CE538– Parallel Computer Architecture

Homework 1

Due on Thursday, February 20, 2014

ASSIGNMENTS SHOULD BE TURNED IN INDIVIDUALLY

1. Papers [1][2][3] at the end of this document focus on some of the issues that we covered on our first class. They discuss how technology shapes future architectures, moving from single- to multi-core architectures [1], and what may be some of the technology limitations for future scaling of multi-core architectures [2][3]. You should read the papers and write an up to two page summary (in English) trying to think on some of the following problems. How has technology (e.g. clock frequency, power dissipation, etc.) affected processor architectures in the last ten years and how is it probable to affect it in the future? Have the predictions of [1] been confirmed according to what we know now? Can you predict how future processors will look like in 2025?
2. **Branch prediction.** Suppose we have a deeply pipelined processor, for which we implement a branch target buffer (BTB) for the conditional branches only. Assume that the misprediction penalty is always 5 cycles and the buffer miss penalty is always 2 cycles. Assume 90% hit rate and 90% accuracy (in case of a BTB hit), and 15% branch frequency. How much faster is the processor with the branch target buffer versus a processor that has a fixed 2-cycle branch penalty? Assume a base CPI without branch stalls of 1. Note: study example of page 123 in H&P, v. 4 before you attempt this problem.
3. **Cache performance optimization.** Exercises 5.6, 5.7 of H&P, version 4. Assume that a double precision array element is 8 bytes. Assume that each array occupies 8 KB of the cache. Note: Study examples at page 307 and 308 before you attempt this exercise.
4. **Προσομοίωση ιεραρχίας μνήμης**. Στα πλαίσια αυτής της εργαστηριακής άσκησης, θα προσομοιώσετε την ιεραρχία μνήμης ενός desktop υπολογιστή χρησιμοποιώντας τον προσομοιωτή *cachegrind*. Το λογισμικό *cachegrind* είναι μέρος του λογισμικού *valgrind*, που είναι ένα σύνολο εργαλείων λογισμικού για memory analysis and debugging, και program behavior profiling σε περιβάλλον Linux. Η προσομοίωση της λειτουργίας ενός επεξεργαστή γενικότερα και της ιεραρχίας μνήμης ειδικότερα είναι ένα βασικό κομμάτι της ανάλυσης που κάνει ένας σχεδιαστής υπολογιστικών συστημάτων για να δημιουργήσει συστήματα με αποδεκτή απόδοση στο μικρότερο δυνατό κόστος.

   Στην περίπτωση μας θέλουμε να εξετάσουμε την συμπεριφορά ενός προγράμματος όσον αφορά την αλληλεπίδραση του με την ιεραρχία μνήμης για να καταλάβουμε καλύτερα την λειτουργικότητα των μνημών cache. Επίσης θέλουμε να εξετάσουμε και να εξηγήσουμε την επίδραση που έχουν στην απόδοση της ιεραρχίας μνήμης βελτιστοποιήσεις στον κώδικα και ιδίως στα loops.

   Αρχικός κώδικας

Ο παρακάτω κώδικας αρχικοποιεί έναν πίνακα δύο διαστάσεων. Το πρώτο ερώτημα σας ζητάει να κάνετε compile τον κώδικα αυτόν και να τον τρέξετε για να χρονομετρήσετε την εκτέλεση του. Η πιο απλή μέθοδος για

```
#include <stdlib.h>
#define N 32768

int main () {
 int *a = (int *)malloc(N*N*sizeof(int));
 for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    *(a+i*N+j) = i+j;
  return (0);
}
```

```
#include <stdlib.h>
#define N 32768

int main () {
 int *a = (int *)malloc(N*N*sizeof(int));
  for (int j = 0; j < N; j++)
   for (int i = 0; i < N; i++)
     *(a+i*N+j) = i+j;
  return (0);
}
```

αυτό είναι με την χρήση της εντολής *time* του Linux.

```
% time ./simple_row
```

Κάντε ακριβώς το ίδιο για τον κώδικα στα δεξιά του πρώτου ο οποίος επιτελεί ακριβώς την ίδια εργασία, απλά όμως έχει αλλάξει την σειρά του εσωτερικού (for-j) με το εξωτερικό loop (for-i):

```
% time ./simple_col
```

Ποιος είναι ο χρόνος εκτέλεσης των δύο προγραμμάτων στον υπολογιστή σας; Συμπληρώστε τον παρακάτω πίνακα με τις απαντήσεις σας.

