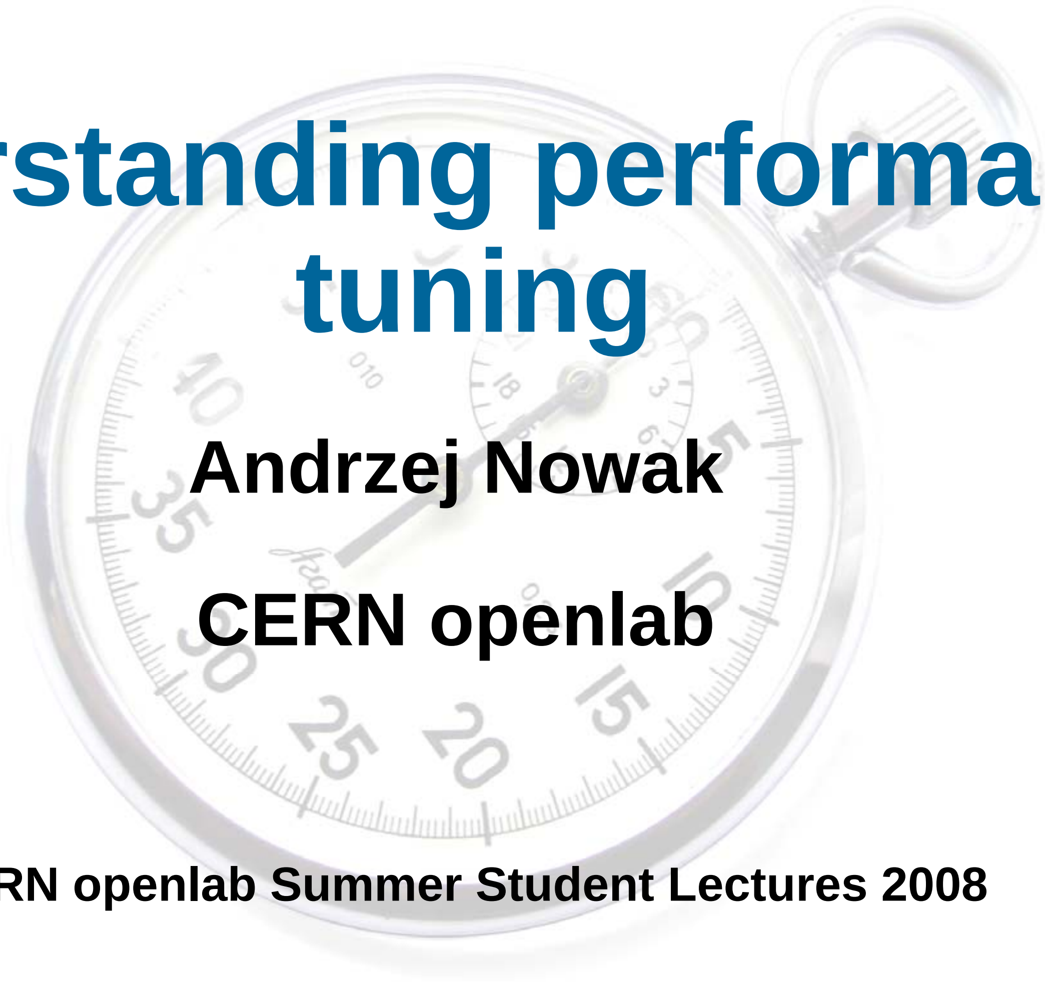
Computer Architecture and Performance Tuning

Understanding performance tuning

Andrzej Nowak

CERN openlab

CERN openlab Summer Student Lectures 2008



Contents

1. **Software performance tuning in general**
2. **Drilling down on performance figures**
3. **General tips**

- **In this talk, we focus on x86_64 processors (Intel Core and friends, AMD Athlon/Barcelona, etc)**

Performance tuning in general

Improving application performance

- **Question #1 – “Why is it SOOOOOO SLOW?”**
- **Exchanging hardware**
  - Removing common bottlenecks
  - New CPU, new hard drive, more memory
  - New, new, new…
- **Replacing whole software components**
  - Replacing shared or external libraries
- **Improving existing code**
- **Performance monitoring will give you the answer**
  - It allows you to find the things you could change in your setup to improve performance

