



Increased performance with AMD CodeAnalyst™ software and Instruction-Based Sampling

Paul J. Drongowski
AMD CodeAnalyst Software Team

June 19, 2008

AMD CodeAnalyst™ software for Linux®

AMD CodeAnalyst is a profiling suite designed to help identify and investigate performance hot-spots.

AMD CodeAnalyst software is integrated with OProfile and is licensed as open source software (GPL version 2.)

Its graphical user interface assists profile collection and analysis. Command line use is supported via OProfile.

Profile collection is designed to be system-wide and low overhead.

Programs can be analyzed using time-based profiling, event-based profiling and **Instruction-Based Sampling** – an innovation offered by quad-core AMD Opteron™ and AMD Phenom™ processors.

Presentation outline

AMD CodeAnalyst in action

- Time-based profiling
- Event-based profiling
- Tuning

Instruction-Based Sampling

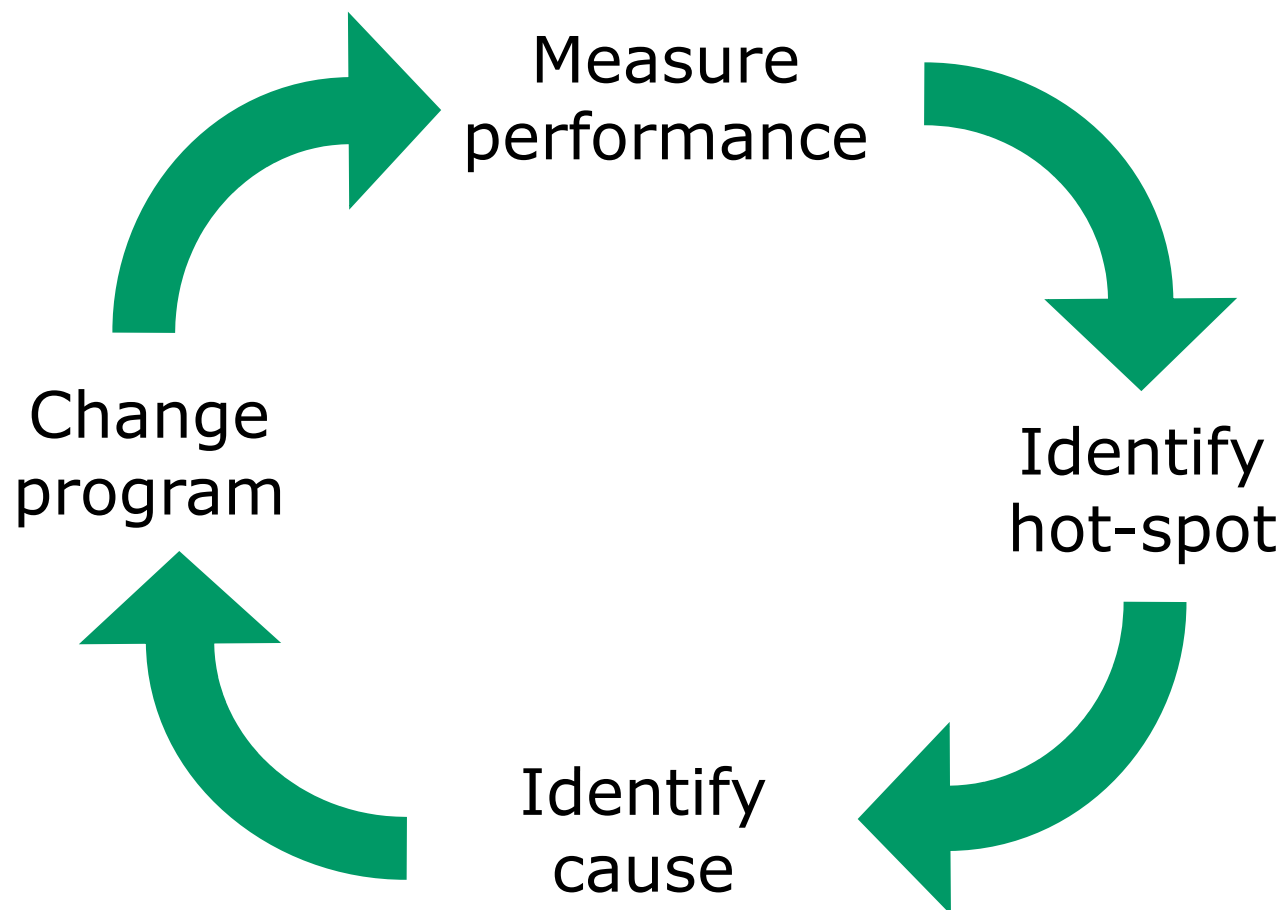
How to get AMD CodeAnalyst software

On-line resources for program performance tuning



AMD CodeAnalyst™ Software in action

Performance tuning cycle



Kinds of analysis

Identify hot-spots

- “Where is the application or system spending its time?”
- Use time-based profiling

Identify cause

- “How well is the application using the CPU and Memory?”
- “What are the performance issues in a code region?”
- Use event-based profiling
- Use Instruction-Based Sampling

Example program

SPEC CPU2000 179.art benchmark

- Simulates a neural network.
- Operates on arrays.
- Performance is cache/memory access sensitive.

On-line references

- [Compilers and more: Are optimizing compilers important?](#)
Michael Wolfe, The Portland Group, Inc.
- [Compilers and more: Gloptimizations](#), Michael Wolfe, The Portland Group, Inc.

Program preparation

AMD CodeAnalyst™ software monitors running software components:

- Applications
- Libraries
- Operating system kernel

No change to source code is necessary.

Symbolic information is needed to break down data by function or source line.

Compile and build application with debug information:

```
gcc -O2 -g -o art scanner.c
```


Beware of conventional wisdom

Conventional wisdom — “*just compile with -O2 and you’ll be OK*” — is sometimes detrimental!

Compile option	Execution time
-O0	121.097 sec
-O2	165.673 sec
-O2 -mfpmath=sse,387 -march=amdfam10	74.920 sec

