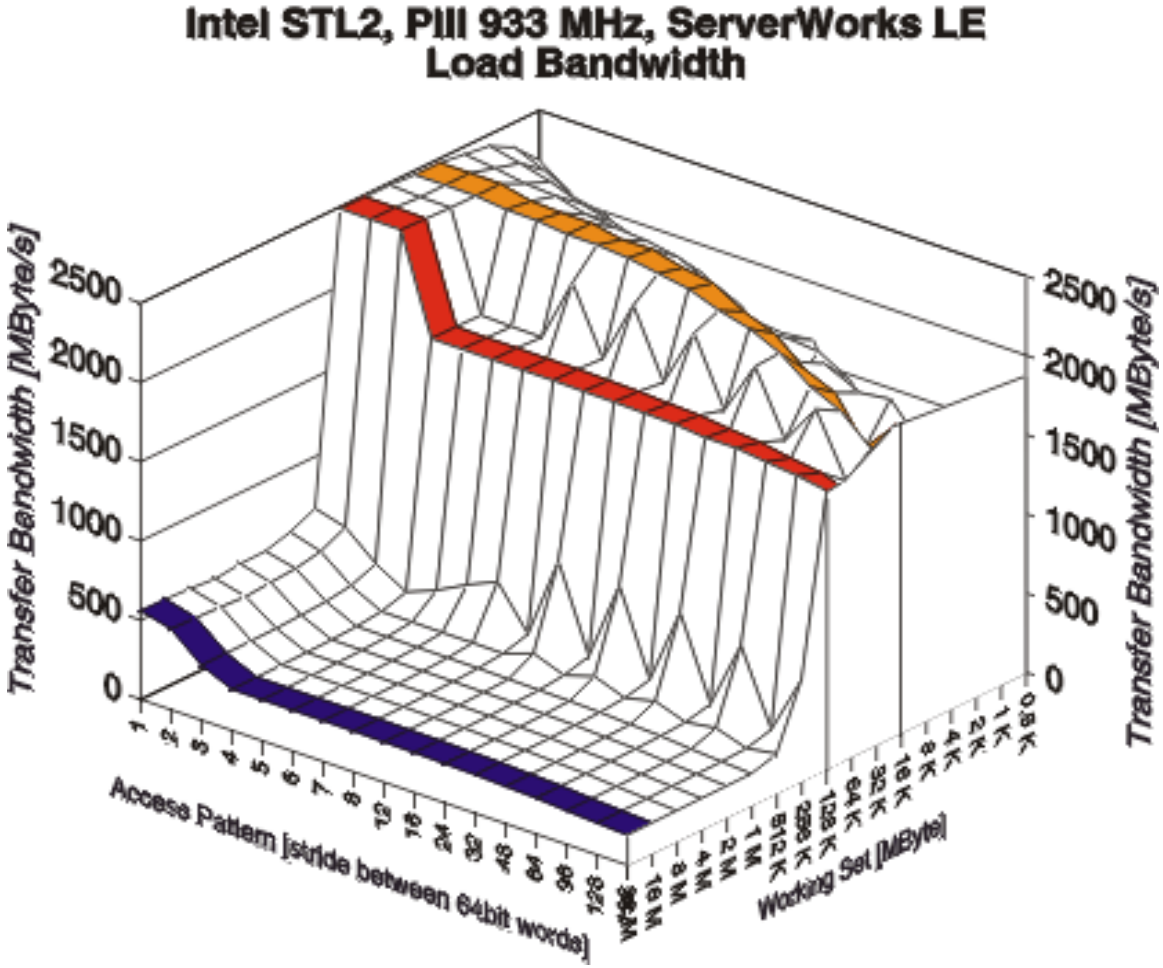
Collecting Information on Locality
(Data Accesses)

Irina Chihaia Tuduce and Thomas Gross
Laboratory for Software Technology
ETH Zurich, Switzerland
http://www.lst.inf.ethz.ch/

