CS528
Caching And APP Classification

A Sahu
Dept of CSE, IIT Guwahati

Outline

- Data Access Optimization
- Roofline Model
- Caching optimization
- App classification based DA: N/N, N^2/N^2, N^3/N^2

[Ref: Hager Book, PDF uploaded to Website]

Hashing Vs Caching

USABILITY

- Simple Hashing: Direct Map Cache
  - Example: Array
  - int A[10], each can store one element
  - Data stored in Addr%10 location
- Array of List
  - Int LA[10], each can store a list of element
  - Data stored in List of (Addr%10)th location
  - List size is limited in Set Associative Cache
- List of Element
  - Full Associative Cache
  - All data stored in one list

Direct/Random Access to Element

MIXED

Serial/Associative Access to Element

TIME

Program Cache Behavior: Hit/Miss

Cache Model

- Direct mapped 8 word per line

Program

```
int A[128];
for(i=0;i<128;i++){
    A[i]=i;
}
```

- Assume &A=000000, **Behavior of only Data**
- Scalar variable {i} mapped to register
- Data have to moved from cache/memory

Cache perf. : Data Size <= Cache Size

```
int A[128];
for(i=0;i<128;i++){
    A[i]=i;
}
```

Scalar mapped to register
Vector mapped to memory

1:7= 1miss:7hit

1:7	**0**	**A[0]**	**A[1]**	**A[2]**					**A[7]**
2:14	**1**	**A[8]**	**A[9]**						**A[15]**
3:21	**2**	**A[16]**	**A[17]**						**A[23]**
	14								
16:112	**15**								**A[127]**

Strided access: Reduce locality

```
for(i=0;i<N;i++){
    for (j=0;j<N;j++){
        a[i][j]=i*j
      }
}      //*(a+i*N+j), j++
```

Row major access: Stride 1, improve locality, cache hit

```
for(i=0;i<N;i++){
    for (j=0;j<N;j++){
        a[j][i]=i*j
      }
}   //*(a+j*N+i), j++
```

Column major access: Stride N, No locality, cache miss dominates

Matrix mult.c

```
int A[8][8], B[8][8], C[8][8];
for(i=0;i<8;i++){
        for(j=0;j<8;j++){
         S=0;
          for(k=0;k<8;k++)
                  S=S+B[i][k]*C[k][j];
          A[i][j]=S;
        }
}
```

Data Size > Cache Size

- (64+64+64) > 128 words
- When we get into cache it can take benefit

```
for(k=0;k<8;k++)
                    S=S+B[i][k]*C[k][j];
```

- Inner loop execute for **64 times**
  - We have to get B[j] once will have 1miss/7 hit
  - C[k] have to bring every time 8miss
  - Total = **7h+9m**
- 2^{nd} loop A have one miss in 8 access (1miss/7hit)
  - Total for A= 8m+56h
- Total program : 64*(**7h+9m**)+8m+56h=504h+584m
- ***Miss Probability = 584/(504+584)=0.5367***

Improving Locality

Matrix Multiplication example

$$[C] = [A] \times [B]$$

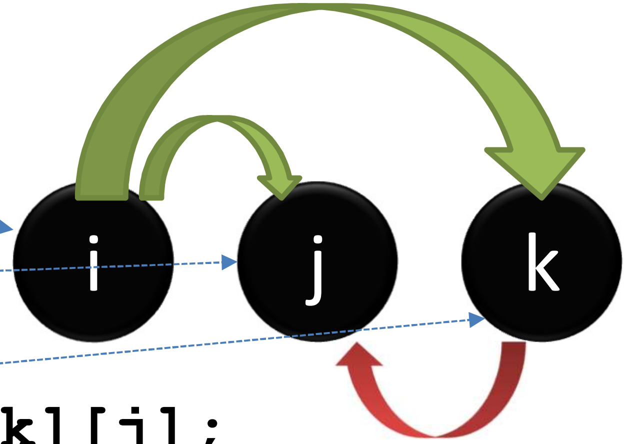
$$L \times M \qquad\qquad L \times N \qquad\qquad N \times M$$

Cache Organization for the example

- Cache line (or block) = 8 matrix elements.
- Matrices are stored row wise.
- **Cache can't accommodate a full row/column.**
  - **L, M and N are so large w.r.t. the cache size**
  - After an iteration along any of the three indices, when an element is accessed again, it results in a miss.
- Ignore misses due to conflict between matrices.
  - As if there was a **separate cache for each matrix**.

