Visualization of Data Layout and Access of Parallel Program for Productive Performance Analysis and Tuning

1. Abstract

Current performance tools such as TAU, Vampire, Paraver, Jumpshot, Scalasca, Peekperf, EXPERT performance-analysis environment, Cilkview, HPCToolkit, etc. provide measurement and visualization of performance and scalability of parallel program execution to help users for performance analysis and tuning. They however do not provide enough and intuitive insight on how data are layer-out and accessed during parallel execution, thus relying users' expertise to manually diagnose issues related to memory access, such as shared cache contention, false sharing and memory bandwidth optimization. We propose a visualization tool dealing with displaying data layout and parallel program access in clear picture of array distribution and computation distribution, maps of program data to the physical NUMA memory region, pattern of memory access, contention on memory bandwidth and shared cache (bandwidth or size). Visualization of data layout will make user sound of peak stack or heap memory usage and peak read/write memory bandwidth contention. Location of memory allocation is a critical factor as NUMA or cache coherence effect can hugely affect the performance of the computation. In order to tackle this challenge, visualization of memory location will help user to identify this bottleneck for the performance issue. To get started, we are thinking of pictorial presentation of 1) program data access graph and 2) data layout/access in memory layout for different programs to get clear insight of the memory usage. This will allow us to find solution for the problem with a greater ease.

2. Introduction

The behavior of parallel programs on advanced computer architectures is often extremely complex, and hardware or software performance monitoring of such programs can generate vast quantities of data. Analysis of the performance of high-performance computing (HPC) systems and applications is hence a very significant and complex task. Thus, it seems natural to use visualization techniques to gain insight into the behavior of parallel programs so that their performance can be understood and improved. The performance of most of the HPC applications is limited by the available memory bandwidth instead of the processor's floating-point performance. Modern microarchitectures integrate the memory controller directly on the processor chip, in order to increase the memory bandwidth. This give rise to a non-uniform memory access (NUMA) behavior in multi socket configurations. In NUMA architecture, as local memory accesses are faster than remote memory accesses, the memory access time depends on the memory location relative to a processor core. If shared memory parallel applications are not coded effectively and if the number of remote accesses are high than local accesses then they may exhibit poor performance on NUMA architectures. Thus, an effective performance visualization tool promises performance optimization in NUMA-aware memory allocation environment. Several performance tools are capable of analyzing the data access of shared memory parallel applications on NUMA architectures. However, most of the tools does not give the details on how data are layered-out and accessed during parallel execution. Some of the tools provide just the quantitative profiling results for memory performance such as number of cache misses, number of memory accesses and simply show the bars of collective numbers for the whole program executions.

Consequently, users need to co-related performance numbers with program data access and memory behavior. Hence it needs lots of manual work and expertise on the part of user too to identify performance bottleneck and provide solutions. Most of the times, solutions are trial-and-test approach which are quite tedious. In this work, we explore a new approach for performance visualization using program data access graph and data layout/access in memory graph using GraphML.

3. Contributions:

The main challenges of memory optimization are NUMA and its first touch policy, cache coherence and false sharing, memory bandwidth contention and shared cache contention for both bandwidth and size. Considering these challenges, following are the contributions of this work:

Visualization of:

- Location of memory allocation: The work uses the program data access graph and data layout access in memory graph for computation and array distribution for displaying the layout of memory allocation to ease the process of optimization.
- Stack/heap Memory Usage of the application in the system
- Read/write memory bandwidth contention
- Level 1/level 2 Cache Hit/Miss evaluation, mostly focusing on shared cache

The above contribution will allow the users to gather data to get more insight into NUMA performance. Thus the work will create a glass box of memory behavior to the users who have no expertise. It will measure and aggregate NUMA-related events and will associate them with source code contexts, such as functions and statements. It will gather instruction and data address pairs to associate instructions that access memory with the variables that they touch. It will provide a wealth of information to guide NUMA optimization, including information about how and where to distribute data to maximize local accesses and reduce memory bandwidth contention.

