VI-HPS Plenary
Acumem Virtual Performance Expert

Mats Nilsson, VP Engineering
mats@acumem.com

Workflow – initial preparations

- Establish a baseline
  - It is ESSENTIAL to have a repeatable, correct test case.
- Compiler flags
  - Use the correct compiler options
  - Acumem VPE requires debug information to be able to point back to source code.

    Thus: compile with -g

    Some compilers (e.g., Intel) require extra flags to cause debug information to be emitted for inlined functions – check the compiler manual.

    Prefer Dwarf debugging format over Stabs if the compiler has that option.

Workflow – the obvious

- Avoid unnecessary work
  - Select the right algorithms
  - Avoid copying data
    - In and out of data structures
    - On the stack as function call parameters.
  - Judiciously break encapsulation to expose internal data without copying
  - Use data types of correct size.

Repeated copying of data will paradoxically cause the miss ratio appear to go down – most accesses will be satisfied by L1.

Workflow – utilization

- Improve utilization
  - hot/cold fields
  - Sparse data representation
  - Alignment
  - Dynamic allocations
  - Invalid loop order

Workflow – data reuse

- Look for reuse opportunities
  - Blocking
  - Loop fusion

  (somewhat related:)
- Reduce local memory footprint
  - Loop fission

Workflow – final adjustments

- Coping with remaining latency problems
  - Arrange for better hw-prefetchability
  - Adding prefetch instructions
  - MLP (Memory Level Parallelism, move instructions that miss in cache closer together)
- Help the prefetcher
  - Differences in prefetch strategy between different CPU:s. We model a consequtive cache-line prefetcher, such as found in
  - Remove irregular data structures and algorithms
  - Pad data to be on separate pages
  - Measure effects of varying degree of loop unrolling

