Programming Parallel and Distributed Systems: Seminar Project Proposals

Frank Feinbube, Felix Eberhardt, Max Plauth, Prof. Andreas Polze
Operating Systems and Middleware Research Group
Hasso Plattner Institute

Overview

Themenwünsche bis zum 06.05.2015 per E-Mail an Frank.Feinbube@hpi.de oder einfach bei uns vorbeikommen :)

Hasso Plattner Institut

Algorithm Optimization

- EDC Graph Search, Hyrise
- Speeded Up Robust Features
- … or anything really

Feature Benchmarks

- Intel Transactional Memory
- Stream Stores vs. Coherent Stores
- Prefetching

Tools

Programming Languages / Models

- PGAS: Fortress, X10, UPC, …
- Scala, Java, JavaScript, C#, …
- CUDA, OpenCL, OpenACC, …

Platforms

- (hierarchical) NUMA systems
- Intel Xeon Phi
- GPU Computing

Linux Kernel experiments

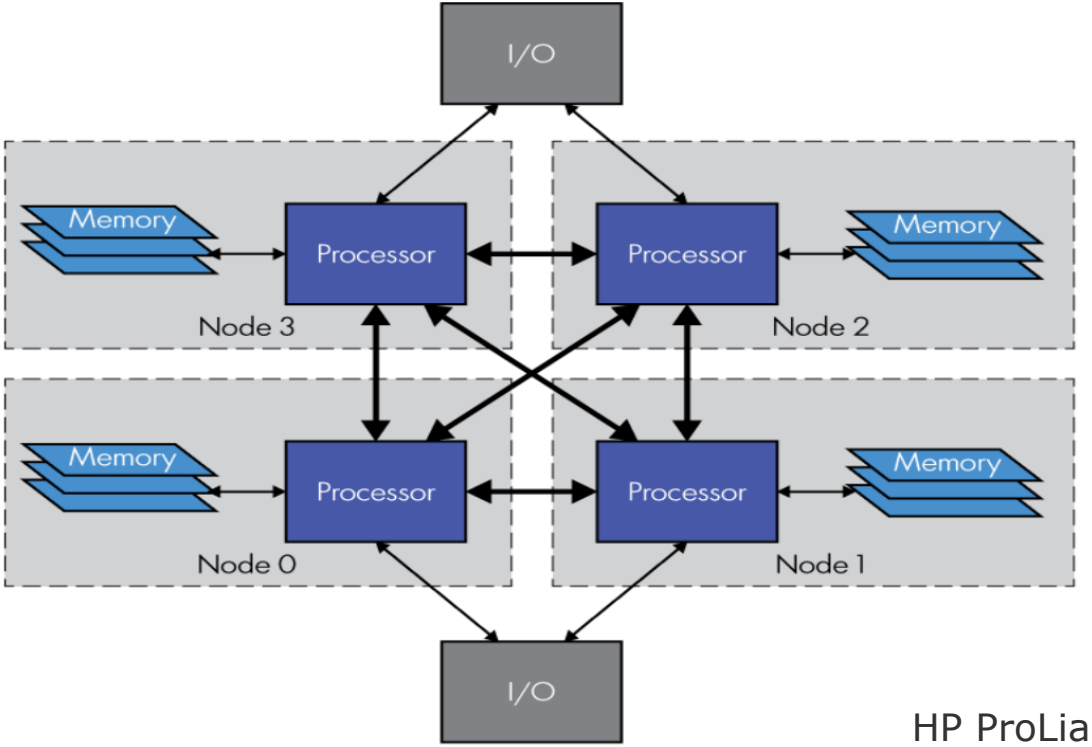
Performance Predictions

Platforms

Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

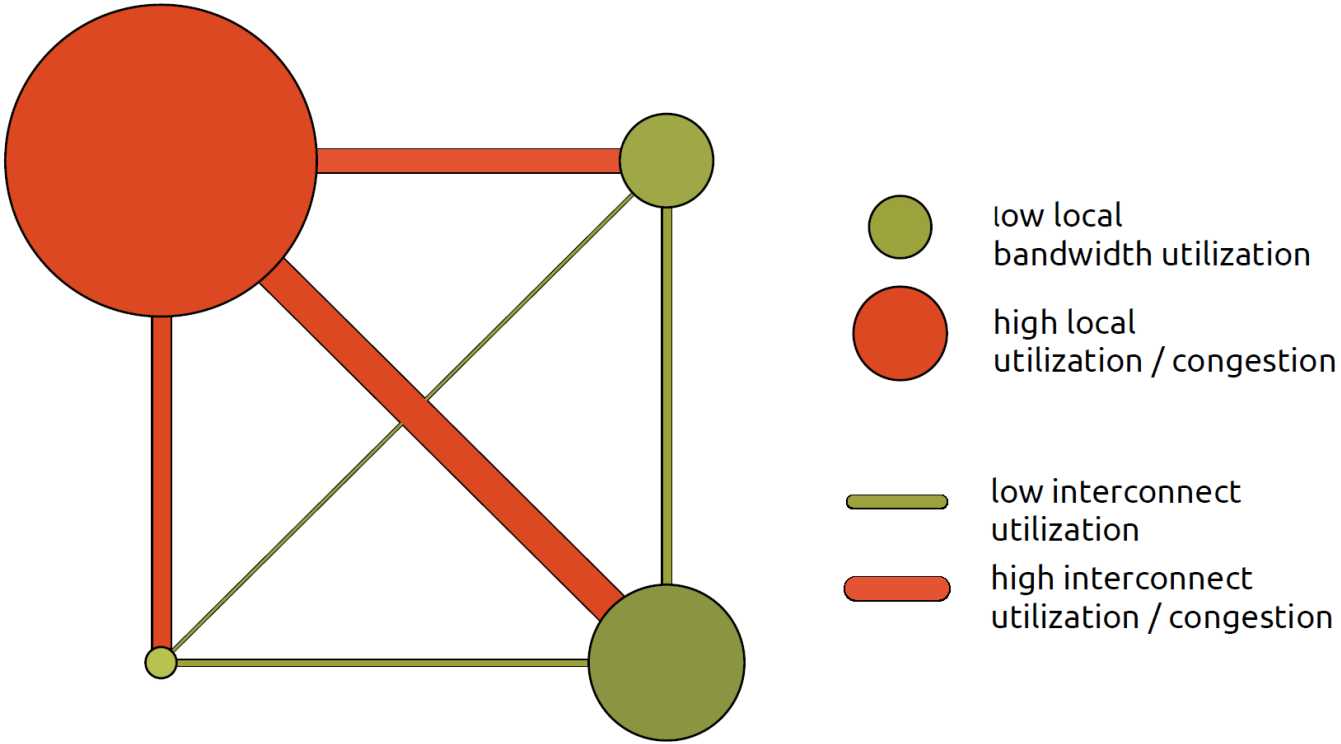
(Hierarchical) NUMA Systems

Classical NUMA-System

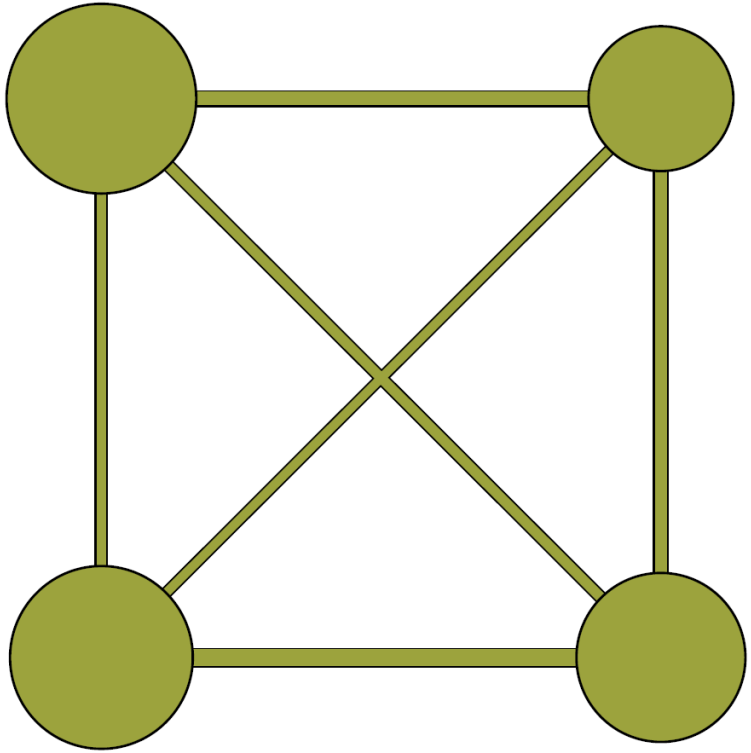


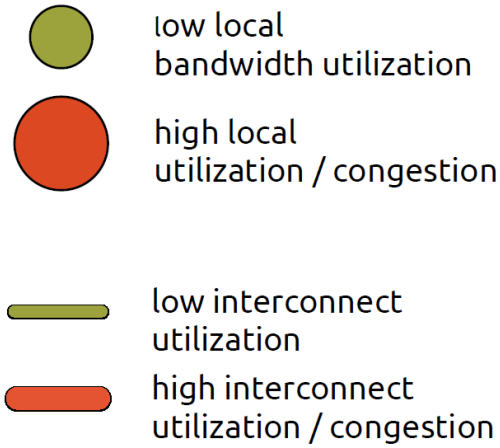
HP ProLiant DL580 G7

NUMA challenges

