

PERFORMANCE ANALYSIS IN A NUTSHELL AN INTRO WITH SCORE-P, SCALASCA, AND VAMPIR

OCTOBER 13, 2022 I MICHAEL KNOBLOCH



Mitglied der Helmholtz-Gemeinschaft

DISCLAIMER

Tools will *not* automatically make you, your applications or computer systems more productive.

However, they can help you understand **how** your parallel code executes and **when / where** it's necessary to work on correctness and performance issues.



PERFORMANCE: AN OLD PROBLEM

3



Difference Engine

"The most constant difficulty in contriving the engine has arisen from the desire to reduce the time in which the calculations were executed to the shortest which is possible."

> Charles Babbage 1791 – 1871

> > NG

TODAY: THE "FREE LUNCH" IS OVER

- Moore's law is still in charge, but
 - Clock rates no longer increase
 - Performance gains only through increased parallelism
- Optimizations of applications more difficult
 - Increasing application complexity
 - Multi-physics
 - Multi-scale
 - Increasing machine complexity
 - Hierarchical networks / memory
 - More CPUs / multi-core
 - Accelerators
 - Modular supercomputer architecture
- Every doubling of scale reveals a new bottleneck!



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten New plot and data collected for 2010-2015 by K. Rupp



TUNING BASICS

- Successful performance engineering is a combination of
 - Careful setting of various tuning parameters
 - The right algorithms and libraries
 - Compiler flags and directives
 - Correct machine usage (mapping and bindings)
 - ...
 - Thinking !!!
- Measurement is better than guessing
 - To determine performance bottlenecks
 - To compare alternatives
 - To validate tuning decisions and optimizations
 - After each step!
- Modeling is extremely useful but very difficult and rarely available
 - Allows to evaluate performance impact of optimization without implementing it
 - Simplifies search in large parameter space



PERFORMANCE ENGINEERING WORKFLOW





PERFORMANCE METRICS

- What can be measured?
 - A count of how often an event occurs
 - E.g., the number of MPI point-to-point messages sent
 - The duration of some interval
 - E.g., the time spent these send calls
 - The size of some parameter
 - E.g., the number of bytes transmitted by these calls
- Derived metrics
 - E.g., rates / throughput
 - Needed for normalization

CENTRE

EXAMPLE METRICS

- Execution time
- Number of function calls
- CPI
 - CPU cycles per instruction
- FLOPS
 - Floating-point operations executed per second



EXECUTION TIME

- Wall-clock time
 - Includes waiting time: I/O, memory, other system activities
 - In time-sharing environments also the time consumed by other applications
- CPU time
 - Time spent by the CPU to execute the application
 - Does not include time the program was context-switched out
 - Problem: Does not include inherent waiting time (e.g., I/O)
 - Problem: Portability? What is user, what is system time?
- Problem: Execution time is non-deterministic
 - Use mean or minimum of several runs



CLASSIFICATION OF MEASUREMENT TECHNIQUES

- How are performance measurements triggered?
 - Sampling
 - Code instrumentation
- How is performance data recorded?
 - Profiling / Runtime summarization
 - Tracing
- How is performance data analyzed?
 - Online No suitable tools anymore
 - Post mortem







Forschungszentrum

INSTRUMENTATION



- Can be done in various ways
- Advantage:
 - Much more detailed information
- Disadvantage:
 - Processing of source-code / executable necessary
 - Large relative overheads for small functions

```
{
    int i;
    Enter("main");
    for (i=0; i < 3; i++)
        foo(i);
    Leave("main");
    return 0;
}
void foo(int i)
{
    Enter("foo");
    if (i > 0)
        foo(i - 1);
    Leave("foo");
}
```

Forschungszentrum

JÜLICH

CENTRE

SUPERCOMPUTING

INSTRUMENTATION TECHNIQUES

- Static instrumentation
 - Program is instrumented prior to execution
- Dynamic instrumentation
 - Program is instrumented at runtime
- Code is inserted
 - Manually
 - Automatically
 - By a preprocessor / source-to-source translation tool
 - By a compiler
 - By linking against a pre-instrumented library / runtime system
 - By binary-rewrite / dynamic instrumentation tool



CRITICAL ISSUES

- Accuracy
 - Intrusion overhead
 - Measurement itself needs time and thus lowers performance
 - Perturbation
 - Measurement alters program behaviour
 - E.g., memory access pattern
 - Accuracy of timers & counters
- Granularity
 - How many measurements?
 - How much information / processing during each measurement?
- Tradeoff: Accuracy vs. Expressiveness of data



CLASSIFICATION OF MEASUREMENT TECHNIQUES

- How are performance measurements triggered?
 - Sampling
 - Code instrumentation
- How is performance data recorded?
 - Profiling / Runtime summarization
 - Tracing
- How is performance data analyzed?
 - Online
 - Post mortem