Matrix Multiplication : Code I

```
for (i = 0; i < L; i++)
  for (j = 0; j < M; j++)
    for (k = 0; k < N; k++)
      C[i][j] += A[i][k] * B[k][j];
```

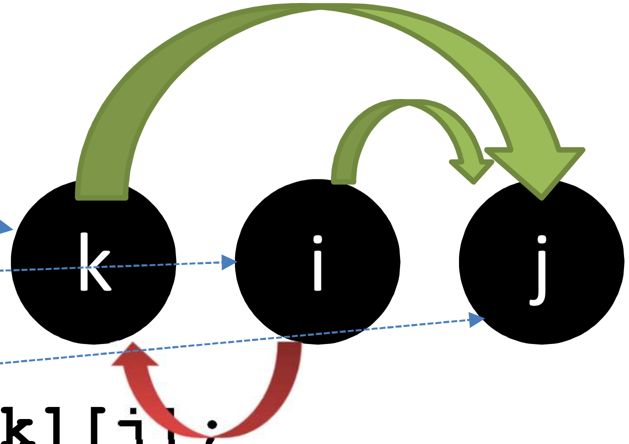


	C	A	B
accesses	LM	LMN	LMN
misses	LM/8	LMN/8	LMN

```
Total misses = LM(9N+1)/8
L=M=N=100; miss=100*100*901/8=1,126,250
```

Matrix Multiplication : Code II

```
for (k = 0; k < N; k++)
  for (i = 0; i < L; i++)
    for (j = 0; j < M; j++)
      C[i][j] += A[i][k] * B[k][j];
```

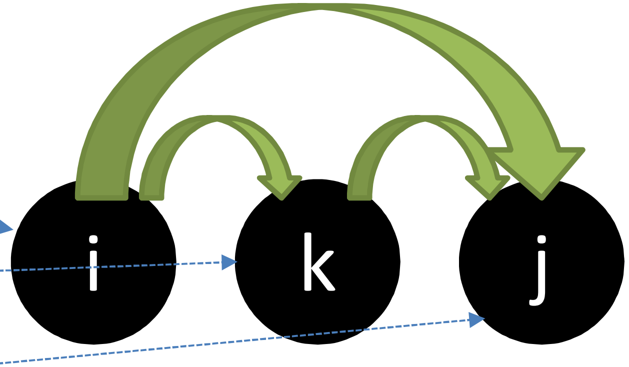


	C	A	B
accesses	LMN	LN	LMN
misses	LMN/8	LN	LMN/8

```
Total misses = LN(2M+8)/8
L=M=N=100; miss=100*100*208/8=260,000
```

Matrix Multiplication : Code III

```
for (i = 0; i < L; i++)
  for (k = 0; k < N; k++)
    for (j = 0; j < M; j++)
      C[i][j] += A[i][k] * B[k][j];
```



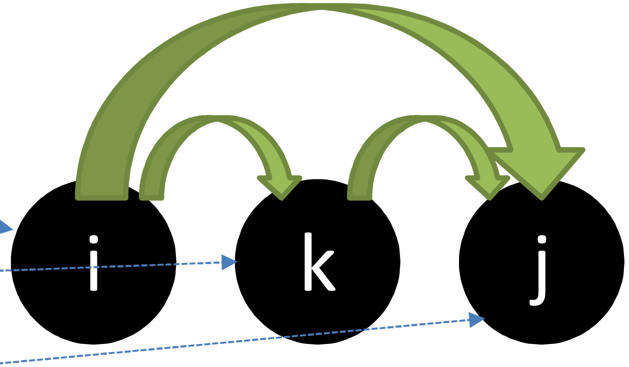
	C	A	B
accesses	LMN	LN	LMN
misses	LMN/8	LN/8	LMN/8