Memory System Performance



http://www.cs.inf.ethz.ch/CoPs/ECT/

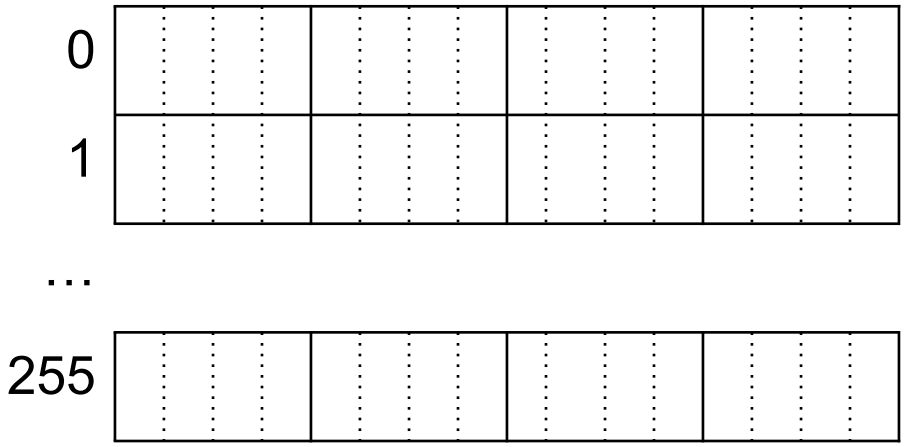
Isn’t the miss rate enough?

```
typedef struct {
  int a,
      b,
      c,
      d;
} my_struct_type;

my_struct_type s[1024];
```

Cache Example 1

L1 cache: 16kB, 64B blocks, direct mapped (256 lines)

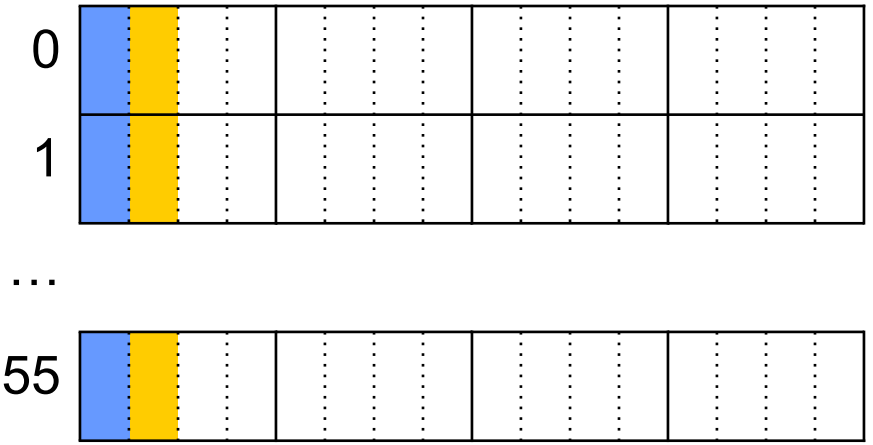


s[i] = 16B (4x4B) → s[0]-s[3] same cache line L1

s[0]-s[1023]: 256 cache lines

Example - Locality

```
for (i=0; i<1024; i+=4) {
  r += s[i].a;
  s += s[i].b;
}
```



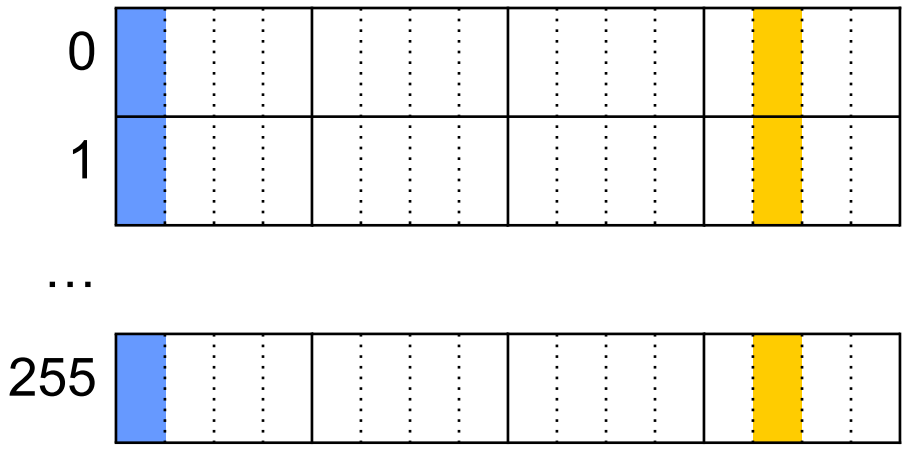
Total: 256 L1misses

sequential accesses: 256

(data access refers to next memory cell)

Example - Locality

```
for (i=0; i<1024; i+=4) {
  r += s[i].a;
  s += s[i+3].b;
}
```