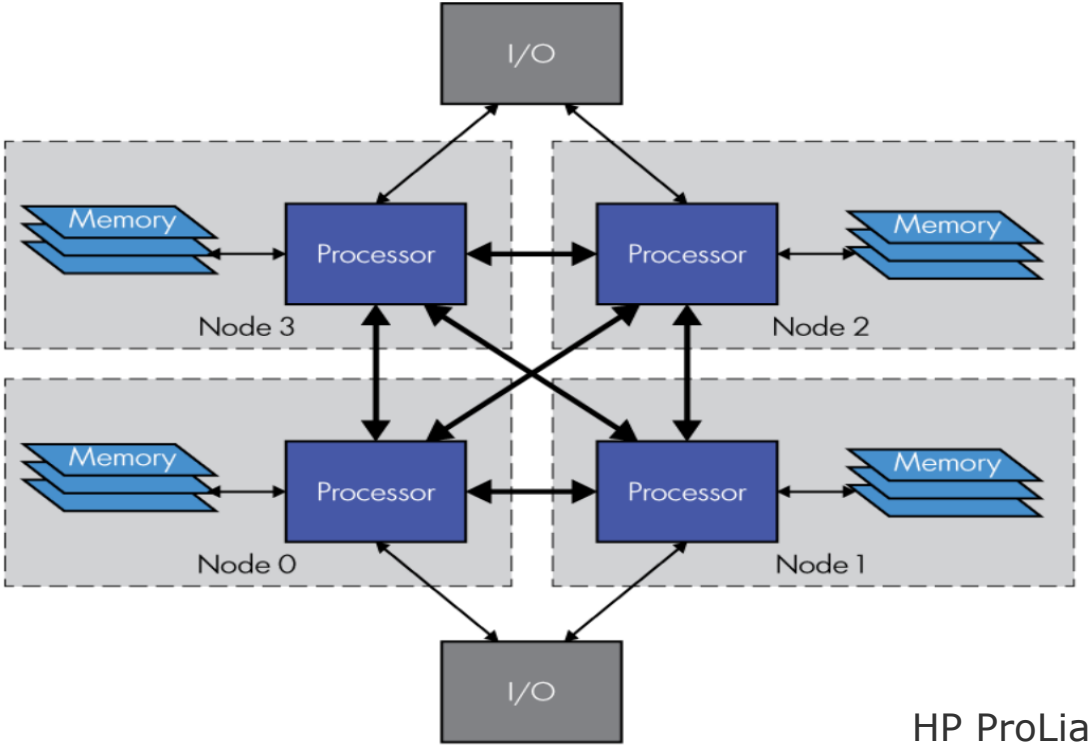


NUMA challenges

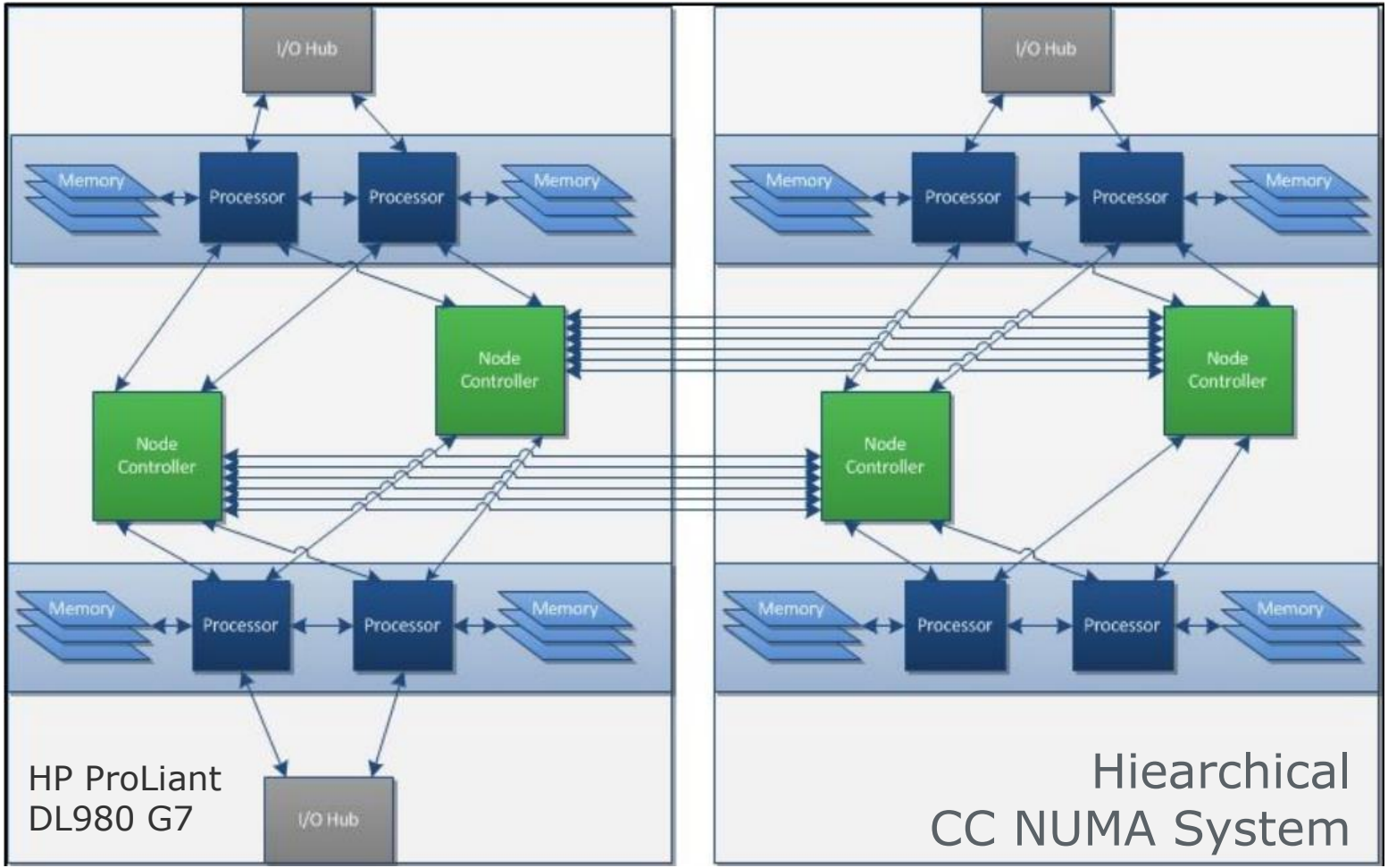




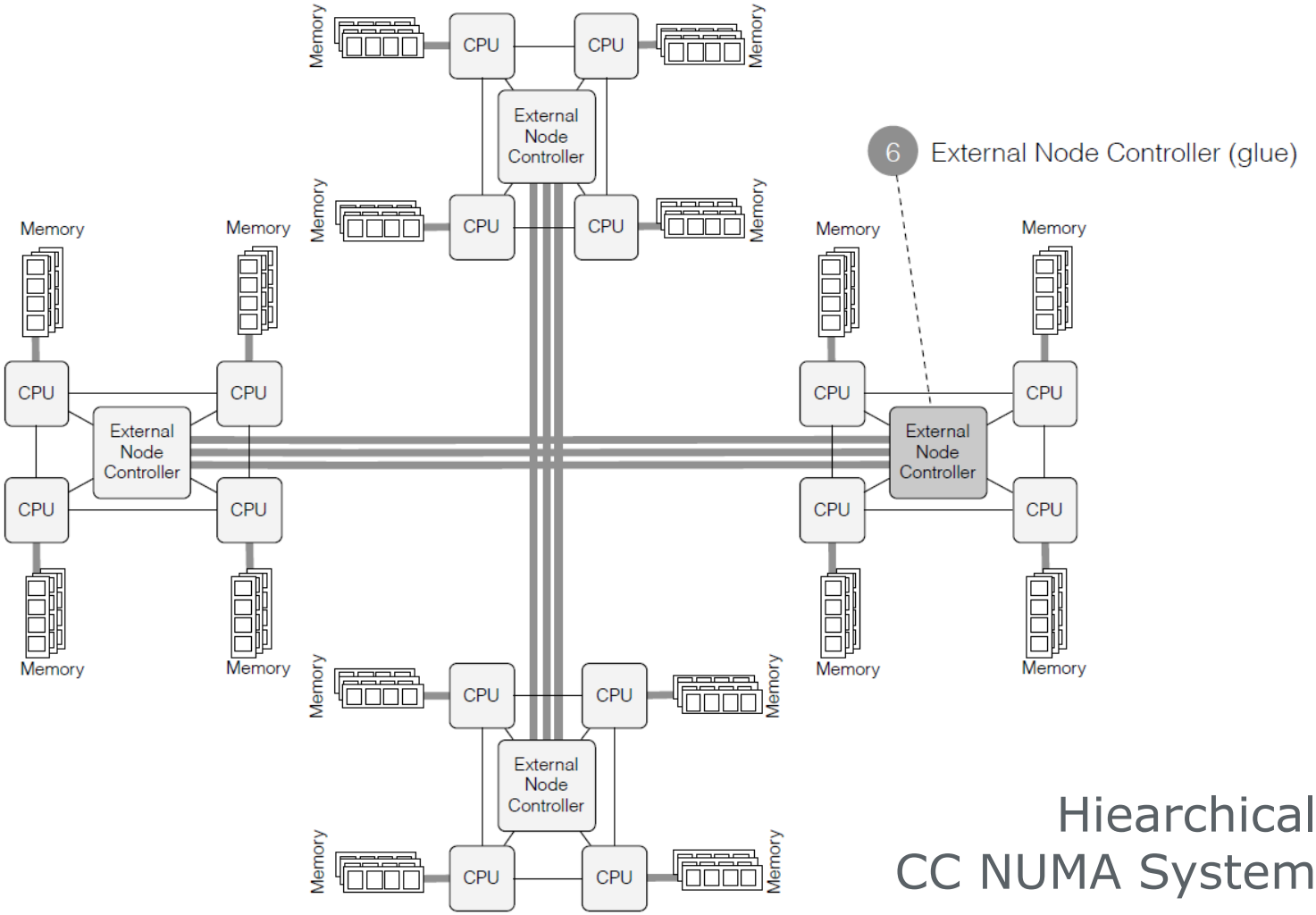
Classical NUMA-System



HP ProLiant DL580 G7



8 sockets glued



16 sockets glued

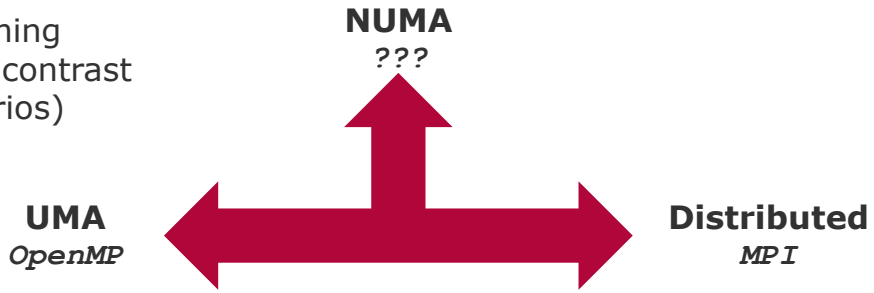
today up 32 is possible

Hiearchical CC NUMA System

Core Research Question: Best practices for hierarchical NUMA environments?

- To date: best practices and optimization techniques focus on either
  - Parallel Shared Memory Systems (UMA; e.g. with OpenMP)
  - Or Distributed Message-Passing Systems (e.g. with MPI)
- Pure NUMA optimizations have been mostly neglected, because
  - The performance penalties were moderate
  - There is no intuitive programming metaphor for NUMA so far (in contrast to UMA and Distributed scenarios)
  - UMA and Distributed allow for portable performance

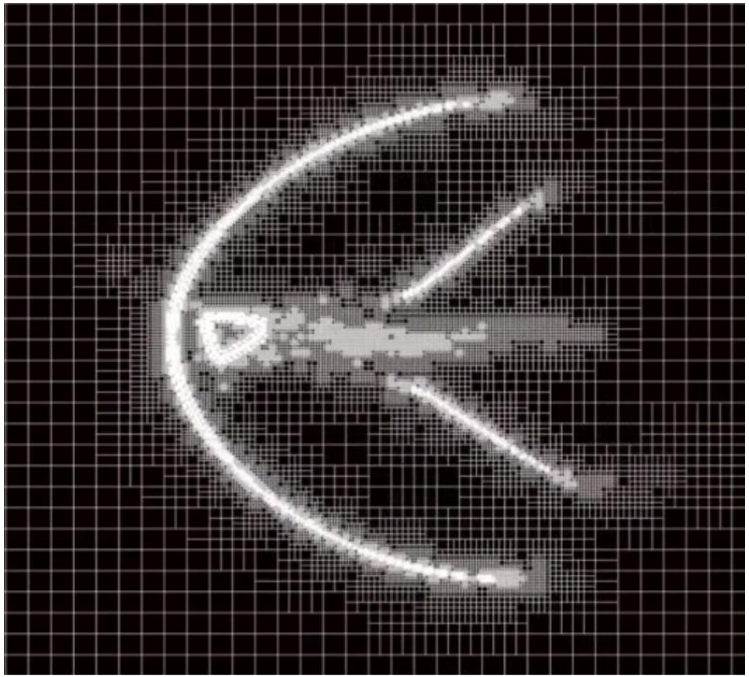
- The emergence of hierarchical cache-coherent NUMA systems requires:
  - Novel Portable Optimization Techniques and Best Practices
  - NUMA-aware Tools, Libraries, Programming Models, Patterns, Distribution Schemes, …
  - Considering Topology and Hardware Characteristics