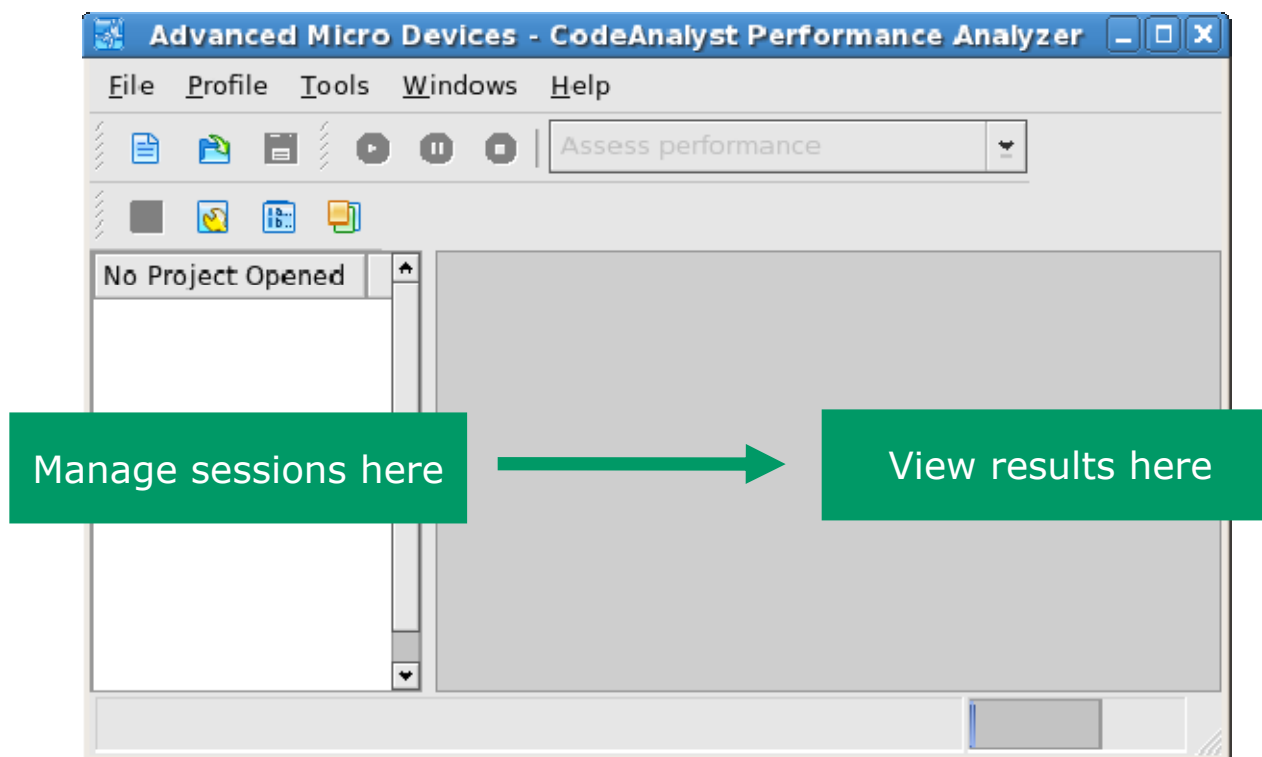
Slowdown is due to data cache and data TLB misses.
A floating point performance kick can come from SSE2.
We will use the third case as the baseline measurement.



Time-based profiling

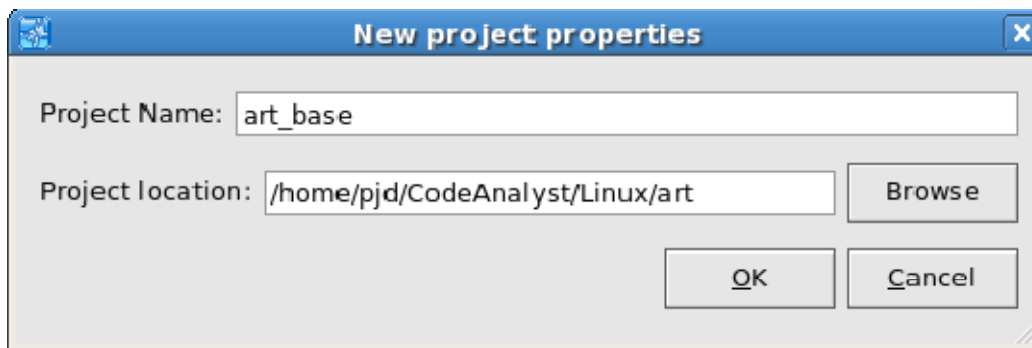
Graphical user interface

Start AMD CodeAnalyst™ software from command line.
No command line options are required.
Usage model is project- and session-oriented.



Create a new project

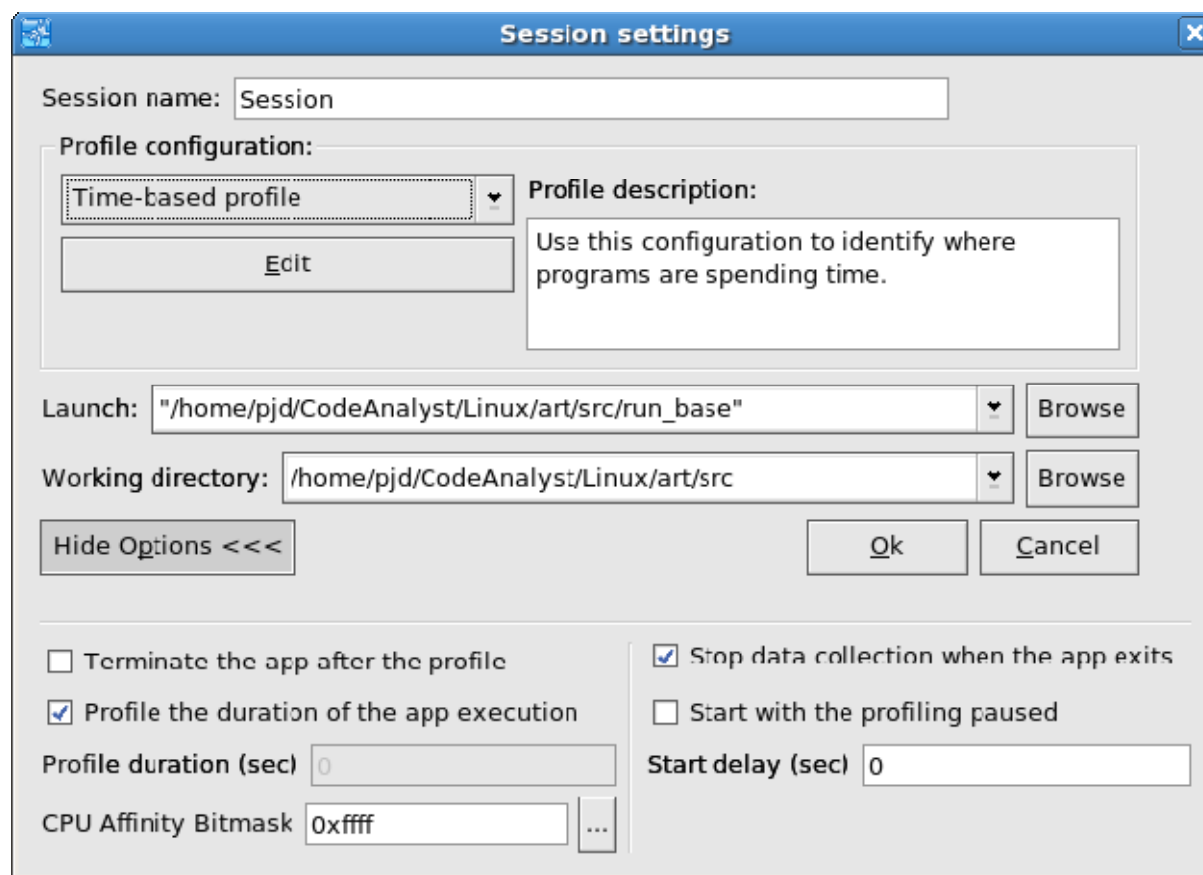
Select "File > New" to create a new project.
Enter a name for the new project in the dialog box.
Choose a directory to hold the project.
Directory is not restricted to any specific (sub)tree.



Prepare to measure performance

Enter the path to the application program (or test script) and set the working directory.

Choose the kind of analysis to perform and click "OK."



