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	40.4 s
Optimized	11 666 M	361 336 K	1 798 s	11 666 M	383 609 K	27.1 s

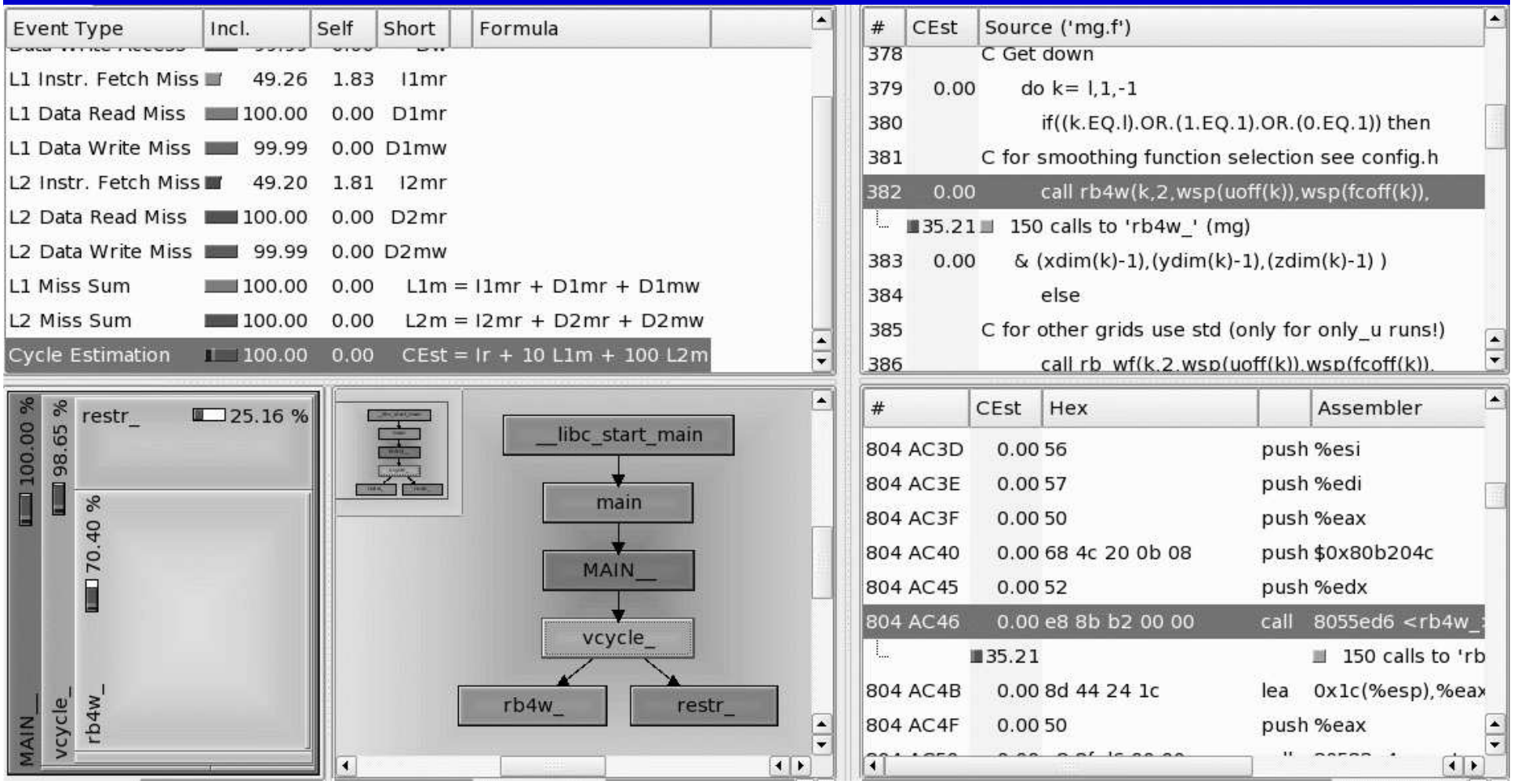
Visualization - Requirements

- 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)
- Extensibilty: Component Architecture

Visualization - Screenshot



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?