Lab 4 – Cache Performance

Nathan Paternoster

Description

We ran a series of experiments on performing matrix multiplication in different arrangements to analyze cache performance. We graphed the resulting miss rates from the different combinations of i, j, and k single matrix multiplication. Based on our graphs, we concluded that KIJ has the best miss rate (followed by IKJ) and KJI has the worst miss rate.

The larger the block size for the block matrix multiplication, the better. However we found that any block size above 16 did not produce efficient code.

The algorithm we used for our block matrix multiplication is shown below. We re-arranged the outer three for loops to get the different i, j, k combinations.

#include <stdlib.h>

#include <iostream>

#include <stdio.h>

#include <fstream>

#include <string>

using namespace std;

int main() {

ifstream fin("/home/grayjin/project4/Matrix32");

if (fin.fail())

return -1;

int i1, j1, k1;

fin >> i1 >> j1 >> k1;

int size = i1;

int a[i1][k1], b[k1][j1], c[i1][j1];

/\* a \*/

for (int x=0; x<i1; x++) {

for (int y=0; y<k1; y++)

fin >> a[y][x];

}

/\* b\*/

for (int x=0; x<k1; x++) {

for (int y=0; y<j1; y++)

fin >> b[y][x];

}

int B = 16;

int n = i1;

int sum = 0;

int i, j, k;

for (i = 0; i<n; i+=B) {

for (j = 0; j<n; j+=B) {

for (k = 0; k<n; k+=B) {

/\* B x B mini matrix multiplications \*/

for (int i0 = i; i0 < i+B; i0++) {

for (int j0 = j; j0 < j+B; j0++) {

for (int k0 = k; k0 < k+B; k0++) {

c[i0][j0] += a[i0][k0]\*b[k0][j0]; }

}

}

}

fin.close();

return 0;

}

Post-Lab Report

Our performance results showed mostly consistent data for single-matrix multiplication. We verified that KIJ and IKJ are the most efficient algorithms and JKI and KJI are the least efficient. Greater associativity resulted in lower miss rates. Interestingly, our L1 data cache seemed to perform the worst of the three caches. Greater matrix sizes showed mostly slower performance with a few exceptions.

Using the block multiplication algorithm provided for us we seemed to obtain very similar results between i, j, k algorithms. However, although the miss rates were nearly uniform, the total number of misses varied between algorithms. In general we noticed that block multiplication seemed to favor larger matrix sizes (probably because larger matrix sizes allow the algorithm to make full use of sub-matrix blocks).

Results

All of the graphs show miss rate (%) in the y-axis and matrix size in the x-axis. Different i, j, k algorithms are shown in different colors.

**L1 Instruction Cache**

**Single Matrix Multiplication**

**1-Way**

**2-Way**

**4-Way**

**8-Way**

**L1 Data**

**Single Matrix Multiplication**

**1-Way**

**2-Way**

**4-Way**

**8-Way**

**L2 Cache**

**Single Matrix Multiplication**

**1-Way**

**2-Way**

**4-Way**

**8-Way**