4. Motivation

Systems with NUMA architecture are challenging to program efficiently. If the data accesses performed by a multithreaded program are remote, there will be bottleneck of contention for limited bandwidth between NUMA nodes. This problem can be more serious if a large data arrays are mapped to a single NUMA node and many threads compete for the limited bandwidth in and out of that node. This situation is very common than one might think in NUMA architecture. By default, current Linux systems employ a “first-touch” policy to bind pages of memory newly-allocated from the operating system to memory banks directly attached to the NUMA domain where page resides on the thread that first accesses it. Because of this, if a data array is initialized by a single thread, but multiple threads process the data later, severe contention can arise. Tailoring a program for efficient execution on systems with multiple NUMA nodes requires identifying and adjusting data and computation layouts to minimize each thread’s use of remote data and avoid contention for bandwidth between NUMA nodes. This motivated us to design a visualization tool that can provide an insight into NUMA-related performance losses that are important for guiding optimization of multithreaded programs.

5. Program data access Graph

In the visualization tool, the Program data access graph shows the array and computation distributions from program level. It is generated by complier analysis with support of language extensions. This can be illustrated by following program and its corresponding Program data access graph.

```
Program:
1.  float A[N][N];
2.  float B[N][N];
3.  float C[N][N];
4.  for(i=0;i<N;i++)
5.    for(j=0;j<N;j++)
          a.  A[i][j]=B[i][j]=C[i][j]=0;          // Sequential initialization

6.  #pragma omp parallel for map(tofrom:C ; distribute[BLOCK])
7.  map(to:A ; distribute[BLOCK]) map(to:B ; distribute[BLOCK])
8.  for(i=0;i<N;i++)
          a.  #pragma omp parallel for map(tofrom:C ; distribute[BLOCK])
          b.  map(to:A ; distribute[BLOCK]) map(to:B ; distribute[BLOCK])
          c.  for(j=0;j<N;j++) {
                  C[i][j]=A[i][j]+B[i][j];
              }
```

OpenMP Parallel Program-1

In above program, line 1-3 deals with declaration of matrices A, B and C. Line 4, 5 does sequential initialization of matrices to zero. Line 6, 7 does BLOCK distribution of matrices A, B and C and aligns them. Thus, 6-8 initializes the outer for parallel for operation of matrix addition. Line 8.a - 8.c performs parallel for operation for inner loop of matrix addition. At the program beginning, the team consists of a single thread. A 'parallel' construct splits the current thread into a new team of threads for the duration of the next block/statement, after which the team merges back into one. 'For' divides the work of the for-loop among the threads of the current team. It does not create threads, it only divides the work amongst the threads of the currently executing team. Thus, 'parallel for' is a shorthand for two commands at once: parallel and for. Parallel creates a new team, and for splits that team to handle different portions of the loop. Thus, the corresponding program data access graph for this OpenMP Parallel Program-1 is represented in Figure 1. In Figure1. the first two nodes 0 and 1 represents the two threads generated by outer parallel for loop. The next two nested nodes 0 and 1 represent new two threads generated by inner parallel for loop for each thread in outer for loop. Figure 1. only represents the array distribution for matrix A. The highlighted portion of the matrix under the nodes show the range of matrix on which the corresponding thread nodes work on. The different color schemes show the mapping of each portion of the matrix with the virtual memory. Similarly the mapping will be shown for matrix B and C. Thus this allows user to understand which thread is responsible for which part of the array or loop computation. It also allow user to understand the mapping of array in virtual region.