PROFILING / RUNTIME SUMMARIZATION

- Recording of aggregated information
 - Total, maximum, minimum, ...
- For measurements
 - Time
 - Counts
 - Function calls
 - Bytes transferred
 - Hardware counters
- Over program and system entities
 - Functions, call sites, basic blocks, loops, ...
 - Processes, threads
- Profile = summarization of events over execution interval



TYPES OF PROFILES

- Flat profile
 - Shows distribution of metrics per routine / instrumented region
 - Calling context is not taken into account
- Call-path profile
 - Shows distribution of metrics per executed call path
 - Sometimes only distinguished by partial calling context (e.g., two levels)
- Special-purpose profiles
 - Focus on specific aspects, e.g., MPI calls or OpenMP constructs
 - Comparing processes/threads



TRACING

- Recording detailed information about significant points (events) during execution of the program
 - Enter / leave of a region (function, loop, ...)
 - Send / receive a message, ...
- Save information in event record
 - Timestamp, location, event type
 - Plus event-specific information (e.g., communicator, sender / receiver, ...)
- Abstract execution model on level of defined events
- Event trace = Chronologically ordered sequence of event records





TRACING PROS & CONS

- Tracing advantages
 - Event traces preserve the temporal and spatial relationships among individual events (* context)
 - Allows reconstruction of dynamic application behavior on any required level of abstraction
 - Most general measurement technique
 - Profile data can be reconstructed from event traces
- Disadvantages
 - Traces can very quickly become extremely large
 - Writing events to file at runtime may causes perturbation



TECHNOLOGIES AND THEIR INTEGRATION



REMARK: NO SINGLE SOLUTION IS SUFFICIENT!



A combination of different methods, tools and techniques is typically needed!









SCORE-P AND SCALASCA



SCORE-P

- Infrastructure for instrumentation and performance measurements
- Instrumented application can be used to produce several results:
 - Call-path profiling: CUBE4 data format used for data exchange
 - Event-based tracing: OTF2 data format used for data exchange
- Supported parallel paradigms:
 - Multi-process: MPI, SHMEM
 - Thread-parallel: OpenMP, Pthreads
 - Accelerator-based: CUDA, OpenCL, OpenACC, ROCm, Kokkos
- Open Source; portable and scalable to all major HPC systems
- Initial project funded by BMBF
- Further developed in multiple 3rd-party funded projects









SCORE-P OVERVIEW



Mitglied der Helmholtz-Gemeinschaft

CUBE

- Parallel program analysis report exploration tools
 - Libraries for XML+binary report reading & writing
 - Algebra utilities for report processing
 - GUI for interactive analysis exploration
 - Requires Qt4 ≥4.6 or Qt 5
- Originally developed as part of the Scalasca toolset
- Now available as a separate component
 - Can be installed independently of Score-P, e.g., on laptop or desktop
 - Latest release: Cube v4.6 (April 2021)





ANALYSIS PRESENTATION AND EXPLORATION - CUBE

- Representation of values (severity matrix) on three hierarchical axes
 - Performance property (metric)
 - Call path (program location)
 - System location (process/thread)
- Three coupled tree browsers
- Cube displays severities
 - As value: for precise comparison
 - As colour: for easy identification of hotspots
 - Inclusive value when closed & exclusive value when expanded
 - Customizable via display modes







ANALYSIS PRESENTATION



CENTRE

Forschungszentrum

Mitglied der Helmholtz-Gemeinschaft

AUTOMATIC TRACE ANALYSIS

- Idea
 - Automatic search for patterns of inefficient behaviour
 - Classification of behaviour & quantification of significance
 - Identification of delays as root causes of inefficiencies



- Guaranteed to cover the entire event trace
- Quicker than manual/visual trace analysis
- Parallel replay analysis exploits available memory & processors to deliver scalability



SCALASCA TRACE TOOLS: OBJECTIVE

- Development of a scalable trace-based performance analysis toolset for the most popular parallel programming paradigms
 - Current focus: MPI, OpenMP, and (to a limited extend) POSIX threads
- Specifically targeting large-scale parallel applications
 - Demonstrated scalability up to 1.8 million parallel threads
 - Of course also works at small/medium scale
- Latest release:
 - Scalasca v2.6 coordinated with Score-P v7.0 (April 2021)



SCALASCA TRACE TOOLS: FEATURES

- Open source, 3-clause BSD license
- Fairly portable
 - IBM Blue Gene, Cray XT/XE/XK/XC, SGI Altix, Fujitsu FX systems, Linux clusters (x86, Power, ARM), Intel Xeon Phi, ...
- Uses Score-P instrumenter & measurement libraries
 - Scalasca v2 core package focuses on trace-based analyses
 - Supports common data formats
 - Reads event traces in OTF2 format
 - Writes analysis reports in CUBE4 format
- Current limitations:
 - Unable to handle traces
 - with MPI thread level exceeding MPI_THREAD_FUNNELED
 - containing Memory events, CUDA/OpenCL device events (kernel, memcpy), SHMEM, or OpenMP nested parallelism
 - PAPI/rusage metrics for trace events are ignored



