Memory Optimization

D. Jiménez, E. Morancho and À. Ramírez

April 23, 2016

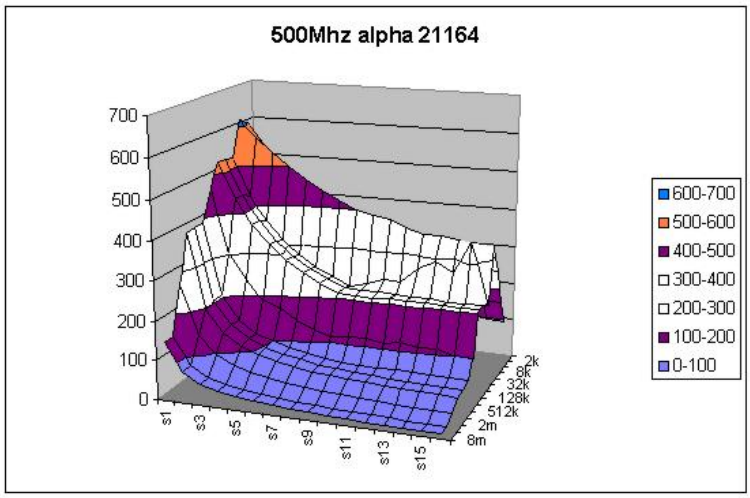
Index

1

Getting knowledge of Our machine

1. Think how you would make a program that helps you to figure out the number of caches and their approximate size. Explain it briefly.
2. Think how you would make a program that helps you to figure out the cache line size of a certain level of cache. Explain it briefly.
3. `hierarchy.c` program (from *Computer Systems. A Programmer's Prespective* book, R. Bryant and D. O'Hallaron) let graphically show the memory hierarchy behaviour of your machine. This program performs memory accesses to different *workloads* (vectors of 1Kbytes up to 16 Mbytes) with different *strides* (element by element, every each element, etc., where each element is 4 bytes). With that, we want to evaluate the capacity of a cache and the cache line sizes.
   (a) Run the program forwarding the output to a file (for instance , `mountain.csv`).
   (b) Open libreOffice and do `insert+sheet from file` to load file `mountain.csv`. WARNING: maybe your openoffice configuration of decimals use "." or ",". In this case you have to see if the mountain.csv use the same format: "." or ",". Otherwise, nothing will be seen on the figure.
   (c) In order to figure out the approximate number of caches and their size you can make a new figure using column 10.
   (d) In order to figure out the approximate cache line size you can make a new figure using rows of $1M$ of the table.

   The following figure has been obtained for an Alpha 21164 processor, representing the bandwidth for each stride/workload of the test.



4. The *distribution sort* algorithm copies (distributes) the elements of a vector to another vector depending on a certain number of bits of the keys to be distributed.

   (a) Open the `distribution.c` program and try to understand:
   - How the vector index is updated depending on the value of the most significant bits of the elements of S
   - How the vector index is updated on the following loop in the code
   - How the vector index is used to copy data from S to D

   (b) Open `distribution.sh` *shellscript* and note that this *shellscript* executes the distribution sort with two different inputs, and varying the number of bits of the digit. Once you run the *shellscript* you will have two files:
   i. `100000000-0-results.txt`: run of the program with parameters `n_elements`=100000000, `bits` from 1 to 22, `data_distribution`=0 (data is ramdomly distributed)
   ii. `100000000-1-results.txt`: run of the program with parameters `n_elements`=100000000, `bits` from 1 to 22, `data_distribution`=1 (data is distributed with an order)

   (c) Compile the program and run the *shellscript*.

   (d) Why is there performance difference between those two executions (data distributions)?
   i. Perform the profiling with `oprofile` looking at the TLB events (`tlb_accesses`) and L2 misses events (`LLC_REFS`), using the following parameters for the program: `n_elements`=100000000, `bits`=18, `distribution`=0 and `n_elements`=100000000, `bits`=18, `distribution`=1.
   ii. Justify your answer based on your timings and the profiling you have just done.

2

Data aligment

5. The unaligned memory accesses may affect the program performance. In this exercise, you have to run the programs of `munge_vectors` directory, and make a figure in order to show the execution time depending on the data alignment.

