Scientific approaches: NUMA Profilers/analyzing runtime behavior

Non-Uniform Memory Access (NUMA) Seminar

Malte Swart

Hasso-Plattner-Institut

10 December 2014

Motivation

Problem

We have our new powerful NUMA system.
But our application does not scale as it does on UMA systems.
What can we do?

Motivation (2)

What can we do?

Upgrade the kernel: We have already a current kernel - so no automatic improvement by new scheduling techniques . . .

Look at the source code: We did not write the application, so no real change for improvement there

Using performance counter: as we see in the last presentation

Analyze our program with profilers: Let's do it . . .

What is achievable?

- We concentrate on remote memory accesses, local caches mostly irrelevant in comparison / not NUMA specific
- Next: identify common problems that we can optimize and need to identify

Remote usage after allocation

Issue

Data is create on one NUMA node, but only used on another

Solutions

- Create data directly on other node
- Copy data on first access (if copying is amortized)
- Migration thread to node with data

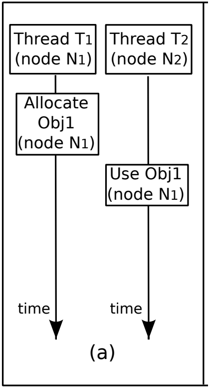

Figure : Remote usage after allocation

Alternate remote accesses to an object

Issue

Data is read by multiple NUMA nodes, but only from one at a time (concurrent but not parallel)

Possible Solutions

- Pin threads to NUMA node with their data
- Migrate threads over time to their data

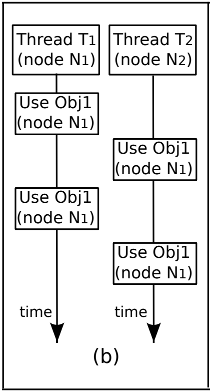

Figure : Alternate remote accesses to an object

Parallel remote accesses to an object

Issue

Parallel access by multiple NUMA nodes

Solution

- Duplicate data, if not or rarely changed (more memory needed)
- Move on thread to the other node (might result in load imbalance)

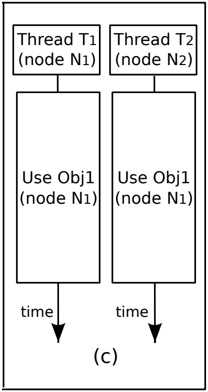

Figure : Concurrent remote accesses to an object

Existing Profiles

Two common types of profiles (example after [1]):

1. instruction/code-orientated profiles (line 4: 100% latency)
2. data-oriented profiles (Array A: line 4: 1% latency; Array B: line 4 - 10% latency, Array C: line 4 - 89% latency)

```
for (int i=0; i < n; i++) {
  for (int j=0; j < n; j++) {
     for (int k=0; k < n; k++) {
        A[i, j, k] = A[i, j, k] + B[j, i, k] + C[k, j, i]
     }
  }
}
```

Traditional profiles concentrate on cache optimization and code hot spots

Existing Profiles (2)

Problem

Information about code or data itself is less useful
UMA profilers not very helpful for NUMA issues

Challenge

Thread on Node x accessed data from Node y
$\rightarrow$ Profilers with more information are needed

SNPERF - a ccNUMA Profiling Tool

- One of the earliest NUMA profiler (Developed around 2001)
- Designed for Origin2000 systems
- Basic on simple performance counters to measure memory bandwidth saturation

SNPERF Examples (1)

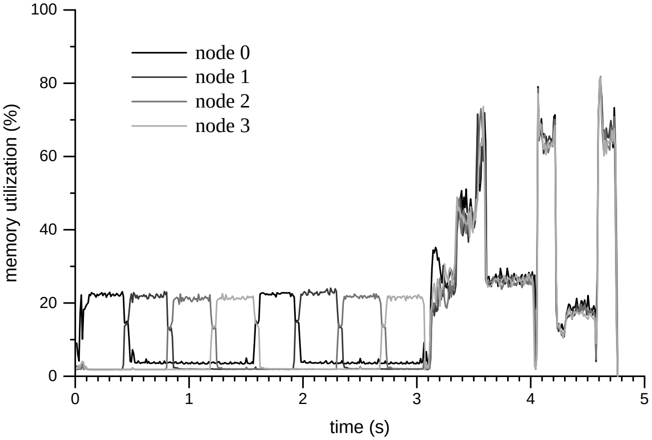

Figure : FFT memory utilization profile on four nodes

SNPERF Examples (2)