SCALASCA WORKFLOW



EXAMPLE: "LATE SENDER" WAIT STATE



- Waiting time caused by a blocking receive operation posted earlier than the corresponding send
- Applies to blocking as well as non-blocking communication



EXAMPLE: CRITICAL PATH



- Shows call paths and processes/threads that are responsible for the program's wall-clock runtime
- Identifies good optimization candidates and parallelization bottlenecks



EXAMPLE: ROOT-CAUSE ANALYSIS



- Classifies wait states into direct and indirect (i.e., caused by other wait states)
- Identifies *delays* (excess computation/communication) as root causes of wait states
- Attributes wait states as *delay costs*



TRACE ANALYSIS REPORT





EVENT TRACE VISUALIZATION WITH VAMPIR

- Visualization of dynamic runtime behaviour at any level of detail along with statistics and performance metrics
- Alternative and supplement to automatic analysis
- Typical questions that Vampir helps to answer
 - What happens in my application execution during a given time in a given process or thread?
 - How do the communication patterns of my application execute on a real system?
 - Are there any imbalances in computation, I/O or memory usage and how do they affect the parallel execution of my application?

Timeline charts

 Application activities and communication along a time axis



Summary charts

 Quantitative results for the currently selected time interval



CENTRE

VAMPIR DISPLAYS







SCORE-P/CUBE CASE STUDY - HEMELB



10/11/2022ed der Helmholtz-Gemeinschaft

HEMELB (SUPERMUC-NG: NO GPUS)

- 3D macroscopic blood flow in human arterial system developed by UC London (UK)
 - lattice-Boltzmann method tracking fluid particles on a lattice grid with complex boundary conditions
 - exascale flagship application of EU H2020 HPC Centre of Excellence for Computational Biomedicine
- HemeLB open-source code and test case: www.hemelb.org
 - C++ parallelized with MPI [+ CUDA unused]
 - Intel Studio 2019u4 compiler and MPI library (v19.0.4.243)
 - configured with 2 'reader' processes (intermediate MPI file writing disabled)
 - MPI-3 shared-memory model employed within compute nodes to reduce memory requirements when distributing lattice blocks from reader processes
 - Focus of analysis 5,000 time-step (500µs) simulation of cerebrovascular "circle of Willis" geometry
 - 6.4µm lattice resolution (21.15 GiB): 10,154,448,502 lattice sites
- Executed on SuperMUC-NG Lenovo ThinkSystem SD650 (LRZ):
 - 2x 24-core Intel Xeon Platinum 8174 ('Skylake') @ 3.1GHz
 - 48 MPI processes/node, 6452 (of 6480) compute nodes: 309,696 MPI processes
 - 190x speed-up from 864 cores: 80% scaling efficiency to over 100,000 cores
- ⇒ Identification & quantification of impact of load balance and its variation











HEMELB@SNG STRONG SCALING OF FOA RUNSIMULATION



HEMELB@SNG STRONG SCALING EFFICIENCY OF FOA RUNSIMULATION

Compute nodes	24	32	48	64	96	128	192	256	384	512	768	1024	1536	2048	3072	4096	6452
Processes	1152	1536	2304	3072	4608	6144	9216	12288	18432	24576	36864	49152	73728	98304	147456	196608	309696
Global scaling efficiency	0.79	0.79	0.84	0.80	0.82	0.75		0.73	0.72	0.73	0.74	0.68	0.68	0.65	0.62	0.57	0.45
- Parallel efficiency	0.79	0.80	0.87	0.83	0.86	0.80		0.75	0.74	0.74	0.77	0.71	0.72	0.70	0.72	0.70	0.73
Load balance efficiency	0.79	0.80	0.88	0.84	0.86	0.80		0.75	0.74	0.75	0.78	0.72	0.74	0.72	0.74	0.73	0.80
Communication efficiency	1.00	1.00	1.00	1.00	1.00	1.00		1.00	1.00	0.99	0.99	0.99	0.98	0.98	0.97	0.96	0.92
- Computation scaling	1.00	0.99	0.96	0.96	0.95	0.93		0.98	0.98	0.98	0.96	0.96	0.94	0.93	0.87	0.81	0.61
Instructions scaling	1.00	1.00	1.00	1.00	1.00	1.00		1.00	1.00	1.00	0.99	0.97	0.94	0.89	0.79	0.67	0.45
IPC scaling	1.00	0.99	0.96	0.96	0.95	0.93		0.98	0.98	0.99	0.98	0.99	1.00	1.04	1.11	1.21	1.36
IPC	1.411	1.395	1.353	1.355	1.342	1.316		1.377	1.387	1.396	1.383	1.390	1.417	1.473	1.566	1.704	1.919
											Key:	<0.65	<0.75	<0.85	<0.95	<1.00	>1.00

Global scaling efficiency fairly good around 80%, before degrading at larger scales

- Parallel efficiency deteriorating following Load balance efficiency
 - Communication efficiency excellent throughout
- Computation scaling (relative to 1152 processes) very good except at largest scale
 - Degradation of Instructions scaling partially compensated by improving IPC scaling