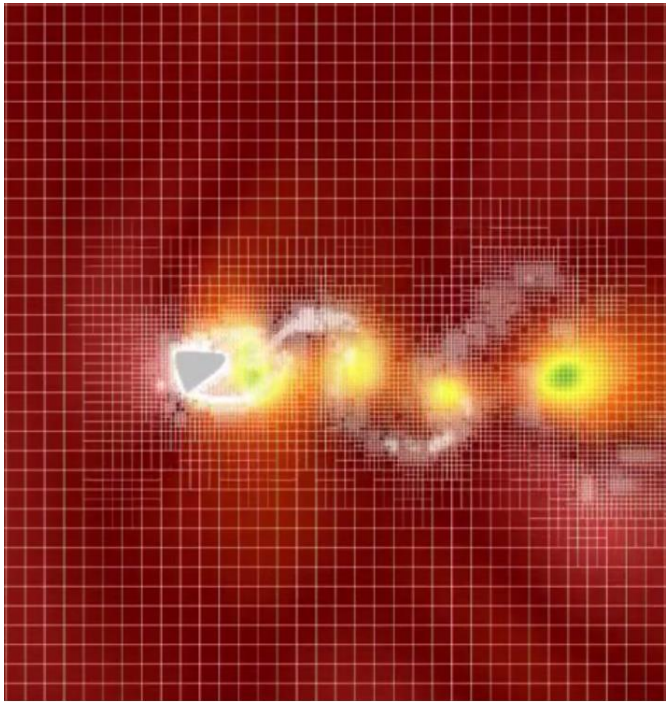


Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

GPU Computing

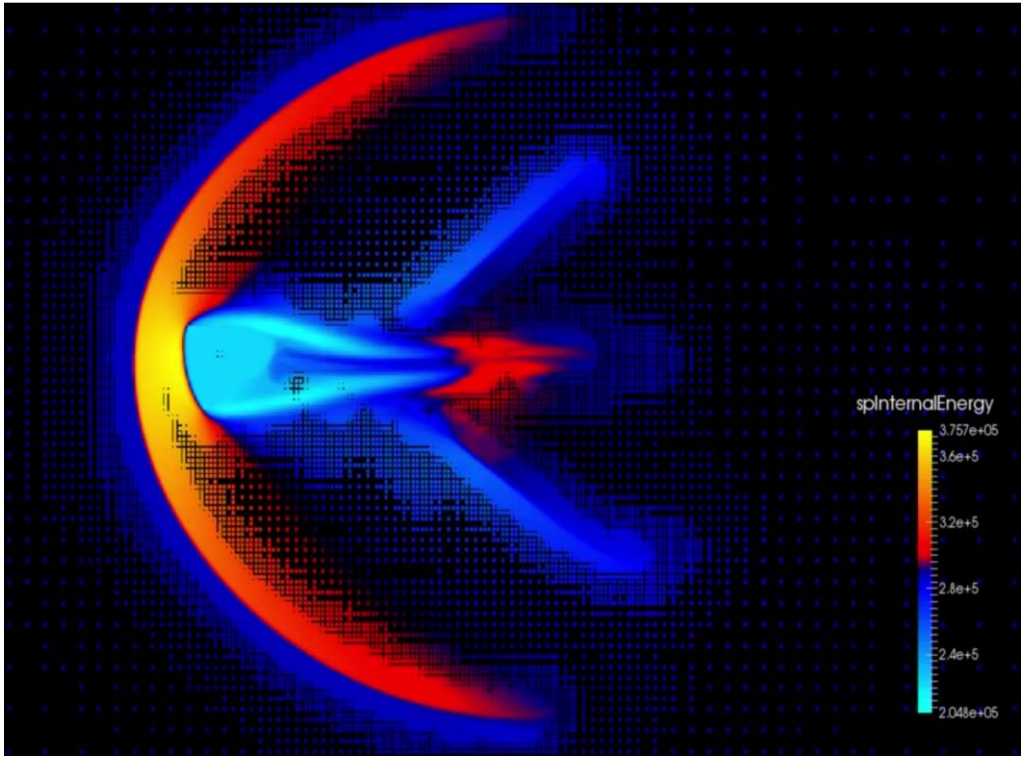
GPU Computing + Dynamic Parallelism





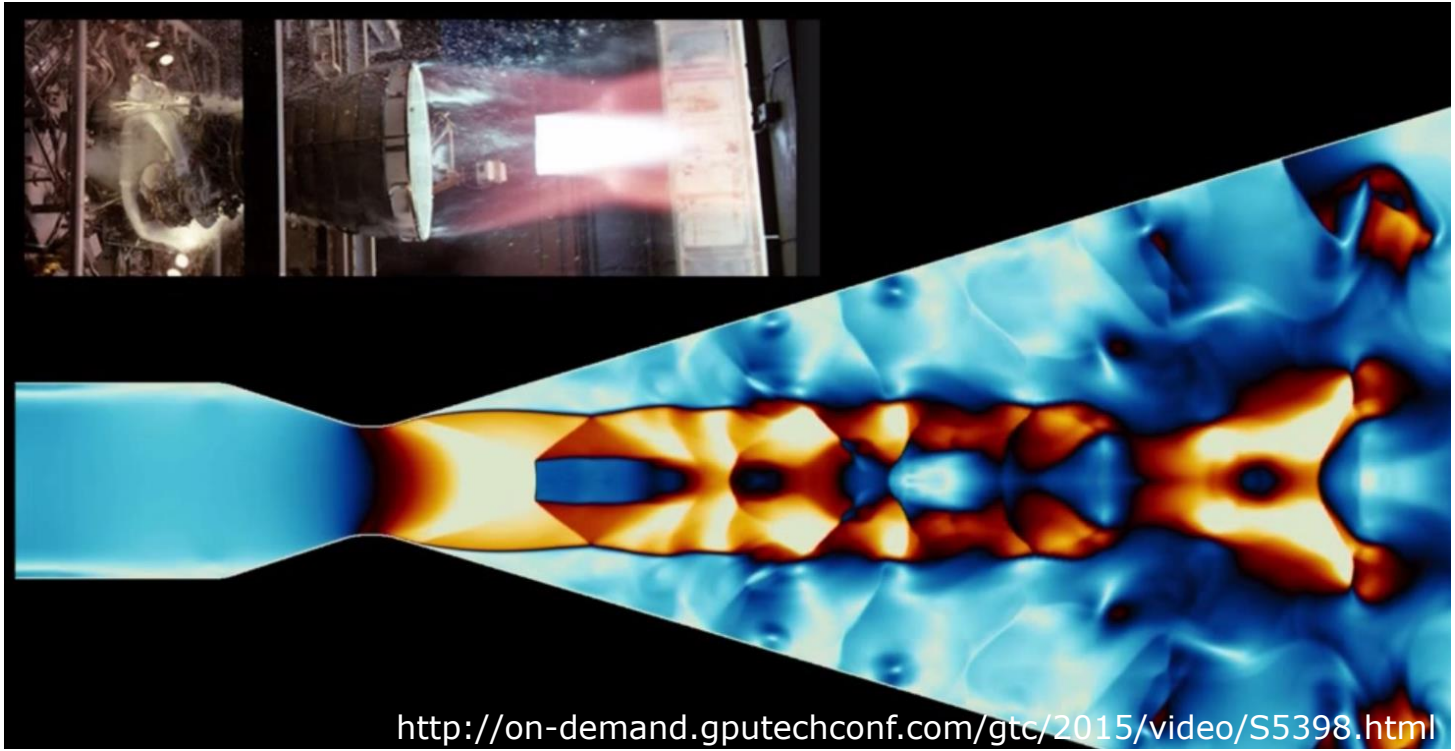
http://on-demand.gputechconf.com/gtc/2015/video/S5398.html

= Science at the next Level



„[…] this would be 300 million grid nodes. We did this on a single GPU.“

= Science at the next Level



Our FSOC K20 + Phi machine would have been the 3rd most powerful computer in the world in 2004

Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

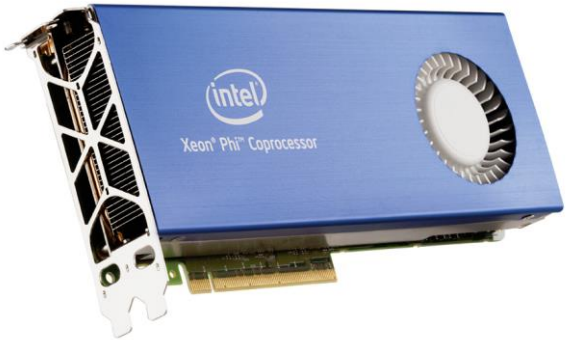
Intel Xeon Phi

Xeon Phi Hardware

60 Cores based on P54C architecture (Pentium)

- > 1.0 Ghz clock speed; 64bit based x86 instructions + SIMD
- 1x 25 MB L2 Cache (=512KB per core) + 64 KB L1
  - Cache coherency
- 8 (to 32) GB of DDR5
- **4 Hardware Threads per Core** (240 logical cores)
  - No Multicore / Hyper-Threading
  - Think graphics-card hardware threads
  - Only one runs = memory latency hiding
  - Switched after each instruction!! -> use 120 or 240 threads for the 60 cores

- 512 bit wide VPU with new ISA KCi
  - **No support for MMX, SSE or AVX**
  - Could handle 8 doule precision floats/16 single precision floats
  - Always structured in vectors with 16 elements



Operating System: minimal, embedded Linux

```
johannes.henning@tesla: ~
[root@tesla-mic0 /root]# ls /bin/
IMB-MPI1            fdflush             mknod               powertop
addgroup            fgrep               mktemp              printenv
adduser             fsync
apr-1-config        ganglia
ash                 getopt
base64              gmetric
busybox             grep
busybox.setuidroot  gstat
cat                 gunzip
catv                gzip
chattr              hostnam
chgrp               hush
chmod               ionice
chown               iostat
coi_daemon          ip
cp                  ipaddr
cpio                ipcalc
cpuinfo             iplink
cttyhack            iproute
date                iprule
dd                  iptunne
delgroup            kill
deluser             limits
df                  linux32
dmesg               linux64
dnsdomainname       ln
dumpkmap            login
echo                ls
ed                  lsattr
egrep               lzop
false               mkdir
```

```
johannes.henning@tesla: ~
tesla:/ # ssh mic0
[root@tesla-mic0 /root]# cat /proc/cpuinfo | tail -n 22
model           : 1
model name      : 0b/01
stepping        : 3
cpu MHz         : 1052.630
cache size      : 512 KB
physical id     : 0
siblings        : 240
core id         : 59
cpu cores       : 60
apicid          : 239
initial apicid  : 239
fpu             : yes
fpu_exception   : yes
cpuid level     : 4
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic mtrr mca pat fxsr ht syscall nx lm re
p_good nopl lahf_lm
bogomips        : 2114.13
clflush size    : 64
cache_alignment : 64
address sizes   : 40 bits physical, 48 bits virtual
power management:

[root@tesla-mic0 /root]#
```

Linux Standard Base (LSB) Core libraries.

