Visualization of Memory Access Behavior on Hierarchical NUMA Architectures

Benjamin Weyers, Christian Terboven, Dirk Schmidl, Joachim Herber, Torsten W. Kuhlen, Matthias S. Müller, Bernd Hentschel
JARA – High Performance Computing
IT Center, RWTH Aachen University
52074 Aachen, Germany

Email: {weyers,terboven,schmidl,herber,kuhlen,mueller,hentschel}@itc.rwth-aachen.de

Abstract—The available memory bandwidth of existing high performance computing platforms turns out as being more and more the limitation to various applications. Therefore, modern microarchitectures integrate the memory controller on the processor chip, which leads to a non-uniform memory access behavior of such systems. This access behavior in turn entails major challenges in the development of shared memory parallel applications. An improperly implemented memory access functionality results in a bad ratio between local and remote memory access, and causes low performance on such architectures. To address this problem, the developers of such applications rely on tools to make these kinds of performance problems visible. This work presents a new tool for the visualization of performance data of the non-uniform memory access behavior. Because of the visual design of the tool, the developer is able to judge the severity of remote memory access in a time-dependent simulation, which is currently not possible using existing tools.

I. INTRODUCTION

The performance of various HPC applications is limited by the available memory bandwidth instead of the processor’s floating-point performance. For example, in the class of iterative linear equation system solvers a sparse matrix-vector product is most often the application hotspot. These matrix-vector products are memory-bound with a computational complexity of $\frac{1}{N}$ according to the roofline model [1] if the vectors fit into the last level cache of the system. In order to increase the memory bandwidth, modern microarchitectures integrate the memory controller directly on the processor chip. This leads to a non-uniform memory access (NUMA) behavior in multi-socket configurations: the memory access time depends on the memory location relative to a processor core, as local memory accesses are faster than remote memory accesses. Shared memory parallel applications, such as some iterative linear equation system solvers, may exhibit poor performance on NUMA architectures if they are not properly prepared and the ratio of remote to local memory accesses is high. Consequently, NUMA-aware memory allocation and corresponding memory access form a performance optimization challenge in which proper tool support promises an increase of the programmer’s productivity.

Several performance tools are capable of analyzing the data access of shared memory parallel applications on NUMA architectures. However, most of these tools only provide a statistical summary and analysis of the data access behavior for the application as a whole. This allows the programmer or analyst to identify NUMA problems in a given setting, but does not help to analyze the application regarding its different phases during runtime. This level of insight can only be provided by time-resolved analysis, which is related to the underlying hardware architecture. The tools that are capable of displaying information about the memory access behavior over time do not relate this to the individual machine architecture but to threads. This limits the insight a user of these tools may gain because first, the distribution of threads on the machine is not necessarily obvious, second, without considering the architecture the severity of remote memory access is hard to judge, and finally, the parallelism may not necessarily be expressed by means of threads but rather by employing tasking.

In this work, we explore a new visualization approach to gain insight into the time-dependent data access pattern of an investigated application. Based on hardware performance counter analysis, the local memory read and write bandwidth, as well as the remote memory traffic is sampled and visually related to the actual system architecture. A dynamic small-multiples visualization design has been implemented to get a general idea of the architecture-dependent counter data. It further enables the user to zoom into the presentation to gain more detailed views to, e.g., the memory access behavior of one specific processor. The visual design is extended by a slider widget that enables the user to navigate interactively through the data along the time axis. This feature guides the understanding of the application behavior to finally discover and resolve NUMA-related performance bugs. The applicability of our approach is demonstrated on a large shared memory machine that exhibits a hierarchical NUMA architecture. Four server boards with four sockets each are connected by a second-level cache-coherent interconnected to a 16 socket shared memory machine. Consequently, the achievable performance on this system depends on achieving a NUMA-aware data distribution and employing a corresponding data access pattern in the application.

In summary the main contributions are:

- A visualization for time-resolved hardware performance counter data.
- A visual correlation of time-resolved data access information to the actual hardware architecture.
- A methodology to pinpoint NUMA-related performance issues based on the new visualization and analysis tool.
The paper is organized as follows. Section II presents related work, which focuses on existing performance and visualization tools developed in context of the analysis of NUMA architecture. Section III introduces an application use case, which shows the capabilities of the newly developed performance visualization tool presented in Section IV. This exemplary analysis is discussed in Section V. The paper ends with a short conclusion and some discussion of future work in Section VI.