[POP CoE scaling efficiency model: www.pop-coe.eu]



INITIAL TREE PRESENTATION: TIME OF MPI_GATHER PER MPI PROCESS



SUPERCOMPUTING CENTRE

TOPOLOGICAL PRESENTATION: STALLS MEM ANY FOR HANDLEACTORS



ADVISOR: POP EFFICIENCY ASSESSMENT FOR RUNSIMULATION



Mitglied der Helmholtz Ready

JÜLICH SUPERCOMPUTING CENTRE

3D macroscopic blood flow in human arterial system developed by UC London (UK) Jattice-Boltzmann method tracking fluid particles on a lattice grid with complex how

- lattice-Boltzmann method tracking fluid particles on a lattice grid with complex boundary conditions
- exascale flagship application of EU H2020 HPC Centre of Excellence for Computational Biomedicine
- HemeLB open-source code and test case: www.hemelb.org
 - C++ parallelized with MPI + CUDA (in development)
 - GCC/8.3.0 compiler, CUDA/10.1.105 and ParaStationMPI/5.4 library
 - configured with 2 'reader' processes and intermediate MPI file writing
 - rank 0 'monitor' process doesn't participate in simulation



- Focus of analysis 2,000 time-step (each 100µs) simulation of CBM2019_Arteries_patched geometry
 - 1.78 GiB: 66,401,494 lattice sites, 1+38 iolets
- Executed on JUWELS-Volta (@JSC):
 - 2x 20-core Intel Xeon Platinum 8168 ('Skylake') CPUs + 4 Nvidia V100 'Volta' GPUs
 - 4* MPI processes/node (one per GPU), 32 (of 56) compute nodes: 129 MPI processes

⇒ Identification & quantification of impact of load balance and its variation





HEMELB (JUWELS-VOLTA)

TREE: TIME FOR ASYNCH. CUDA KERNELS ON SEPARATE CUDA STREAMS

CubeGUI-4.5.0: scorep_hemepure_gpu_20a+IO_129_trace\summary.cubex File Display Plugins Help Absolute Absolute Absolute > System Metric tree Call tree Flat view System tree 🚺 Statistics 14974.45 Time (sec) ✓ □ 0.83 hemepure_gpu_20a.scorep ➤ □ - machine Linux View 1.51e7 Visits (occ) ✓ □ 0.02 main ✓ □ - node jwc09n033.adm09.juwels.fzj.de 5.10e11 Bytes transferred (bytes) 0.00 MPI Initialized ✓ □ - MPI Rank 0 202268 MPI file operations (occ) 122.79 MPI Init 11.73 Master thread Topologies 269.69 Computational imbalance (sec) 12890.80 SimulationMaster ✓ □ - MPI Rank 1 1.86e9 io bytes read (bytes) 1324.46 RunSimulation 9.48 Master thread >
 193.65 ~ Simulation Master 7.44e9 io bytes written (bytes) 0.00 CUDA[0:7] 8.91 MPI Finalize 0.03 CUDA[0:17] General 0.40 BUFFER FLUSH 2.22 CUDA[0:23] 0.41 CUDA[0:26] 396.53 hemelb::GPU CollideStream mMidFluidCollision mWallCollision sBE 13.12 hemelb::GPU CollideStream Iolets NashZerothOrderPressure 0.41 CUDA[0:28] 14.74 hemelb::GPU CollideStream wall sBB iolet Nash 0.08 CUDA[0:31] 8.21 hemelb::GPU StreamReceived sistr ✓ □ - MPI Rank 2 9.48 Master thread 0.00 CUDA[1:7] 0.05 CUDA[1:17] 2.34 CUDA[1:23] 0.36 CUDA[1:26] 0.35 CUDA[1:28] 0.01 CUDA[1:31] > 11.99 MPI Rank 3 > 🗖 11.92 MPI Rank 4 ✓ □ - node jwc09n036.adm09.juwels.fzj.de > 12.48 MPI Rank 5 > 11.23 MPI Rank 6 > 11.35 MPI Rank 7 > 14.40 MPI Rank 8 ✓ □ - node jwc09n039.adm09.juwels.fzj.de > 🗖 12.60 MPI Rank 9 > 🗖 13.32 MPI Rank 10 > 11.54 MPI Rank 11 > 14.03 MPI Rank 12 All (715 elements) 0.00 14974.45 (100.00%) 14974.45 0.00 1757.06 (12.08%) 14541.85 0.00 1757.06 JÜLICH

