COUNTER DESCRIPTION AND FORMULAE

|  |  |  |
| --- | --- | --- |
| 1. | totalBytesRead | Total number of bytes read by CPU |
| 2. | readMissCount | Number of bytes CPU missed when cache is read by CPU |
| 3. | readHitCount | Number of bytes got hit when cache is read by CPU |
| 4. | readReplaceCount | Number of bytes got replaced in cache from memory |
| 5. | readWBCount | Number of dirty bits in cache |
| 6. | readCacheCount | Number of bytes CPU read from cache |
| 7. | readMemFnCount | Number of types read Memory function is called |
| 8. | readCacheFnCount | Number of types read cache function is called |

|  |  |  |
| --- | --- | --- |
| 1. | totalBytesWritten | Total number of bytes written by CPU |
| 2. | writeMissCount | Number of bytes CPU missed when cache is read by CPU |
| 3. | writeHitCount | Number of bytes got hit when cache is read by CPU |
| 4. | writeReplaceCount | Number of bytes got replaced in cache from memory |
| 5. | writeWBCount | Number of dirty bits in cache |
| 6. | writeCacheCount | Number of bytes CPU read from cache |
| 7. | writeMemFnCount | Number of types write Memory function is called |
| 8. | writeCacheFnCount | Number of types write cache function is called |
| 9. | WriteCPU2Cache | Number of bytes CPU wrote to memory (writeThrough) |



#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <inttypes.h>

#include <stdbool.h>

#include <assert.h>

#include <string.h>

*//global constants*

#define cacheSize 262144

#define MAX\_LINES 131072

#define MAX\_WAYS 16

*//counters*

uint64\_t totalBytesRead;

uint64\_t readHitCount;

uint64\_t readMissCount;

uint64\_t readReplaceCount;

uint64\_t readWBCount;

uint64\_t readCacheCount;

uint64\_t readMemFnCount;

uint64\_t readCacheFnCount;

uint64\_t flushCounter;

uint64\_t totalBytesWritten;

uint64\_t writeHitCount;

uint64\_t writeMissCount;

uint64\_t writeReplaceCount;

uint64\_t writeWBCount;

uint64\_t writeCPU2MemCount;

uint64\_t writeCacheCount;

uint64\_t writeMemFnCount;

uint64\_t writeCacheFnCount;

*//global variables*

uint32\_t TAG[MAX\_LINES][MAX\_WAYS];

**bool** VALID[MAX\_LINES][MAX\_WAYS];

**bool** DIRTY[MAX\_LINES][MAX\_WAYS];

uint32\_t LRU[MAX\_LINES][MAX\_WAYS];

uint8\_t ws = 0;

*//void code3(void);*

**void** zeroCache()

{

totalBytesRead = 0;

readHitCount = 0;

readMissCount = 0;

readReplaceCount = 0;

readWBCount = 0;

readCacheCount = 0;

readMemFnCount =0;

readCacheFnCount =0;

totalBytesWritten = 0;

writeHitCount = 0;

writeMissCount = 0;

writeReplaceCount = 0;

writeWBCount = 0;

writeCPU2MemCount = 0;

writeCacheCount = 0;

writeMemFnCount = 0;

writeCacheFnCount = 0;

flushCounter = 0;

**for**(uint64\_t i = 0; i<MAX\_LINES; i++)

{

**for**(uint8\_t j =0; j< MAX\_WAYS; j++)

{

VALID[i][j]=0;

DIRTY[i][j]=0;

TAG[i][j] = 0;

LRU[i][j] = j;

}

}

}

uint32\_t getLine(uint32\_t add, uint8\_t N, uint8\_t BL)

{

uint32\_t line;

uint32\_t B = BL\*2;

uint64\_t L = cacheSize/(B\*N);

uint8\_t b = log2(B);

uint8\_t l = log2(L);

*// uint8\_t t = 24-l-b;*

line = add>>b;

uint32\_t bits = (pow(2,l)-1);

line = line&bits;

**return** line;

}

uint32\_t getTag(uint32\_t add, uint8\_t N, uint8\_t BL)