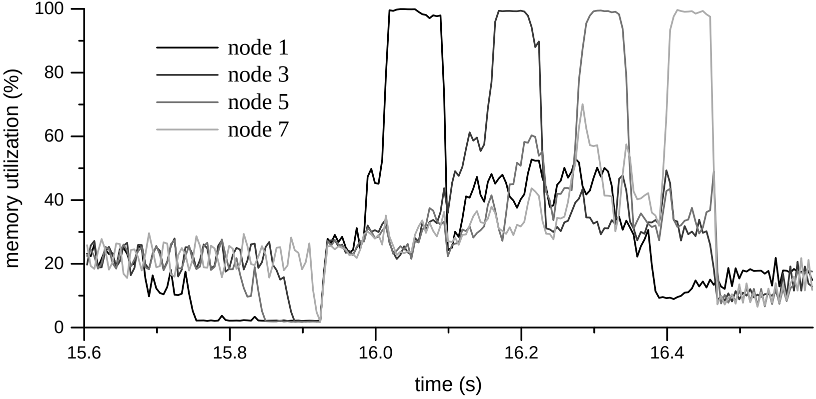

Figure : Unoptimized FFT matrix transpose without staggering

SNPERF Examples (3)

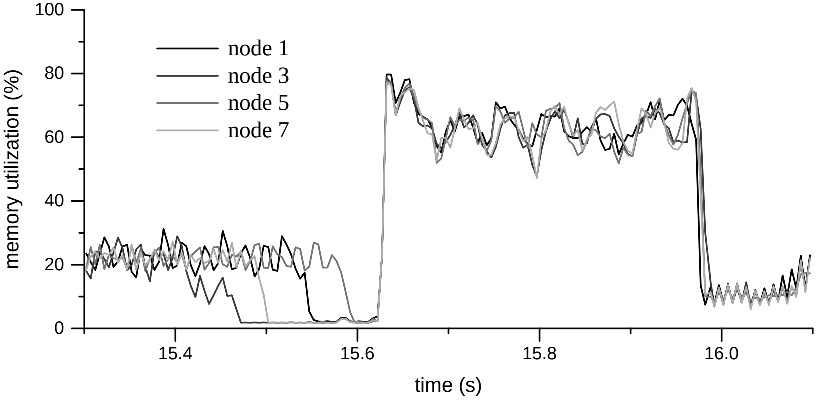

Figure : FFT matrix transpose with optimized staggering

SNPERF Examples (4)

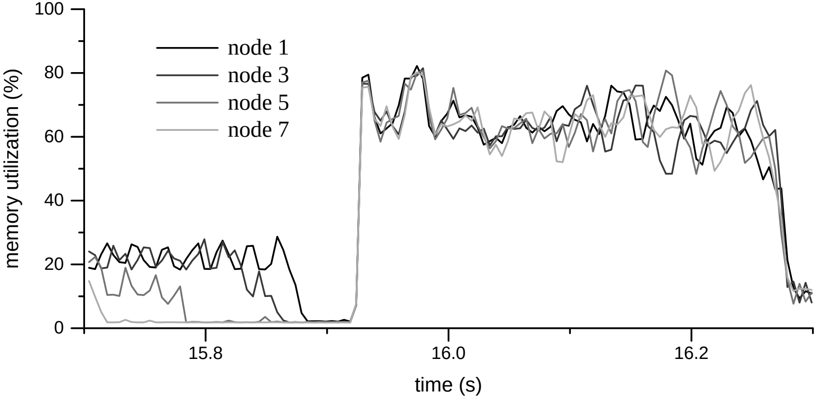

Figure : FFT matrix transpose with optimized staggering

NumaTOP

Question

Does we have a NUMA problem (high remote memory access)?
And no poorly scaling application

NumaTOP

- Live ranking between different running tasks
- Measures local/remote memory access for different processes / nodes
- Some special view about stats of NUMA nodes or memory ranges

NumaTOP Demo

Monitoring 348 processes and 397 threads (interval: 5.0s)

PID	PROC	RMA(K)	LMA(K)	RMA/LMA	CPI	*CPU%
52773	stream-gcc	19092.4	506790.6	0.0	4.16	99.2
52753	numatop	12.4	98.5	0.1	1.53	0.1
143	kworker/1:0	23.9	76.0	0.3	4.52	0.0
336	kworker/16:	17.4	72.1	0.2	5.25	0.0

Figure : Example output of NumaTOP

- RMA(K): remote (non-local NUMA node) memory accesses (in 1000)
- LMA(K): local memory accesses (in 1000)
- RMA/LMA: remote memory percentage - should be low
- CPI: CPU cycles per instruction

vTune

- Specialized profiler from Intel
- Based on performance counters
- But more traditional profiler (cache misses, % operation stalled)

"If [remote memory] percentage is significant (>20%) , consider strategies for improving NUMA access : use a NUMA-aware memory allocator, privatize variables. System tuning : ensure memory is balanced across nodes." [2]

Problem

Generic advices, no hints what to do exactly with a given problem.