SUPERCOMPUTING **CENTRE**





TOPO: TIME FOR ASYNCH. CUDA KERNELS ON SEPARATE CUDA STREAMS



Mitglied der Helmholtz Ready

TOPO: TIME FOR MPI FILE WRITING ON CPU VARIES PER MPI PROCESS



SUPERCOMPUTING

TOPO: TIME FOR CUDA ASYNCHRONOUS MEMORY COPIES IS IMBALANCED



Mitglied der Helmholtz Ready

SUPERCOMPUTING

HEMELB@JUWELS-VOLTA STRONG SCALING OF FOA RUNSIMULATION



- Reference execution with 8ppn
 - multiple processes offloading GPU kernels generally unproductive
- Comparison of versions (4ppn)
 - v1.20a generally better
- Synchronous MPI file writing is the primary bottleneck
- CUDA kernels on GPUs
 - less than half of Simulation time (therefore GPUs mostly idle)
 - total kernel time scales very well (0.93 scaling efficiency)
 - load balance deteriorates (0.95 for single node, 0.50 for 32 nodes)



HEMELB@JUWELS/VOLTA STRONG SCALING EFFICIENCY OF RUNSIMULATION

Simulation time [s]	1n 5p 147.87	2n 9p 88.38	4n 17p 48,13	8n 33p 22.66	16n 65p 13.68	32n 129p 11.67
Global scaling efficiency	0.64	0.53	0.49	0.52	0.43	0.25
– Parallel efficiency	0.64	0.53	0.50	0.54	0.47	0.29
– – Load balance efficiency (GPU)	0.95	0.78	0.73	0.73	0.65	0.50
 – Communication efficiency (GPU) 	0.67	0.68	0.68	0.75	0.73	0.58
 Computation scaling (GPU) 	1.00	1.00	0.99	0.96	0.92	0.87

Only considering GPUs (ignoring all CPU cores, 90% of which are completely unused)

- Single (quad-GPU) node already suffers significant communication inefficiency
 - includes MPI file writing, but doesn't degrade much as additional nodes are included
- Load balance of GPUs deteriorates progressively
- GPU computation scaling remains reasonably good [POP CoE scaling efficiency model: www.pop-coe.eu] Mitglied der Helmholtz-Gemeinschaft



Key:

HEMELB@JUWELS-VOLTA STRONG SCALING OF FOA RUNSIMULATION





SCALASCA CASE STUDY – TEA LEAF



28/04/2016d der Helmholtz-Gemeinschaft

CASE STUDY: TEALEAF

- HPC mini-app developed by the UK Mini-App Consortium
 - Solves the linear 2D heat conduction equation on a spatially decomposed regular grid using a 5 point stencil with implicit solvers
 - Part of the Mantevo 3.0 suite
 - Available on GitHub: https://uk-mac.github.io/TeaLeaf/
- Measurements of TeaLeaf reference v1.0 taken on Jureca cluster @ JSC
 - Using Intel 19.0.3 compilers, Intel MPI 2019.3, Score-P 5.0, and Scalasca 2.5
 - Run configuration
 - 8 MPI ranks with 12 OpenMP threads each
 - Distributed across 4 compute nodes (2 ranks per node)
 - Test problem "5": 4000 × 4000 cells, CG solver

SCALASCA ANALYSIS REPORT **EXPLORATION (OPENING VIEW)**



CENTRE

Forschungszentrum

Additional top-level

trace analysis...

SCALASCA WAIT-STATE METRICS



Forschungszentrum



TEALEAF SCALASCA REPORT ANALYSIS (I)

While MPI communication time and wait states are small (~0.6% of the total execution time)...



Mitglied der Helmholtz-Gemeinschaft

TEALEAF SCALASCA REPORT ANALYSIS (II)



JULICH

Forschungszentrum

SUPERCOMPUTING

CENTRE

Absolute	 Absolute 		Ŧ	Absolute	
Metric tree	🔚 Call t	ree 🔲 Flat view		System tree 🚺 Statistics	🚺 🖸 S
 735.75 Idle threads 1.17e8 Visits (occ) 72 MPI synchronizations (occ) 0 MPI pair-wise one-sided synchroit and synchronications (occ) 1.29e6 MPI communications (occ) 2.37e10 MPI bytes transferred (b) 0.00 Delay costs (sec) 0.00 Delay costs (sec) 0.00 Point-to-point 4.57 Late Sender 0.26 Late Receiver 0.00 Collective 0.00 OpenMP 33.17 Wait at N x N 0.00 OpenMP 59.82 Wait at Barrier 0.00 Idleness delay costs 354.84 Short-term 87.42 Long-term 5.13 MPI point-to-point wait state 99.90 Critical path (sec) 198.95 Computational imbalance 	s (propaga s (direct v c) ((sec)	 7.00 tea_module.tea_exchange 4.76 tea_module.tea_pack_r 1.98 tea_module.tea_send_r 2.99 MPI_frecv 56.82 MPI_Waitall 4.79 tea_module.tea_unpach 6.85 tea_module.tea_pack_t 1.25 tea_module.tea_send_r 1.25 tea_module.tea_unpach 4.95 MPI_frecv 7.10 tea_module.tea_unpach 4.87 tea_module.tea_pack_h 1.92 tea_module.tea_unpach 4.87 tea_module.tea_unpach 4.87 tea_module.tea_unpach 4.87 tea_module.tea_unpach 4.87 tea_module.tea_unpach 4.87 tea_module.tea_unpach 4.87 tea_module.tea_unpach 4.89 tea_module.tea_unpach 3.13 MPI_frecv 4.59 tea_module.tea_unpach 6.98 tea_module.tea_unpach 6.96 tea_module.tea_unpach 3.83 update_halo_kernel_modu 3.55 tea_leaf_kernel_cg_module.tea_unpach 9.87 tea_module.tea_allsum_ 54.90 MPI_Alreduce 	e_ right_ recv_message_right_ k_right_ top_ recv_message_top_ k_top_ eft_ recv_message_left_ k_left_ bottom_ recv_message_botto k_bottom_ le.update_halo_kern tea_leaf_kernel_init_	 0.00 machine Linux 0.00 node jrc0280 24.77 MPI Rank 24.21 MPI Rank 0.00 node jrc0281 20.93 MPI Rank 21.55 MPI Rank 0.00 node jrc0282 23.46 MPI Rank 24.15 MPI Rank 0.00 node jrc0283 19.39 MPI Rank 19.39 MPI Rank 20.40 MPI Rank 20.40 MPI Rank 	0 1 2 3 4 5 6 7