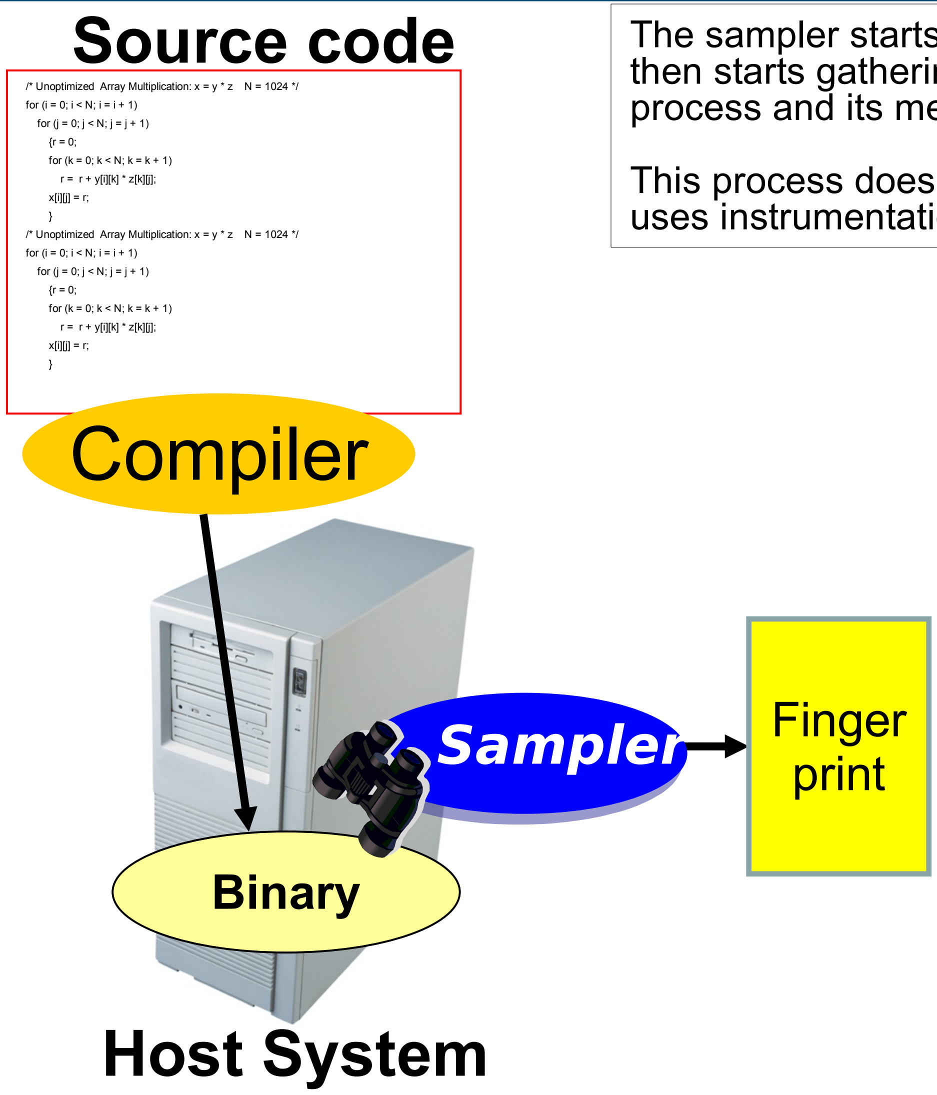
Workflow – upcoming features

- Multicore optimizations
  - Detect false sharing
  - Measure communication efficiency
    - How much of each cache line communicated to a different core is actually used by that core before that cache line is evicted.
  - Classify upgrade misses
- Other serial improvements
  - Avoid cache conflicts
    - Find code that is prone to cause cache conflict misses

Acumem Technology

The sampler starts a process or injects itself into a running process. It then starts gathering rich but sparse information about the target process and its memory accesses.

This process does not involve HW performance counters, but instead, uses instrumentation to obtain this hardware independent fingerprint.



```
/* Unoptimized  Array Multiplication: x = y * z    N = 1024 */
for (i = 0; i < N; i = i + 1)
   for (j = 0; j < N; j = j + 1)
      {r = 0;
      for (k = 0; k < N; k = k + 1)
         r =  r + y[i][k] * z[k][j];
      x[i][j] = r;
      }
/* Unoptimized  Array Multiplication: x = y * z    N = 1024 */
for (i = 0; i < N; i = i + 1)
   for (j = 0; j < N; j = j + 1)
      {r = 0;
      for (k = 0; k < N; k = k + 1)
         r =  r + y[i][k] * z[k][j];
      x[i][j] = r;
      }
```

Acumem Technology

The fingerprint is analyzed, and it is not until in this step that you supply the intended parameters for the target system.