Performance tuning

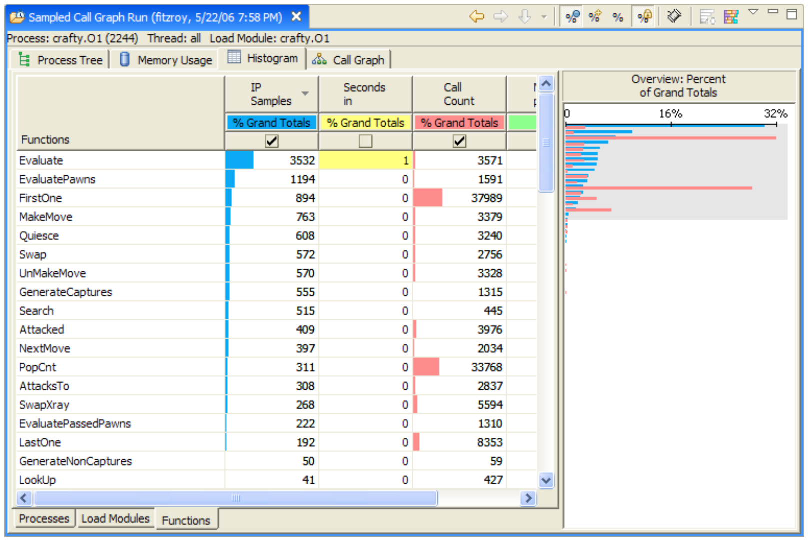
- **Why tune performance?**
  - To get more speed and/or throughput…
  - …or to just keep up with the hardware or previous performance figures
  - Processor clock frequencies don't go up anymore! No free meals since the millennium
- **Who needs performance tuning?**
- **Who can do performance tuning?**
  - Some bottlenecks are really easy to find…
  - … but performance tuning can be VERY tricky
- **Performance tuning is a lot like tuning a car… but you can do well with only one wrench and you don't need all those expensive parts**

Performance tuning levels

- **Source code**
  - Function calls
    - Excessive calls of a function or a group of functions
    - Blocking (i.e. I/O)
  - Loops within your program
    - Iterating over sparse/long structures
  - General characteristics of your program
    - Excessive memory allocations and copying, excessive calculations, checks, malformed conditions, etc.
- **Operating system**
  - Running daemons, limits, co-existing processes, I/O
- **Hardware counter level**
  - Can span all of the above… if well implemented
- **Hardware**
  - Buy new, better hardware… not always possible, even if the money is there

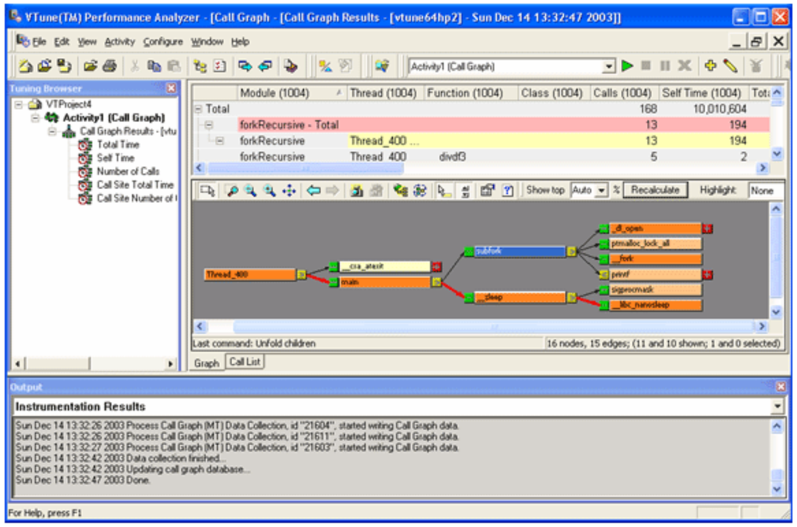
Popular performance tuning software (1)

- **Intel products:**
  - VTune, PTU – very powerful
  - Thread Checker, Thread Profiler – for multithreading
  - Expensive, VTune in Linux requires a precompiled kernel module
- **HP Caliper**


Sampled Call Graph Run (fitzroy, 5/22/06 7:58 PM)
Process Tree
Memory Usage
Histogram
Call Graph
Overview: Percent of Grand Totals
0
16%
32%
Processes
Load Modules
Functions