MemProf

Challange

Combine remote memory access with detailed information about allocation and object properties

→ Generate flow graphs for objects and threads

1. Execute program and dump information about object and thread lifecycle and memory accesses
2. Generate flow graphs (offline - after execution)

MemProf: Object lifecycle tracking

Own dynamic library

- Overrides memory management functions (like `malloc`) – stores profile information and calls original library
- Needs to be loaded manual per LD_PRELOAD or dlsym

MemProf: Thread lifecycle tracking

Own dynamic library

- Overloaded kernel functions `pref_event_task` and `perf_event_comm0`

MemProf: Memory access tracking

performance monitoring units (PMU)

- Microarchitecture profiling technique
- "Instruction Based Sampling" by AMD, "Precise Event Based Sampling" is similar technique by Intel (PMU technique)
- processor selects single instructions on a given frequency
- interrupt containing information about instruction used to process the data
- random based approach $\rightarrow$ variation of results

MemAxes

- Similar profiler like MemProf
- Aggregate profile information from performance monitoring units (PMUs)
- Gathers also information about hardware topology (caches, NUMA nodes . . .)

MemAxes: Semantic Annotations

Semantic Annotations

Developer decides which data structures are interested to profile
Developer can optimal aggregate additional attributes

Listing 1: Profile matrix A

```
#define N 1024
double A[N][N]; // matrix data object

SMRTree *smrt = new SMRTree();
SMRNode *A_SMR =
smrt->addSMR("A", sizeof(double), A, N*N);
```

MemAxes: Semantic Annotations (2)

Listing 2: Aggregate further application specific fields

```
// smrt is from previous example
smrt->addIntegerAttribute("x_coord",-1);
smrt->addIntegerAttribute("y_coord",-1);

void* mat_attribution(SMRNode *smr, struct mem_sample *
    sample)
{
  // Obtain the index of the address
  int bufferIndex = smr->indexOf(sample->daddr);
  // Calculate the x and y indices (row-major)
  sample->setAttribute("x_coord", bufferIndex % N);
  sample->setAttribute("y_coord", bufferIndex / N);
}
```

MemAxes: Working Principle

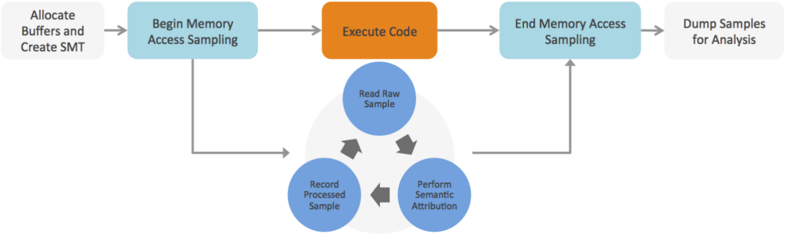

Figure : Basic working principle of MemAxes

MemAxes: Visualization

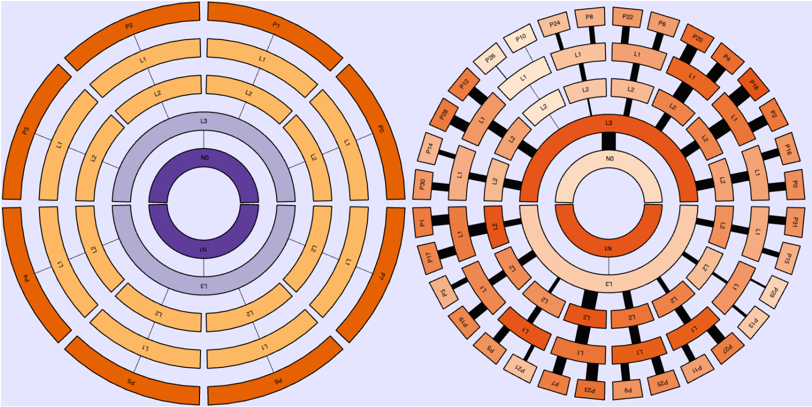

Figure : Visualization principle of memAxes; left: hardware topology in general; right: with cpu latency (color) and usage (line width)