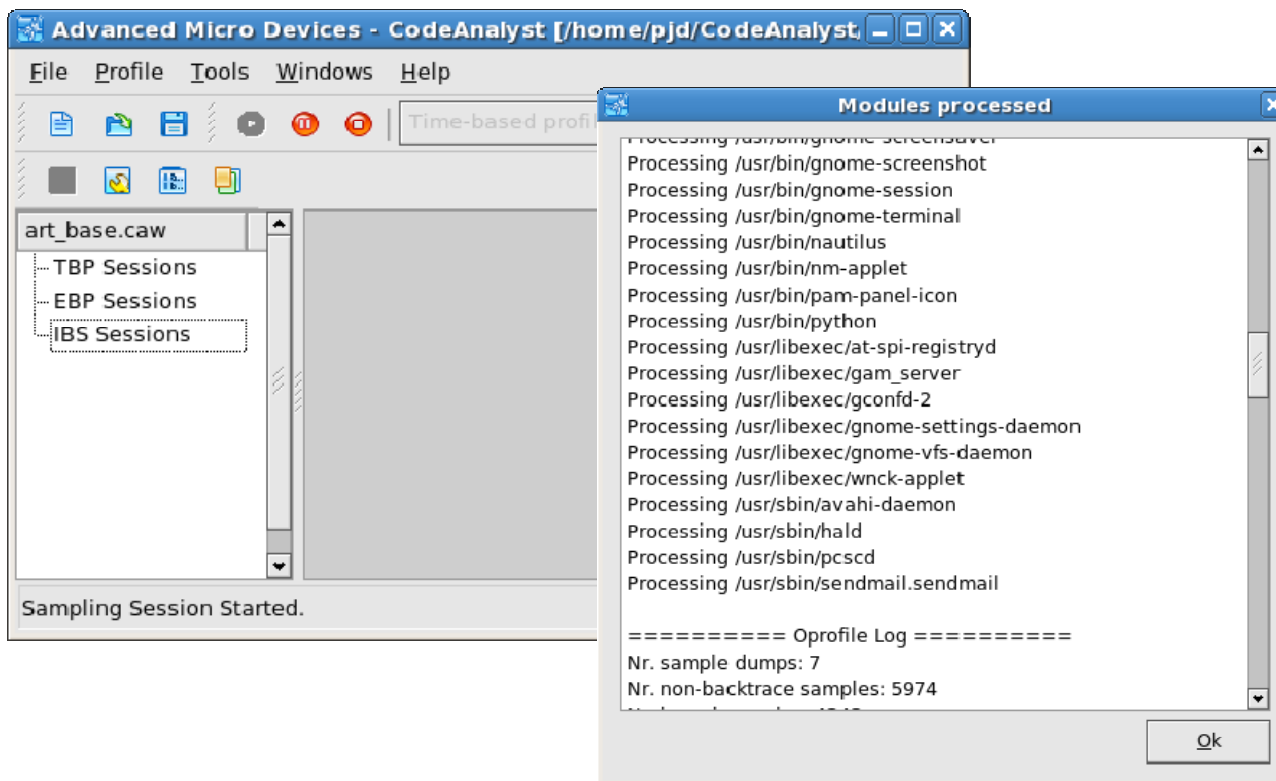
The image shows a "Session settings" dialog box with the following fields and options:

- Session name:** Session
- Profile configuration:**
 - Time-based profile (selected)
 - Edit button
- Profile description:** Use this configuration to identify where programs are spending time.
- Launch:** "/home/pjd/CodeAnalyst/Linux/art/src/run_base" (with a dropdown arrow and a Browse button)
- Working directory:** /home/pjd/CodeAnalyst/Linux/art/src (with a dropdown arrow and a Browse button)
- Buttons:** Hide Options <<<, Ok, Cancel
- Checkboxes:**
 - ☐ Terminate the app after the profile
 - ☒ Profile the duration of the app execution
 - ☒ Stop data collection when the app exits
 - ☐ Start with the profiling paused
- Profile duration (sec):** 0
- Start delay (sec):** 0
- CPU Affinity Bitmask:** 0xffff (with a dropdown arrow)

Start profile data collection

Click the start button  in the toolbar.

AMD CodeAnalyst™ software runs the application, collects data, imports the data and displays results.

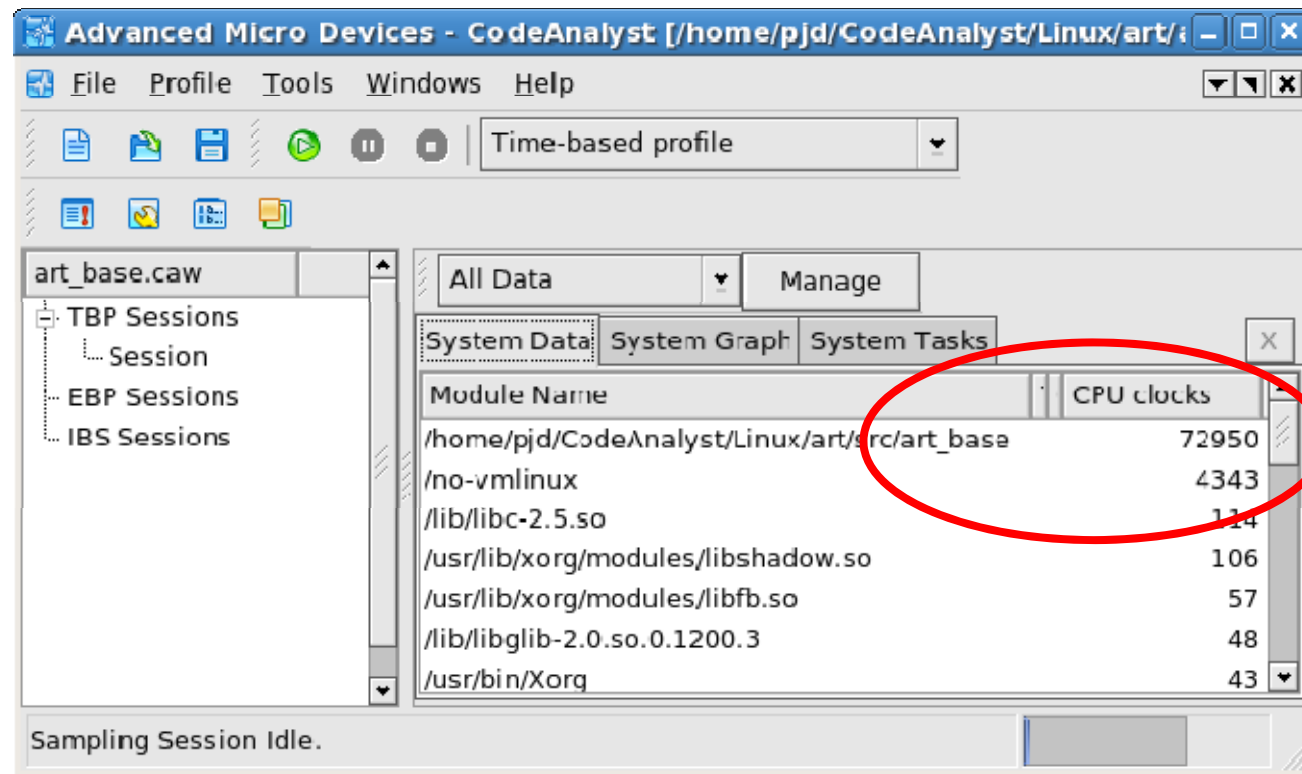


Time-based profiling: modules

Shows time for each module.

