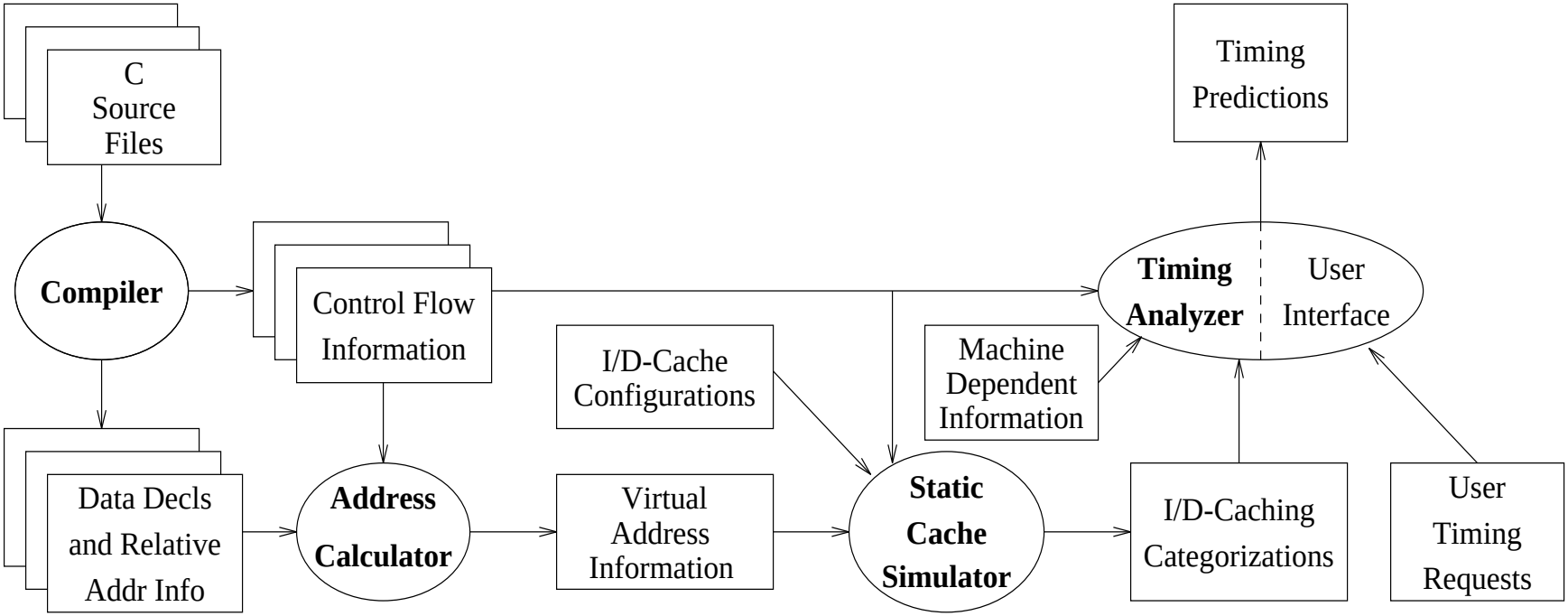
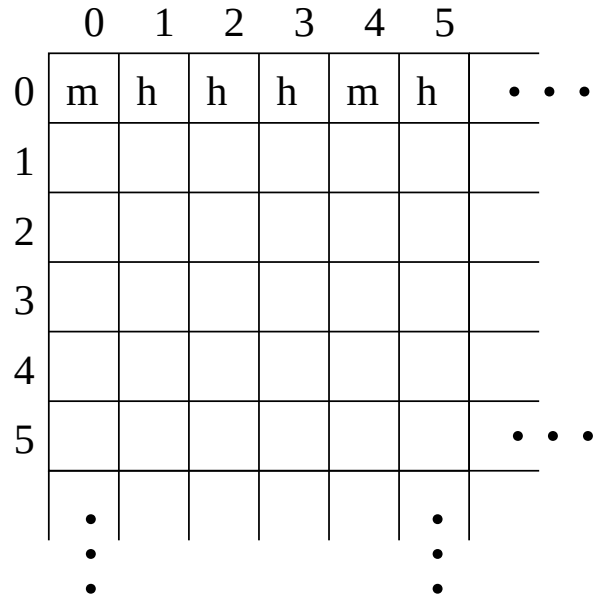
Framework for Timing Predictions