II. RELATED WORK

Performance tools in general aim to collect and provide information about the runtime behavior of applications and thereby support developers in detecting and possibly resolving performance problems in parallel programs. Some of these tools focus on shared memory parallelism, which imposes specific program optimization challenges, in particular on the synchronization of, and load balancing among threads and the memory access behavior. These tools range from simple profilers with dedicated support for OpenMP, as such as ompp [2], to commercial tools offering a variety of analyses, such as the Intel VTune Amplifier XE [3]. As a profile does not consider the chronology of events in an absolute sense, profilers only provide a statistical overview about the program behavior, which does not allow an understanding of the time-resolved memory access behavior; hence they are not considered here.

Tracing tools record the chronology of events, whereas the events themselves may be of various kinds. The LIKWID [4] tool suite provides the tool likwid-perfctr, which measures hardware performance counters, to, e.g., determine the used memory bandwidth. The measurement result can either be provided as a summary after program termination, or as samples over time with a given sampling resolution. The latter mode is employed in this work to collect the basic memory access information, but LIKWID itself does not offer any visual analysis of the raw data. Similar information can also be collected with the VTune tool, which offers two displays: either it shows the consumed memory bandwidth for the whole application, or the GUI can attribute the raw counter information to the individual program threads or program parts. However, it is currently not possible to overlay this information with the machine architecture, nor is it possible to derive metrics like the bandwidth when custom analyses are used. This is the case with uncore counters to measure remote memory accesses. The Vampir tool [5] allows to visualize hardware counter information also for parts of the execution time by zooming in to specific time intervals of the execution. The counter values can be displayed per process or thread but without any connection to the underlying architecture. We argue that annotating the system architecture with information collected at program runtime improves the insight a user may get, particularly for the memory access behavior on NUMA architectures.

In a previous work some of the authors employed the Vampir tool to visualize the remote socket traffic [6] on an Intel Nehalem-EX architecture. The Vampir timeline visualization was made to interpret remote memory accesses as MPI messages and each NUMA node was represented as an MPI process. While this experiment showed the general value of examining these counters, extensive and in particular manual filtering was necessary to reduce the number of accesses shown in the timeline to direct the user towards phases of congestion. It was concluded that more appropriate visualization techniques are required to gain insight on time-resolved hardware counter data.

Nevertheless, various tools for the visual analysis of performance data can be found in the literature. When focusing on visualization tools for NUMA behavior, the selection shrinks. Tao et al. [7] present a tool comprised of various views. One view is specifically developed for visualizing the run-time behavior in form of a hardware architecture-related presentation. Still, this visualization is restricted to up to four cores, where more cores are just visualized in a value-based matrix, which hides the underlying hardware architecture completely. This static implementation is mainly based on the presentation of numerical values and is only capable of color-coding and predefined thresholds. A framework implementation for performance visualization called Rivet has been introduced by Bosch et al. [8]. In contrast to the work of Tao et al., Rivet implements a framework offering various visualization tools for different types of performance data, including NUMA behavior. The main visualization method is based on Gantt diagrams, as the authors also present in [8] to visualize NUMA performance data. Another recently presented performance visualization tool based on Gantt charts is called Aftermath, presented by Drebes et al. [9]. It mainly focuses on a set of data processing features, such as K-means clustering. Nevertheless, neither Rivet nor Aftermath support visualization tools relating time-resolved counter data to the underlying hardware architecture. Further works can be found in close related contexts like the work of Landge et al. [10], which describes a visualization concept for network traffic in massive parallel simulations. These works further not focus on a close relation of the visualization to the underlying hardware architecture.

The work published by Jiang [11] presents a node-link diagram-based visualization of NUMA behavior, which explicitly involves the architecture in the visual design. This implementation focuses on the visualization of accessed data points per memory node neglecting the bandwidth. Furthermore, the presentation of the software does not clarify whether the example is extensible to more than four memory nodes or not. Nevertheless, the visualization offers a simple way to gather a broad overview of whether load balancing is achieved or not, but does not offer any further "zooming" into the data to gain detailed insights.