{

uint32\_t tag;

uint32\_t B = BL\*2;

uint64\_t L = cacheSize/(B\*N);

uint8\_t b = log2(B);

uint8\_t l = log2(L);

uint8\_t t = 24-l-b;

uint32\_t bits = (pow(2,t)-1);

tag = (add>>(l+b));

tag = tag&bits;

**return** tag;

}

**int** findWay(uint32\_t line, uint32\_t tag, uint8\_t N)

{

int8\_t way = -1;

**for**(uint8\_t i = 0; i<N; i++)

{

**if**(TAG[line][i] == tag)

way = i;

}

**return** way;

}

**bool** alwaysValid(uint32\_t line, uint32\_t N)

{

**bool** check=1;

**for**(uint8\_t i=0; i<N; i++)

{

**if**(VALID[line][i]==1)

check &=1;

**else**

check &=0;

}

**return** check;

}

int8\_t findOldestLRU(uint32\_t line, uint8\_t N)

{

uint8\_t lru = 0;

**for**(uint8\_t i = 0; i<N; i++)

{

**if**(LRU[line][i]==(N-1))

lru = i;

}

**return** lru;

}

**bool** isDirty(uint32\_t line, uint8\_t way)

{

**if**((DIRTY[line][way])&&(VALID[line][way]==1))

**return** 1;

**else**

**return** 0;

}

**void** writeBackRead(uint32\_t line, uint8\_t way)

{

readWBCount++;

}

**void** writeBackWrite(uint32\_t line, uint8\_t way)

{

writeWBCount++;

}

**void** invalid(uint32\_t line, uint8\_t way)

{

VALID[line][way] = 0;

DIRTY[line][way] = 0;

}

**void** setTag(uint32\_t line, uint8\_t way, uint32\_t tag)

{

TAG[line][way] = tag;

}

**void** valid(uint32\_t line, uint8\_t way)

{

VALID[line][way] = 1;

}

**void** updateLRU(uint32\_t line, uint8\_t N, uint8\_t way)

{

**for**(uint8\_t i = 0; i<N; i++)

{

**if**(LRU[line][i] < LRU[line][way])

LRU[line][i]++;

}

LRU[line][way] = 0;

}

**void** readCacheToCPU()

{

readCacheCount++;

}

**void** readCache(uint32\_t add, int8\_t N, uint8\_t BL, uint8\_t ws)

{

readCacheFnCount++;

int32\_t line = getLine(add,N,BL);

uint32\_t tag = getTag(add,N,BL);

int8\_t wayCheck = findWay(line,tag,N);

**bool** hit = (wayCheck!=-1);

**if**(!hit)

{

readMissCount++;

uint8\_t way = findOldestLRU(line,N);

**if**(alwaysValid(line,N))

{

readReplaceCount++;

**if**(isDirty(line,way))

{

writeBackRead(line,way);

invalid(line,way);

}

}

setTag(line,way,tag);

valid(line,way);

updateLRU(line,N,way);

}

**else**

{

readHitCount++;

updateLRU(line,N,wayCheck);

}

readCacheToCPU(); *//readCacheCount++*

}

**void** writeCache(uint32\_t add, **int** N, uint8\_t BL, uint8\_t ws)

{

writeCacheFnCount++;

int32\_t line = getLine(add,N,BL);

uint32\_t tag = getTag(add,N,BL);

int8\_t wayCheck = findWay(line,tag,N);

**bool** hit = (wayCheck!=-1);

**if**(!hit && ws!=2) *//not in cache and wb, wta*

{

writeMissCount++;

uint8\_t way = findOldestLRU(line,N);

**if**(alwaysValid(line,N))

{

writeReplaceCount++;

**if**((isDirty(line,way)&&(ws==0)))

{

writeBackWrite(line,way);

invalid(line,way);

DIRTY[line][way]=0;

}

}

setTag(line,way,tag);

valid(line,way);

**if**(ws==0)

DIRTY[line][wayCheck]=1;

updateLRU(line,N,way);

writeCacheCount++;

}

**if**(hit && ws!=2)

{

writeHitCount++;

updateLRU(line,N,wayCheck);

writeCacheCount++;

}

**if** (ws == 0)

DIRTY[line][wayCheck] = 1;

**if**(ws==2)

{

**if**(!hit)

writeMissCount++;