Process: crafty.O1 (2244) Thread: all Load Module: crafty.O1

Functions	IP Samples (% Grand Totals ✓)	Seconds in (% Grand Totals)	Call Count (% Grand Totals ✓)
Evaluate	3532	1	3571
EvaluatePawns	1194	0	1591
FirstOne	894	0	37989
MakeMove	763	0	3379
Quiesce	608	0	3240
Swap	572	0	2756
UnMakeMove	570	0	3328
GenerateCaptures	555	0	1315
Search	515	0	445
Attacked	409	0	3976
NextMove	397	0	2034
PopCnt	311	0	33768
AttacksTo	308	0	2837
SwapXray	268	0	5594
EvaluatePassedPawns	222	0	1310
LastOne	192	0	8353
GenerateNonCaptures	50	0	59
LookUp	41	0	427



	Module (1004)	Thread (1004)	Function (1004)	Class (1004)	Calls (1004)	Self Time (1004)
Total					168	10,010,604
	forkRecursive - Total				13	194
	forkRecursive	Thread_400 ...			13	194
	forkRecursive	Thread_400	divdf3		5	2

Last command: Unfold children — 16 nodes, 15 edges; (11 and 10 shown; 1 and 0 selected)

```
Sun Dec 14 13:32:26 2003 Process Call Graph (MT) Data Collection, id "21604", started writing Call Graph data.
Sun Dec 14 13:32:26 2003 Process Call Graph (MT) Data Collection, id "21611", started writing Call Graph data.
Sun Dec 14 13:32:27 2003 Process Call Graph (MT) Data Collection, id "21603", started writing Call Graph data.
Sun Dec 14 13:32:42 2003 Data collection finished...
Sun Dec 14 13:32:42 2003 Updating call graph database...
Sun Dec 14 13:32:47 2003 Done.
```

Popular performance tuning software (2)

- **gprof**
  - Flat profiles, call lists
  - Recompilation needed
- **oprofile**
  - Flat profiles
  - Kernel driver needed
- **PIN, Valgrind**
  - Instrumentation / Synthetic software CPU
  - Simulate such characteristics as cache misses and branch mispredictions, memory space usage, function call relationships
- **pfmon / perfmon2**
  - Low level access to counters
  - No recompilation needed
  - Kernel patch needed today, but will be a part of the standard Linux kernel

Performance monitoring in hardware

- **Most modern CPUs are able to provide real-time statistics concerning executed instructions…**
- **…via a Performance Monitoring Unit (PMU)**
- **The PMU is spying in real time on your application! (and everything else that goes through the CPU)**
- **Limited number of "sentries" (counters) available, but they are versatile**
- **Recorded occurrences are called events**
- **On the Intel Core microarchitecture:**
  - 2 universal counters: #0, #1
  - 3 specialized counters: #16, #17, #18

Common performance figures

And how to interpret them

Basic information about your program

- **The amount of:**
  - instructions executed
  - processor cycles spent on the program
  - transactions on the bus
- **The amount/percentage of:**
  - memory loads and stores
  - floating point operations
  - vector operations (SIMD)
  - branch instructions
  - cache misses

Advanced information about your program

- **The amount and type of:**
  - micro-ops executed
  - SIMD instructions executed
  - resource stalls within the CPU
- **Cache access characteristics**
  - A rich set on Intel Core CPUs
  - Demand
  - Requests (missed / hit / total / exclusive or shared / store or read)
  - Lines modified / evicted / prefetched

Derived events

- **Too much information available?**
- **Low level and fine grained events can be combined to produce ratios (so called "derived events")**
- **Extensive information:**
  - Intel Manual 248966-016 "Intel 64 and IA-32 Architectures Optimization Reference Manual"
  - AMD CPU-specific manuals, i.e. #32559 "BIOS and Kernel Developer's Guide for AMD NPT Family 0Fh Processors"

A word for the future

Mapping performance monitoring data onto your source code and environment requires care and experience