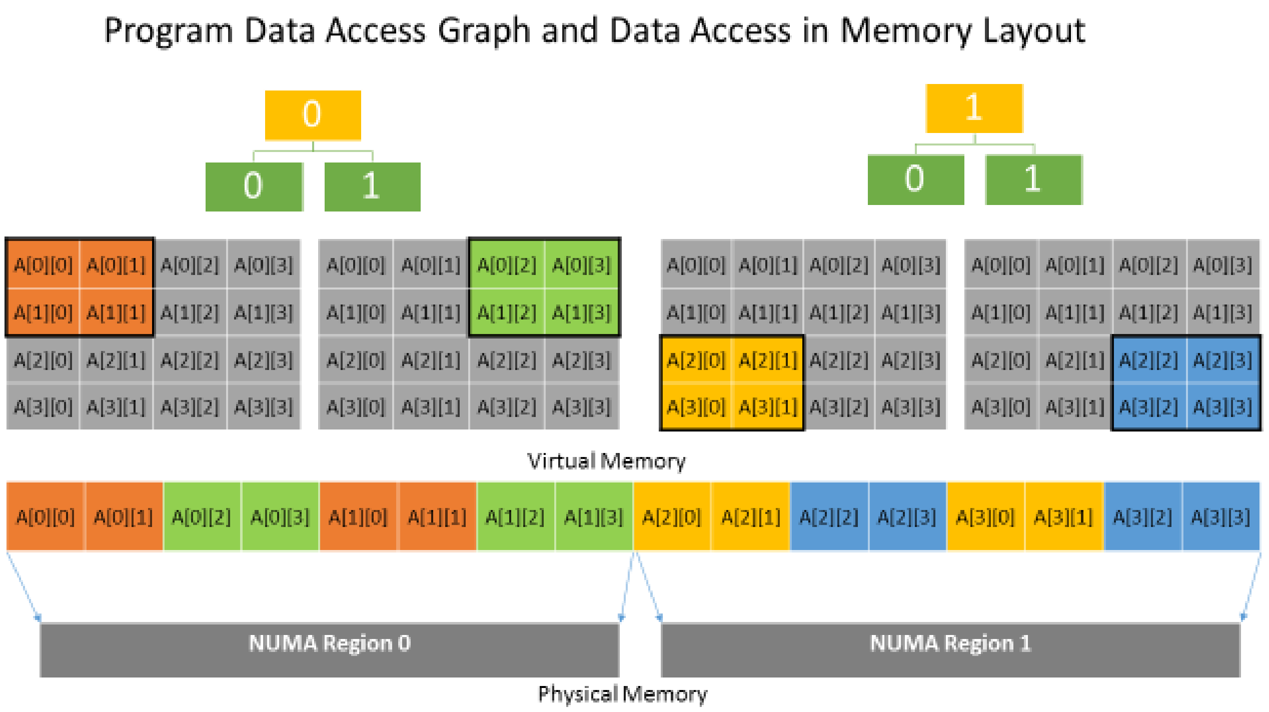

Figure 1. Program data access graph and data access in memory layout

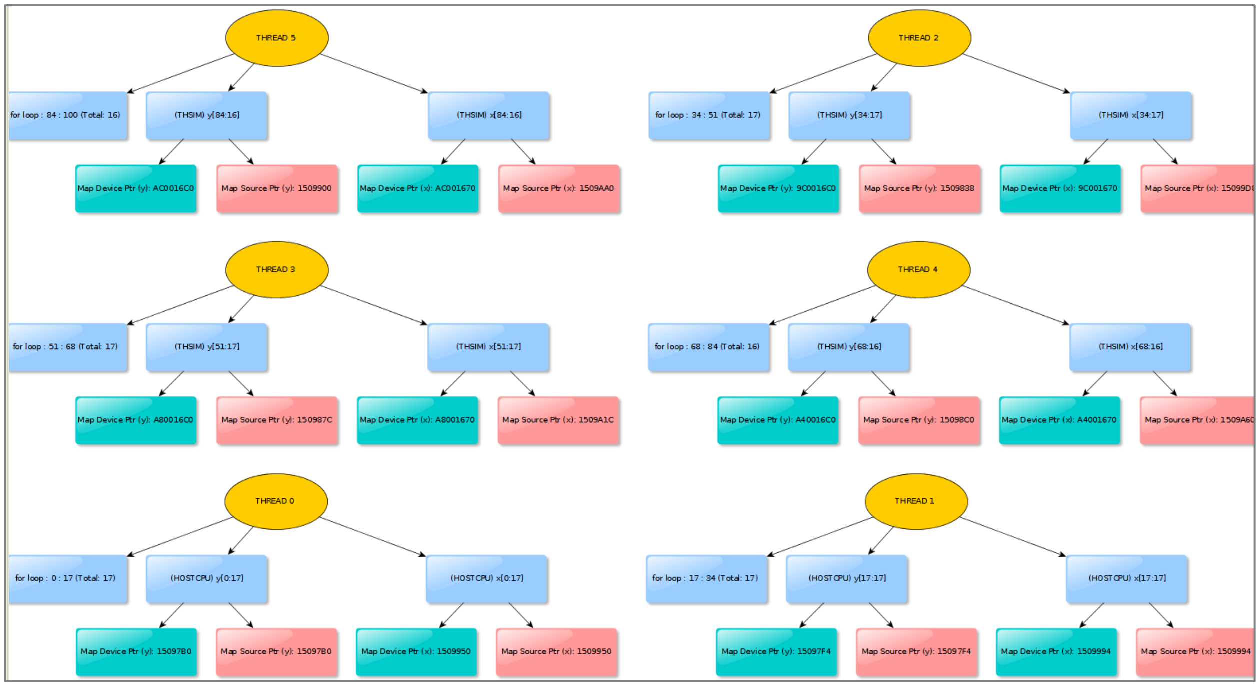
THREAD 5
for loop : 84 : 100 (Total: 16)
(THSIM) y[84:16]
(THSIM) x[84:16]
Map Device Ptr (y): AC0016C0
Map Source Ptr (y): 1509900
Map Device Ptr (x): AC001670
Map Source Ptr (x): 1509AA0
THREAD 2
for loop : 34 : 51 (Total: 17)
(THSIM) y[34:17]
(THSIM) x[34:17]
Map Device Ptr (y): 9C0016C0
Map Source Ptr (y): 1509838
Map Device Ptr (x): 9C001670
Map Source Ptr (x): 15099D
THREAD 3
for loop : 51 : 68 (Total: 17)
(THSIM) y[51:17]
(THSIM) x[51:17]
Map Device Ptr (y): A80016C0
Map Source Ptr (y): 150987C
Map Device Ptr (x): A8001670
Map Source Ptr (x): 1509A1C
THREAD 4
for loop : 68 : 84 (Total: 16)
(THSIM) y[68:16]
(THSIM) x[68:16]
Map Device Ptr (y): A40016C0
Map Source Ptr (y): 15098C0
Map Device Ptr (x): A4001670
Map Source Ptr (x): 1509A6
THREAD 0
for loop : 0 : 17 (Total: 17)
(HOSTCPU) y[0:17]
(HOSTCPU) x[0:17]
Map Device Ptr (y): 15097B0
Map Source Ptr (y): 15097B0
Map Device Ptr (x): 1509950
Map Source Ptr (x): 1509950
THREAD 1
for loop : 17 : 34 (Total: 17)
(HOSTCPU) y[17:17]
(HOSTCPU) x[17:17]
Map Device Ptr (y): 15097F4
Map Source Ptr (y): 15097F4
Map Device Ptr (x): 1509994
Map Source Ptr (x): 1509994

Figure 2. Visualization of array and loop distribution using GraphML

Figure 2 represents the total threads and their corresponding data on which they are operating. It also represent the distribution of array x and y and for loop on CPU/GPU/Simulator. It also shows the range of the distribution of array and for loop on each device. Thus, user get clear insight of data distribution on each thread and device.