Για να καταλάβουμε καλύτερα που οφείλεται αυτή η διαφορά στον χρόνο εκτέλεσης θα χρησιμοποιήσουμε το εργαλείο *cachegrind* για να αναλύσουμε την συμπεριφορά της ιεραρχίας μνήμης του υπολογιστή. Προσοχή ότι εννοούμε τον ίδιο τον υπολογιστή που εκτελείτε το πρόγραμμά σας, και όχι κάποιον simulator. Στην προκειμένη μας ενδιαφέρει η συμπεριφορά μόνο της data cache και όχι της instruction cache[1]. Πριν συνεχίσετε την άσκησή αυτή καλό είναι να διαβάσετε για τον *cachegrind* [1] για να καταλάβετε καλύτερα τις δυνατότητές του.

```
% valgrind --tool=cachegrind ./simple_row
```

```
% cg_annotate --auto=yes cachegrind.out.<XXX> ./simple_row.c
```

Η δεύτερη εντολή (cg_annotate)δεν είναι υποχρεωτική για να καταλάβετε τι συμβαίνει, απλά με αυτήν μπορείτε να δείτε τις αστοχίες (misses) της cache που οφείλονται σε κάθε γραμμή του source code και να δείτε τα χαρακτηριστικά της cache. Το αρχείο cachegrind.out.<XXX> δημιουργείται αυτόματα από το εργαλείο *cachegrind*.

Αφού εκτελέσετε με αυτόν τον τρόπο και τα δύο προγράμματα, βρείτε τον ρυθμό των αστοχιών της L1 Data Cache και της L3 Data Cache. Το *cachegrind* αναφέρει μόνο τις αστοχίες για το πρώτο και το τελευταίο επίπεδο της ιεραρχίας μνήμης, οπότε δεν έχετε τρόπο να μετρήσετε τις αστοχίες της L2 Cache.

Τι παρατηρείτε και πως μπορείτε να το εξηγήσετε; Πως εξηγείτε την μεγάλη διαφορά μεταξύ των χρόνων εκτέλεσης των δύο προγραμμάτων βασιζόμενοι στην πληροφορία που σας δίνει το *cachegrind*; Γιατί δημιούργησε αυτή την μείωση του χρόνου εκτέλεσης η αλλαγή στην σειρά εκτέλεσης των δύο loops;

Απάντηση :
Χαρακτηριστικά Cache:
L1-Instruction Cache:
L1-Data Cache:
L3 Cache (unified):
Πρόγραμμα simple_row:
Χρόνος εκτέλεσης =
D1 miss rate =
LL3 miss rate =

Πρόγραμμα simple_col:
Χρόνος εκτέλεσης =
D1 miss rate =
LL3 miss rate =

[1] Για τόσο μικρά προγράμματα η L1 I-Cache έχει πρακτικά 100% επιτυχία και για αυτό δεν μας απασχολεί ιδιαίτερα η συμπεριφορά της. Γενικά, η data cache είναι πιο προβληματική όσον αφορά την ρυθμό αστοχιών κατά την εκτέλεση ενός προγράμματος και για αυτό οι περισσότερες προσπάθειες βελτιστοποίησης του κώδικα επικεντρώνονται στην data cache.

Βιβλιογραφία

[1] Cachegrind online documentation. *http://valgrind.org/docs/manual/cg-manual.html*

Bibliography

1) Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures, *International Symposium on Computer Architecture (ISCA 2000)*
2) Thousand core chips: a technology perspective, *Design Automation Conference*, (*DAC 2007)*
3) Dark Silicon and the End of Multicore Scaling, *International Symposium on Computer Architecture (ISCA 2011)*