**else**

{

updateLRU(line,N,wayCheck);

writeCacheCount++;

writeHitCount++;

}

}

}

**void** readMemory(**void** \*pmem, uint32\_t size, int8\_t N, uint8\_t BL, uint8\_t ws)

{

readMemFnCount++;

uint32\_t line;

int32\_t lastLine = -1;

uint32\_t add = (uint32\_t)pmem;

**for**(**int** i = 0; i<size; i++)

{

line = getLine(add,N,BL);

**if**(line!=lastLine)

{

readCache(add,N,BL,ws);

lastLine = line;

}

add++;

totalBytesRead++;

}

}

**void** writeMemory(**void** \*pmem, uint32\_t size, uint8\_t N, uint8\_t BL, uint8\_t ws)

{

writeMemFnCount++;

int32\_t line,lastLine = -1;

uint32\_t add = (uint32\_t)pmem;

**int** i = 0;

**for**(i = 0; i<size; i++)

{

line = getLine(add,N,BL);

**if**(line!=lastLine)

{

writeCache(add,N,BL,ws);

lastLine = line;

}

add++;

totalBytesWritten++;

**if**(ws == 1 || ws == 2)

{

**if**(i%2 == 0)

writeCPU2MemCount++; *//writes exactly half the no. of times the mail loop executes because of wr\_bl =1*

}

}

}

**void** flush()

{

uint8\_t N = 1;

uint8\_t BL = 1;

uint32\_t B = BL\*2;

uint64\_t L = cacheSize/(B\*N);

**for**(uint64\_t i = 0; i<L; i++)

{

**for**(uint8\_t j =0; j<N; j++)

{

DIRTY[i][j]=0;

flushCounter++;

}

}

}

**void** writeToFile(FILE \*fp)

{

**if**(fp == **NULL**)

fprintf(fp,"File error\n");

**else**

{

fprintf(fp,"%lld\t, %lld\t, %lld\t, %lld\t, %lld\t, %lld\t, %lld\t \n", totalBytesWritten,writeHitCount,writeMissCount,writeReplaceCount,writeWBCount,writeCPU2MemCount,writeCacheCount);

}

fclose(fp);

}

**void** code()

{

uint8\_t x[524288];

uint8\_t N = 1,BL = 1;

**for**(uint32\_t i=0;i<524288;i++)

{

x[i]=0;

writeMemory(&x[i],**sizeof**(uint8\_t),N,BL,1);

}

printf("1. totalBytesWritten = %"PRIu64"\n",totalBytesWritten);

printf("2. writeHitCount = %"PRIu64"\n",writeHitCount);

printf("3. writeMissCount = %"PRIu64"\n",writeMissCount);

printf("4. writeReplaceCount = %"PRIu64"\n", writeReplaceCount);

printf("5. writeWBCount = %"PRIu64"\n", writeWBCount);

printf("6. writeCPU2MemCount = %"PRIu64"\n",writeCPU2MemCount);

printf("7. writeCacheCount = %"PRIu64"\n",writeCacheCount);

}

**void** code1()

{

uint32\_t data[65536];

uint64\_t sum =0;

uint8\_t N = 1;

uint8\_t BL = 1;

**for**(uint32\_t i=0;i<65536;i++)

{

sum += data[i];

readMemory(&data[i],**sizeof**(uint32\_t),N,BL,1);

}

printf("1. totalBytesRead = %"PRIu64"\n",totalBytesRead);

printf("2. readHitCount = %"PRIu64"\n",readHitCount);

printf("3. readMissCount = %"PRIu64"\n",readMissCount);

printf("4. replaceCount = %"PRIu64"\n", readReplaceCount);

printf("5. readWBCount = %"PRIu64"\n", readWBCount);

printf("6. readCacheCount = %"PRIu64"\n",readCacheCount);

}

**static** **void** memset\_16aligned(**void** \*space, **char** byte, size\_t nbytes)

{

assert((nbytes & 0x0F) == 0);

assert(((uintptr\_t)space & 0x0F) == 0);

memset(space, byte, nbytes);

}

**static** **void** align(size\_t align)

{

uintptr\_t mask = ~(uintptr\_t)(align - 1);