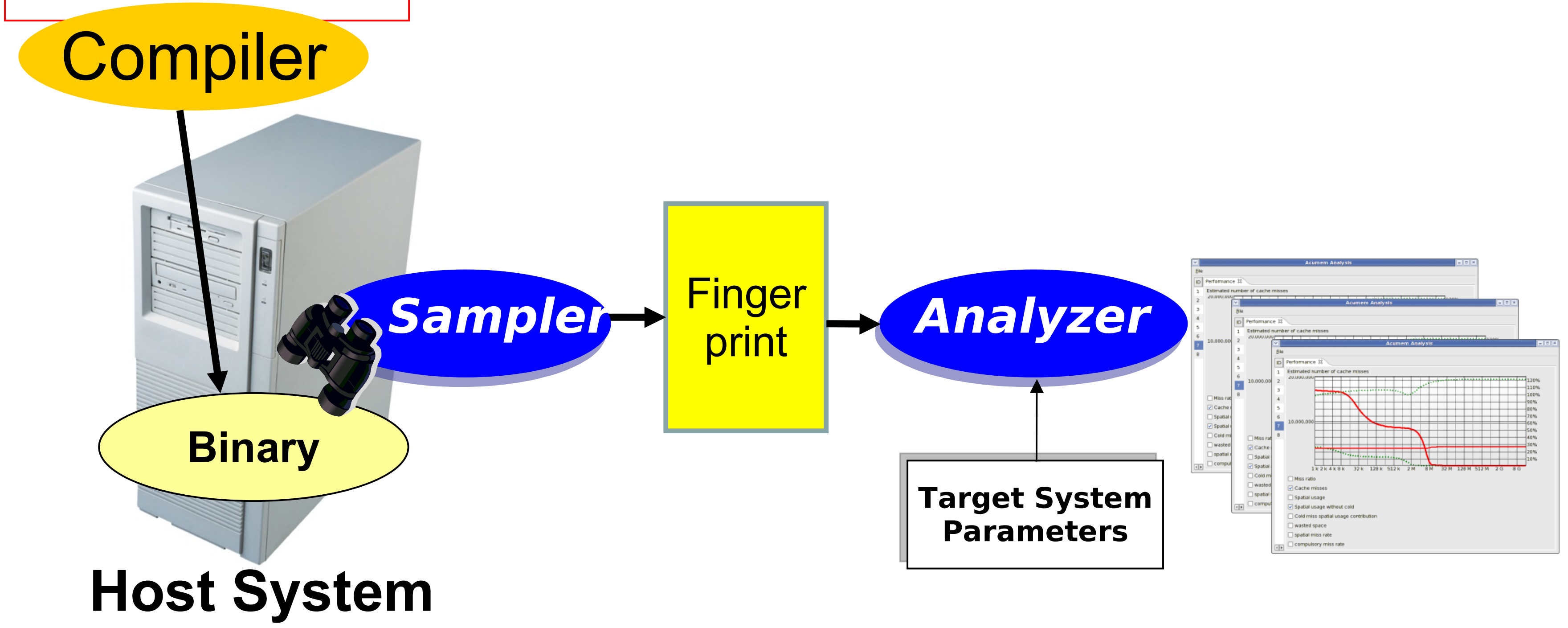
The analysis produces two different outputs:

1. projections of how cache miss ratio, fetch ratio and cache utilization varies with cache size.

...

