Denis Doci

Katarzyna Krawczyk

LABW14

1. Briefly explain your understanding of cache-friendly programs.

Cache-friendly programs work through memory in a linear fashion. This is in would be done instead of jumping around the memory. When this is done, you take advantage of temporal locality. The requested memory is in a near areay, so if you retrieve information from memory the cache will perform better.

1. Name one programming technique that improves cache-friendliness.

Row Major operations:

After doing a row multiplication, we can access information through the rows instead of through the columns. This would increase hit rate and decrease the miss rate since it is the same memory block as was already used. This would be “cache-friendls”

1. Create an instrumented source code (obvious.cc) which implements matrix multiplication (row-column multiplication). In order to quickly complete the task, create a copy of blocking.cc and replace the central part with the new implementation.

include "inst\_none.h"

#include <iostream>

using namespace std;

const int matrixSize = 4;

const int blockingFactor = 2;

int a[matrixSize][matrixSize];

int b[matrixSize][matrixSize];

int c[matrixSize][matrixSize];

int main() {

int i, j, k, jj, kk, r;

// Initialise the matrices arbitrarily

for (i=0; i < matrixSize; i++)

for (j=0; j < matrixSize; j++) {

b[j][i] = i + j; c[j][i] = i - j; a[j][i] = 0;

}

// Work out a = b \* c, using a blocking algorithm

jj = 0;

kk = 0;

while (jj < matrixSize) {

while (kk < matrixSize) {

for (i=0; i < matrixSize; i++)

for (k=kk; k < kk + blockingFactor; k++) {

INST\_R(b[k][i]);

cout << "INST\_R(b[" << i <<"]" << "[" << k <<"])"<<"\n";

INST\_R(c[j][k]);

cout << "INST\_R(c[" << k <<"]" << "[" << j <<"])"<<"\n";

r += b[k][i] \* c[j][k];

}

INST\_R(a[j][i]);

cout << "INST\_R(a[" << i <<"]" << "[" << j <<"])"<<"\n";

a[j][i] = a[j][i] + r;

INST\_W(a[j][i]);

cout << "INST\_W(a[" << i <<"]" << "[" << j <<"])"<<"\n";

}

kk += blockingFactor;

}

kk = 0;

jj += blockingFactor;

}

}

// Display the product

for (i=0; i < matrixSize; i++) {

for (j=0; j < matrixSize; j++) cerr << a[i][j] << ' ';

cerr << endl;

}

}

1. Suppose matrixSize and blockingFactor in blocking.cc are respectively 4 and 2. List all the instrumented codes in blocking.cc line by line in execution order (indicate the exact number of array index). It should look like:

INST\_R(b[0][0])

INST\_R(c[0][0])

INST\_R(b[0][1])

INST\_R(c[1][0])

INST\_R(a[0][0])

INST\_W(a[0][0])

INST\_R(b[0][0])

INST\_R(c[0][1])

INST\_R(b[0][1])

INST\_R(c[1][1])

INST\_R(a[0][1])

INST\_W(a[0][1])

INST\_R(b[1][0])

INST\_R(c[0][0])

INST\_R(b[1][1])

INST\_R(c[1][0])

INST\_R(a[1][0])

INST\_W(a[1][0])

INST\_R(b[1][0])

INST\_R(c[0][1])

INST\_R(b[1][1])

INST\_R(c[1][1])

INST\_R(a[1][1])

INST\_W(a[1][1])

INST\_R(b[2][0])

INST\_R(c[0][0])

INST\_R(b[2][1])

INST\_R(c[1][0])

INST\_R(a[2][0])

INST\_W(a[2][0])

INST\_R(b[2][0])

INST\_R(c[0][1])

INST\_R(b[2][1])

INST\_R(c[1][1])

INST\_R(a[2][1])

INST\_W(a[2][1])

INST\_R(b[3][0])

INST\_R(c[0][0])

INST\_R(b[3][1])

INST\_R(c[1][0])

INST\_R(a[3][0])

INST\_W(a[3][0])

INST\_R(b[3][0])

INST\_R(c[0][1])

INST\_R(b[3][1])

INST\_R(c[1][1])

INST\_R(a[3][1])

INST\_W(a[3][1])

INST\_R(b[0][2])

INST\_R(c[2][0])

INST\_R(b[0][3])

INST\_R(c[3][0])

INST\_R(a[0][0])

INST\_W(a[0][0])

INST\_R(b[0][2])

INST\_R(c[2][1])