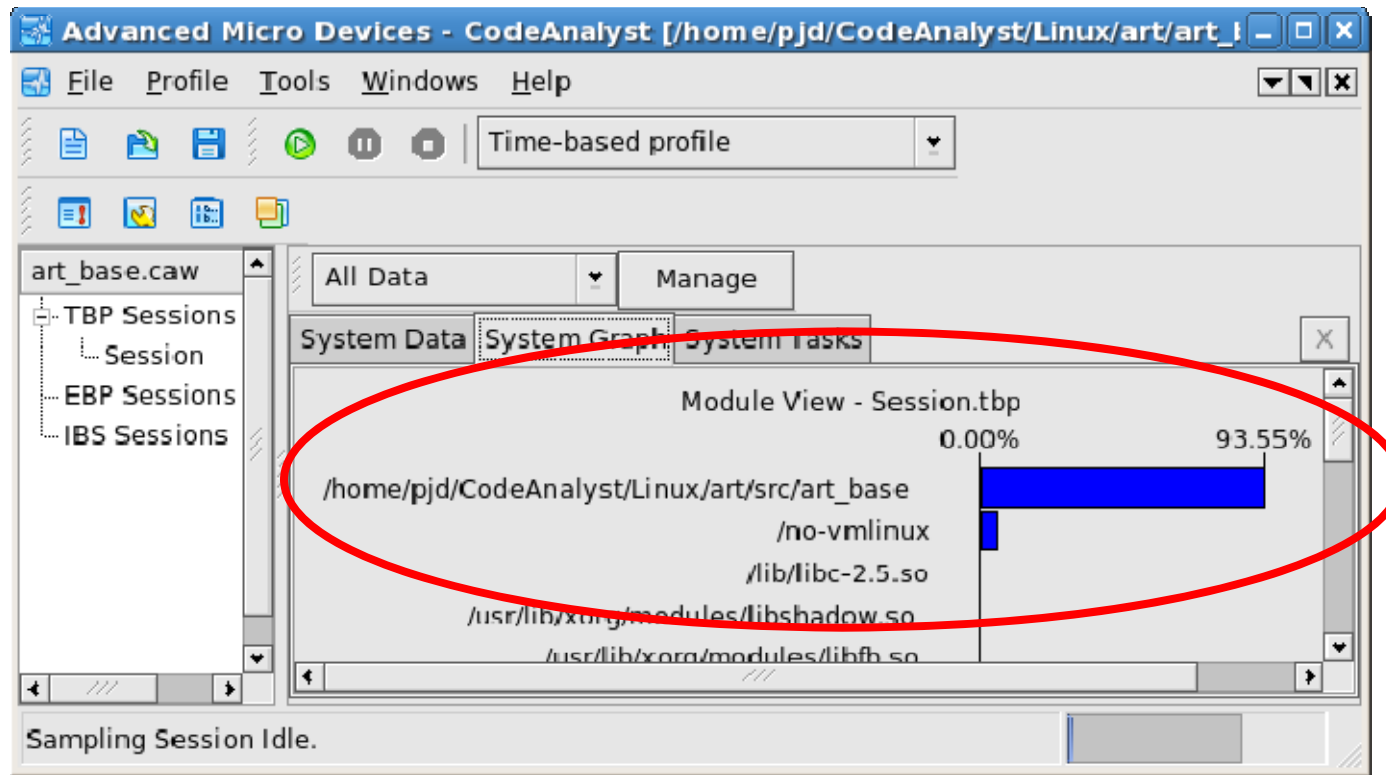
Time is measured by sampling event 0x076 (CPU clocks.)

Double-click a module name to drill down to functions.



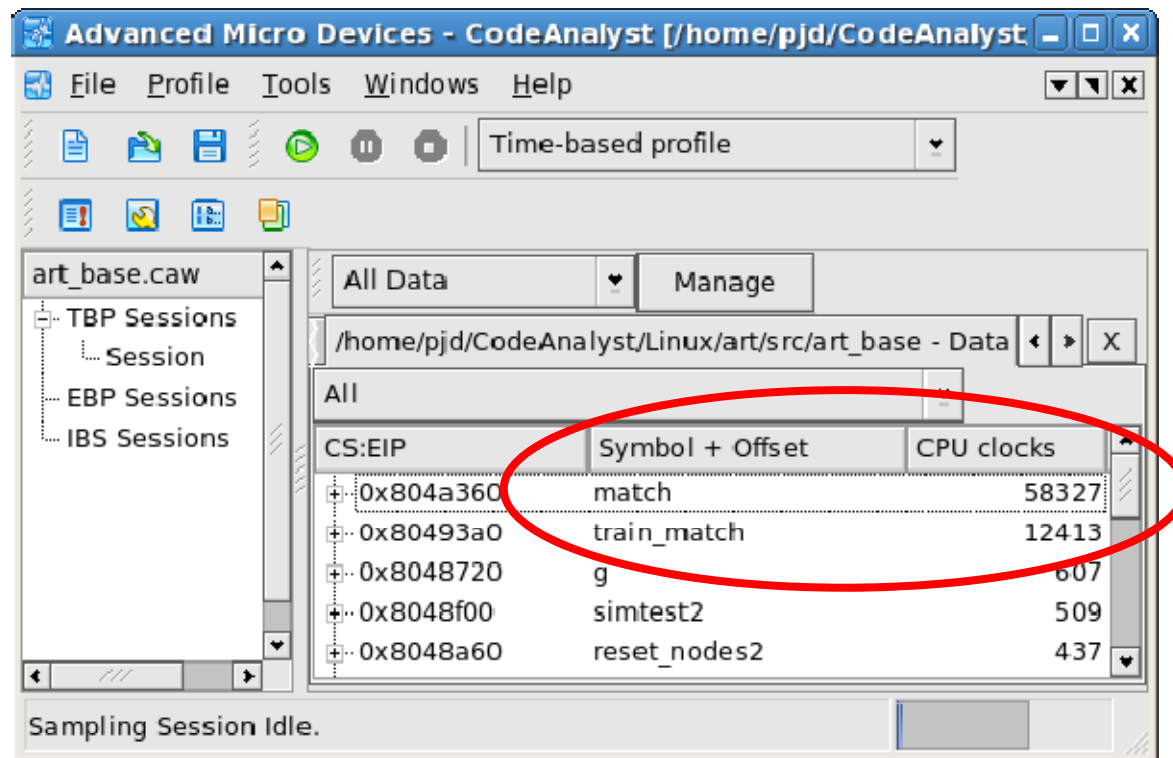
Time-based profiling: modules

Results are shown in chart form, too.