III. APPLICATION CASE STUDY

In order to evaluate our new approach, we use a sparse-matrix-vector-multiplication (SpMXV) kernel on a matrix with about 85 million nonzero elements executed on a large NUMA machine as a case study. SpMXV is relevant for HPC in that it is by far the most time consuming operation in many iterative solvers, for example in the GMRES method. For any SpMXV kernel, the memory bandwidth is the limiting performance factor, consequently it is sensitive to locality. The intuitive approach to parallelize the SpMXV operation is to distribute the matrix rows in even chunks among the threads. However, depending on the matrix structure, this may lead to load imbalances if the nonzero values are not evenly distributed among the rows – a nonzero-based distribution...
may improve load balance and optimize locality [12], as will be discussed below. Considering the compressed row storage (CRS) format, the SpMXV operation is memory efficient in that the matrix data values together with the column values are stored consecutively in memory, resulting in high spatial locality in the SpMXV operation. The memory access pattern on the right hand side vector depends on the matrix structure.

For our experiments, a 16-socket server equipped with Intel Xeon X7550 (Nehalem-EX) CPUs was used, where every CPU is attached to 16 GB of main memory. The Bull Coherence Switch (BCS) is used to combine four Bullx s6010 servers into one shared memory machine. Inside of the 4-socket boards, cache coherence is maintained through the standard Intel Quick Path Interconnect (QPI). Between the boards, the BCS chip uses a directory-based protocol (x-QPI) on-top of the QPI for cache coherence. This makes the machine a hierarchical NUMA machine, with two levels of remote memory: on a different socket of the same board or on a different board connected via the BCS. Figure 1 illustrates the topology of the machine in detail.

We use the LIKWID tool suite to query the hardware performance counters of the Xeon processors. The counter group MEM provided by likwid-perfctr records the bandwidth of the data transferred over the memory buses (light blue lines) as well as over the four QPI links of every processor (solid black lines). Since the BCS chips do not provide an API to query any related performance counters, we cannot observe traffic over the x-QPI links (dashed lines), but we can measure traffic between the socket and the BCS chip. I/O operations are performed via the BCS chip, such that measured traffic between the socket and the BCS chip is either I/O or traffic to remote boards. LIKWID’s sampling mode was used to acquire the counter information for 100 ms intervals. For further information on LIKWID, the reader is referred to the LIKWID documentation [4]. The data was stored in a text file for visualization. Information on the maximum available bandwidth of all links is determined with the help of the STREAM kernel benchmark [13]. The numactl tool provided by the Linux OS was used to pin threads and memory on different sockets, which allows to measure the bandwidth of inter-socket links of the system.

For the investigations made here, we used three different versions of the SpMXV kernel, with different levels of NUMA optimization:

1) not optimized: In this version, the matrix is read from a file serially and only the compute kernel is parallelized using an OpenMP parallel for loop over the rows of the matrix, i.e., the outer loop of the matrix-vector product.
2) distributed memory: Reading the data from a file with a single thread lets the operating system (OS) use the physical memory of one NUMA node to store the complete matrix. This is because the OS applies the so-called first touch strategy for memory allocation. In this optimized SpMXV version, the matrix data structure is first initialized in parallel with
a static parallel for loop, before the file is read. This results in an even distribution of the matrix across all NUMA nodes of the system.

3) load balanced: Parallelizing the outer loop in the matrix-vector product leads to an even distribution of rows over all threads. However, the number of non-zero elements might differ between rows. This leads to a load imbalance, since the work is proportional to the number of non-zero elements and not to the number of rows. In this version, an even distribution of non-zero elements is computed during the matrix setup and later on used for a reproducible load balancing in the actual compute loop.

The next section will introduce the newly developed visualization tool, which is used in Section V to analyze and discuss the gathered data for the different kernel versions.

IV. Visualization of NUMA Memory Access Behavior

The main goal of the interactive and architecture-related visualization tool is to provide the user/developer with a tool to gather insights into the memory access behavior of a NUMA architecture. It is designed to visualize the time-variant data representing the bandwidth on the architecture’s data links.