The CPI figure and its meaning

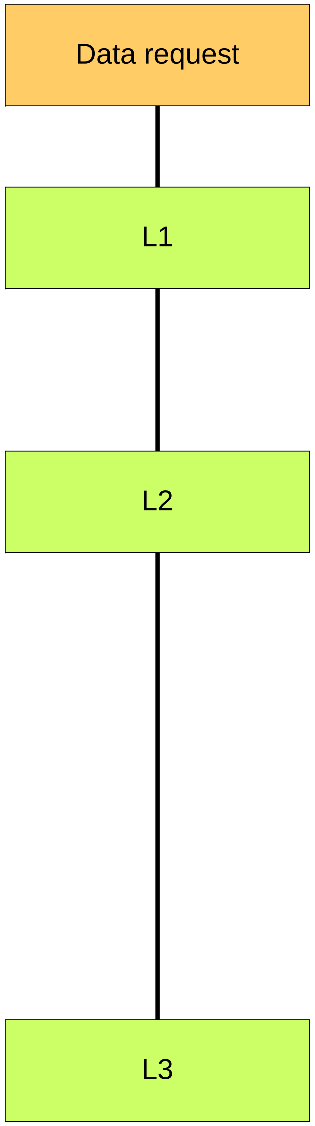
- **CPI – cycles per instruction**
  - Thanks to multiple execution ports (superscalar architecture), more than one instruction can be executed per cycle
  - In Intel Core 2 CPUs, CPI can go as low as 0.25 = 4 instructions per cycle
  - CPI above 1.0 is not impressive
- **The ratio of the number of CPU cycles spent on a program to the number of program instructions retired by the CPU**
- **CYCLES / INSTRUCTIONS**
- **This figure illustrates the CPU usage efficiency, but, like all ratios, can be tricky to interpret**

Cache misses

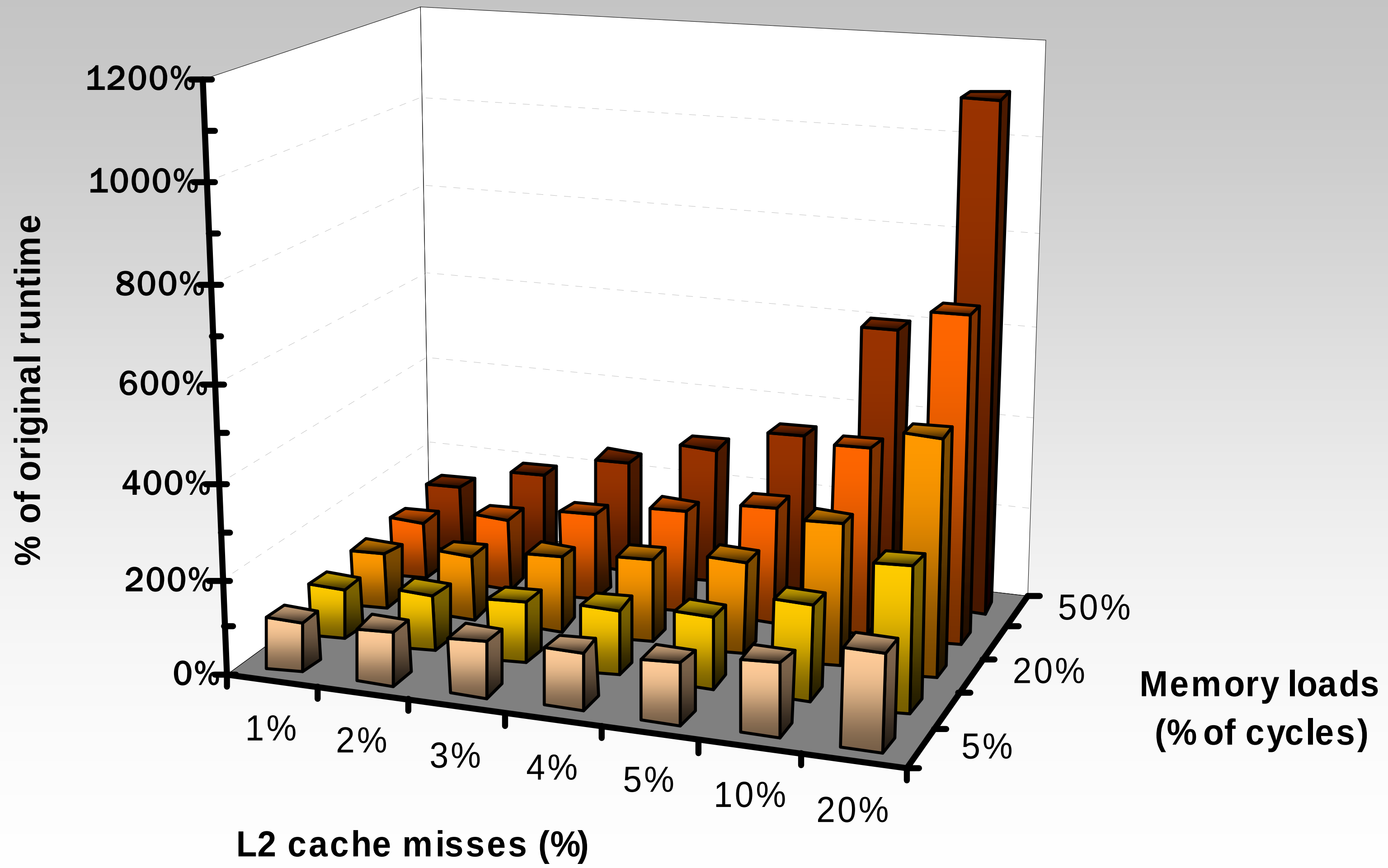
- **If the requested item is not in the polled cache, the next level has to be consulted (cache miss)**
- **Significant impact on performance**
- **Formula:**