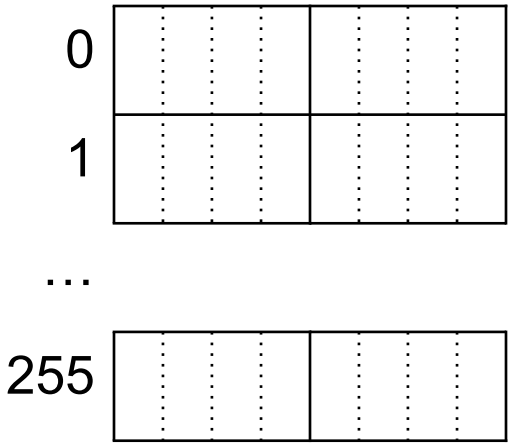
Total: 256 L1misses

accesses to same line L1: 256

… same line L2: 128 (L2 line = 2 x L1 line)

Cache Example 2

L1 cache: 8kB, 32B blocks, direct mapped (256 lines)



s[i] = 16B (4x4B) → s[0]-s[1] same cache line L1

s[0]-s[1024]: 512 cache lines

Example

```
for (i=0; i<1024; i+=2)
  r += s[i].a;
for (i=0; i<1024; i+=2)
  s += s[i].b;
```

256+256 Misses

256+256 Misses

Total: 1024 L1misses

same line L1: 0
same line L2: 512 (L2 line = 2 x L1 line)

Example – Improve Locality

```
for (i=0; i<1024; i+=2) {
  r += s[i].a;
  s += s[i].b;
}
```

256+256 Misses

Hits

Total: 512 L1misses

sequential : 512

same line L2: 256 (L2 line = 2 x L1 line)

Example – Smaller Working Set

```
for (i=0; i<512; i+=2)
   r += s[i].a;
for (i=0; i<512; i+=2)
   s += s[i].b;
for (i=512; i<1024; i+=2)
   r += s[i].a;
for (i=512; i<1042; i+=2)
   s += s[i].b;
```

256 Misses

Hits

256 Misses

Hits

Total: 512 L1misses

random accesses L1: 512
(neither sequential nor to the same line)
same line L2: 256 (L2 line = 2 x L1 line)

Locality Information

- Spatial locality: distance between two consecutive *data* accesses
- Understanding locality key to many program transformations
- Gathering detailed access information is challenging and expensive
  - Can't afford to record all accesses
  - Need simple way to capture program characteristics

Proposal: Locality Information

Classify (data) accesses according to their performance :

- accesses to the same location
- sequential accesses: pipelined transfers
- accesses within the same cache line: whole line fetch
- random accesses: read ahead and prefetching do not help

Good estimation of program locality provided by the frequency of each access type

Counters for Locality Information Gathering

Set of counters to capture accesses according to classification:

- **`C_same`**: accesses to the same location
- **`C_seq`**: sequential accesses
- **`C_line_i`**: accesses to the same cache line for each cache level **`i`**
- random accesses can be computed from these counters and cache hit rates

Possible Implementation

F and S addresses of two consecutive accesses

```
if (F == S) {
  C_same++;
  return;
}
if (|F-S|/access_size == 1) {
  C_seq++;
  return;
}
```

Possible Implementation (cont.)

/* neither same, nor sequential */

L1 cache

```
  if S is a L1 hit {
     determine cache line Line_F of F
     determine cache line Line_S of S
     if (Line_F == Line_S)
          C_line_1++;
     return;
  } else  /* S is a L1 miss */
     continue with L2
```

Random Accesses (Hits)

Capture other accesses to level i

```
N_random_L1 =
  N_hits_L1 - C_same - C_seq -
  - C_line_1

N_random_L2 =
  N_hits_L2 - C_line_2
```

Usage scenarios

- Identify segments of code with poor spatial locality
  → improve code locality

- Locality counters can be a basis for performance analysis
  → Compare execution time before and after transformation, insights about effectiveness of a transformation.

Summary

- Use performance monitoring counters to capture a program's spatial locality

- Few counters can provide useful information
  - 2 cache levels: 4 counters
  - 3 cache levels: 5 counters

Thank you!

Caveats & Issues

- Reset counters on cache misses
- C_seq: accesses with stride 1
  - stride is 1 in source language terms
- If we can afford more counters, different strides are interesting