Source code

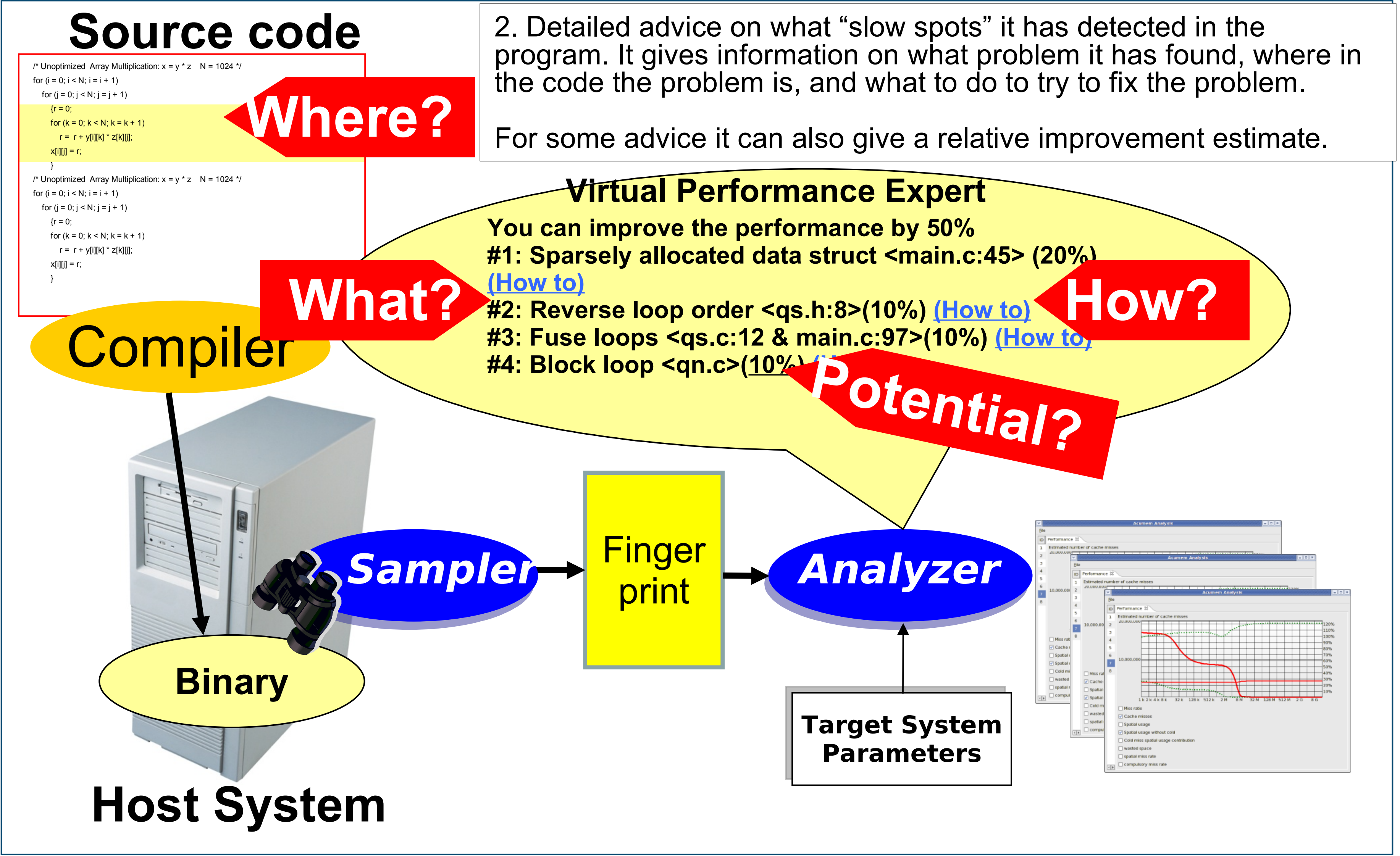
```
/* Unoptimized  Array Multiplication: x = y * z    N = 1024 */
for (i = 0; i < N; i = i + 1)
   for (j = 0; j < N; j = j + 1)
      {r = 0;
      for (k = 0; k < N; k = k + 1)
         r =  r + y[i][k] * z[k][j];
      x[i][j] = r;
      }
/* Unoptimized  Array Multiplication: x = y * z    N = 1024 */
for (i = 0; i < N; i = i + 1)
   for (j = 0; j < N; j = j + 1)
      {r = 0;
      for (k = 0; k < N; k = k + 1)
         r =  r + y[i][k] * z[k][j];
      x[i][j] = r;
      }
```



Acumem Technology

2. Detailed advice on what “slow spots” it has detected in the program. It gives information on what problem it has found, where in the code the problem is, and what to do to try to fix the problem.

For some advice it can also give a relative improvement estimate.



```
/* Unoptimized  Array Multiplication: x = y * z    N = 1024 */
for (i = 0; i < N; i = i + 1)
   for (j = 0; j < N; j = j + 1)
      {r = 0;
      for (k = 0; k < N; k = k + 1)
         r =  r + y[i][k] * z[k][j];
      x[i][j] = r;
      }
/* Unoptimized  Array Multiplication: x = y * z    N = 1024 */
for (i = 0; i < N; i = i + 1)
   for (j = 0; j < N; j = j + 1)
      {r = 0;
      for (k = 0; k < N; k = k + 1)
         r =  r + y[i][k] * z[k][j];
      x[i][j] = r;
      }
```

Virtual Performance Expert

You can improve the performance by 50%
#1: Sparsely allocated data struct <main.c:45> (20%) (How to)
#2: Reverse loop order <qs.h:8>(10%) (How to)
#3: Fuse loops <qs.c:12 & main.c:97>(10%) (How to)
#4: Block loop <qn.c>(10%)

Getting started

- RWTH environment

```
$ module load acumem
```

- Starting a sampling
  - Start program and sample from beginning

```
$ sample -o <output filename> -r <binary> <parameters>
```

  - Attach to running process. Then type “q” to quit and detach. The process will continue running, as if nothing happened.

```
$ sample -o <output filename> -p <pid>
```

Getting started 2

- Preparing/viewing a report

  ```
  $ report -i <fingerprint filename> -o <reportname> -c <target cache size>
  ```

  This creates a file “<reportname>.html” and a directory “<reportname>”.