INST\_R(b[0][3])

INST\_R(c[3][1])

INST\_R(a[0][1])

INST\_W(a[0][1])

INST\_R(b[1][2])

INST\_R(c[2][0])

INST\_R(b[1][3])

INST\_R(c[3][0])

INST\_R(a[1][0])

INST\_W(a[1][0])

INST\_R(b[1][2])

INST\_R(c[2][1])

INST\_R(b[1][3])

INST\_R(c[3][1])

INST\_R(a[1][1])

INST\_W(a[1][1])

INST\_R(b[2][2])

INST\_R(c[2][0])

INST\_R(b[2][3])

INST\_R(c[3][0])

INST\_R(a[2][0])

INST\_W(a[2][0])

INST\_R(b[2][2])

INST\_R(c[2][1])

INST\_R(b[2][3])

INST\_R(c[3][1])

INST\_R(a[2][1])

INST\_W(a[2][1])

INST\_R(b[3][2])

INST\_R(c[2][0])

INST\_R(b[3][3])

INST\_R(c[3][0])

INST\_R(a[3][0])

INST\_W(a[3][0])

INST\_R(b[3][2])

INST\_R(c[2][1])

INST\_R(b[3][3])

INST\_R(c[3][1])

INST\_R(a[3][1])

INST\_W(a[3][1])

INST\_R(b[0][0])

INST\_R(c[0][2])

INST\_R(b[0][1])

INST\_R(c[1][2])

INST\_R(a[0][2])

INST\_W(a[0][2])

INST\_R(b[0][0])

INST\_R(c[0][3])

INST\_R(b[0][1])

INST\_R(c[1][3])

INST\_R(a[0][3])

INST\_W(a[0][3])

INST\_R(b[1][0])

INST\_R(c[0][2])

INST\_R(b[1][1])

INST\_R(c[1][2])

INST\_R(a[1][2])

INST\_W(a[1][2])

INST\_R(b[1][0])

INST\_R(c[0][3])

INST\_R(b[1][1])

INST\_R(c[1][3])

INST\_R(a[1][3])

INST\_W(a[1][3])

INST\_R(b[2][0])

INST\_R(c[0][2])

INST\_R(b[2][1])

INST\_R(c[1][2])

INST\_R(a[2][2])

INST\_W(a[2][2])

INST\_R(b[2][0])

INST\_R(c[0][3])

INST\_R(b[2][1])

INST\_R(c[1][3])

INST\_R(a[2][3])

INST\_W(a[2][3])

INST\_R(b[3][0])

INST\_R(c[0][2])

INST\_R(b[3][1])

INST\_R(c[1][2])

INST\_R(a[3][2])

INST\_W(a[3][2])

INST\_R(b[3][0])

INST\_R(c[0][3])

INST\_R(b[3][1])

INST\_R(c[1][3])

INST\_R(a[3][3])

INST\_W(a[3][3])

INST\_R(b[0][2])

INST\_R(c[2][2])

INST\_R(b[0][3])

INST\_R(c[3][2])

INST\_R(a[0][2])

INST\_W(a[0][2])

INST\_R(b[0][2])

INST\_R(c[2][3])

INST\_R(b[0][3])

INST\_R(c[3][3])

INST\_R(a[0][3])

INST\_W(a[0][3])

INST\_R(b[1][2])

INST\_R(c[2][2])

INST\_R(b[1][3])

INST\_R(c[3][2])

INST\_R(a[1][2])

INST\_W(a[1][2])

INST\_R(b[1][2])

INST\_R(c[2][3])

INST\_R(b[1][3])

INST\_R(c[3][3])

INST\_R(a[1][3])

INST\_W(a[1][3])

INST\_R(b[2][2])

INST\_R(c[2][2])

INST\_R(b[2][3])

INST\_R(c[3][2])

INST\_R(a[2][2])

INST\_W(a[2][2])

INST\_R(b[2][2])

INST\_R(c[2][3])

INST\_R(b[2][3])

INST\_R(c[3][3])

INST\_R(a[2][3])

INST\_W(a[2][3])

INST\_R(b[3][2])

INST\_R(c[2][2])

INST\_R(b[3][3])

INST\_R(c[3][2])

INST\_R(a[3][2])

INST\_W(a[3][2])

INST\_R(b[3][2])

INST\_R(c[2][3])

INST\_R(b[3][3])

INST\_R(c[3][3])

INST\_R(a[3][3])

INST\_W(a[3][3])