   (a) Analyze the source codes that you will find on the directory, compile them and run them.

   (b) Create a figure for each program execution, and another for all of them together.

   (c) Try to justify the results obtained.

3

Memory Bandwidth

6. Using the same code of `munge_vectors` of previous exercise:

   (a) Create a figure for all the executions, but in this case, normalized by the number of accesses done. In order to create this figure you should modify comment out a printf of the `munge` function to print out *cyc/numberofaccesses*. The number of accesses is computed by (memoryEnd - memoryInit)/sizeof(data).

   (b) What are the memory accesses that take maximum profit of the memory bandwidth: 8-bit, 16-bit, 32-bit or 64-bit accesses based on the normalized results? Which is the memory bandwidth of our machine to load/store to/from the general purpose registers based on the normalized results and the assembler code?

7. Optimize your best `swap.c` program version (exercise of problem collection) so that the memory bandwidth exploitation is improved.

4

Spatial Locality

8. As a part of a graphic engine, there is a routine that has to fill a buffer with green colour before printing next frame. The screen is `ROWSxCOLUMNS` pixels.

   (a) Analyze the `rgb.c` code. When runnig it, redirect the output to a file.

   (b) Obtain the first cache level misses (references to the second cache level) and the second cache level misses of the original code. Remember to use `-g` compilation flag in order to be able to visualize profiling at source code level.

   (c) Propose a first optimization of this code so that we can better exploit the spatial locality of the data. Compile and execute in order to obtain the new timing.

   (d) Propose a second optimization that also reduces the number of memory accesess. Compile and execute in order to obtain the new timing.

   (e) Compute the speedup of the second optimitzation code compared to the first optimization code. Why don't you get any speedup?

5

Temporal Locality

9. It is very common to find a matrix multiply computation on graphic engines. The naive basic code of that operation does not usually exploit very well the temporal and spatial locality. You have the matrix multiply code in the `mult1.c` file.

   (a) Compile with -O3 (`make mult1.3`).

   (b) Codify a *blocking* version of the matrix per matrix function in order to exploit the temporal locality of memory accesses to `C` matrix.

   (c) Run and do timing of the *blocking* version using different *blocking* sizes for $N = 1024$ and $N = 2048$.

   (d) Analyze the non-blocking version and the *blocking* version with the best blocking size you have found, and compare them. In order to do that analysis use profiling with `oprofile` using execution cycles, L2 cache misses (`LLC_REFS`) and TLB misses (`tlb_accesses`) events.

10. The PCA entreprise wants to sort their workers by NID very fast.

    Each worker is represented with a *struct*:

```
#define MAX_NOM 40
#define MAX_DIAS_MES 31
#define MAX_CATEGORIA 40

typedef struct
{
  long long int NID;              /* Identification number*/
  char Nom[MAX_NOM];
  char Cognoms[MAX_NOM];
  int  horesMes[MAX_DIAS_MES];
  char Categoria[MAX_CATEGORIA];
  unsigned int ptrClauForaneaDepartament;
  unsigned int ptrClauCategoria;
} Templeat
```

   (a) Program `empleats.c` sorts a parametrizable number of workers. In order to do that, the worker information is generated in a random way. In order to sort the workers quicksort (`qsort`) algorithm is used.

   (b) Make a figure showing the cycles spent per sorted worker (CPW, Cycles per worker), varying the number of workers to sort. In order to make that process automatic it would be usefull to create a *shellscript*. The figure can be created using LibreOffice.

   (c) Optimize the application under the point of view of system calls. Compute the speedup obtained compared to the original code.

(d) Now, the enterprise also wants to maintain the address information of the worker, the first and last name of the couple (if he/she has). If there is something that is not applicable, the field will be kept in blank. Activate the EXTES define of the *struct* definition and analyze the program performance with a figure where you show again CPW. Compare the performance of the previous code and the actual code.

(e) Optimize the program so that the size of the structure (activated or not EXTES) does not influence too much in the performance of the sorting process. This optimization should improve the spatial and temporal locality exploitation of the sorting.

(f) Compare that optimized code with the original code for different number of empleats and make a figure comparing their CPW.