- Viewing the report

  ```
  $ firefox <reportname>.html
  ```

MPI

- Sampling MPI programs requires using a wrapper script, like the following:

```
$ cat wrapper.sh
sample -o sample-file$MPD_JRANK.smp a.out

$ mpiexec -n 2 sh -c wrapper.sh
```

- Prepare one report for each rank

```
$ report -i sample-file0.smp -o report0 -c 1M
$ report -i sample-file1.smp -o report1 -c 1M
```

Adv. Sampler options

- It is seldom necessary to sample the entire program.

  A few iterations of your outer loop usually gives enough coverage.

  If you have knowledge about where the different phases in your program begin and end, you can control the sampling to cover only a specific part of your program.

  - After a certain period of time

    ```
    $ sample ... -d <seconds> ...          (d = delay)
    $ sample ... -t <seconds> ...          (t = timeout)
    ```

  - upon reaching a function

    ```
    $ sample ... --start-at-function <function name> ...
    $ sample ... --stop-at-function <function name> ...
    ```

  - upon executing an instruction (address)

    ```
    $ sample ... --start-at-address <hex address> ...
    $ sample ... --stop-at-address <hex address> ...
    ```

Adv. Sampler options

- Line sizes to sample (default 1,32,64,4096

  This is to allow modelling different architectures

  ```
  $ sample ... --line-sizes=1,64,128,4096 ...
  ```

  (1,4096 required for full analysis functionality)

Adv. Reporter options

- Replacement policy

  `$ report ... -r <replacement policy> ...` (lru, default:random)

- Line size

  `$ report ... -l <target line-size> ...`

  (must be one of the sampled sizes)

- Stack depth

  Analysis optionally takes call stack contents into account for separating issues.