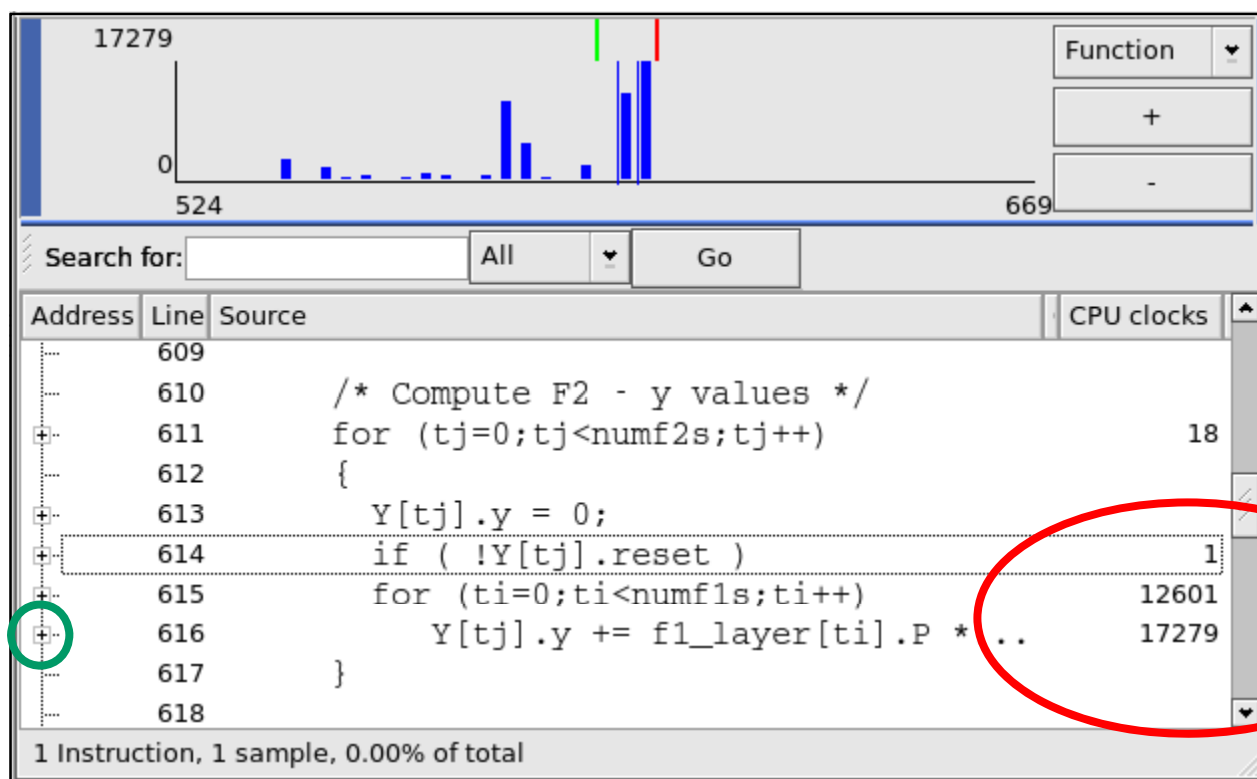
Time-based profiling: functions

Shows a function-by-function time breakdown.
Identify candidates for investigation and tuning.
Double-click a function name to drill down to source.



Time-based profiling: source

Identify code within “match” to investigate and tune.
Click on an expansion box  to expose instructions.



Time-based profiling: instructions

Time samples are distributed throughout the hot loop.

Address	Line	Source	CPU clocks
	616	Y[tj].y += fl_layer[ti].P *...	17279
0..		mov 0x804ca80,%esi	
0..		mov 0x804ca9c,%eax	
0..		mov 0x804cae8,%edx	
0..		mov 0x804cb00,%edi	
0..		movl \$0x0,0xffffffffd0(%ebp)	
0..		mov %esi,0xffffffffc0(%ebp)	
0..		mov %eax,0xffffffffc8(%ebp)	
0..		mov %edx,0xffffffffb8(%ebp)	
0..		mov 0xffffffffc8(%ebp),%esi	60
0..		movsd 0x24(%edx),%xmm0	756
0..		mov (%esi,%ecx,4),%eax	296
0..		mov 0xfffffffffa8(%ebp),%esi	4960
0..		mulsd (%eax,%esi,1),%xmm0	4880
0..		addsd %xmm1,%xmm0	4843
0..		movapd %xmm0,%xmm1	69
0..		movsd %xmm0, (%ebx)	1415

1 Instruction, 1 sample, 0.00% of total

What do we know at this point?

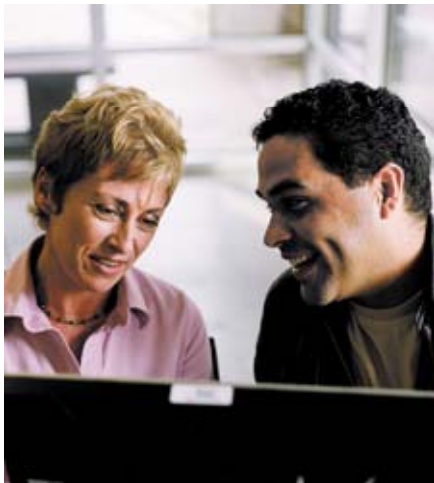
The function `match()` consumes the most time.

There are several hot loops in `match()`.

There are two loops that are the hottest:

- “Compute F1 layer – P values”
- “Compute F2 – y values”

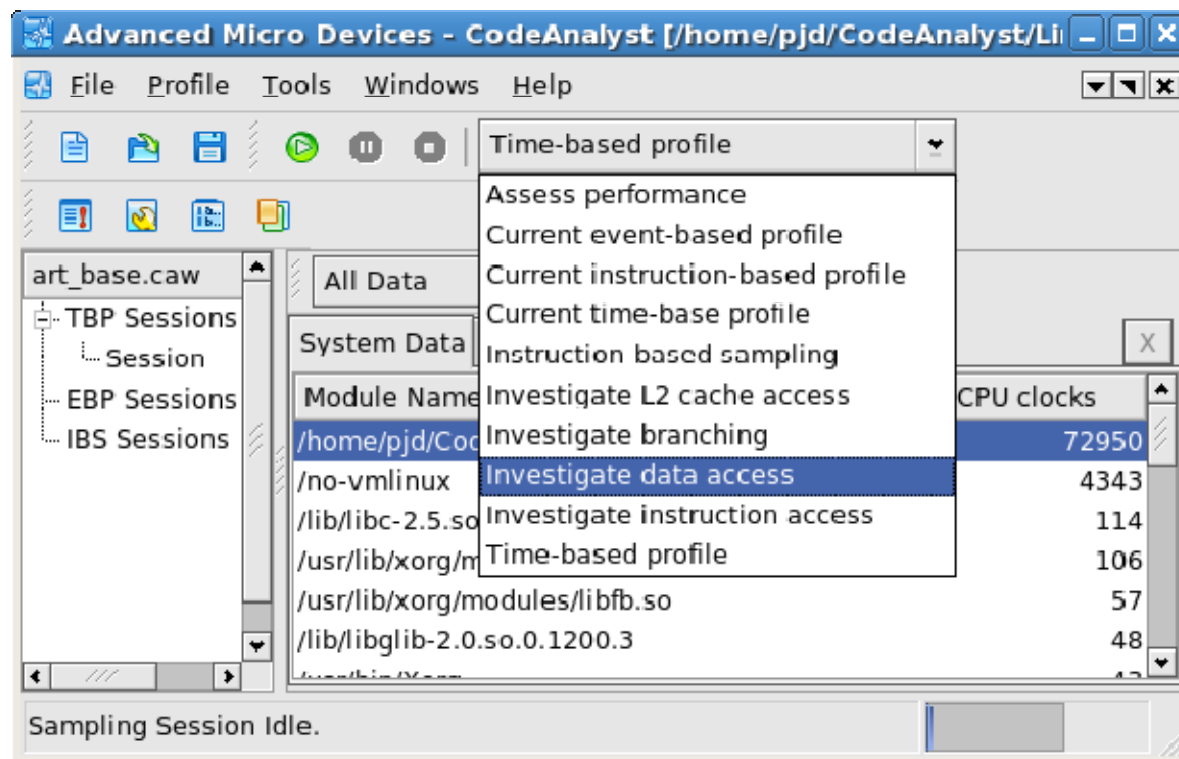
Next, let's use event-based profiling to identify performance issues.