...they directly cause a significant amount of the OpenMP thread idleness

TEALEAF SCALASCA REPORT ANALYSIS (III)

The "Wait at NxN" collective wait states are mostly caused by the first 2 OpenMP do loops of the solver (on ranks 5 & 1, resp.)...



TEALEAF SCALASCA REPORT ANALYSIS (IV)



...while the MPI pointto-point wait states are caused by the 3rd solver do loop (on rank 1) and two loops in the halo exchange



TEALEAF SCALASCA REPORT ANALYSIS (V)

SUPERCOMPUTING

CENTRE

Forschungszentrum

CubeGUI-4.4.3: scorep_tea_leaf_baseline_8x12_trace/trace.cubex <@jri11> File Display Plugins Help Restore Setting * Save Settings Absolute Absolute **v** Absolute Statistics E Flat view System tree O Su Metric tree Call tree 0.00 machine Linux T35.75 Idle threads 0.57 pack kernel module.tea pack message 5.68 !\$omp parallel @pack kernel.f90:61 - 0.00 node jrc0280 1.17e8 Visits (occ) • 0.48 MPI Rank 0 2.28 tea module.tea send recv message left 72 MPI synchronizations (occ) 39.00 MPI Rank 1 Othe D 0 MPI pair-wise one-sided synchronizations I.60 tea module.tea unpack left - 0.00 node jrc0281 1.29e6 MPI communications (occ) I.31 tea module.tea pack bottom Image: 0.29 MPI Rank 2

 0 MPI file operations (occ)
 I.01 tea module.tea send recv message botto I 1.53 MPI Rank 3 2.37e10 MPI bytes transferred (bytes) I 0.62 tea module.tea unpack bottom 0.67 update halo kernel module.update halo kernel - 0.00 node jrc0282 0.00 Delay costs (sec) I 2.72 MPI Rank 4 D.04 tea leaf kernel cg module.tea leaf kernel init • □ 0.00 MPI I 20.14 MPI Rank 5 4.37 tea module.tea allsum
 □ 0.00 Point-to-point
 - 0.00 node jrc0283 4.57 Late Sender Image: 0.29 MPI Rank 6 0.03 tea leaf kernel cg module.tea leaf kernel solve I 0.26 Late Receiver Image: 0.33 MPI Rank 7 0.07 \$ omp parallel @tea leaf cg.f90.186
 □ 0.00 Collective
 I4.89 !somp do @tea leaf cg.f90:187 0.10 Wait at Barrier 0.02 !\$omp implicit barrier @tea leaf cg.f90:20 Image: Book and Barry D 0.00 Late Broadcast = 0.02 tea leaf kernel cg module.tea leaf kernel solve Image: somp parallel @tea leaf cg.f90:234 -
 0.00 OpenMP
 0.00
 0 59.82 Wait at Barrier 19.08 !\$omp do @tea leaf cg.f90:247 0.02 !somp implicit barrier @tea leaf cg.f90:25 □ 0.00 Idleness delay costs 354.84 Short-term 0.22 tea leaf kernel cg module.tea leaf kernel solve Image: 87.42 Long-term 20.37 !somp do @tea leaf cg.f90:294 5.13 MPI point-to-point wait states (propaga) 0.12 !somp implicit barrier @tea leaf cg.f90:30 5.13 MPI point-to-point wait states (direct v 99.90 Critical path (sec) >
 0.02 tea leaf kernel module.tea leaf kernel finalise 9590.08 Performance impact (sec) ▷ □ 0.00 field summary 198.95 Computational imbalance (sec) All (96 elements) Þ. 1 H I 540.18 0.00 64.78 (74.10%) 87.42 0.00 0.00 87.42 (16.18%) 0.00 (0.00%) 64.78 Deselected "!\$omp implicit barrier @tea_leaf_cg.f90:302" JULICH

Various OpenMP do loops (incl. the solver loops) also cause OpenMP thread idleness on other ranks via propagation