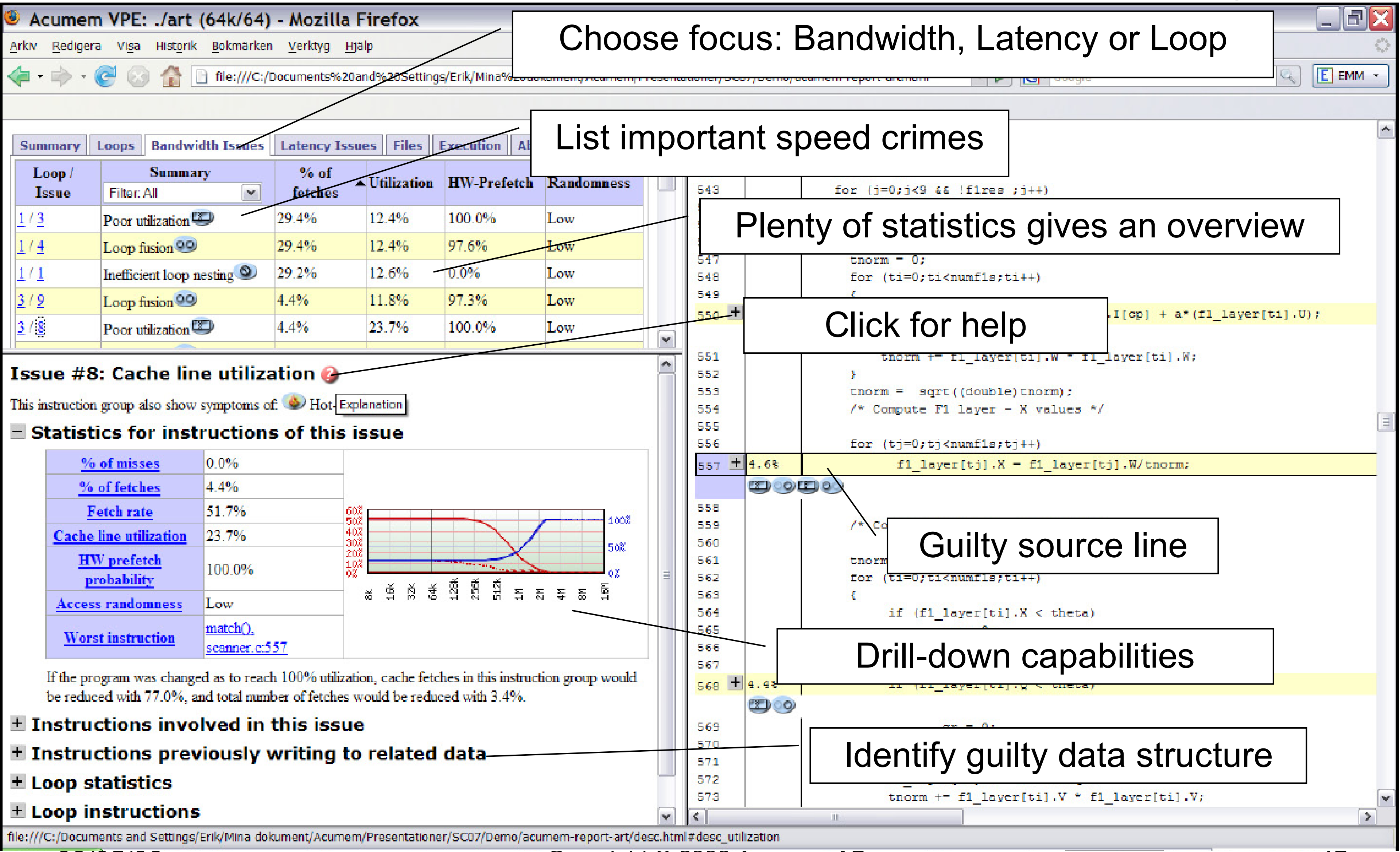
  `$ report ... -d <depth> ...` (0 – 32, default 1)

- Hide unimportant issues

  Hide issues with insufficient number of fetches

  `$ report ... -p <percent of total fetches> ...` (0 – 100, default 1)

Acumem VPE


Acumem VPE: ./art (64k/64) - Mozilla Firefox
Arkiv Redigera Visa Historik Bokmärken Verktyg Hjälp
file:///C:/Documents%20and%20Settings/Erik/Mina%20dokument/Acumem/Presentationer/SC07/Demo/acumem-report-art/html
EMM
Summary
Loops
Bandwidth Issues
Latency Issues
Files
Execution
Choose focus: Bandwidth, Latency or Loop
List important speed crimes
Plenty of statistics gives an overview
Click for help
Guilty source line
Drill-down capabilities
Identify guilty data structure
60%
50%
40%
30%
20%
10%
0%
100%
50%
0%
8k
16k
32k
64k
128k
256k
512k
1M
2M
4M
8M
16M
file:///C:/Documents and Settings/Erik/Mina dokument/Acumem/Presentationer/SC07/Demo/acumem-report-art/desc.html#desc_utilization

Loop / Issue	Summary Filter: All	% of fetches	▲Utilization	HW-Prefetch	Randomness
1 / 3	Poor utilization	29.4%	12.4%	100.0%	Low
1 / 4	Loop fusion	29.4%	12.4%	97.6%	Low
1 / 1	Inefficient loop nesting	29.2%	12.6%	0.0%	Low
3 / 9	Loop fusion	4.4%	11.8%	97.3%	Low
3 / 8	Poor utilization	4.4%	23.7%	100.0%	Low

Issue #8: Cache line utilization

This instruction group also show symptoms of: Hot- Explanation

− **Statistics for instructions of this issue**

% of misses	0.0%
% of fetches	4.4%
Fetch rate	51.7%
Cache line utilization	23.7%
HW prefetch probability	100.0%
Access randomness	Low
Worst instruction	match(), scanner.c:557

If the program was changed as to reach 100% utilization, cache fetches in this instruction group would be reduced with 77.0%, and total number of fetches would be reduced with 3.4%.