**void** \*mem = malloc(1024+align-1);

**void** \*ptr = (**void** \*)(((uintptr\_t)mem+align-1) & mask);

assert((align & (align - 1)) == 0);

*// printf("0x%08" PRIXPTR ", 0x%08" PRIXPTR "\n", (uintptr\_t)mem, (uintptr\_t)ptr);*

memset\_16aligned(ptr, 0, 1024);

free(mem);

}

**void** test3(uint8\_t N,uint8\_t BL,uint8\_t ws)

{

uint32\_t x[131072];

uint32\_t i;

uint32\_t S = **sizeof**(i);

writeMemory(&i,S,N,BL,ws);

**for**(i = 0;i<131072;i++)

{

x[i]++;

readMemory(&i,S,N,BL,ws);

readMemory(&x[i],S,N,BL,ws);

writeMemory(&x[i],S,N,BL,ws);

readMemory(&i,S,N,BL,ws);

writeMemory(&i,S,N,BL,ws);

}

printf("1. totalBytesWritten = %"PRIu64"\n",totalBytesWritten);

printf("2. writeHitCount = %"PRIu64"\n",writeHitCount);

printf("3. writeMissCount = %"PRIu64"\n",writeMissCount);

printf("4. writeReplaceCount = %"PRIu64"\n", writeReplaceCount);

printf("5. writeWBCount = %"PRIu64"\n", writeWBCount);

printf("6. writeCPU2MemCount = %"PRIu64"\n",writeCPU2MemCount);

printf("7. writeCacheCount = %"PRIu64"\n",writeCacheCount);

printf("8. writeMemFnCount = %"PRIu64"\n",writeMemFnCount);

printf("9. writeCacheFnCount = %"PRIu64"\n",writeCacheFnCount);

printf("\n");

printf("1. totalBytesRead = %"PRIu64"\n",totalBytesRead);

printf("2. readHitCount = %"PRIu64"\n",readHitCount);

printf("3. readMissCount = %"PRIu64"\n",readMissCount);

printf("4. replaceCount = %"PRIu64"\n", readReplaceCount);

printf("5. readWBCount = %"PRIu64"\n", readWBCount);

printf("6. readCacheCount = %"PRIu64"\n",readCacheCount);

printf("7. readMemFnCount = %"PRIu64"\n",readMemFnCount);

printf("8. readCacheFnCount = %"PRIu64"\n",readCacheFnCount);

}

*//cholesky functions*

**void** choldc(**double** \*\*a, **int** n, **double** p[],uint8\_t N, uint8\_t BL, uint8\_t ws)

{

**int** i,j,k,x,y;

**double** sum;

**double** sum1;

**double** l[n][n];

**double** sum2 = 0;

writeMemory(&sum2,**sizeof**(sum2),N,BL,ws);

writeMemory(&x,**sizeof**(x),N,BL,ws);

readMemory(&n,**sizeof**(n),N,BL,ws);

**for**(x =0; x<n;x++)

{

writeMemory(&y,**sizeof**(y),N,BL,ws);

readMemory(&n,**sizeof**(n),N,BL,ws);

**for**(y = 0; y<n; y++)

{

readMemory(&x,**sizeof**(x),N,BL,ws);

readMemory(&y,**sizeof**(y),N,BL,ws);

writeMemory(&l[x][y],**sizeof**(l[x][y]),N,BL,ws);

l[x][y] = 0;

readMemory(&y,**sizeof**(y),N,BL,ws);

writeMemory(&y,**sizeof**(y),N,BL,ws);

readMemory(&n,**sizeof**(n),N,BL,ws);

}

readMemory(&x,**sizeof**(x),N,BL,ws);

writeMemory(&x,**sizeof**(x),N,BL,ws);

readMemory(&n,**sizeof**(n),N,BL,ws);

}

writeMemory(&i,**sizeof**(i),N,BL,ws);

readMemory(&n,**sizeof**(n),N,BL,ws);

**for** (i=0;i<n;i++)

{

readMemory(&i,**sizeof**(i),N,BL,ws);

readMemory(&a[i][i],**sizeof**(a[i][i]),N,BL,ws);

writeMemory(&sum1,**sizeof**(sum1),N,BL,ws);

sum1 = a[i][i];

writeMemory(&k,**sizeof**(k),N,BL,ws);

readMemory(&i,**sizeof**(i),N,BL,ws);

