

# Analysis of Memory Access Bottlenecks

ORNL Young Investigators Symposium 2008  
Oak Ridge, October 13-15, 2008

Josef Weidendorfer

Lehrstuhl für Rechnertechnik und Rechnerorganisation  
Institut für Informatik, Technische Universität München

## Memory Access Bottlenecks

Main problem: “Memory wall”

= Increasing performance gap main memory vs. processor

Getting worse with:

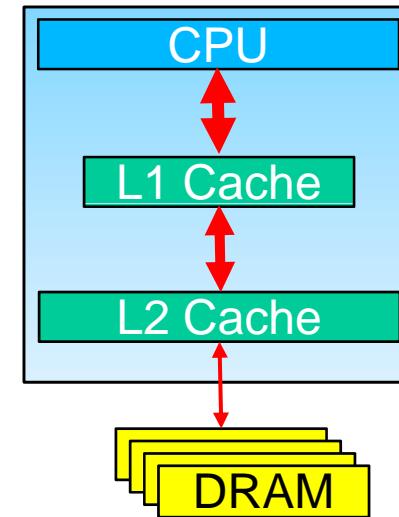
- multicore processors (need to share memory connection)
- trend to NUMA even on small systems (latency)

→ Applications slow down because of memory access issues  
(even old code which ran well in the past)

# Memory Access Bottlenecks

## Solution: Caches

- exploit **locality** of memory accesses (temporal / spatial)
- lowers access latency by putting data copies into fast memory

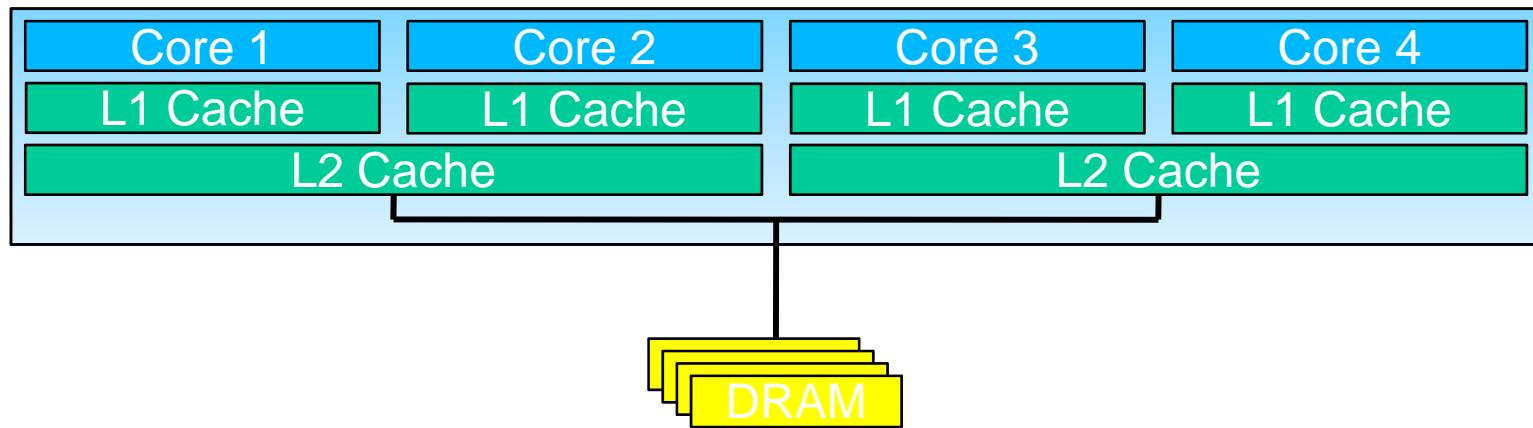


„Bad memory access behavior“ = poor exploitation of caches

# How to get rid of Memory Access Bottlenecks

## Optimization strategies

- improve temporal locality by reordering accesses
- improve spatial locality by changing data layout
- prefetch data needed in the future
- exploit shared caches in Multicore



## How to get rid of Memory Access Bottlenecks

Nice. But how?

Ideal world: automatically avoided... if not:

Key questions (when looking at a given code):

- is it worth the effort?
- which source lines expose bad access locality?
- which data structures need to be layout in a better way?
- when is prefetching useful?
- how do threads share data (to find best mapping to cores)?

→ Need for performance analysis tools

## Performance Analysis Tools

- Hardware performance counters (cache misses ...)
  - read counter values / sample code position at overflows
  - raw hardware events are not always enough
- Another solution: Architecture simulation
  - measurement method does not influence results
  - simple cache model
  - execution-driven using runtime instrumentation

## Performance Analysis Tools

Is it worth?

Suppose only cache hits...

Bad access locality?

Stack reuse distance  $r(t)$

*“Number of memory cells accessed between current and previous access of same cell”*

Example:  $1_1 2_2 3_3 2_4 3_5 2_6 3_7 1_8$

$$r(8)=2$$



## Performance Analysis Tools

Is it worth?

Suppose only cache hits...

Bad access locality?

Stack reuse distance  $r(t)$

Bad data layout?

Cacheline usage

Example:

- for one cacheline load:  
“60 of 64 bytes not used before eviction”
- for whole program:  
“30% of data loaded in cache not used”

## Performance Analysis Tools

Is it worth?

Suppose only cache hits...

Bad access locality?