The tool design follows Shneiderman’s visual information seeking mantra: “Overview first, zoom and filter, then details-on-demand” [14, p. 2]. The overview is achieved by a small multiples design presenting the complete data, as can be seen in Figure 2. The zoom step is implemented by an interactive change of the viewport onto the small multiples area, which reduces the number of visible tiles and, simultaneously, enlarges the visible ones. The filtering step is implemented as the interactive selection and dynamic navigation of the visualized time interval using a slider widget, as it can be seen at the bottom of Figure 2. To address the time-variant property of the data, the tool is capable to animate the visualization of the whole data set. Therefore, the defined time interval is moved along the time axis and the visualization gets adapted accordingly. Details-on-demand is currently implemented by two operations, which the user can apply to the visualization to show or hide additional information (the max value and the sub-intervals, which lie over the threshold). A closer view to all components will be presented, below.

A. Overview—Small Multiples

The design concept “Small multiples” was introduced by Tufte in [15]. The concept proposes the multiple use of the same type of graphic or chart organized in a grid of tiles to make differences in a data set visible. Our tool uses a small
multiples design to visualize the time-dependent bandwidth related to each port in a 16-socket NUMA architecture using a BCS (as proposed in Section III). The small multiples design is further split into four groups to represent the NUMA architecture. Thereby, the visualization becomes directly related to the underlying architecture, which simplifies the matching of the visualized data to the relevant parts of the hardware architecture. The NUMA architecture is shown in Figure 1 while the corresponding small multiples visualization can be seen in Figure 2. For instance, the upper left grid of graphs in Figure 2 represents the memory access behavior of node $N_0$ as indicated in Figure 1.

Every single tile represents the time-dependent bandwidth (in MByte/s) of either memory buses and QPI links or of the BCS interconnect as simple two-dimensional function graph. The graph is further extended by a predefined threshold that is represented as a simple line. The threshold can be individually specified as bandwidth limit relevant for a specific scenario supporting the analysis. The four $4 \times 5$ matrix shows the bandwidth between the different sockets in a node and to the bandwidth to the BCS. The diagonal of the left $4 \times 4$ submatrix presents the memory bus’ bandwidth of every processor on one node, while the other graphs represents the indicated QPI links. In case of QPI links, the direction of data flow is read from row to column in the small multiples design. For instance, in case of node $N_0$ the data bandwidth from socket $S_0$ to $S_3$ is visualized in the tile in the first row, third column (as can be seen in Figure 3). The separated right column in the matrices represent the bandwidth of the processors to the BCS interconnect. It should be noted that the visualization of the BCS interconnect represents the bandwidth of the BCS interconnect from every socket to the BCS interface. The bandwidth of the BCS interconnect between nodes has not been visualized due to missing hardware counter for this parameter. Nevertheless, it would be possible to simply aggregate the measured bandwidth and extend the design with a fifth tile that presents the bandwidth from a node to BCS. As the initial design focus on the existing hardware counter and the underlying architecture, this extension is left for future work. In case of memory bus and BCS access, no direction is given due to the data captured or the provided hardware counter, respectively.

To enhance the overview in the small multiples representation (as shown in Figure 2), the single tiles are colored according to the average bandwidth of the displayed time interval. The used color mapping relates blue or green to low bandwidth and red to high bandwidth, yellow lies in between. Nevertheless, the color mapping can be individually defined through simple changes of parameters in the initialization file of the tool. They are shown as part of the visualization and serve as explanation (see Figure 2, right). Finally, the coloring offers the extension of the initial design to more complex architectures. If more nodes are involved, the design comprise more tiles up to a certain limit where the resolution of the used visualization hardware limits the presentation. In this case, a layered approach can be implemented, which comprises parts of the architecture into meta-tiles on a higher layer that represent average bandwidth measures arising from the subsumed part of the architecture.

B. Zoom & Filtering—Interactive Change of the Viewport and Dynamic Navigation

After the user has an overview of the data using the initial small multiples visualization, she is able to continuously zoom into the presentation. Thereby, different subsets of tiles can be focused (see Figure 3) down to one single tile (see Figure 4, left). It is further possible to translate the viewport of the small multiples presentation, such that the user is able to select different tiles or sets of tiles on one zoom level.

