real world design project. It is mandatory that you elect a project team leader whose function, among other things, is to call the design meetings. In your first project review meeting, you should partition the work so each team member should have a clear responsibility and a fair portion of the workload. You should also have a plan for weekly design review meetings (expected 4-5 times in total).

Ethics for Team Work

Although discussion of the project in general terms is to be expected, members of different groups should not exchange source code or project implementation details. Also, it is expected that the contributions of each member of group will be clearly detailed, so care should be taken to make sure that each person contributes a fair amount. Therefore, your

team strategy should be geared to prevent bottlenecks where group progress is entirely dependent on one person. A well-considered strategy and a clearly defined interface may mitigate such difficulties.

References

[1] M. Denneau and H. S. Warren Jr., "64-bit Cyclops: Principles of Operation," IBM Watson Research Center, Yorktown Heights, NY, Tech. Rep., April 2005.

[2] E. Garcia, R. Khan, K. Livingston, I. E. Venetis, and G. Gao, "Dynamic percolation - mapping dense matrix multiplication on a many-core architecture," *CAPSL Technical Memo 98*, June 2010. [Online]. Available: ftp://ftp.capsl.udel.edu/pub/doc/memos/memo098.pdf

[3] E. Garcia, I. E. Venetis, R. Khan, and G. Gao, "Optimized Dense Matrix Multiplication on a Many-Core Architecture," in *Proceedings of the Sixteenth International Conference on Parallel Computing (Euro-Par 2010), Part II*, ser. Lecture Notes in Computer Science, vol. 6272. Ischia, Italy: Springer, 2010, pp. 316–327. [Online]. Available: http://www.capsl.udel.edu/pub/doc/papers/ElkinGarcia-EuroPar2010.pdf