Busybox minimal shell environment

First step for portable applications: Discovering and assessing the NUMA topology

Objectives for portable performance:

- Identify Application Bottlenecks
  - At development time
  - To (Re-)Design algorithm accordingly
- Acquiring Topology Information
  - At application starting time
  - To create and map threads and data accordingly

State-of-the-Art NUMA Tools:

- ACPI distance values
- Linux sysfs
- Libnuma: numactl
- Hwloc lstopo
- MemAxes
- Linux Perf
- numatop
- Intel Performance Counter Monitor
- Intel Vtune
- MLC (Memory Latency Checker)

Linux sysfs

Information provided:

- Nodes (sockets)
- ACPI distance values of nodes and CPUs
- Mapping of CPUs to nodes
- Cache sizes, levels, associativity, cacheline size
- Cache sharing of CPUs

Restrictions:

- Linux only

Macintosh HD — ssh — 88×23

```
Felix.Eberhardt@side:/sys/devices/system/cpu/cpu0> ls
cache  cpufreq  crash_notes  node0  thermal_throttle  topology
Felix.Eberhardt@side:/sys/devices/system/cpu/cpu0> cd topology/
Felix.Eberhardt@side:/sys/devices/system/cpu/cpu0/topology> cat core_siblings_list
0-14,240-254
Felix.Eberhardt@side:/sys/devices/system/cpu/cpu0/topology> cd ../node0/
Felix.Eberhardt@side:/sys/devices/system/cpu/cpu0/node0> cat distance
10 16 19 16 50 50 50 50 50 50 50 50 50 50 50 50
Felix.Eberhardt@side:/sys/devices/system/cpu/cpu0/node0> cat cpulist
0-14,240-254
Felix.Eberhardt@side:/sys/devices/system/cpu/cpu0/node0> cd ..
Felix.Eberhardt@side:/sys/devices/system/cpu/cpu0> cd topology/
Felix.Eberhardt@side:/sys/devices/system/cpu/cpu0/topology> cat thread_siblings_list
0,240
Felix.Eberhardt@side:/sys/devices/system/cpu/cpu0/topology> cd ../cache/
Felix.Eberhardt@side:/sys/devices/system/cpu/cpu0/cache> ls
index0  index1  index2  index3
Felix.Eberhardt@side:/sys/devices/system/cpu/cpu0/cache> cd index0
Felix.Eberhardt@side:/sys/devices/system/cpu/cpu0/cache/index0> ls
coherency_line_size  physical_line_partition  size
level                shared_cpu_list          type
number_of_sets       shared_cpu_map           ways_of_associativity
Felix.Eberhardt@side:/sys/devices/system/cpu/cpu0/cache/index0>
```

Libnuma
numactl --hardware

Information provided:

- Nodes (sockets)
- ACPI distance values of nodes and CPUs
- Mapping of CPUs to nodes

Restrictions:

- Linux only
- Available as library to be used in applications to query system devices

Macintosh HD — ssh — 88×23

```
node 15 cpus: 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 465 466 467 46
8 469 470 471 472 473 474 475 476 477 478 479
node 15 size: 753648 MB
node 15 free: 622078 MB
node distances:
node   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
  0:  10  16  19  16  50  50  50  50  50  50  50  50  50  50  50  50
  1:  16  10  16  19  50  50  50  50  50  50  50  50  50  50  50  50
  2:  19  16  10  16  50  50  50  50  50  50  50  50  50  50  50  50
  3:  16  19  16  10  50  50  50  50  50  50  50  50  50  50  50  50
  4:  50  50  50  50  10  16  19  16  50  50  50  50  50  50  50  50
  5:  50  50  50  50  16  10  16  19  50  50  50  50  50  50  50  50
  6:  50  50  50  50  19  16  10  16  50  50  50  50  50  50  50  50
  7:  50  50  50  50  16  19  16  10  50  50  50  50  50  50  50  50
  8:  50  50  50  50  50  50  50  50  10  16  19  16  50  50  50  50
  9:  50  50  50  50  50  50  50  50  16  10  16  19  50  50  50  50
 10:  50  50  50  50  50  50  50  50  19  16  10  16  50  50  50  50
 11:  50  50  50  50  50  50  50  50  16  19  16  10  50  50  50  50
 12:  50  50  50  50  50  50  50  50  50  50  50  50  10  16  19  16
 13:  50  50  50  50  50  50  50  50  50  50  50  50  16  10  16  19
 14:  50  50  50  50  50  50  50  50  50  50  50  50  19  16  10  16
 15:  50  50  50  50  50  50  50  50  50  50  50  50  16  19  16  10
Felix.Eberhardt@side:~>
```

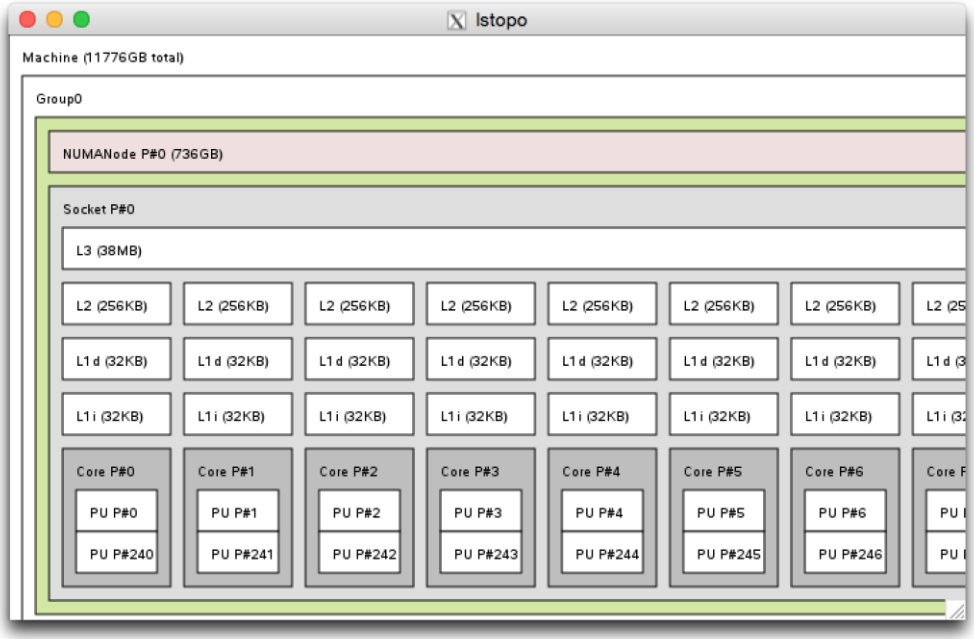
Hwloc
lstopo

Information provided:

- Nodes (sockets)
- ACPI distance values of nodes and CPUs
- Mapping of CPUs to nodes
- Grouping of nodes according to distance values
- Whole memory hierarchy

Restrictions:

- Several platforms: Windows, Linux, BSD, ...
- Available as library to be used in applications to query system devices



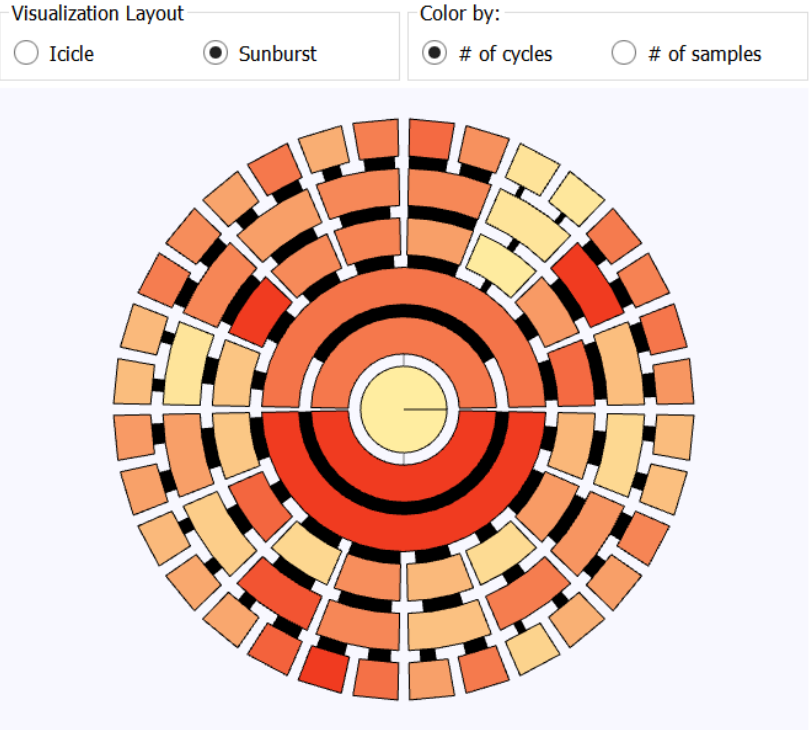
MemAxes

Information provided:

- Nice visualization of several nodes and the memory hierarchy
- Able to see the bottleneck or misplacement of threads and data

Restrictions:

- Research prototype
- The data collection part is missing



Linux Perf

Information provided:

- Ability to read processor specific performance counter
- Can collect profile data and compare them to different runs
- Several extensions: memory profiling, cache-to-cache sharing
- Abstraction layer for kernel and hardware events

Restrictions:

- Linux only

Macintosh HD — ssh — 88×23

```
Felix.Eberhardt@side:~> perf stat echo hello world!
hello world!

 Performance counter stats for 'echo hello world!':

          0.668209 task-clock                #    0.590 CPUs utilized
                 1 context-switches          #    0.001 M/sec
                 0 CPU-migrations            #    0.000 M/sec
               160 page-faults               #    0.239 M/sec
           2126171 cycles                    #    3.182 GHz
           1717825 stalled-cycles-frontend   #   80.79% frontend cycles idle
     <not counted> stalled-cycles-backend
            842824 instructions              #    0.40  insns per cycle
                                             #    2.04  stalled cycles per insn
            170968 branches                  #  255.860 M/sec
              8223 branch-misses             #    4.81% of all branches

       0.001131902 seconds time elapsed

Felix.Eberhardt@side:~>
```

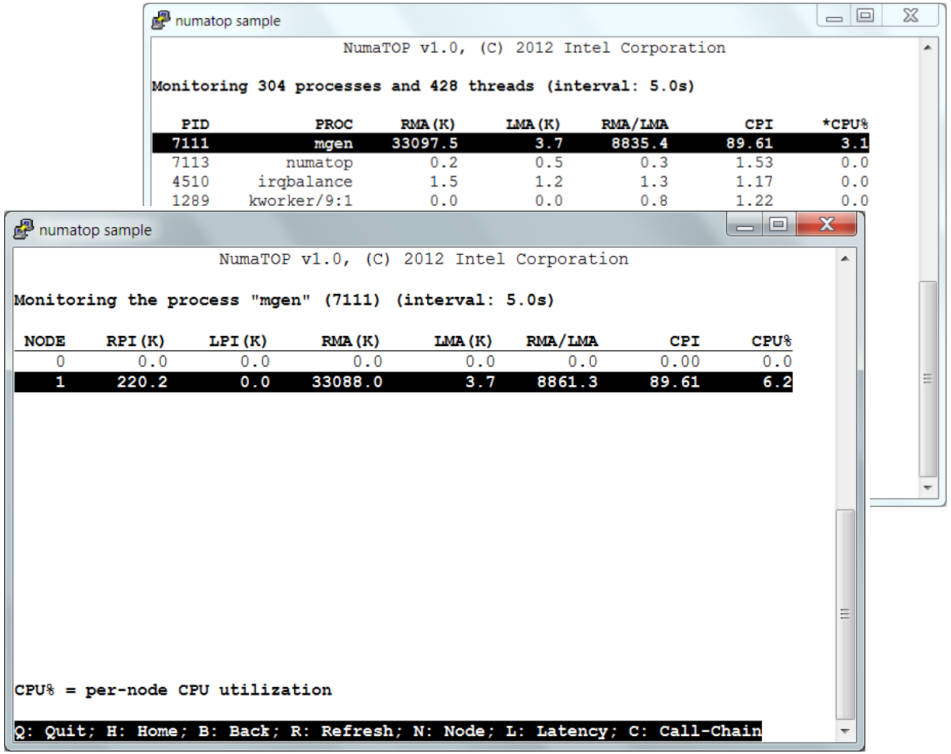
numatop

Information provided:

- Similar to `top` tool
- Shows NUMA specific metrics
- Uses instruction sampling
- Memory view to find out which memory addresses are accessed frequently by remote nodes
- Ability to collect stacktraces

Restrictions:

- Linux only, Kernel 3.9 or later
- Intel processors only



Intel Performance Counter Monitor

Information provided:

- API for Intel specific performance counters
- Core and Uncore events
- QPI links and memory controller utilization

Restrictions:

- Available on Windows and Linux
- Intel processors only

Macintosh HD — ssh — 88×15

```
Time elapsed: 2 ms
Called sleep function for -1000 ms
-- NODE0 Memory (MB/s):      122.78  --||-- NODE1 Memory (MB/s):       96.83  --
-- NODE2 Memory (MB/s):      108.48  --||-- NODE3 Memory (MB/s):       34.88  --
-- NODE4 Memory (MB/s):       27.90  --||-- NODE5 Memory (MB/s):       30.37  --
-- NODE6 Memory (MB/s):       15.58  --||-- NODE7 Memory (MB/s):       22.02  --
-- NODE8 Memory (MB/s):       15.62  --||-- NODE9 Memory (MB/s):       17.57  --
-- NODE10 Memory (MB/s):       16.99  --||-- NODE11 Memory (MB/s):       19.49  --
-- NODE12 Memory (MB/s):       18.72  --||-- NODE13 Memory (MB/s):       20.38  --
-- NODE14 Memory (MB/s):       30.88  --||-- NODE15 Memory (MB/s):    19845.82  --
---------------------------------------||---------------------------------------
--                  System Read Throughput(MB/s):  10757.21                  --
--                 System Write Throughput(MB/s):   9687.10                  --
--               System Memory Throughput(MB/s):  20444.32                  --
----------------
```

Macintosh HD — ssh — 88×26

```
Intel(r) QPI traffic estimation in bytes (data and non-data traffic outgoing from CPU/so
cket through QPI links):

                QPI0     QPI1     QPI2    |  QPI0   QPI1   QPI2
-----------------------------------------------------------------------------------------
------
 SKT    0       114 M     29 M     30 M   |    0%     0%     0%
 SKT    1        46 M     53 M     24 M   |    0%     0%     0%
 SKT    2        47 M     36 M     27 M   |    0%     0%     0%
 SKT    3        33 M     35 M     44 M   |    0%     0%     0%
 SKT    4        61 M     15 M     22 M   |    0%     0%     0%
 SKT    5        62 M     22 M     15 M   |    0%     0%     0%
 SKT    6        46 M     16 M     19 M   |    0%     0%     0%
 SKT    7        19 M     16 M     48 M   |    0%     0%     0%
 SKT    8        89 M     17 M     22 M   |    0%     0%     0%
 SKT    9        47 M     37 M     21 M   |    0%     0%     0%
 SKT   10        35 M     29 M     22 M   |    0%     0%     0%
 SKT   11        28 M     26 M     35 M   |    0%     0%     0%
 SKT   12        44 M     17 M     22 M   |    0%     0%     0%
 SKT   13        34 M     32 M     31 M   |    0%     0%     0%
 SKT   14        32 M     25 M     31 M   |    0%     0%     0%
 SKT   15        25 M     25 M     48 M   |    0%     0%     0%
-----------------------------------------------------------------------------------------
------
Total QPI outgoing data and non-data traffic: 1679 M
```

Memory Latency Checker: mlc

Information provided:

- Latency and Bandwidth measurements for various Read / Write Scenarios
- Measures caching performance as well
- Can modify prefetcher settings

Restrictions:

- Available on Windows and Linux
- Intel processors only

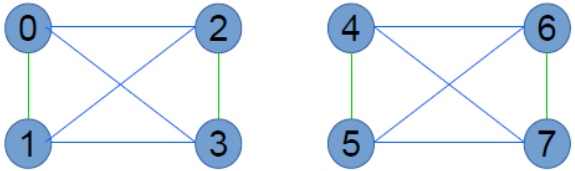
Idle latency

Soc	0	1	2	3	4	5	6	7
0	37.30	49.90	74.70	73.50	84.40	82.60	80.30	82.00
1	49.40	36.30	73.80	76.90	85.00	82.30	84.40	86.10
2	75.00	75.60	35.00	46.40	75.30	75.40	78.10	81.80
3	69.60	67.60	47.30	35.10	83.50	81.70	78.20	81.50
4	77.10	78.60	77.50	78.90	34.80	48.90	70.00	75.30
5	79.80	76.50	79.80	80.90	46.20	35.10	69.20	69.80
6	75.70	74.10	80.00	77.10	67.60	67.80	35.10	46.50
7	83.70	84.00	84.00	82.70	69.90	70.30	47.40	34.90

mlc vs ACPI

ACPI: SLIT

```
[...]
node distances:
node   0   1   2   3   4   5   6   7
  0:  10  12  17  17  19  19  19  19
  1:  12  10  17  17  19  19  19  19
  2:  17  17  10  12  19  19  19  19
  3:  17  17  12  10  19  19  19  19
  4:  19  19  19  19  10  12  17  17
  5:  19  19  19  19  12  10  17  17
  6:  19  19  19  19  17  17  10  12
  7:  19  19  19  19  17  17  12  10
```



Idle latency

Soc	0	1	2	3	4	5	6	7
0	37.30	49.90	74.70	73.50	84.40	82.60	80.30	82.00
1	49.40	36.30	73.80	76.90	85.00	82.30	84.40	86.10
2	75.00	75.60	35.00	46.40	75.30	75.40	78.10	81.80
3	69.60	67.60	47.30	35.10	83.50	81.70	78.20	81.50
4	77.10	78.60	77.50	78.90	34.80	48.90	70.00	75.30
5	79.80	76.50	79.80	80.90	46.20	35.10	69.20	69.80
6	75.70	74.10	80.00	77.10	67.60	67.80	35.10	46.50
7	83.70	84.00	84.00	82.70	69.90	70.30	47.40	34.90

ACPI: SLIT	Normalized latency
10	10
12	11.1
17	22.7
19	25.5

A hint, but not very accurate;

Weird effects for larger machines

State-of-the-Art NUMA Tools by objective

- Acquiring Topology Information
  - ACPI distance values
  - Linux sysfs
  - Libnuma: numactl
  - Hwloc lstopo
  - MemAxes
  - MLC (Memory Latency Checker)
- Identify Application Bottlenecks
  - Linux Perf
  - numatop
  - Intel Performance Counter Monitor
  - Intel Vtune

Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

One tool to rule them all

Omnitool

Idea: Performance Engineer - 1:n tools -> 1:1 Tools 1:n Plugins

- Generic profiling tool with advanced visualization capabilities
- Use database as backend to store all results from different runs
  - Compare different runs, soucre code annotations, collect stacktraces, …

- What information can be derived from the data provided by the tools?
- How can it be accumulated, digested, represented?
- How can it be visualized?
- How can workload be characterized?
- Can performance be predicted?
- What are the interesting metrics?
- …

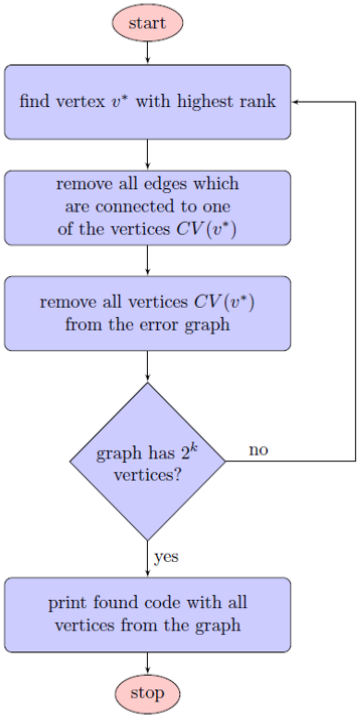
Algorithm Optimizations

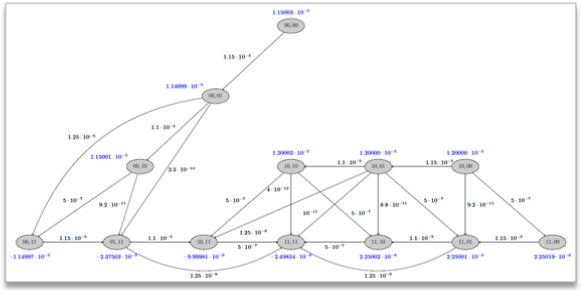
Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

EDC Graph Search

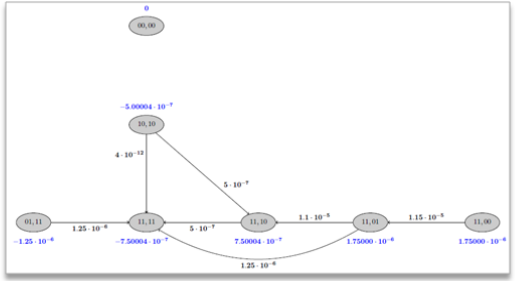
Overview:
EDC Graph Search Algorithm

- Graph-based algorithm used to derive error detection codes from error models
- Performance bottleneck is a maximum search in the graph
- Our straight forward UMA parallelization shows close to ideal speedups in a unified memory scenario
- And demonstrates severe performance degradations of NUMA





Transformation per loop iteration



Represents Performance Bottlenecks #9 and #12 in Berkeley Taxonomy

Bottleneck: select_codeword

```
long long int select_codeword()
{
    long long int best, i;
    double best_rank, rank;

    best = -1;
    best_rank = -1;

    for (i = 0; i<vertices; i++) {
        rank = get_rank(i);
        if (rank > best_rank) {
            best = i;
            best_rank = rank;
        }
    }

    return best;
}
```

- Looking for a maximum
  - = Typical reduction operation
- Characteristics
  - Commutative, associative
  - Input-Array is not changed
- Special:
  - Looking for an index, comparing to a value
- Approach
  - Parallelization of the loop

Straight-forward UMA parallelization with OpenMP

```
long long int select_codeword()
{
    long long i
    double best_rank, rank;

    best = -1;
    best_rank =

    for (i = 0; i<vertices; i++) {
        rank = get_rank(i);
        if (rank > best_rank) {
            best = i;
            bes
        }
    }

    return best;
}
```

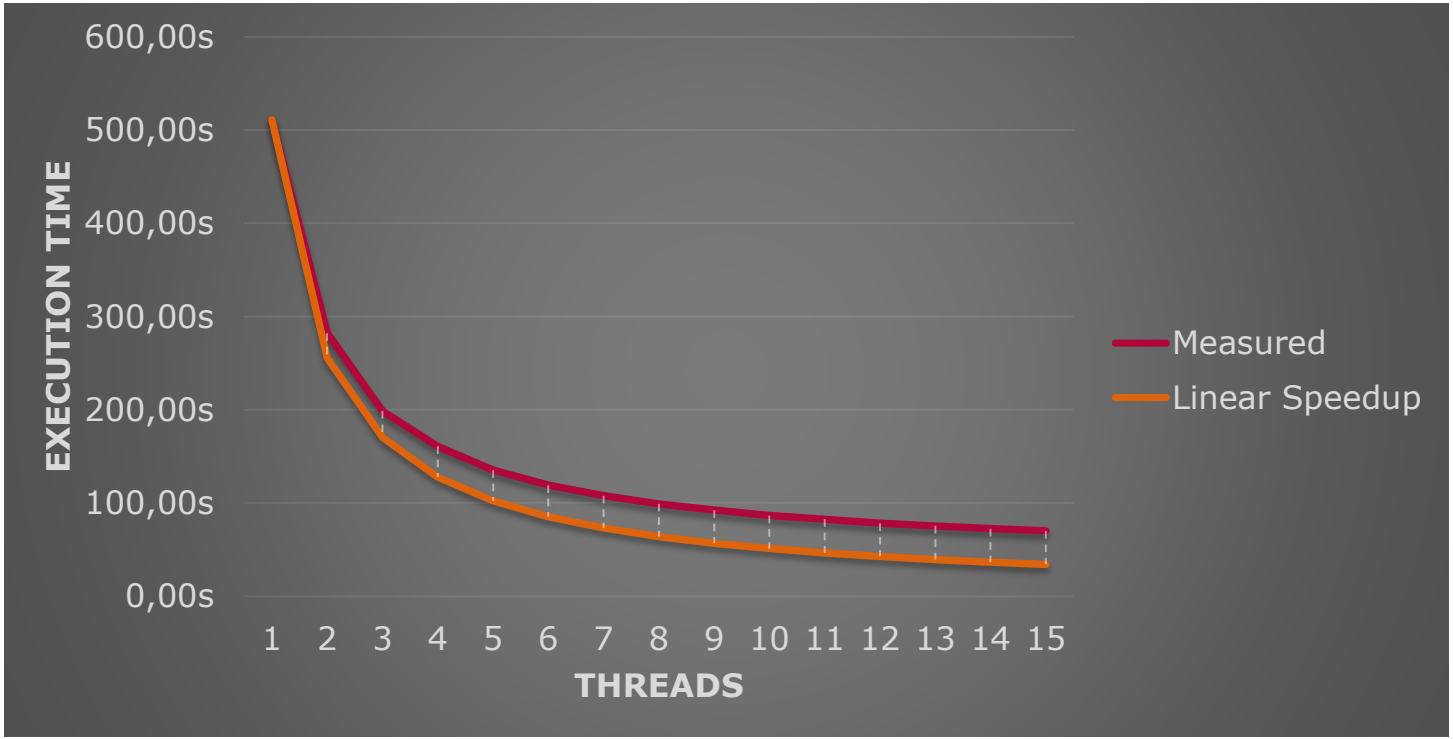
Local variables (copies)