Filtering of the data is currently implemented based on an interactive selection of the displayed time interval and, thus, the selected sub data set. Therefore, the user can reposition the handles of a slider, which is accessible in the lower part of the tool’s window (see Figure 2, bottom). The width represents the selected interval relating to the complete time interval provided by the data. The selection directly affects the shown time interval in the various graphs/tiles and also affects the graphs’ coloring through the implicitly changing mean value.
Beside the specification of the selected time interval, the user is further able to navigate through the data along the time axis by dragging the slider’s middle handle to the left and right. Finally, the user is able to let the data be visualized dynamically by pressing the start button shown on the left of the slider (see Figure 2, bottom). The tool starts to continuously change the presented interval along the time axis in an infinite loop.

C. Details-on-Demand—Extending the Graph with Detailed Information

The last part of Shneiderman’s mantra refers to the details-on-demand metaphor. It describes a technique which enables the user to extend the visualization with detailed information on demand. In our tool, two functions for details-on-demand are currently implemented:

1) Max value: The user is able to display the max value of the currently selected time interval in the line graph.
2) Interval(s) over threshold: The user is able to highlight all sub-intervals, which show a higher bandwidth compared to a predefined threshold.

These features can be combined arbitrarily (see Figure 4). Both tools can further support the user guidance on which time period to select and where are relevant effects visible in the data. The activation is done interactively through buttons, which are accessible in the lower part of the window, beside the slider (see Figure 2, bottom right).

D. Implementation

The implementation is based on the ViSTA toolkit [16]. ViSTA offers an API for the implementation of interactive 3D visualization applications. Although the presented visualization application does not use a three-dimensional visualization, the rendering of a three dimensional scene offers certain benefits for the implementation of the zooming and refocusing of the viewport. To realize this, the camera’s orientation is fixed in the scene. This results in a very flexible navigation and zooming feature, allowing the camera to be moved towards, away from, as well as across the small multiples visualization using the mouse. The color mapping legend as well as the widgets are rendered as overlays, such that these elements are always visible. Nevertheless, the user is able to turn them off using the keyboard.

V. DISCUSSION

Finally, the tool was applied to visualize performance data measured for all three versions of the SpMXV kernel, as described in Section III. As it is the case with the SpMXV operation employed in an iterative solver, the kernel was executed repeatedly to induce time-variant behavior.

Figure 5 shows the traffic on all links during the matrix-vector multiplication for the not optimized version, where all the data resides on the first NUMA-node. Roughly the first 80 seconds are used to read the input matrix from a file, therefore we navigate to the remainder of the program to investigate the actual compute kernel. It can be observed, that the memory traffic on socket 0 is the performance bottleneck in this version, since the traffic is close to the maximum of the link all the time. All connections to socket 0 from the same board and all connections through the BCS link to other boards are also in use, since data needs to be read from socket 0, but the links are not at the limit of their capacity as shown by the green or lightblue color of their corresponding line graphs/tiles.

Figure 6 shows the performance data for the distributed memory case of the SpMXV kernel. All memory controllers are in use and the runtime could be reduced significantly from
Fig. 7. Visualized memory and inter-node traffic for the SpMXV kernel, where the data and the computational load have been distributed evenly over all threads.

about 150 seconds to 25 seconds. However, all links on the board and all BCS links are still used significantly during execution. The traffic on the connections inside a board and to other boards is nearly the same, but because of the lower available bandwidth on the BCS links, these connections turn out to be the bottleneck. The reason is that we distributed the data evenly after the matrix has been allocated. This was done using parallel loops with a static schedule over the value, index, and row vector of the matrix. During computation, two nested loops are used: the outer one over the rows and the inner one over all elements within a row. Here, the outer loop was parallelized, resulting in an even distribution of rows over all threads. Since not every row has the same number of non-zero elements, this does not match the data distribution completely. The remote accesses that result from this mismatch can be observed in Figure 6.