Event-based profiling

Event-based profiling

Observe how the program behaves on CPU and memory.
Define and test hypothesis about a performance issue.
Choose the kind of analysis to perform.



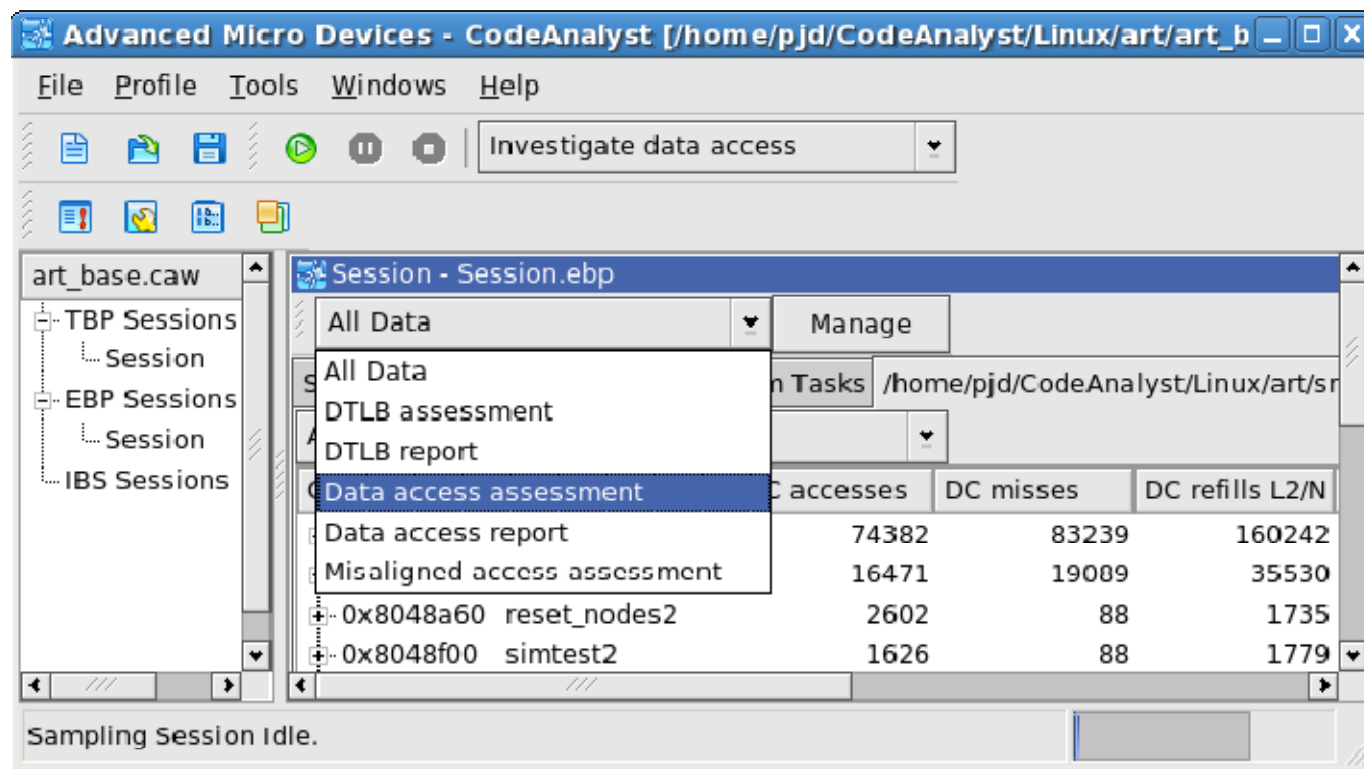
Kinds of analysis

Choose from predefined profile configurations.
Or, define a custom profile configuration.

Profile configuration	Focus
Time-based profile	Hot-spot identification
Assess performance	Overall evaluation ("triage")
Investigate data access	Memory access patterns / locality
Investigate L2 cache access	Memory access patterns / locality
Investigate branching	Branch and return prediction
Investigate instruction access	Code placement / locality
Instruction-Based Sampling	Region-specific fetch and execution

Event-based profiling

Display and drill-down are similar to time-based profiling. Views are offered depending upon available event data. Views show computed measurements such as ratios.



Hot-spot: Compute F2 - y values

Instruction-level event breakdown for inner loop.

Memory access events are distributed across inner loop.

Address	Instruction	DC access	DC miss	DTLB miss	Misaligns
804a651	mov 0xffffffffc8(%ebp),%esi	135	230	2	84
804a654	movsd 0x24(%edx),%xmm0	1474	2150	36	794
804a659	add \$0x3c,%edx	4047	5983	147	3486
804a65c	mov (%esi,%ecx,4),%eax	1148	2035	40	1029
804a65f	mov 0xfffffffffa8(%ebp),%esi	8810	14053	306	8971
804a662	inc %ecx	5404	7810	160	6559
804a663	cmp %ecx,0xfffffffffb8(%ebp)	474	852	21	199
804a666	mulsd (%eax,%esi,1),%xmm0	6691	11892	270	2500
804a66b	addsd %xmm1,%xmm0	5118	7919	137	2328
804a66f	movapd %xmm0,%xmm1	208	388	5	156
804a673	movsd %xmm0,(%ebx)	3038	4968	63	1703
804a677	jg 804a651 <match+0x2f1>	4081	5724	128	1795

Hot-spot: Compute F2 - y values

```
/* Compute F2 - y values */
```

```
for (tj = 0 ; tj < numf2s ; tj++)
```

```
{
```

```
    Y[tj].y = 0 ;
```

```
    if ( !Y[tj].reset )
```

```
    {
```

```
        for (ti = 0 ; ti < numf1s ; ti++)
```

```
        {
```

```
            Y[tj].y += f1_layer[ti].P * bus[ti][tj] ;
```

```
        }
```

```
    }
```

```
}
```

Long stride through memory



Inefficient cache line access



Inefficient cache line access

```
typedef struct {
```

```
    double *I;
```

```
    double W;
```

```
    double X;
```

```
    double V;
```

```
    double U;
```

```
    double P;
```

```
    double Q;
```

```
    double R;
```

```
} f1_neuron;
```

```
f1_neuron *f1_layer;
```

```
...
```