+ **Instructions involved in this issue**

+ **Instructions previously writing to related data**

+ **Loop statistics**

+ **Loop instructions**

```
543          for (j=0;j<9 && !f1res ;j++)
547             tnorm = 0;
548             for (ti=0;ti<numf1s;ti++)
549             {
550 +                            .I[cp] + a*(f1_layer[ti].U);
551                tnorm += f1_layer[ti].W * f1_layer[ti].W;
552             }
553             tnorm =  sqrt((double)tnorm);
554             /* Compute F1 layer - X values */
555
556             for (tj=0;tj<numf1s;tj++)
557 + 4.6%         f1_layer[tj].X = f1_layer[tj].W/tnorm;
558
559             /* Co
560
561             tnorm
562             for (ti=0;ti<numf1s;ti++)
563             {
564                if (f1_layer[ti].X < theta)
565
566
567
568 + 4.4%
569
570
571
572
573                tnorm += f1_layer[ti].V * f1_layer[ti].V;
```

Acumem VPE Issues

- Issues
  - Utilization
  - Loop nesting
  - Irregular access patterns
  - Temporal/spatial Blocking
  - Loop Fusion
  - SW prefetch analysis
  - Hot-spot

Explained in the following pages...

Issue: Utilization

- Complains about sparse memory usage.
- In the context of this issue, not all data in a cache line is used.
- Low utilization causes:
  - Too high memory bandwidth usage, reduces scalability
  - Effective cache size decreased – cache filled with unused data
  - Reduced Spatial locality benefits – each cache line contains less useful data, need to go to memory more often
- See online help for possible reasons.

Issue: Loop nesting

- Acumem VPE has detected that a loop is not going through a data area in a efficient manner.
- This causes high cache miss ratios, and possibly excessive TLB misses.
- Consider turning loop levels around

Issue: Irregular access patterns

- Random, or pseudo-random access patterns have been detected.
- This prevents hw prefetcher from working optimally, leading to high degree of cache misses.
- It could be related to memory placement, such as following a linked list, or searching through a tree causes accesses to jump around in memory
- It could be related to the algorithm (e.g., graph algorithms)
- It could be a desired (?) feature – hash tables.

Issue: Temporal/Spatial Blocking

- Acumem VPE has detected that there is an opportunity to rearrange loops to get more immediate data reuse.
- Acumem VPE will perform some data dependency checks to filter out those opportunities that are deemed improbable.
- It will try to identify the interesting loop levels.

  See online manual for more explanations on a structured way to perform blocking.

Issue: Loop Fusion

- Detect that two different loops use the same data.
- By moving loops closer together, or fusing them completely, then caches will not have to be reloaded for the second use.

Issue: Prefetch

- Acumem VPE judges how well prefetch instructions are placed (by you or the compiler).
- Three cases:
  - They must perform useful work, i.e., they should pull in data which wasn't already loaded
  - They must appear sufficiently close to the place where data is used, or else data may be evicted before use.
  - They must appear sufficiently far from the place where data is used, or else data will not have arrived to the cache

Issue: “Hotspot”

- A catch-all issue – we see a lot of cache misses/fetches but no other negative signs
- If there is a lot of misses, adding prefetch instructions or changing the layout to help the HW prefetcher might help.

Acumem VPE Supporting information

- Source and instruction references
  - Source code navigation
  - Instruction address/type
  - Call stack presentation/analysis
  - Subexpression identification
  - Drill down: loop → instruction groups → instructions
- Metrics
  - Fetch ratio
  - Miss ratio
  - Randomness
  - HW prefetchability
  - Cache line utilization
  - Estimates of how fixing problems affect fetch ratio

More information

- The manual is installed with each report. There are Hyperlinks from report into relevant parts of the manual.

- Web site
  http://www.acumem.com

- Further inquiries:
  info@acumem.com

- Mats Nilsson
  mats@acumem.com