  LAST LEVEL CACHE MISSES / LAST LEVEL CACHE REFERENCES

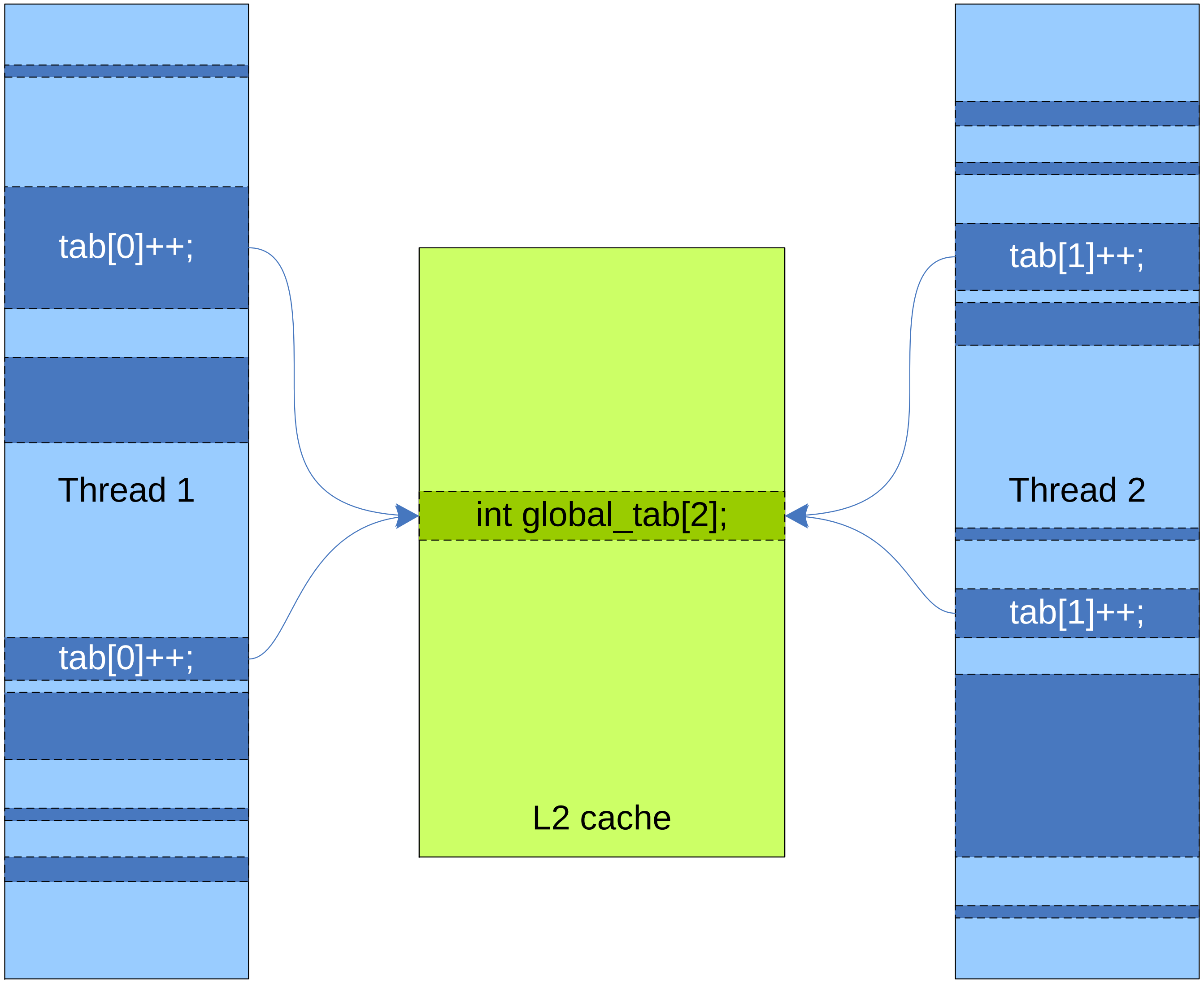
- **Tips:**
  - A L2 cache hit ratio below 95% is considered to be catastrophic! (=5% miss)
  - Usually the figure should be above 99%
  - The overall cache miss rate might be low (misses / total instructions), but the resource stalls figure might be high; always check the cache miss percentage