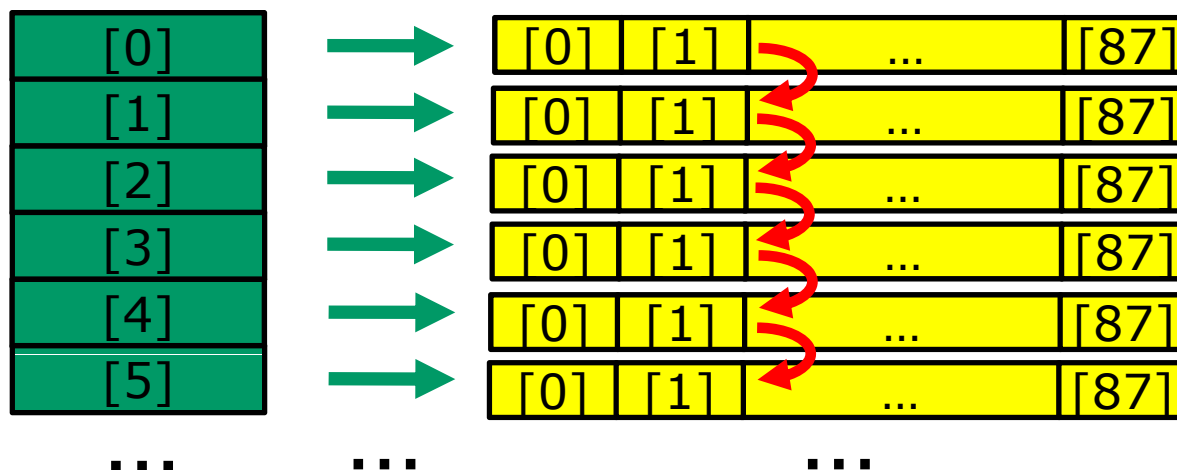
```
f1_layer = (f1_neuron *)malloc(numfls * sizeof (f1_neuron));
```

Aligns to 8-byte boundary.

Accesses only one member from each array element.

- Uses 8 bytes out of 60 bytes.
- Cache line is 64 bytes.

Long stride through memory



```
for (ti = 0 ; ti < numfls ; ti++)
{
    Y[tj].y += fl_layer[ti].P * bus[ti][tj] ;
}
...
bus = (double **)malloc(numfls*sizeof(double *));
...
bus[i] = (double *)malloc(numf2s*sizeof(double));
```



Tuning

Reorganize using struct-of-arrays

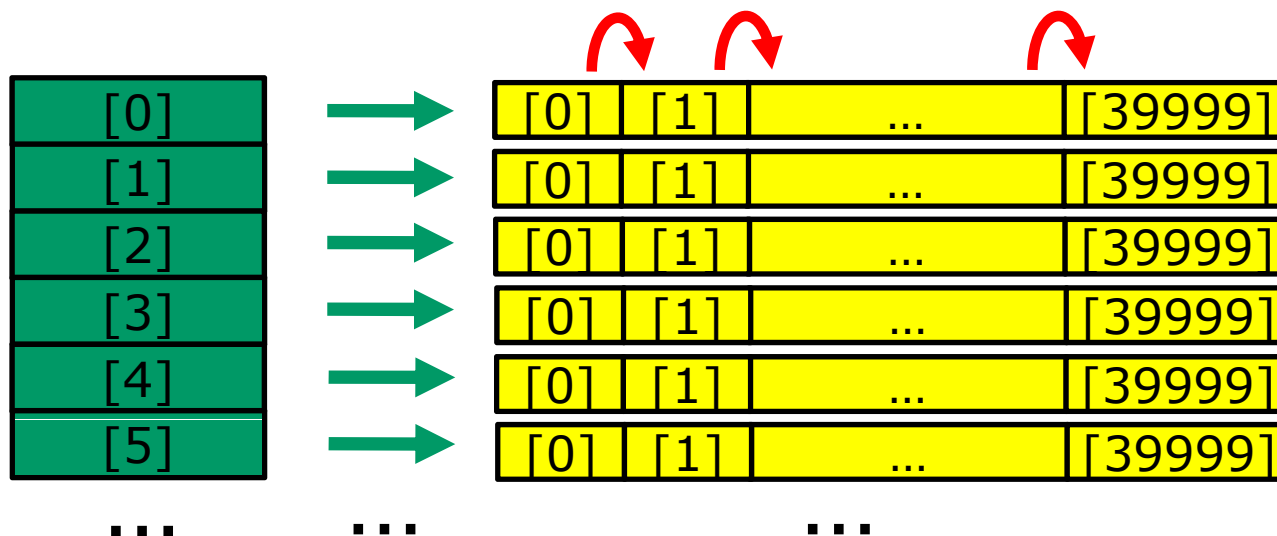
```
typedef struct {
    double **I;
    double *W;
    double *X;
    double *V;
    double *U;
    double *P;
    double *Q;
    double *R;
} f1_neuron;
```

Access each element sequentially.

- Maximize use of cache line.
- Exploit hardware prefetching.

```
f1_neuron f1_layer;
...
f1_layer.I = (double**)malloc(numfls * sizeof(double*)) ;
f1_layer.W = (double*)malloc(numfls * sizeof(double)) ;
...
```

Transpose array and indexing



```
for (ti = 0 ; ti < numfls ; ti++)
{
    Y[tj].y += fl_layer.P[ti] * bus[tj][ti];
}
...
bus = (double **)malloc(numf2s*sizeof(double *)) ;
...
bus[i] = (double *)malloc(numfls*sizeof(double)) ;
```

Measure the example code after tuning

Execution time is drastically reduced for this example.

Memory behavior is improved.

Column “DTLB miss” shows misses in both L1 and L2.

DTLB misses that miss L1 and hit L2 are also reduced, but are not shown below.

Test case	Time	DC access	DC miss	DTLB miss	Misaligns
Baseline	74.920 s	97490	103884	1988	74219
struct-of-arrays	55.725 s	93593	79260	841	22653
Transpose	39.984 s	84165	24826	52	73868
Both	25.456 s	85967	1787	25	22465



Instruction-Based Sampling

Instruction-based sampling

New hardware measurement technique in quad-core AMD Opteron™ and AMD Phenom™ processors.

Designed to isolate performance issues to specific instructions.

- EBP: Events are distributed throughout a code region.
- IBS: Events are associated with exactly the instructions that caused the events.

Captures a rich set of event data in a single test run.

Enables new forms of analysis and optimization.

- “Data-centric” analysis identifying hot data structures.
- Precise mispredict information to guide code straightening.

Instruction-based sampling

IBS selects and monitors specific operations as they are performed by the processor.

Event information — including the instruction pointer value — is reported when an operation completes.

Instruction pointer value allows precise event attribution.

IBS monitors two kinds of operations: fetch and op.

- IBS fetch: Information about instruction fetch.
- IBS op: Information about instruction execution.

Information gathered by IBS

IBS fetch

Fetch address

Completion status

Fetch latency

Instruction cache miss

L1 instruction TLB miss

L2 instruction TLB miss

Translation page size

IBS op

Instruction pointer value

Load/store flags

Operand (data) address

Data cache miss

Data cache miss latency

Misaligned access

L1 data TLB miss

L2 data TLB miss

Translation page size

Branch/return flags

Branch mispredict/taken

Remote/local access

EBP events: Compute F2 – y values

EBP event breakdown for the **optimized** inner loop.
Memory access events are distributed across inner loop.
Cannot conclusively associate events with instructions.

Address	Instruction	DC access	DC miss	DTLB miss	Misaligns
804a880	mov 0xffffffffc0(%ebp),%ebx	558	17		16
804a883	mov %edx,%eax	3012	48	1	603
804a885	movsd 0xfffffffff8(%ebx,%edx,8),%xmm0				
804a88b	mov 0xfffffffffa4(%ebp),%ebx	670	9		38
804a88e	mulsd 0xfffffffff8(%ebx,%edx,8),%xmm0	4082	76	1	2232
804a894	inc %edx	1			
804a895	cmp %edi,%eax	927	29		244
804a897	addsd %xmm1,%xmm0	14738	253	3	11740
804a89b	movapd %xmm0,%xmm1				
804a89f	movsd %xmm0,(%ecx)	1585	46		20
804a8a3	j1 804a880 <match+0x320>	2771	90		1349

IBS events: Compute F2 – y values

IBS op event breakdown for the **optimized** inner loop.