TEALEAF SCALASCA REPORT ANALYSIS (VI)



JULICH

Forschungszentrum

SUPERCOMPUTING

CENTRE

	Absolute *	Absolute	Absolute	*
	Metric tree	Call tree	System tree Statistics	Su 🔹
e Critical Path also ighlights the three solver loops	 735.75 Idle threads 1.17e8 Visits (occ) 72 MPI synchronizations (occ) 0 MPI pair-wise one-sided synchronizations 1.29e6 MPI communications (occ) 0 MPI file operations (occ) 2.37e10 MPI bytes transferred (bytes) 0.00 Delay costs (sec) 0.00 Point-to-point 4.57 Late Sender 0.26 Late Receiver 0.00 Collective 0.00 Late Broadcast 0.00 OpenMP 5.9.82 Wait at Barrier 0.00 Idleness delay costs 354.84 Short-term 87.42 Long-term 5.13 MPI point-to-point wait states (propagation of the states (direct volume) 5.13 MPI point-to-point wait states (direct volume) 99.90 Critical path (sec) 198.95 Computational imbalance (sec) 	 0.00 timer 0.00 set_field_module.set_field_ 0.00 set_field_module.timestep_ 0.08 tea_leaf_module.tea_leaf_ 0.08 tea_leaf_module.tea_leaf_ 0.01 tea_leaf_kernel_cg_module.tea_leaf_kernel_init_ 0.84 tea_module.tea_allsum_ 0.01 tea_leaf_kernel_cfey_module.tea_leaf_kernel_cd 0.03 tea_leaf_kernel_gfmodule.tea_leaf_kernel_solv 0.05 !\$omp parallel @tea_leaf_cg.f90:186 37.74 !\$omp do @tea_leaf_cg.f90:187 0.02 !\$omp implicit barrier @tea_leaf_kernel_solv 0.02 !\$omp implicit barrier @tea_leaf_kernel_solv 0.04 tea_leaf_kernel_cg_module.tea_leaf_kernel_solv 0.07 !\$omp implicit barrier @tea_leaf_cg.f90:26 0.03 !\$omp implicit barrier @tea_leaf_cg.f90:27 0.03 !\$omp implicit barrier @tea_leaf_cg.f90:284 0.07 !\$omp implicit barrier @tea_leaf_kernel_solv 0.03 !\$omp implicit barrier @tea_leaf_cg.f90:284 0.07 !\$omp implicit barrier @tea_leaf_cg.f90:30 0.03 !\$omp implicit barrier @tea_leaf_cg.f90:30 0.03 !\$omp implicit barrier @tea_leaf_cg.f90:30 0.01 tea_leaf_kernel_module.tea_leaf_kernel_solv 0.03 !\$omp implicit barrier @tea_leaf_cg.f90:30 0.03 !\$omp implicit barrier @tea_leaf_cg.f90:30 0.01 tea_leaf_kernel_module.tea_leaf_kernel_solv 	 0.00 machine Linux 0.00 node jrc0280 38.93 MPI Rank 0 38.93 MPI Rank 1 0.00 node jrc0281 13.35 MPI Rank 2 13.35 MPI Rank 3 0.00 node jrc0282 0.91 MPI Rank 4 38.44 MPI Rank 5 0.00 node jrc0283 0.26 MPI Rank 6 0.32 MPI Rank 7 	
	4 F	4	All (96 elements)	*

TEALEAF SCALASCA REPORT ANALYSIS (VII)



SUPERCOMPUTING

CENTRE

Absolute	*	Nosolute	٣	Absolute	
Metric tree		Call tree 📔 Flat view		System tree 🚺 Statistics	Su 🤇
 735.75 Idle threads 1.17e8 Visits (occ) 72 MPI synchronizations (occ) 0 MPI pair-wise one-sided synchronizations (occ) 1.29e6 MPI communications (occ) 0 MPI file operations (occ) 2.37e10 MPI bytes transferred (bytes) 0.00 Delay costs (sec) 0.00 Point-to-point 4.57 Late Sender 0.00 Collective 0.10 Wait at Barrier 0.00 OpenMP 59.82 Wait at Barrier 0.00 Idleness delay costs 354.84 Short-term 87.42 Long-term 5.13 MPI point-to-point wait states (piece) 9.41 Imbalance 959.08 Performance impact (sec) 	ropaga irect v	 0.72 MPI Waitall 0.00 tea_module.tea_unpack_right_ 0.20 tea_module.tea_pack_top_ 0.20 tea_module.tea_send_recv_message_top_ 0.21 tea_module.tea_unpack_top 0.21 tea_module.tea_unpack_top 0.22 tea_module.tea_pack_left_ 0.32 tea_module.tea_send_recv_message_left_ 0.32 tea_module.tea_unpack_left_ 0.31 tea_module.tea_pack_bottom_ 0.28 tea_module.tea_unpack_bottom_ 0.28 tea_module.tea_unpack_bottom_ 0.18 tea_module.tea_unpack_bottom_ 0.12 update_halo_kernel_module.update_halo_kernel_module.update_halo_kernel 0.02 tea_leaf_kernel_cg_module.tea_leaf_kernel_init_ 0.09 tea_module.tea_allsum		 0.00 machine Linux 0.00 node jrc0280 0.03 MPI Rank 0 3.07 MPI Rank 1 0.00 node jrc0281 0.01 MPI Rank 2 0.28 MPI Rank 3 0.00 node jrc0282 0.02 MPI Rank 4 2.30 MPI Rank 5 0.00 node jrc0283 0.01 MPI Rank 6 0.01 MPI Rank 7 	4