Parallel for loop

Safe creation of the result

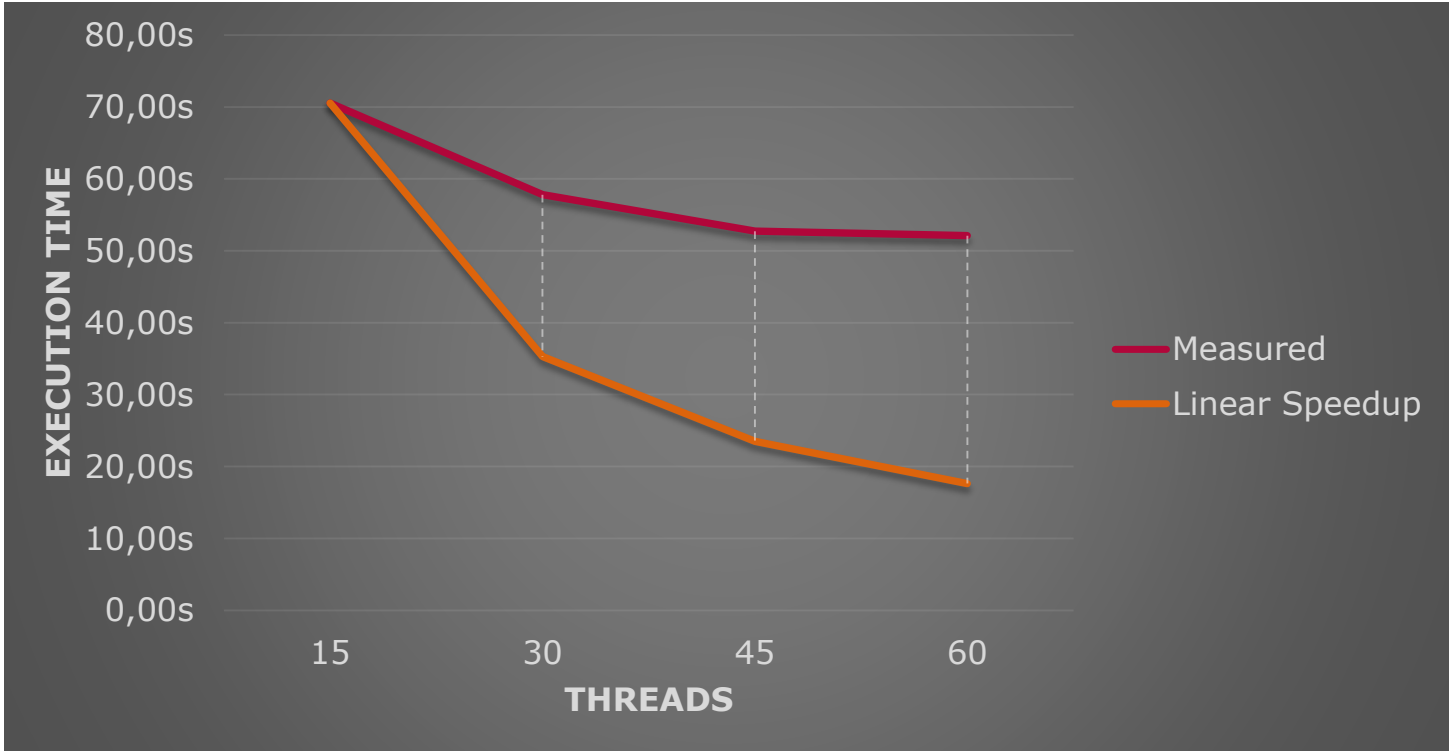
```
long long int select_codeword()
{
    long long int best, i;
    double best_rank, rank;
    double local_best, local_best_rank;

    best      = -1;
    best_rank = -1;

#pragma omp parallel private(rank, local_best, local_best_rank, i)
    {
        local_best = -1;
        local_best_rank = -1;

#pragma omp for schedule(guided)
        for (i = 0; i<vertices; i++) {
            rank = get_rank(i);
            if (rank > local_best_rank) {
                local_best = i;
                local_best_rank = rank;
            }
        }
#pragma omp critical
        if (local_best_rank > best_rank) {
            best = local_best;
            best_rank = local_best_rank;
        }
    }

    return best;
}
```

Single-Processor execution times: We achieve close-to optimal speedup



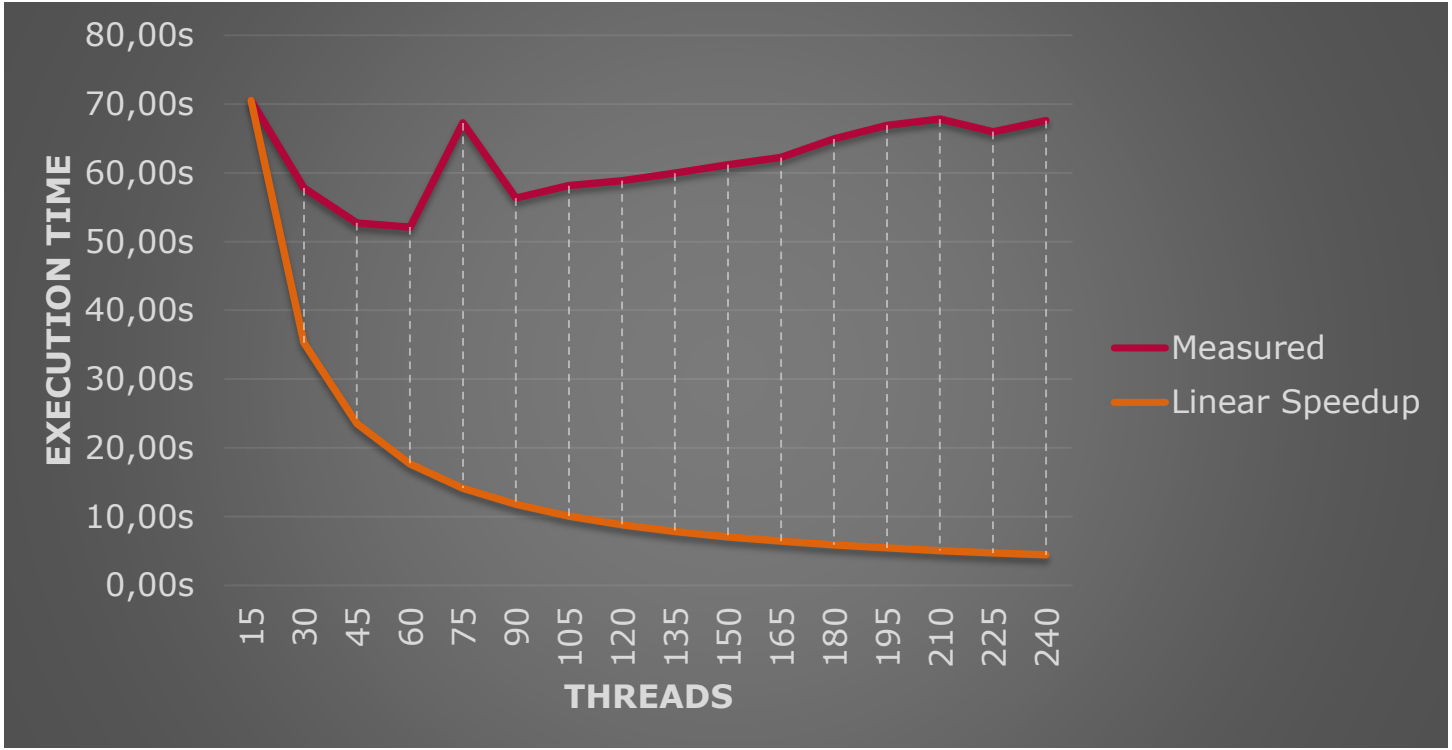
The past

Multi-Processor execution times: Speedup degradations with each additional processor



The present

Multi-Blade execution times: Using more blades decreases the performance



The future

Conclusion: Case Study #1

Lessons Learned:

- Hierarchical cache-coherent NUMA systems can become severe performance bottlenecks for naive UMA parallelizations
- Distributed execution performance is expected to experience even stronger performance degradations due to the bottleneck
- Especially problematic for Graphics Models and Graph Traversal Algorithms
  - Due to the strong interdependence and indirections

Next Steps:

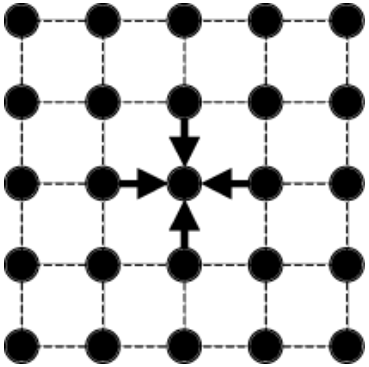
- Explore and generalizable algorithm redesign approaches to identify and exploit localities of clustered sub-graphs:
  - Multithreaded scaling is limited because of dependencies of different iterations
  - Only intra parallelization
  - Measure execution with respect to data and thread placement

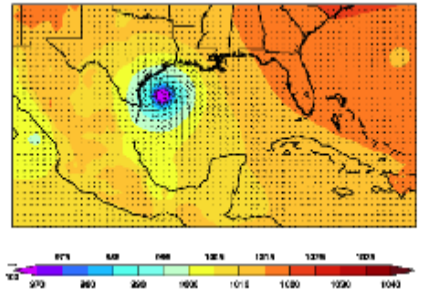
Speeded Up Robust Features

Speeded Up Robust Features (SURF) Algorithm

- **Data**: a regular multidimensional grid; access is regular and statically determinable (strided)
- **Computation**: sequence of grid updates (all points are updated using values from a small neighborhood); updates are logically concurrent
- In practice implemented as sequential sweep through computation domain (in place or two grid versions)

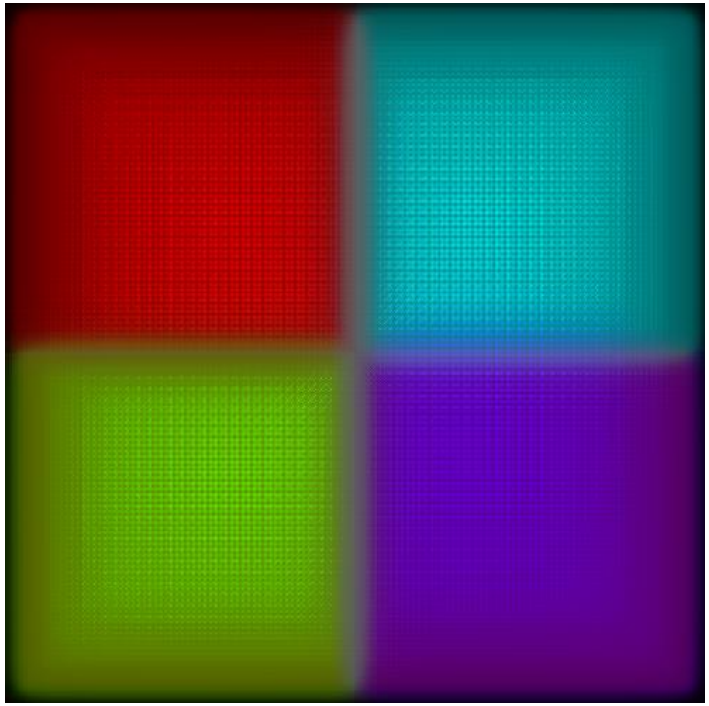
- **Uniprocessor Mapping**: highly vectorizable, points can be visited in any order
  - Spatial locality to use of long cache lines
  - Temporal locality to allow cache reuse (small grids)
- **Parallel Mapping**: subgrid per processor
  - Communication and synchronization for boundaries (=ghost cells, surface to volume ratio important)
  - Latency hiding: increased number of overhead zones and exchanging more data less frequently

