

# A Tool Suite for Simulation Based Analysis of Memory Access Behavior

ICCS 2004, 9 June 2004  
Krakow, Poland

Josef Weidendorfer\*, Markus Kowarschik\*\*, Carsten Trinitis\*

\* Technische Universität München,  
Germany



\*\* Universität Erlangen-Nürnberg,  
Germany



# Overview

- Introduction
- Scope, Motivation
- Simulation Based Profiling
- Techniques, Results
- Visualization
- Requirements, Implementation
- Future Work

# Introduction

- Scope of this Work: Project DiME
  - Cache-Optimized Library for Sequential PDE Solvers (Multigrid)
  - Best Parameters for Standard Techniques (Blocking/Padding) searched at Compile Time for given Platform
  - Benefit on Alpha around +100%, on Intel P4 +20% (far from Peak Perf.)

# Introduction

- Scope of this Work: Project DiME
- Current Approaches
  - Try to understand Memory Access Behavior on recent HW (OOO, Speculation, HW Pf.)
  - Separate Effects (using Simulation):
    - (1) Simulation with Simple Cache Model
    - (2) + HW/SW Prefetching
    - (3) Real Hardware
  - New Techniques (e.g. aware of HW Pf.)

# Introduction

- Scope of this Work: Project DiME
- Current Approaches
- Here:
  - Presentation of Tools developed so far
  - Focus on Analysis of Sequential Code (Profiling, no Tracing)
  - General Usability

# Simulation Based Profiling

- Trapping Memory Access Events
  - Via Runtime Instrumentation (Valgrind/Linux on x86)
  - No recompilation needed
  - „Instrument on demand“ (JIT like)
  - Switchable (at function enter / interactive)
    - Delivery Off (Slowdown factor 3 on HPC-Code)
    - Delivery On (Slowdown factor 30 – 100)

# Simulation Based Profiling

- Trapping Memory Access Events
- Cache Simulator
  - Read/Write Accesses, Hits/Misses on L1/L2
  - Misses because of HW/SW Prefetching

# Simulation Based Profiling

- Trapping Memory Access Events
- Cache Simulator
- Event Collection/Processing (on the fly)
  - Dynamic Context (Caller Chains) Call Graph
  - Cache Events attributed to Context, Thread



# Simulation Based Profiling - Results

Amount of Profiling Data can get Huge !

Command	Call chain length limit $n_{max}$					
	0	1	2	5	10	20
bzip2 libm.so.6 Nodes	408	850	1 004	1 329	1 332	1 132
(Compressor) Arcs	538	688	861	1 113	1 113	1 113
cc1 ct_main-i.c Nodes	1 519	5 157	8 352	22 060	41 164	44 899
(C compiler) Arcs	6 741	10 881	15 905	34 282	52 191	54 722
konqueror Nodes	21 500	55 829	91 713	251 449	420 871	507 507
(KDE Browser) Arcs	51 052	90 629	147 958	315 838	470 032	544 487

# Simulation Based Profiling - Results

- Multigrid Code:
- „Standard“ Strategies work  
(Blocking on 2 Iterations: Half L2 Misses)
  - Cache Events comparable to Reality

	Simulation			Real Measurement	
	Instr. exec.	L2 Misses	Runtime	Instr. retired	L2 Lines In Runtime
Standard	11 879 M	751 421 K	1 865 s	11 879 M	777 131 K
Optimized	11 666 M	361 336 K	1 798 s	11 666 M	383 609 K

# Visualization - Requirements

- GUI enabling fast Browsing
- Zoomability from Overview to Details
- Visualizations focusing on Performance Problems
- Language/Tool Independence
- Extensibility  
(Input Data/Visualization Types)

# Visualization - KCachegrind

- „Fast“ Browsing: Currently C++/QT
- Zoomability: Library/Function/Line/Instruction
- Visualizations:
  - Call Graph, Tree Map, Annotations
- Independence:
  - Use Debug Info + Standard Tools (Disass.)
  - Tailored to Simulation, but Converters exist for Data from Statistic Sampling (OProfile, PFMon)
- Extensibility: Component Architecture

# Visualization - Screenshot

The screenshot displays a performance analysis tool with several panels:

- Event Type Panel:** Shows cache miss statistics for L1 and L2 instructions.
 

Event Type	Incl.	Self	Short	Formula
L1 Instr. Fetch Miss	49.26	1.83	I1mr	
L1 Data Read Miss	100.00	0.00	D1mr	
L1 Data Write Miss	99.99	0.00	D1mw	
L2 Instr. Fetch Miss	49.20	1.81	I2mr	
L2 Data Read Miss	100.00	0.00	D2mr	
L2 Data Write Miss	99.99	0.00	D2mw	
L1 Miss Sum	100.00	0.00	L1m = I1mr + D1mr + D1mw	
L2 Miss Sum	100.00	0.00	L2m = I2mr + D2mr + D2mw	
Cycle Estimation	100.00	0.00	CEst = Ir + 10 L1m + 100 L2m	
- Source ('mg.f') Panel:** Shows C code snippets.
 

```

      378 C Get down
      379 0.00 do k= 1,1,-1
      380 if((k.EQ.!).OR.(1.EQ.1).OR.(0.EQ.1)) then
      381 C for smoothing function selection see config.h
      382 0.00 call rb4w(k,2,wsp(uoff(k)),wsp(fcoff(k)),
      383 35.21 150 calls to 'rb4w_' (mg)
      384 & (xdim(k)-1),(ydim(k)-1),(zdim(k)-1) )
      385 else
      386 C for other grids use std (only for only_u runs!)
      call rb_wf(k,2,wsp(uoff(k)),wsp(fcoff(k)).
      
```
- Call Graph Panel:** A flowchart showing the execution flow from `__libc_start_main` to `main`, `MAIN_`, `vcycle_`, `rb4w_`, and `restr_`.
 

```

      graph TD
      __libc_start_main --> main
      main --> MAIN_
      MAIN_ --> vcycle_
      vcycle_ --> restr_
      vcycle_ --> rb4w_
      
```
- Assembly Panel:** Shows assembly instructions with cycle estimates.
 

#	CEst	Hex	Assembler
804 AC3D	0.00 56		push %esi
804 AC3E	0.00 57		push %edi
804 AC3F	0.00 50		push %eax
804 AC40	0.00 68 4c 20 0b 08		push \$0x80b204c
804 AC45	0.00 52		push %edx
804 AC46	0.00 e8 8b b2 00 00		call 8055ed6 <rb4w_
804 AC4B	0.00 8d 44 24 1c		lea 0x1c(%esp),%eax
804 AC4F	0.00 50		push %eax

# Future Work

- Simulation
  - Locality Metrics (temporal, spatial, reuse dist.)
  - Relation to Data Structures, Compression Techniques (e.g. Address differences)
- According Visualizations

**Thanks for Listening!**  
**Any Questions?**