Forschungszentrum

...with imbalance (time on critical path above average) mostly in the first two loops and MPI communication

TEALEAF SCALASCA REPORT ANALYSIS (VIII)



Forschungszentrum



TEALEAF SCALASCA REPORT ANALYSIS (IX)



Forschungszentrum

CENTRE

CubeGUI-4.4.3: scorep_tea_leaf_baseline_8x12_trace/trace.cubex <@jri11> File Display Plugins Help Restore Setting * Save Settings x-rot: 0 v-rot: 0 Peer distribution Absolute Absolute E Flat view Process x Thread Metric tree 🔚 Call tree Sunburst • 0.11 tea module.tea send recv message top
 □ 0.00 Execution
 4.47 tea module.tea unpack top 8546.03 Computation 4.98 tea module.tea pack left - 0.00 MPI I 0.17 tea module.tea send recv message left 3.09 tea module.tea unpack left • 4.64 Management I 3.65 tea module.tea pack bottom 0.14 Synchronization

 0.00 Communication
 I 0.12 tea module.tea send recv message botto 11.31 Point-to-point 4.39 tea module.tea unpack bottom 11.04 update halo kernel module.update halo ke 4.87 Late Sender ▶ ■ 6.26 tea leaf kernel cg module.tea leaf kernel init 0.26 Late Receiver 3.59 Collective Description of the second s 0.00 Early Reduce - 0.21 tea leaf kernel cg module.tea leaf kernel solve 0.00 Early Scan 1.70 \$ 000 parallel @tea leaf cg.f90:186 0.00 Late Broadcast 33.50 Wait at N x N 3442.11 !\$omp do @tea leaf cg.f90:187 1.40 N x N Completion 0.00 !somp implicit barrier @tea leaf cg.f90:20 0.34 tea leaf kernel cg module.tea leaf kernel solve ▶ □ 0.00 One-sided 2.35 !\$omp parallel @tea leaf cg.f90:234 I 0.00 File I/O 3424.97 !\$omp do @tea leaf cg.f90:247 - □ 0.00 OpenMP 0.00 !\$omp implicit barrier @tea leaf cg.f90:25 I 140.42 Management

 0.00 Synchronization
 • 0.27 tea leaf kernel cg module.tea leaf kernel solve 1614.53 !somp parallel @tea leaf cg.f90:284 1.19 tea leaf kernel module.tea leaf kernel finalise D 0.00 Explicit 37.84 Implicit 0.26 field summary 75.11 Wait at Barrier 0.00 tea module.tea finalize I 0.00 Critical b. 1 H I . 0.00 0.00 9594.86 0.00 8546.03 0.00 0.00 8546.03 (89.07%) 3424.97 (40.08%) 100.00 JULICH SUPERCOMPUTING

...and 2nd do loop mostly balanced within each rank, but vary considerably across ranks...

TEALEAF SCALASCA REPORT ANALYSIS (X)

Forschungszentrum



TEALEAF ANALYSIS SUMMARY

- The first two OpenMP do loops of the solver are well balanced within a rank, but are imbalanced across ranks
 - Requires a global load balancing strategy
- The third OpenMP do loop, however, is imbalanced within ranks,
 - causing direct "Wait at OpenMP Barrier" wait states,
 - which cause indirect MPI point-to-point wait states,
 - which in turn cause OpenMP thread idleness
 - Low-hanging fruit
- Adding a SCHEDULE (guided) clause reduced
 - the MPI point-to-point wait states by ~66%
 - the MPI collective wait states by ~50%
 - the OpenMP "Wait at Barrier" wait states by ~55%
 - the OpenMP thread idleness by ~11%
 - → Overall runtime (wall-clock) reduction by ~5%





SUMMARY



Mitglied der Helmholtz-Gemeinschaft

TAKE AWAY MESSAGES

- Many performance analysis tools exist for a reason
 - Different measurment and analysis techniques
 - Instrumentation vs. Sampling
 - Profiling vs. Tracing
 - Different hardware support
 - Vendor specific tools, e.g. NVIDIA NSIGHT COMPUTE, Intel VTune
 - Verndor agnostic tools, e.g. Score-P ecosystem, TAU, HPCToolkit
- Tools don't automagically increase performance
 - Performance analysis is a daunting task, requires experience
 - Performance tuning requires domain and architecture knowledge

© Successful performance engineering often is a collaborative effort