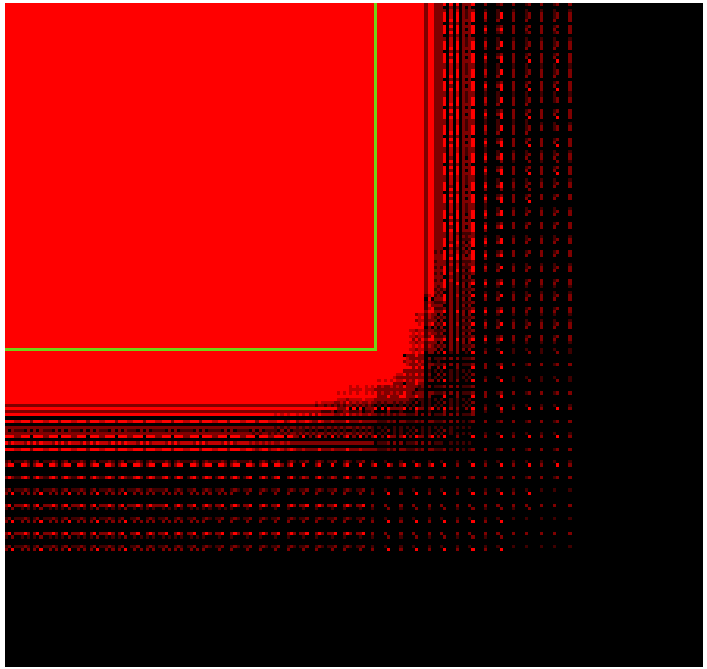




Represents Performance Bottlenecks #3 and #5 in Berkeley Taxonomy

Memory access pattern for SURF



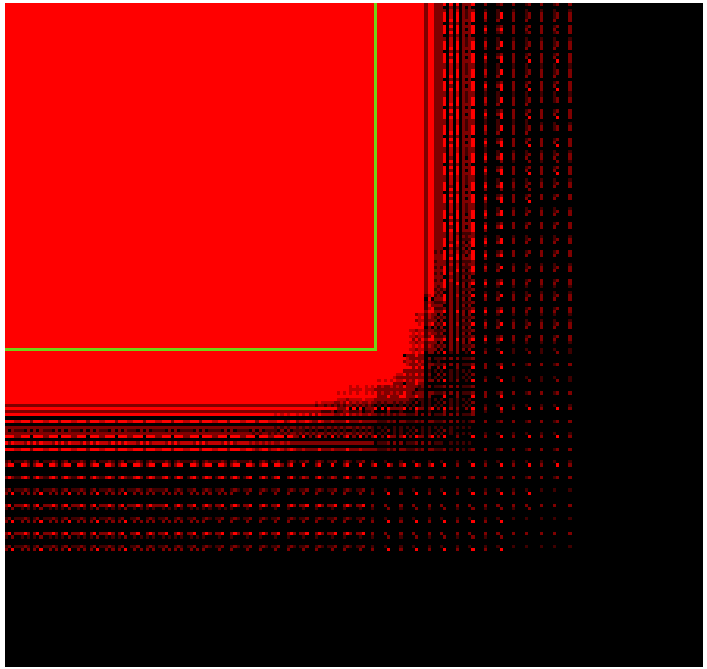


The golden ratio

Lessons Learned:

- Perfect example for NUMA
  - Too large for UMA (huge images from astronomy, maps, medical systems, …)
  - Huge overheads with Distributed approach

- **Next Steps:**
  - Study the golden ratio
  - Develop a cost model
  - Generalize



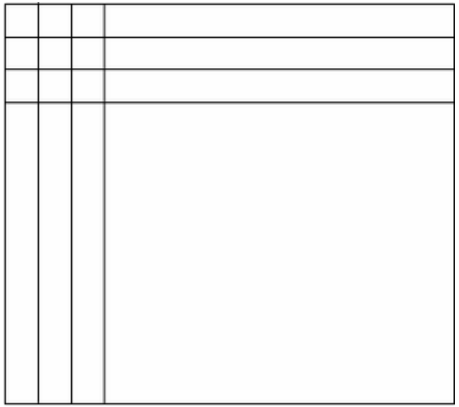
Matrix-Matrix-Multiplications

Case Study #2: Matrix Multplications

- Classic Vector and Matrix operations: VxV, MxV, MxM

- Example:

```
do i=1,n
  do j=1,n
    do k=1,n
      a(i,j) = a(i,j) + b(i,k)*c(k,j)
    enddo
  enddo
enddo
```

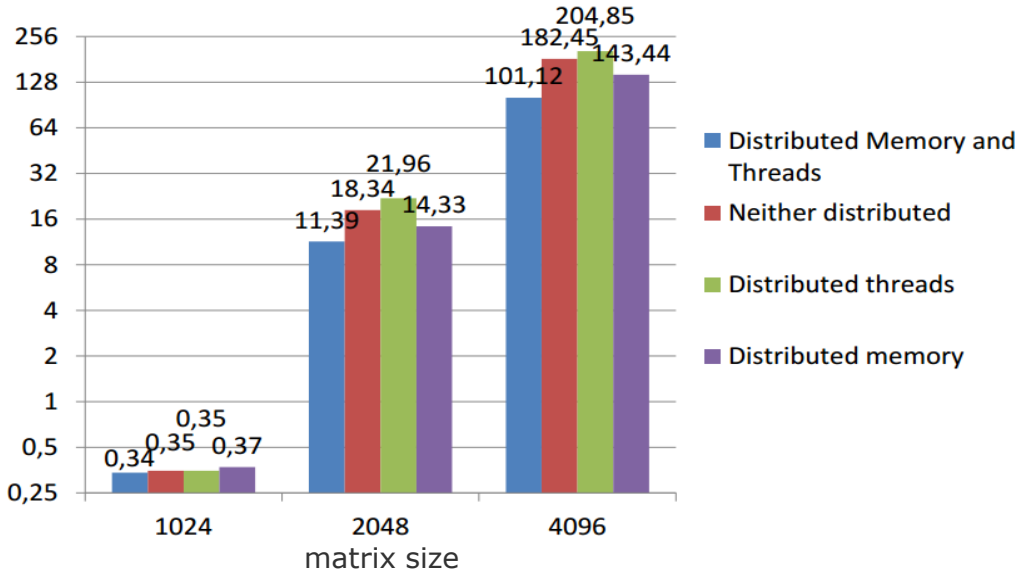


Represents Performance Bottleneck #1 in Berkeley Taxonomy: „Dense Linear Algebra"

- **Data layout**: continuous array
- **Computation**: on elements, rows, columns or matrix blocks
- **Uniprocessor Mapping**: block algorithms to exploit cache
- **Parallel Mapping**:
  - **Issues**: memory hierarchy, data distribution for load balancing critical
  - **Best**: 2D block cyclic distributions and computation/communication overlap

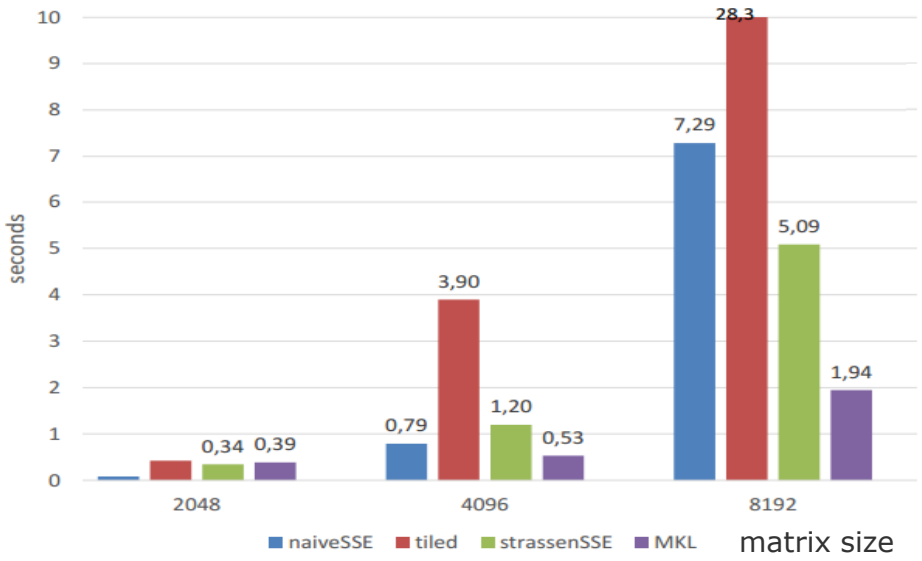
Execution time of thread and memory placements on an 8-node NUMA system with 128

- Hardware is highly optimized for algorithm like this: caching, prefetching,…
- Thus the penalty for ignoring NUMA is only factor 1.5x to 2x

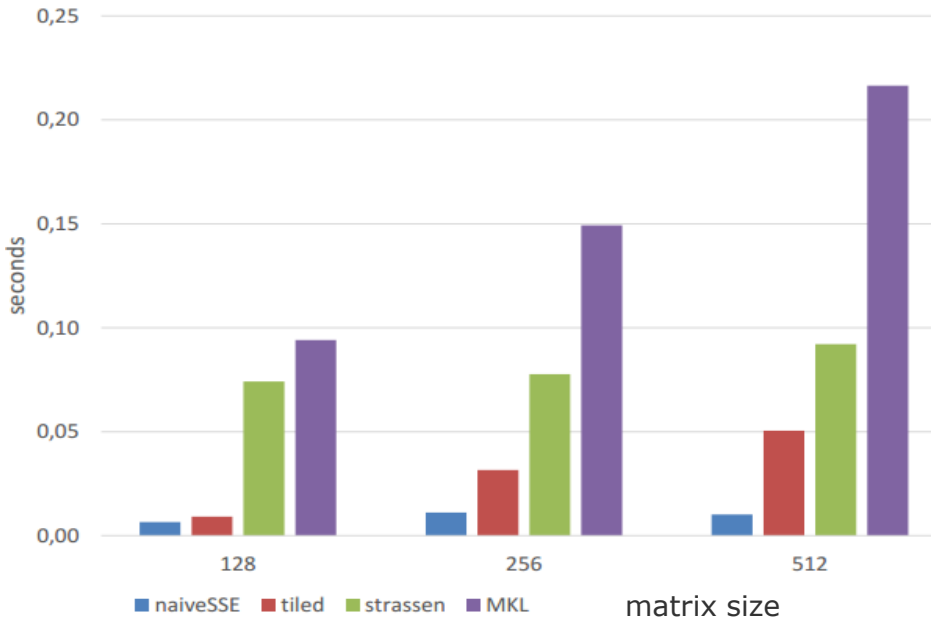


Execution time of naive, SSE-based, Strassen, and MKL matrix multiplications for larger matrices.

- Intel provides highly optimized implementations in the Math Kernal Library (MKL)
- MxM implementation is a collection of algorithms -> the best is selected



Execution time of naive, SSE-based, Strassen, and MKL matrix multiplications for small matrices.