Ops are precisely associated with instructions.

No mispredicted branches (good prediction.)

Address	Instruction	All ops	Load/Store	Branch	Mispredict
804a880	mov 0xffffffffc0(%ebp),%ebx	2031	2031		
804a883	mov %edx,%eax	1994			
804a885	movsd 0xfffffffff8(%ebx,%edx,8),%xmm0	2150	2150		
804a88b	mov 0xffffffffa4(%ebp),%ebx	2450	2450		
804a88e	mulsd 0xfffffffff8(%ebx,%edx,8),%xmm0	2455	2455		
804a894	inc %edx	2385			
804a895	cmp %edi,%eax	8635			
804a897	addsd %xmm1,%xmm0	8703			
804a89b	movapd %xmm0,%xmm1	8947			
804a89f	movsd %xmm0,(%ecx)	3071	3071		
804a8a3	jnl 804a880 <match+0x320>	3364		3364	0

IBS events: Compute F2 – y values

IBS op event breakdown for the **optimized** inner loop.

Events are precisely associated with instructions.

Can identify the instruction with misaligned accesses.

Address	Instruction	Load	Store	DC miss	DTLB miss	Misalign
804a880	mov 0xffffffffc0(%ebp),%ebx	2031		1		
804a883	mov %edx,%eax					
804a885	movsd 0xfffffffff8(%ebx,%edx,8),%xmm0	2150		25		
804a88b	mov 0xffffffffa4(%ebp),%ebx	2450		1		
804a88e	mulsd 0xfffffffff8(%ebx,%edx,8),%xmm0	2455		70	3	
804a894	inc %edx					
804a895	cmp %edi,%eax					
804a897	addsd %xmm1,%xmm0					
804a89b	movapd %xmm0,%xmm1					
804a89f	movsd %xmm0,(%ecx)		3071	1		564
804a8a3	j1 804a880 <match+0x320>					

Misaligns are due to the only store in the loop.

struct alignment

“Note that the alignment of any given struct or union type is required by the ISO C standard to be at least a perfect multiple of the lowest common multiple of the alignments of all of the members of the struct or union in question.”

```
typedef struct {
    double y ;
    int reset ;
} xyz;

xyz *Y;
...
Y = (xyz *)malloc(numf2s*sizeof(xyz));
...
for (ti = 0 ; ti < numf1s ; ti++)
{
    Y[tj].y += f1_layer.P[ti] * bus[tj][ti] ;
}
```


struct alignment

Add alignment attribute to struct definition.
Or, reorganize into struct-of-arrays.

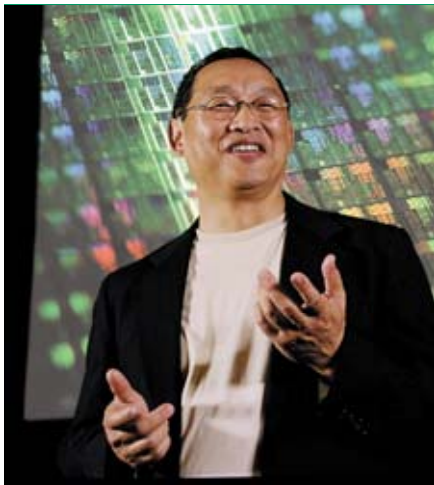
```
typedef struct {
    double y ;
    int reset ;
} __attribute__((aligned(8))) xyz;

xyz *Y;
...
Y = (xyz *)malloc(numf2s*sizeof(xyz));
...
for (ti = 0 ; ti < numf1s ; ti++)
{
    Y[tj].y += f1_layer.P[ti] * bus[tj][ti] ;
}
```

struct alignment

Misaligned accesses were eliminated in this example.
This final change did not produce a speed-up, however.

Function	Optimized case	Aligned case
match	17687	0
train_match	4372	0
g	174	0
weightadj	44	0



Summary

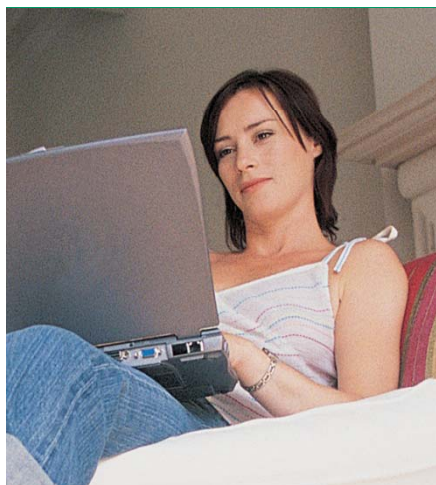
Summary

AMD CodeAnalyst™ software makes it easier for software engineers to analyze performance.

- Time-based profiling to identify hot-spots.
- Event-based profiling and Instruction-Based Sampling to investigate causes for slow-down in a hot-spot.
- Performance is cache/memory access sensitive.

Instruction-Based Sampling allows rich, pinpoint analysis in a single test run.

IBS is available now on quad-core AMD Opteron™ and AMD Phenom™ processors.



On-line resources

How to get AMD CodeAnalyst™ software

AMD CodeAnalyst license is available without charge!

Download it from AMD Developer Central

<http://developer.amd.com/codeanalyst>

RPMs and source tarballs are available:

- V2.7: RHEL 4U4, 4U5, 5U1; Fedora 7, 8; SLES 10 SP1; OpenSUSE 10.2, 10.3
- V2.6: RHEL 5; Fedora 6; SLES 10
- V2.5: RHEL 4U4; SLES 9 SP3
- See the web site for the current support matrix.

Also available for Microsoft Windows®

E-mail: support.codeanalyst@amd.com

On-line resources

Available at AMD Developer Central

“An introduction to analysis and optimization with AMD CodeAnalyst”

<http://developer.amd.com/documentation/articles/Pages/111820052.aspx>

“Basic performance measurements for AMD Athlon™ 64 and AMD Opteron™ Processors”

<http://developer.amd.com/documentation/articles/Pages/1212200690.aspx>

“Instruction-Based Sampling: A new performance analysis technique for AMD Family 10h processors”

http://developer.amd.com/assets/AMD_IBS_paper_EN.pdf

On-line resources

Available at AMD Developer Central

“Software Optimization Guide for AMD Family 10h Processors”

<http://developer.amd.com/documentation/guides/Pages/default.aspx>

“BIOS and Kernel Developer’s Guide for AMD Family 10h Processors” or “BKDG” for descriptions of hardware performance events

<http://developer.amd.com/documentation/guides/Pages/default.aspx>

Other AMD CPU tools including performance libraries

<http://developer.amd.com/CPU/Pages/default.aspx>

Trademark Attribution

AMD, the AMD Arrow logo, AMD Opteron, AMD Phenom, AMD Athlon and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Linux is a registered trademark of Linus Torvalds. Other names used in this presentation are for identification purposes only and may be trademarks of their respective owners.

©2008 Advanced Micro Devices, Inc. All rights reserved.