**for** (k=0;k<=i-1;k++)

{

readMemory(&i,**sizeof**(i),N,BL,ws);

readMemory(&k,**sizeof**(k),N,BL,ws);

readMemory(&a[i][k],**sizeof**(a[i][k]),N,BL,ws);

readMemory(&sum2,**sizeof**(sum2),N,BL,ws);

writeMemory(&sum2,**sizeof**(sum2),N,BL,ws);

sum2 += a[i][k]\*a[i][k];

readMemory(&k,**sizeof**(k),N,BL,ws);

writeMemory(&k,**sizeof**(k),N,BL,ws);

readMemory(&i,**sizeof**(i),N,BL,ws);

}

readMemory(&sum1,**sizeof**(sum1),N,BL,ws);

readMemory(&sum2,**sizeof**(sum2),N,BL,ws);

writeMemory(&sum,**sizeof**(sum),N,BL,ws);

sum = sum1-sum2;

writeMemory(&sum2,**sizeof**(sum2),N,BL,ws);

sum2 = 0;

readMemory(&sum,**sizeof**(sum),N,BL,ws);

**if**(sum<=0)

printf("choldc failed \n");

readMemory(&sum,**sizeof**(sum),N,BL,ws);

readMemory(&i,**sizeof**(i),N,BL,ws);

writeMemory(&p[i],**sizeof**(p[i]),N,BL,ws);

p[i] = sqrt(sum);

writeMemory(&j,**sizeof**(j),N,BL,ws);

readMemory(&n,**sizeof**(n),N,BL,ws);

**for** (j=0;j<n;j++)

{

readMemory(&j,**sizeof**(j),N,BL,ws);

readMemory(&i,**sizeof**(i),N,BL,ws);

readMemory(&a[i][j],**sizeof**(a[i][j]),N,BL,ws);

writeMemory(&sum1,**sizeof**(sum1),N,BL,ws);

sum1=a[i][j];

writeMemory(&k,**sizeof**(k),N,BL,ws);

readMemory(&i,**sizeof**(i),N,BL,ws);

**for** (k=0;k<=i-1;k++)

{

readMemory(&i,**sizeof**(i),N,BL,ws);

readMemory(&k,**sizeof**(k),N,BL,ws);

readMemory(&a[i][k],**sizeof**(a[i][k]),N,BL,ws);

readMemory(&j,**sizeof**(j),N,BL,ws);

readMemory(&k,**sizeof**(k),N,BL,ws);

readMemory(&a[j][k],**sizeof**(a[j][k]),N,BL,ws);

readMemory(&sum2,**sizeof**(sum2),N,BL,ws);

writeMemory(&sum2,**sizeof**(sum2),N,BL,ws);

sum2 += a[i][k]\*a[j][k];

readMemory(&k,**sizeof**(k),N,BL,ws);

writeMemory(&k,**sizeof**(k),N,BL,ws);

readMemory(&i,**sizeof**(i),N,BL,ws);

}

readMemory(&sum1,**sizeof**(sum1),N,BL,ws);

readMemory(&sum2,**sizeof**(sum2),N,BL,ws);

writeMemory(&sum,**sizeof**(sum),N,BL,ws);

sum = sum1-sum2;

writeMemory(&sum2,**sizeof**(sum2),N,BL,ws);

sum2 = 0;

readMemory(&i,**sizeof**(i),N,BL,ws);

readMemory(&p[i],**sizeof**(p[i]),N,BL,ws);

readMemory(&sum,**sizeof**(sum),N,BL,ws);

readMemory(&j,**sizeof**(j),N,BL,ws);

readMemory(&i,**sizeof**(i),N,BL,ws);

writeMemory(&a[j][i],**sizeof**(a[j][i]),N,BL,ws);

a[j][i]=sum/p[i];

readMemory(&i,**sizeof**(i),N,BL,ws);

readMemory(&j,**sizeof**(j),N,BL,ws);