- **Lessons Learned**:
  - With hierarchical NUMA there is room for improvement everywhere

Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

HYRISE

Planned Case Study #5: In-Memory DB Technology

- Combined column and row store
- Map/Reduce
- Single and multi-tenancy
- Lightweight compression
- Insert only for time travel
- Real-time replication
- Working on integers
- SQL interface on columns and rows
- Active/passive data store
- Minimal projections
- Group key
- Reduction of software layers
- Dynamic multi-threading
- Bulk load of data
- Object-relational mapping
- Text retrieval and extraction engine
- No aggregate tables
- Data partitioning
- Any attribute as index
- No disk
- On-the-fly extensibility
- Analytics on historical data
- Multi-core/ parallelization

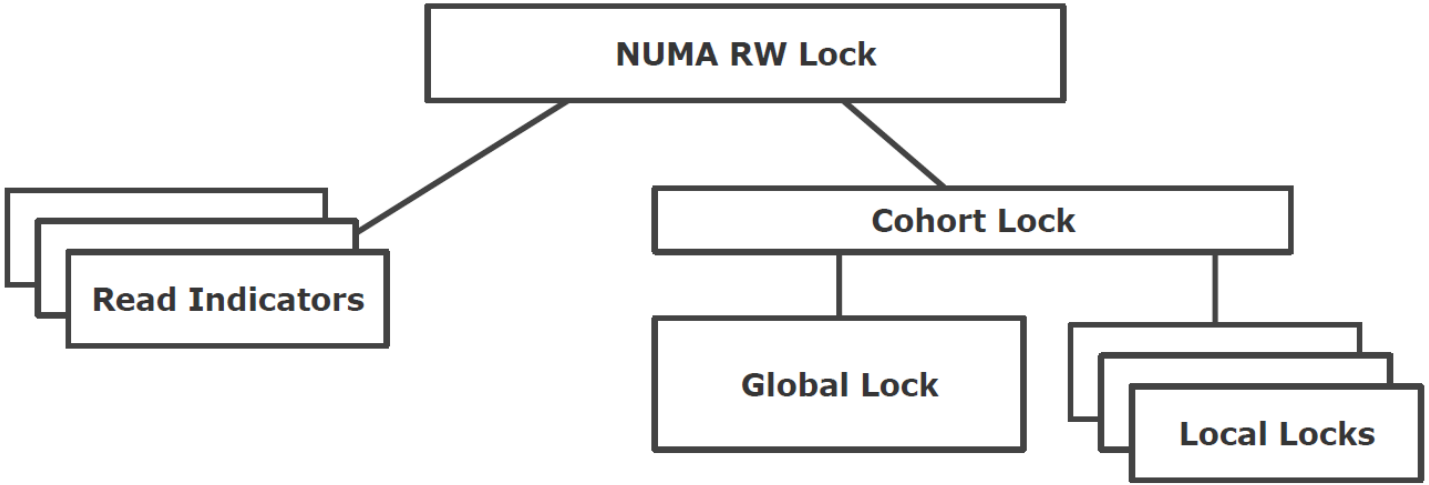
Ideas

- Collect benchmarks with different workload characteristics
- Measure execution of workloads
- Review hotspots and bad placement of threads and data
- Extend memory allocator, thread scheduler

Feature Benchmarks / Experiments

Reader/Writer Locks

NUMA-aware Reader/Writer Locks



This can be implemented „under the hood"

- One Read Indicator Counter per NUMA Node
- One Global Lock
- Plus one local lock per NUMA core

"Perf mem rec" for Pthread RW Lock

```
Samples: 64K of event 'cpu/mem-loads/pp', Event count (approx.): 1322361
 52,27%          58183  L1 hit
 18,35%           1969  LFB hit
 10,80%            787  L3 miss
 10,02%           2362  L2 hit
  5,38%            511  L3 hit
  2,06%             86  Remote Cache (1 hop) hit
  1,01%            555  Uncached hit
  0,10%              3  Remote RAM (1 hop) hit
```

Execution time: 9.097s

"Perf mem rec" for NUMA RW Lock with CK

```
Samples: 89K of event 'cpu/mem-loads/pp', Event count (approx.): 942862
 80,00%          88295  L1 hit
  9,17%            524  LFB hit
  7,35%            699  L2 hit
  2,33%            141  L3 miss
  0,82%            101  L3 hit
  0,32%            181  Uncached hit
```

Execution time: 7.783s (85.6%)

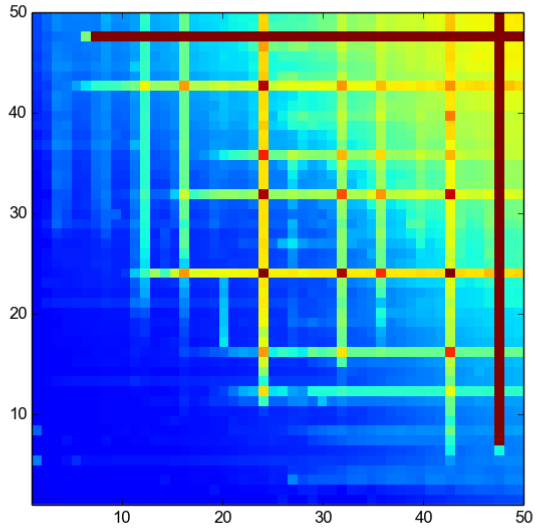
Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

SSE Anomalies + Prefetching

Performance Anomalies for the Gaussian Blur: Non-Vectorized version, 15 threads, single socket

- Evaluation of input images ranging from 1000x1000 pixels to 50000x50000 pixels
- Observation (example):
  - 11999x11999: good performance
  - 12000x12000: 5 to 6 times slower!
  - 12001x12001: good performance
- Massive cache misses on store operations occur for particular image sizes resulting in enormous performance breakdown (bars in the heatmap)
- This anomaly occurred for various stencil sizes (we evaluated 9x9 to 21x21)

Heatmap: Average time per pixel



x/y: width/height of the image in kilopixels

Color: execution time per megapixel

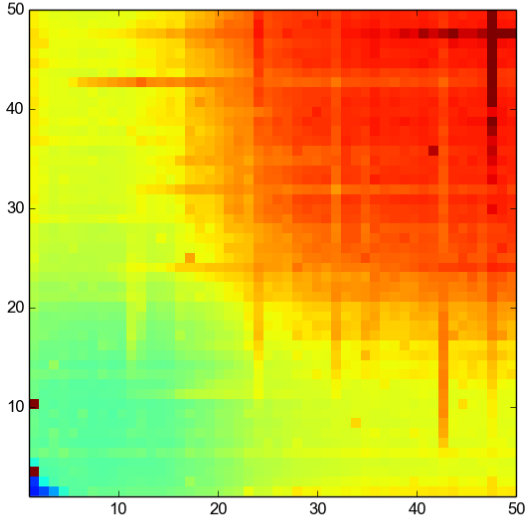
Blue ~ 10ms/megapixel; Red >= 100ms

Bars: enormous **abrupt** performance breakdowns

Performance Anomalies for the Gaussian Blur: AVX2 Vectorized version, 15 threads, single socket

- Evaluation of input images ranging from 1000x1000 pixels to 50000x50000 pixels
- More than 5 times faster than Non-Vectorized
- Observation:
  - Same anomalies as in the non-vectorised version
- Massive cache misses on store operations occur for particular image sizes resulting in enormous performance breakdown (bars in the heatmap)
- This anomaly occurred for various stencil sizes (we evaluated 9x9 to 21x21)

Heatmap: Average time per pixel



x/y: width/height of the image in kilopixels

Color: execution time per megapixel

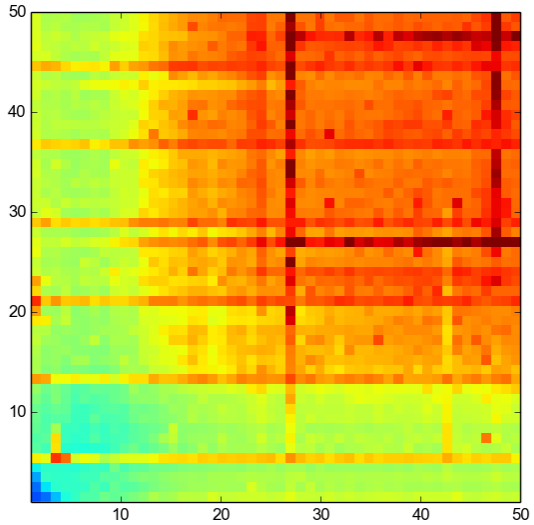
Blue ~ 1ms/megapixel; Red >= 20ms

Bars: enormous **abrupt** performance breakdowns

Performance Anomalies for the Gaussian Blur: AVX2 Vectorized version, 30 threads, single socket

- Evaluation of input images ranging from 1000x1000 pixels to 50000x50000 pixels
- Using hyperthreads: more than 6 times faster than Non-Vectorized
- Observation:
  - Same anomalies as in the non-vectorised version
- Massive cache misses on store operations occur for particular image sizes resulting in enormous performance breakdown (bars in the heatmap)
- This anomaly occurred for various stencil sizes (we evaluated 9x9 to 21x21)

Heatmap: Average time per pixel



x/y: width/height of the image in kilopixels

Color: execution time per megapixel

Blue ~ 1ms/megapixel; Red >= 15ms

Bars: **asymmetric** enormous **abrupt** performance breakdowns

Stream Stores vs. Coherent Stores

Intel Transactional Memory

Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Object Orientation

Object Orientation

Runtimes:

- Use one JVM per Node:
  - Two JVMs on two nodes -> 54% faster
  - Four JVMs on four nodes -> 79% faster

- Objects are different from float arrays!
  - Huge overhead to copy / move objects in hierarchical NUMA systems
  - Various algorithms (methods) work on objects and collections of objects, each with its own NUMA friendly distribution requirements
  - Even worse: Dictionary-based object implementations (JavaScript)

Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Linux Kernel Experiments

Linux Kernel Experiments

- Compare different kernel versions with NUMA balancing on/off
- What can be done to solve the problems of higher layers?
  - Design extensions to the thread and data allocation API with respect to NUMA systems
  - Design necessary runtime collection of metrics for placement decisions
- Malloc anomalies: Linux start to zero pages, when we ask for them
  - Instead of not doing so OR doing so, when it is idle…

- Get creative! :)

Cooperating with Fujitsu?

Performance Prediction / Planning

Task / Data Mapping and Performance rating

- How many Threads do I need to start? Where?
- How do I need to distribute = move/copy the data?
- Queuing theory applied to hybrid systems, demonstrates the feasibility for CPU / GPU scenarios.
  - Can this be applied to hierarchical NUMA as well? How?
  - Are there other similar theories / models / methods?
- Related Work
  - IBM Paper: http://www.ac.uma.es/~siham/pact09_workstealing.pdf
  - "The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling"

Programming Parallel and Distributed Systems: Seminar Project Proposals

Frank Feinbube, Felix Eberhardt, Max Plauth, Prof. Andreas Polze
Operating Systems and Middleware Research Group
Hasso Plattner Institute

Overview

Themenwünsche bis zum 06.05.2015 per E-Mail an Frank.Feinbube@hpi.de oder einfach bei uns vorbeikommen :)

Hasso Plattner Institut

Algorithm Optimization

- EDC Graph Search, Hyrise
- Speeded Up Robust Features
- … or anything really

Feature Benchmarks

- Intel Transactional Memory
- Stream Stores vs. Coherent Stores
- Prefetching

Tools

Programming Languages / Models

- PGAS: Fortress, X10, UPC, …
- Scala, Java, JavaScript, C#, …
- CUDA, OpenCL, OpenACC, …

Platforms

- (hierarchical) NUMA systems
- Intel Xeon Phi
- GPU Computing

Linux Kernel experiments

Performance Predictions