L2 Cache miss impact (simplified)



False sharing



Branch prediction

- Branch prediction is a process inside the CPU which determines whether a conditional branch in the program is anticipated by the hardware to be taken or not
- Typically: prediction based on history
- The effectiveness of this hardware mechanism heavily depends on the way the software is written
- The penalty for a mispredicted branch is usually severe (the pipelines inside the CPU get flushed and execution stalls for a while)

Branch prediction ratios

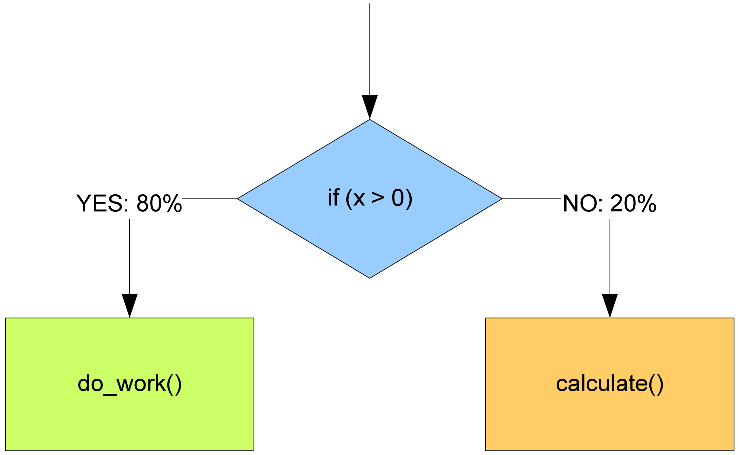
- **The percentage of branch instructions**

  BRANCH INSTRUCTIONS / ALL INSTRUCTIONS

- **The percentage of mispredicted branches**

  MISPREDICTED BRANCHES / BRANCH INSTRUCTIONS

  - The number of correctly predicted branches is typically very high (80%+), up to 99%



Floating point operations

- **Often a significant portion of work of an application**
- **May be accelerated using SSE (SIMD)**
- **Related events on the Intel Core microarchitecture:**
  - “traditional” x87 FP ops
  - Packed/Scalar single computational SIMD
  - Packed/Scalar double computational SIMD
  - SIMD micro-ops
- **Non computational SIMD instructions can also be counted**

Relating to code (1)

- **CPI problems**
  - Doing too many operations?
  - Large latency instructions in the code?
  - Using vector instructions?
- **Cache misses, false sharing**
  - Memory access characteristics
  - Data structures and their layout
  - Does your program fit in the cache?
  - Help the hardware prefetcher!