**if**(i<j)

{

readMemory(&j,**sizeof**(j),N,BL,ws);

readMemory(&i,**sizeof**(i),N,BL,ws);

readMemory(&a[j][i],**sizeof**(a[j][i]),N,BL,ws);

readMemory(&j,**sizeof**(j),N,BL,ws);

readMemory(&i,**sizeof**(i),N,BL,ws);

writeMemory(&l[j][i],**sizeof**(l[j][i]),N,BL,ws);

l[j][i] = a[j][i];

}

**else**

{

readMemory(&j,**sizeof**(j),N,BL,ws);

readMemory(&i,**sizeof**(i),N,BL,ws);

writeMemory(&l[j][i],**sizeof**(l[j][i]),N,BL,ws);

l[j][i] = 0;

}

readMemory(&j,**sizeof**(j),N,BL,ws);

writeMemory(&j,**sizeof**(j),N,BL,ws);

readMemory(&n,**sizeof**(n),N,BL,ws);

}

readMemory(&i,**sizeof**(i),N,BL,ws);

writeMemory(&i,**sizeof**(i),N,BL,ws);

readMemory(&n,**sizeof**(n),N,BL,ws);

}*//for loop of i ends here*

}

**void** cholsl(**double** \*\*a, **int** n, **double** p[], **double** b[], **double** x[],uint8\_t N, uint8\_t BL, uint8\_t ws)

{

**int** i,k; **double** sum;

writeMemory(&i,**sizeof**(i),N,BL,ws);

readMemory(&n,**sizeof**(n),N,BL,ws);

**for** (i=0;i<n;i++)

{

readMemory(&i,**sizeof**(i),N,BL,ws);

readMemory(&b[i],**sizeof**(b[i]),N,BL,ws);

writeMemory(&sum,**sizeof**(sum),N,BL,ws);

sum=b[i];

readMemory(&i,**sizeof**(i),N,BL,ws);

writeMemory(&k,**sizeof**(k),N,BL,ws);

**for** (k=i-1;k>=0;k--)

{

readMemory(&i,**sizeof**(i),N,BL,ws);

readMemory(&k,**sizeof**(k),N,BL,ws);

readMemory(&a[i][k],**sizeof**(a[i][k]),N,BL,ws);

readMemory(&k,**sizeof**(k),N,BL,ws);

readMemory(&x[k],**sizeof**(x[k]),N,BL,ws);

readMemory(&sum,**sizeof**(sum),N,BL,ws);

writeMemory(&sum,**sizeof**(sum),N,BL,ws);

sum = sum - a[i][k]\*x[k];

readMemory(&k,**sizeof**(k),N,BL,ws);

writeMemory(&k,**sizeof**(k),N,BL,ws);

}

readMemory(&i,**sizeof**(i),N,BL,ws);

readMemory(&p[i],**sizeof**(p[i]),N,BL,ws);

readMemory(&sum,**sizeof**(sum),N,BL,ws);

readMemory(&i,**sizeof**(i),N,BL,ws);

writeMemory(&x[i],**sizeof**(x[i]),N,BL,ws);

x[i]=sum/p[i];

readMemory(&i,**sizeof**(i),N,BL,ws);

writeMemory(&i,**sizeof**(i),N,BL,ws);

readMemory(&n,**sizeof**(n),N,BL,ws);

} *// for loop of i ends here*

readMemory(&n,**sizeof**(n),N,BL,ws);

writeMemory(&i,**sizeof**(i),N,BL,ws);

**for** (i=n-1;i>=0;i--)

{

readMemory(&i,**sizeof**(i),N,BL,ws);

readMemory(&x[i],**sizeof**(x[i]),N,BL,ws);

writeMemory(&sum,**sizeof**(sum),N,BL,ws);

sum=x[i];

readMemory(&i,**sizeof**(i),N,BL,ws);

writeMemory(&k,**sizeof**(k),N,BL,ws);

readMemory(&n,**sizeof**(n),N,BL,ws);

**for** (k=i+1;k<n;k++)

{

readMemory(&i,**sizeof**(i),N,BL,ws);

readMemory(&k,**sizeof**(k),N,BL,ws);

readMemory(&a[k][i],**sizeof**(a[k][i]),N,BL,ws);

readMemory(&k,**sizeof**(k),N,BL,ws);

readMemory(&x[k],**sizeof**(x[k]),N,BL,ws);

readMemory(&sum,**sizeof**(sum),N,BL,ws);

writeMemory(&sum,**sizeof**(sum),N,BL,ws);

sum = sum - a[k][i]\*x[k];

readMemory(&k,**sizeof**(k),N,BL,ws);

writeMemory(&k,**sizeof**(k),N,BL,ws);

readMemory(&n,**sizeof**(n),N,BL,ws);