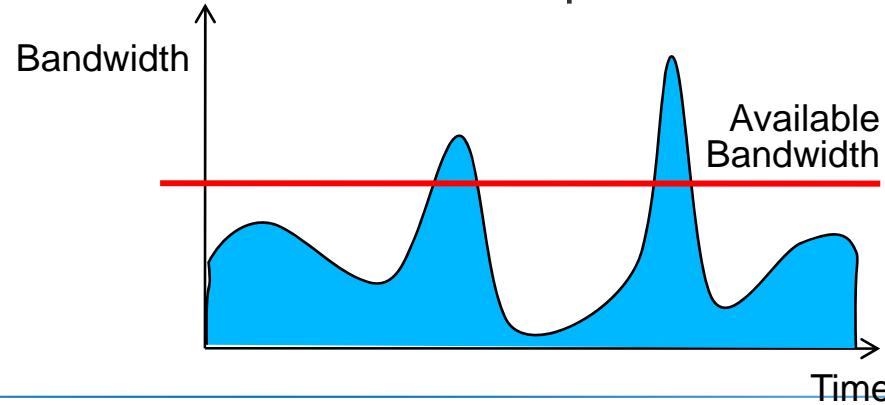
Stack reuse distance  $r(t)$

Bad data layout?

Cacheline usage

Prefetching useful?

Load/Store bandwidth requirement



## Performance Analysis Tools

Is it worth?

Suppose only cache hits...

Bad access locality?

Stack reuse distance  $r(t)$

Bad data layout?

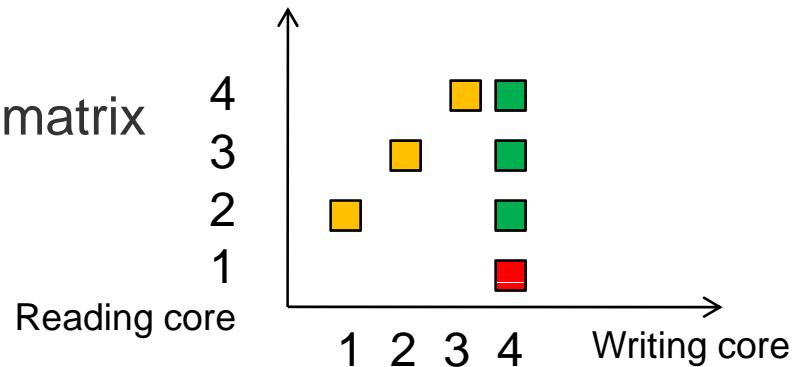
Cacheline usage

Prefetching useful?

Load/Store bandwidth requirement

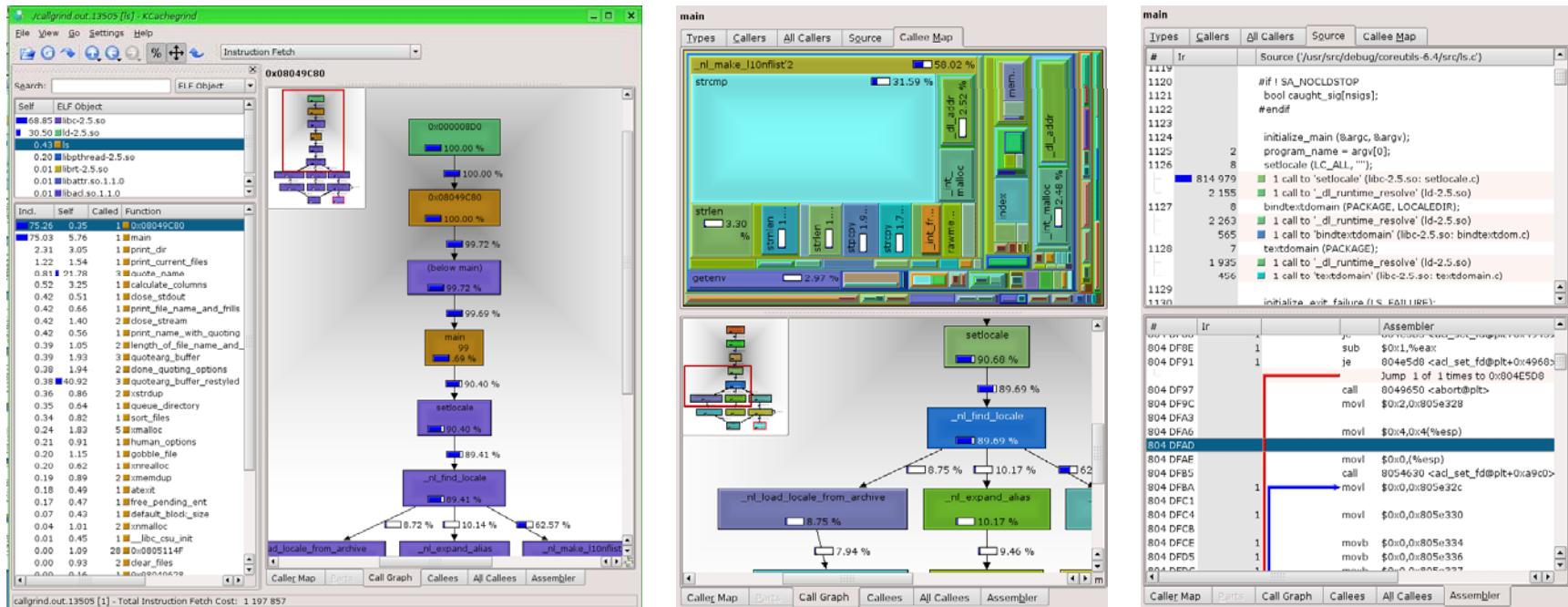
Data sharing?

Communication matrix  
(heat map)



# Our Tool Suite: Callgrind / KCachegrind

Open source, based on Valgrind, runs on Linux



## Outlook

Simulation with less slowdown

- simulation only in representative sample intervals

Integration with AutoPin

- suggestion of good pinning candidates

# Questions?