Spatial Locality: items whose addresses are near one another tend to be referenced close together in time

```
/* row order sum */
int a[100][100];
main()
{
   int i, j, sum;

   sum = 0;
   for (i = 0; i < 100; i++)
      for (j = 0; j < 100; j++)
         sum += a[i][j];
}
```

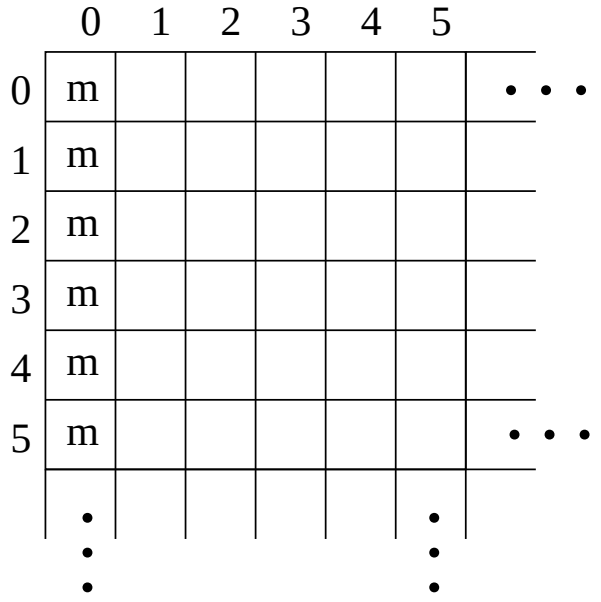


Categorization: `c 25 2500` (from `[m h h h m h h h ... m h h h]`)

Cache Size: 256 bytes, 16 lines, 16 bytes/line

```
/* column order sum */
int a[100][100];
main()
{
    int i, j, sum;

    sum = 0;
    for (j = 0; j < 100; j++)
        for (i = 0; i < 100; i++)
            sum += a[i][j];
}
```



Categorization: `m` (from `[m m m m m m m m ... m m m m]`)

Cache Size: 256 bytes, 16 lines, 16 bytes/line

Worst Case Loop Analysys (WCLA)

$$\text{WCET} = \text{path time} \times \text{iterations}$$

If data reference in path:

- If hit, use: *base_time* (assume 10)
- If miss, use: *base_time* + *miss_penalty* (assume 10 + 9 = 19)

Look at categorization on each iteration and add in time. Takes 50 steps:

`mhhhmhhhmhhhmhhhmhhhmhhhmhhhmhhhmhhhmhhhmhhhmhhhmh`

$19 + 10 + 10 + 10 + 19 + 10 + 10 + 10 + \ldots + 19 + 10$

Or, can deal with all misses first. Takes 2 steps:

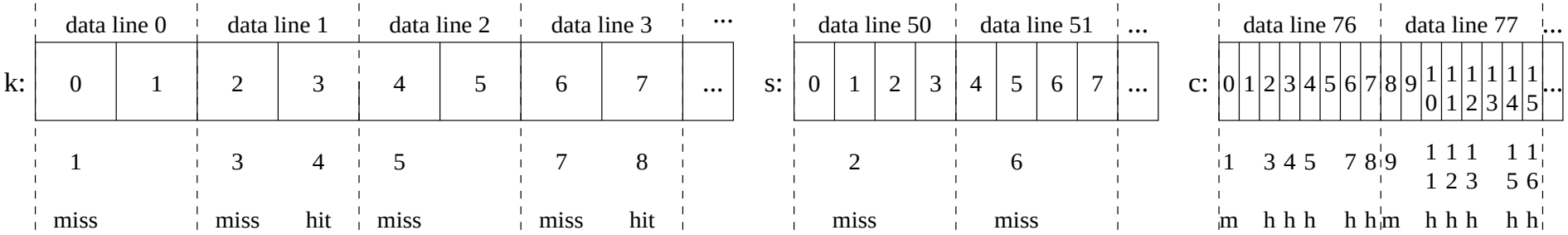
`mmmmmmmmmmmmmhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh`

$13 \times 19 \; + \; 37 \times 10$

What if there is conditional control flow and, thus, multiple paths through the loop?