}

readMemory(&i,**sizeof**(i),N,BL,ws);

readMemory(&p[i],**sizeof**(p[i]),N,BL,ws);

readMemory(&sum,**sizeof**(sum),N,BL,ws);

readMemory(&i,**sizeof**(i),N,BL,ws);

writeMemory(&x[i],**sizeof**(x[i]),N,BL,ws);

x[i]=sum/p[i];

readMemory(&i,**sizeof**(i),N,BL,ws);

writeMemory(&i,**sizeof**(i),N,BL,ws);

readMemory(&n,**sizeof**(n),N,BL,ws);

}*//for loop of i ends here*

}

*// main*

**int** main()

{

*//cholesky; matrix genration for computation by accessing a file genrated by matlab*

**int** i,j;

**double** \*\*l;

uint16\_t n=256;

l = (**double**\*\*)malloc(n\***sizeof**(**double**\*));

**for**(**int** row = 0;row<n;row++)

l[row]=(**double** \*)malloc(n\***sizeof**(**double**));

**double** p[n],x[n];

**double** b[256];*//generate 256 values by generating it in matlab and accesing it through a file*

**double**\* mat\_b=malloc(n\***sizeof**(**double**\*));

FILE \*fb;

fb=fopen("exp3.xls", "r+");

**if**(fb == **NULL**)

printf("File Error");

**else**

{

**for**(j = 0; j < n; j++)

{

**if** (!fscanf(fb, "%lf", &mat\_b[j]))

**break**;

b[j] = mat\_b[j];

}

}

fclose(fb);

**double**\*\* mat=malloc(n\***sizeof**(**double**\*));

**for**(i=0;i<n;++i)

mat[i]=malloc(n\***sizeof**(**double**));

**double**\*\* mat\_t=malloc(n\***sizeof**(**double**\*));

**for**(i=0;i<n;++i)

mat\_t[i]=malloc(n\***sizeof**(**double**));

FILE \*file;

file=fopen("exp2.xls", "r+");

**if**(file == **NULL**)

printf("File Error");

**else**

{

**for**(i = 0; i < n; i++)

{

**for**(j = 0; j < n; j++)

{

**if** (!fscanf(file, "%lf", &mat[i][j]))

**break**;

l[i][j] = mat[i][j];

}

}

}

fclose(file);

printf("\n");

*//creating a text file for storing values of the result*

FILE \*fp;

fp = fopen("‎⁨test.txt","w+");

align(16);

**int** loop3\_count = 0;

**for**(uint8\_t N = 1; N<=16; N = N\*2)

{

**for**(uint8\_t BL = 1; BL<=8; BL = BL\*2)

{

**for**(uint8\_t ws = 0; ws<3; ws++)

{

printf("N = %d, BL = %d, WS = %d\n", N,BL,ws);

**for**(i = 0; i < n; i++)

{

**for**(j = 0; j < n; j++)

mat\_t[i][j] = l[i][j];

}

zeroCache();

choldc(mat\_t,n,p,N,BL,ws);

cholsl(mat\_t,n,p,b,x,N,BL,ws);

test3(N,BL,ws);

fprintf(fp,"%lld\t %lld\t %lld\t %lld\t %lld\t %lld\t %lld\t %lld\t %lld\t \t %lld\t %lld\t %lld\t %lld\t %lld\t % lld\t %lld\t %lld\t %d\t %d\t %d\t \n", totalBytesWritten,writeHitCount,writeMissCount,writeReplaceCount,writeWBCount,writeCPU2MemCount,writeCacheCount, writeMemFnCount,writeCacheFnCount, totalBytesRead, readHitCount, readMissCount, readReplaceCount, readWBCount, readCacheCount, readMemFnCount, readCacheFnCount, N, BL, ws);

fflush(fp);

flush();

loop3\_count++;

}

}

}

printf("Number of Loops occurred = %d\n",loop3\_count);

}