Relating to code (2)

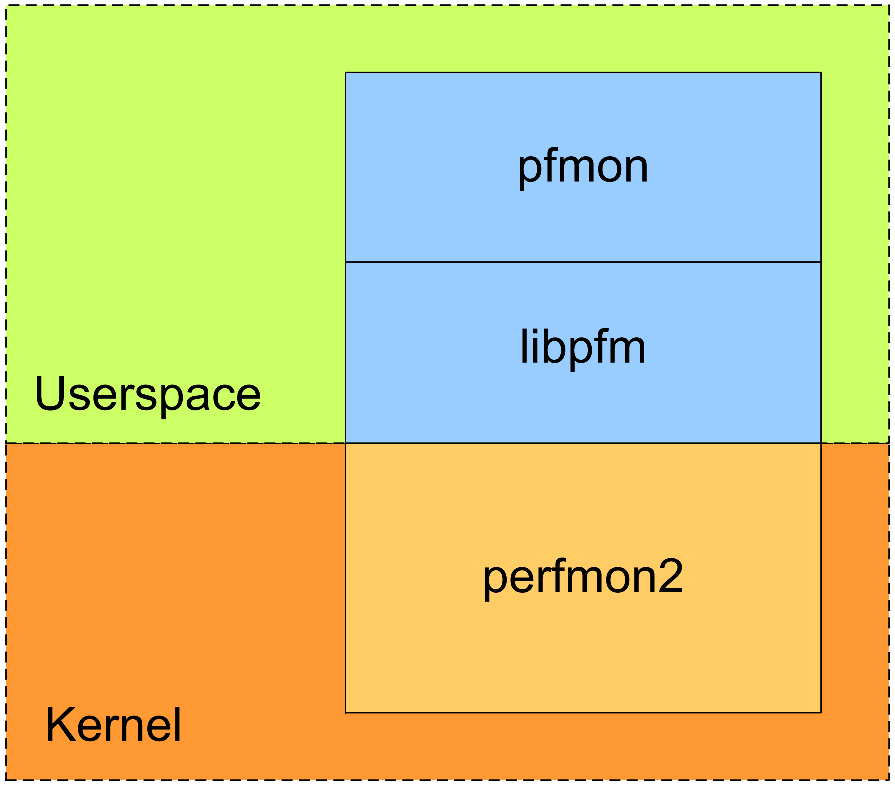
- **Many mispredicted branches**
  - Is there a way to restructure the code?
  - Is there a way to make the “ifs” more predictable?
  - Rearranging conditions and loops
  - Too many jumps / function calls?
- **Excessive floating point operations**
  - Does everything need to be calculated?
  - Could some results be reused?

Perfmon2 & pfmon

A real-world performance monitoring framework example

Perfmon2 architecture

- An example of a robust performance monitoring framework for Linux
- **perfmon2 – kernel part**
- **libpfm – userspace interface for perfmon**
- **pfmon – “example” userspace application, perfmon2 client**



Perfmon2

- **Resides in the kernel**
  - Currently available as a kernel patch
- **Being merged into the Linux kernel mainline**
  - Perfmon2 will eventually be available by default in the Linux kernel (after 2.6.27)
- **Very basic functionality, keeping the kernel patch slim**
- **Support for numerous architectures:**

  x86, x86-64, ia64, powerpc, cell / ps3, mips, sparc
- **Supported in Red Hat since a long time (which is the base for Scientific Linux)**

Pfmon overview

- **Console based interface to libpfm/perfmon2**
- **Provides convenient access to performance counters**
- **Wide range of functionality:**
  - Counting events
  - Sampling in regular intervals
  - Flat profile
  - System wide mode
  - Triggers
  - Different data readout “plug-ins” (modules) available

Events

- **Many events in the CPU can be monitored**
  - A comprehensive list is dependent on the CPU and can be extracted from the manufacturer's manuals
- **On some CPUs (i.e. Intel Core), some events have bit-masks which limit their range, called "unit masks" or "umasks"**
- **In pfmon:**
  - Getting a list of supported events: **`pfmon –l`**
  - Getting information about an event: **`pfmon –i`** ***eventname***

Basic modes

- **Counting**
  - Example: How many instructions did my application execute?
  - Example: How many times did my application have to stop and wait for data from the memory?
- **Sampling**
  - Reporting results in “regular” intervals
  - Example: every 100’000 cycles record the number of SSE operations since the last sample
- **Profiling**
  - Example: how many cycles are spent in which function?
  - Example: how many cache misses occur in which function?
  - Example: which code address is the one most frequently visited? (looking for hotspots)

Q & A

This research project has been supported by a Marie Curie Early Stage Research Training Fellowship of the European Community's Sixth Framework Programme under contract number (MEST-CT-2004-504054)