Total misses = LN(2M+1)/8

L=M=N=100; miss=100*100*201/8=251,250

Matrix Multiplication : Code III

```
for (i = 0; i < L; i++)
   for (k = 0; k < N; k++)
      for (j = 0; j < M; j++)
         C[i][j] += A[i][k] * B[k][j];
```



All most all modern processor uses

- Cache block pre-fetch
- When ith block is getting used i+1 block prefetched
- **Perfect overlap : only three cache miss**
  - **Each for A, B, C**

Algorithm Classification and Access Optimization

- O(N)/O(N) : If the # of arithmetic Ops and data transfer (LD/ST) are proportional to Loop Length N
  - Optimization potential is limited
  - Example Scalar Product, vector add, sparse MVM
- Memory bound for large N
- Compiler generated code achieve good perf.
  - Using software pipelining and loop nests

Loop fusion for O(N)/O(N)

```
for(i=0;i<N;i++)
    A[i]=B[i]+C[i];   //Bc=3W/1F
for(i=0;i<N;i++)
    Z[i]=B[i]+E[i];   //Bc=3W/1F
```

```
for(i=0;i<N;i++){
    A[i]=B[i]+C[i];
    Z[i]=B[i]+E[i];
}
```

B_C=5W/2F
No need get to
B[i]again

$O(N^2)/O(N^2)$: OPS/DataTransfer

- Typical two loop nests with loop strip count N
  - $O(N^2)$ operation for $O(N^2)$ loads and stores
- Example: dense MVM, Mat add, MatTrans
- MVM : -> Covert both access to row access

```
for(i=0;i<N;i++) {
     tmp=C[i]
     for(j=0;j<N;j++)  tmp=A[i][j]*B[j]
     C[i]=tmp
}
```

  - Row I of A and vector B
  - Original Bc=2W/2F but → 2W*m/2F
  - m is miss rate of cache for Row access

$O(N^3)/O(N^2)$: OPS/DataTransfer

- Typical three loop nests
  - $O(N^3)$ operation for $O(N^2)$ loads and stores
- Example: dense Matrix Mulltiplication
- Implementation of cache Bound
  - Already studied : loop interchange
  - **Blocking : Strassen multiplication, will be discussed later**

Multiprocessor Programming using Theading

Threading Language and Support

- Initially threading used for
  - Multiprocessing on single core : earlier days
  - Feeling/simulation of doing multiple work simultaneously even if on one processor
  - Used TDM, time slicing, Interleaving
- Now a day threading mostly used for
  - To take benefit of multicore
  - Performance and energy efficiency
  - We can use both TDM and SDM

Threading Language and Support

- Pthread: POSIX thread
  - Popular, Initial and Basic one
- Improved Constructs for threading
  - c++ thread : available in c++11, c++14
  - **Java thread : very good memory model**
    - **Atomic function, Mutex**
- Thread Pooling and higher level management
  - OpenMP (loop based)
  - Cilk (dynamic DAG based)

Pthread, C++ Thread, Cilk and OpenMP

```
pthread_t tid1, tid2;
pthread_create(&tid1, NULL, Fun1, NULL);
pthread_create(&tid2, NULL, Fun2, NULL);
pthread_join(tid1, NULL);
pthread_join(tid2, NULL);
```

Pthread

```
thread t1(Fun1);
thread t1(Fun2, 0, 1, 2); // 0, 1,2 param to Fun2
t1.join();
t2.join();
```

C++
thread

```
#pragma omp parallel for
for(i=0;i<N;i++)
     A[i]=B[i]*C[i];
//Auto convert serial code to threaded code
// $gcc -fopenmp test.c; export OMP_NUM_THREADS=10; ./a.out
```

```
cilk fib (int n){//Cilk dynamic parallism, DAG recursive code
      if (n<2) return n;
      int x=spawn fib(n-1); //spawn new thread
      tnt y=spawn fib(n-2); //spawn new thread
      sync;
      return x+y;
}
```