- Could go through all iterations and choose longest path on each, or
- Choose longest path and replicate it for as long as it is guaranteed to be the longest.

Iteration in which each array element will be accessed and the cache result



Group Categorizations:

```
k[i]: c 50   from [m h m h ... m h ]
s[i]: c 25   from [m h h h m h h h ... m h h h]
c[i]: c 13   from [m h h h h h h h ... m h h h h h h h m h h h]
```

Timing Analysis Steps for Loop in WCLA Algorithm Example.:

step	start iter	longest path cycles	iterations handled	additional cycles	total cycles
1	1	20+18=38	13	20+((20-4)*12)+(18*13)=446	446
2	14	20+9=29	37	((20-4)*37)+(9*37)=925	1371
3	51	17+9=26	25	((17-4)*25)+(9*25)=550	1921
4	76	20+0=20	24	(20-4)*24=384	2305
5	100	20+0=20	1	20-4=16	2321

Test Programs

Cache Size: 512 bytes, 16 lines, 32 bytes/line

Name	Num Bytes	Description or Emphasis
Matcnta	40060	Counts and Sums Nonnegative Values in a 100x100 Integer Matrix
Matcntb	460	Counts and Sums Nonnegative Values in a 10x10 Integer Matrix
Matmula	30044	Multiplies 2 50x50 Matrices into a third 50x50 Integer Matrix
Matmulb	344	Multiplies 2 5x5 Matrices into a third 5x5 Integer Matrix
Matsuma	40044	Sums Nonnegative Values in a 100x100 Integer Matrix
Matsumb	444	Sums Nonnegative Values in a 10x10 Integer Matrix
Sorta	2044	Bubblesort of 500 Integer Array into Ascending Order
Sortb	444	Bubblesort of 100 Integer Array into Ascending Order
Statsa	16200	Calcs Sum, Mean, Var, Std Dev, Cor Coef of 2 arrays of 1000 doubles
Statsb	600	Calcs Sum, Mean, Var, Std Dev, Cor Coef of 2 arrays of 25 doubles
Des	1346	Data Encryption Standard—Encrypts and Decrypts 64 bits

WCET Measurements for Data Caching

Name	Hit Ratio	Observed Cycles	Estimated Cycles	Estim. Ratio	Naive Ratio
Matcnta	71.86%	1,143,014	1,143,023	**1.000**	1.148
Matcntb	70.73%	12,189	12,189	**1.000**	1.148
Matmula	62.81%	7,245,830	7,952,807	**1.098**	1.240
Matmulb	89.40%	11,396	11,396	**1.000**	1.332
Matsuma	71.86%	1,122,944	1,122,953	**1.000**	1.151
Matsumb	69.98%	11,919	11,919	**1.000**	1.152
Sorta	97.06%	4,768,228	9,826,909	**2.061**	2.883
Sortb	99.40%	188,696	371,977	**1.971**	2.915
Statsa	90.23%	1,237,698	1,447,572	**1.170**	1.290
Statsb	89.21%	32,547	37,246	**1.144**	1.290
Des	75.71%	155,340	191,564	**1.233**	1.448

WCET Measurements for Set-Associative Instruction Caching

Program	Observed Cycles	Estimated Cycles	Est./Obs. Ratio	Naive Ratio
Matcnta	443,754	443,790	**1.00**	9.99
Matmulta	1,430,538	1,430,538	**1.00**	10.00
Matsuma	343,628	343,646	**1.00**	9.99
Sorta	3,130,692	6,249,474	**2.00**	10.00
Statsa	183,491	192,518	**1.05**	9.94
Des	95,877	109,069	**1.14**	5.58