In Figure 7, the behavior for the load balanced version of the SpMXV kernel is shown. Here, the compute kernel has been changed to distribute the work according to the number of non-zeros in the matrix. Consequently, every thread performs the same amount of computations and every thread accesses exactly the same blocks of data it initialized, which should lead to good data locality. Surprisingly, we observed significant traffic between two neighboring sockets. Again, the traffic is the same on the QPI and BCS links, but on the BCS links it reaches the limit of the available bandwidth. It turned out, that the Linux OS used huge pages (2 MB) for the matrix. Between two neighboring threads, there is always one shared page where data for both threads is located. For 128 threads this results in 127 boundaries times 2 MB, so 254 MB out of the 950 MB of the matrix are on shared pages. The transparent use of huge pages can be switched off under Linux which results in the use of standard 4 KB pages, so only 504 KB of data resides on shared pages. Figure 8 shows how this further reduced the amount of traffic on the links and it also reduces the runtime from about 11 seconds to 6.3 seconds. The effect of the transparent huge pages feature was not expected initially and its discovery underlines the value of this visualization.

In conclusion, the visualization approach chosen in this work combines architecture information with hardware performance counter data. This allows to investigate the utilization of all links in the system and to identify performance bottlenecks during execution, as described above. With the help of SpMXV, we have shown that different performance problems can be detected, e.g., bad memory affinity, which leads to a lot of remote traffic. Other tools allow to investigate the raw hardware counter data, but they require a detailed hardware knowledge from the user. The tool presented here directly emphasizes the performance bottleneck from a hardware point of view, as has been shown for the SpMXV kernel. This delivers insights into the application execution which were very hard to gather before. To conclude, we are able to judge the severity of remote memory access using the visualization because of the representation of the architecture in the visualization.

VI. CONCLUSION AND FUTURE WORK

This work presented a new approach to visualize data gathered by performance counters for NUMA architectures. The interactive visualization tool is based on Shneiderman’s information seeking mantra: “Overview first, zoom and filter, then details-on-demand”. The result is an interactive visualization application providing an overview of the presented data using a small multiples design, zooming and filtering by interactively changing the camera perspective in the underlying 3D visualization, as well as details-on-demand capabilities through an interactive extension of the small-multiples-tiles. The applicability of this tool has been evaluated in an application case.
study, which uses an SpMXV kernel on a matrix with about 85 million nonzero elements in three different implementation alternatives. The experiment has been executed on a 16-socket server, with at structure that is reflected in the overall design of the visualization. Data is gathered using LIKWID in three different scenarios: not optimized, distributed memory, and load balanced. The gathered data has been discussed using the newly implemented visualization tool showing the applicability of the whole approach. By the visual representation of the underlying hardware architecture in the visual design of the tool, we are able to judge the severity of remote memory access. We have shown that a time-resolved visualization of the memory access pattern can provide valuable insight into the application behavior.

Future work shall connect our tool with the actual program source, meaning that when a memory access hotspot is identified, the corresponding source code location is shown. We plan to achieve this by employing the Score-P measurement infrastructure [17], which can provide the time-resolved hardware counter analysis in addition to appropriately instrumenting the application code.

To further extend the visual analytic capabilities of the presented visualization tool, it is planned to add linked histograms as a further view to the application. These histograms will present the occurrence of specific bandwidth intervals in the measurement related to the architecture. Therefore, the user will be able to switch (interactively) between the histogram and the currently implemented line graphs in the small multiples design. Furthermore, we plan to offer histograms of sub sets or of the whole data set, which are linked to the small multiple design. By selecting a bar in one of those diagrams, the relevant parts in the small multiples view will be highlighted. Thus, the user will be able to navigate through the data using this multiple view approach. Finally, we plan to apply a user study to identify possible extensions to the visualization tool and the overall analysis process to identify continuous research questions in context of architecture-related performance visualization.

ACKNOWLEDGMENT

Some of the tests were performed with computing resources granted by JARA-HPC from RWTH Aachen University under project jara0001. Parts of this work were funded by the German Federal Ministry of Research and Education (BMBF) under Grant Number 01IH13001D(Score-E) and by the European Union in context of the FET flagship project “The Human Brain Project”.

REFERENCES