MemAxes: Example

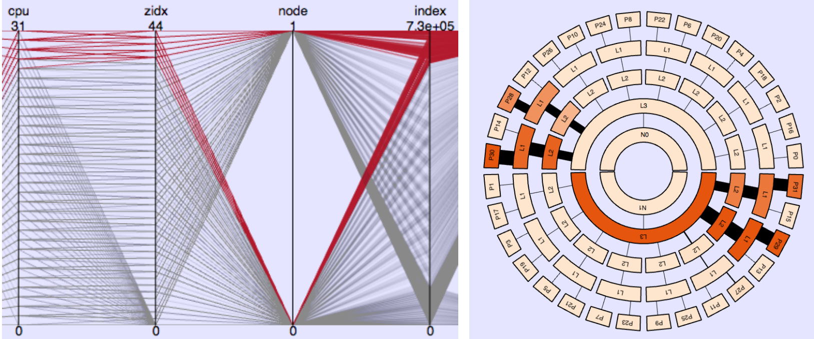

Figure : Application with unoptimized affinities

MemAxes: Example (2)

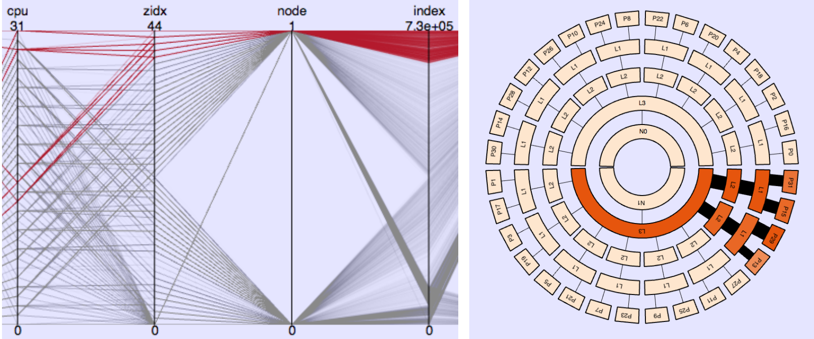

Figure : Same application with optimized affinities

Tips for algorithm groups

- Profilers support identifying the cause of intensive remote memory access
- NumaTOP is a good start (easily installable, check whether we have a NUMA problem)
- vTune could give some hints in what direction to look
- MemProf not usable as it currently depends on AMD profiling instructions
- MemAxes is the most powerful tool, but take a little bit more time to use it (smaller code adjustments) and I was unable to find the tracing sources itself

Summary

- Runtime behavior of NUMA applications is difficult to understand / predict
- Concrete tracing information needed to identify issues
- Automatic tools (e.g. kernel scheduling) not always able to produce optimal placing
- Profilers have currently only limited support for identify NUMA issues
- NUMA profilers are an active research and development field
- Even with this information may it be complicated to optimize your application
- NUMA problems remain performance problems

References

Images are extracted from the corresponding papers!

- Lachaize, Renaud, Baptiste Lepers, and Vivien Quéma. **MemProf: A Memory Profiler for NUMA Multicore Systems.** USENIX Annual Technical Conference. 2012. (https://www.usenix.org/system/files/conference/atc12/atc12-final229.pdf)
- MemProf source code. https://github.com/Memprof
- Alfredo Giménez, Todd Gamblin, Barry Rountree, Abhinav Bhatele, Ilir Jusufi, Peer-Timo Bremer, and Bernd Hamann. **Dissecting On-Node Memory Access Performance: A Semantic Approach.** In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (to appear), SC '14, November 2014. LLNL-CONF-658626. (http://charm.cs.illinois.edu/~bhatele/pubs/pdf/2014/sc2014c.pdf)

References (2)

- Davis, Alan L., and Uroš Prestor. **The ccNUMA memory profiler.** Proc. of the 4th IEEE Workshop on Workload Characterization. 2001. (http://www.cs.utah.edu/~ald/pubs/CC-numa.pdf)
- SNPERF website. http://www.cs.utah.edu/~uros/snperf/
- NumaTOP v1.0 Documentation (https://01.org/sites/default/files/documentation/numatop_introduction_0.pdf)
- PARSEC Benchmark 2.1 (http://parsec.cs.princeton.edu/)
- [1]: http://www.paradyn.org/CSCADS2013/slides/liu13.pdf
- [2]: Vtune Performance Analyze. http://nsfcac.rutgers.edu/people/irodero/classes/10-11/ece451-566/slides/vtune.pdf