6. Data layout/access in the memory systems Graph

Data Layout in memory systems graph shows how the data is allocated in the NUMA region. Uneven distribution of memory accesses to NUMA region often lead to contention and unnecessary bandwidth saturation in both off-chip and on-chip interconnects, memory controllers and caches. Solution can be, instead of mapping large data objects to a single NUMA domain, in many cases one can reduce associated bandwidth saturation and contention by distributing large data objects across all NUMA regions. This is called as optimization contention reduction. Data layout in memory system graph also tells how the data is moving vertically across the memory hierarchy and horizontally across the cache at the same level. It also allows user to understand mapping between multiple discrete memory spaces such as in accelerators. This gives user the information of the page suffering from uneven memory requests, so that one can use various allocation methods to balance the memory requests. The Data layout/access in the memory systems Graph will also allow analysis of the access patterns across the threads to guide NUMA locality optimization, and thereby identify where and which data pages are bounded to NUMA region. User can visualize the shared data access by multiple cores in the system. The visualization tool also show the mapping of virtual pages to the physical pages at the runtime as shown in figure 1. It shows the mapping of matrix A across NUMA node 0 and NUMA node1. The graph also allows to visualize the pages allocated on each NUMA node as shown in Figure 2, helping user to bind the pages to specific memory location to reduce contention and false sharing.

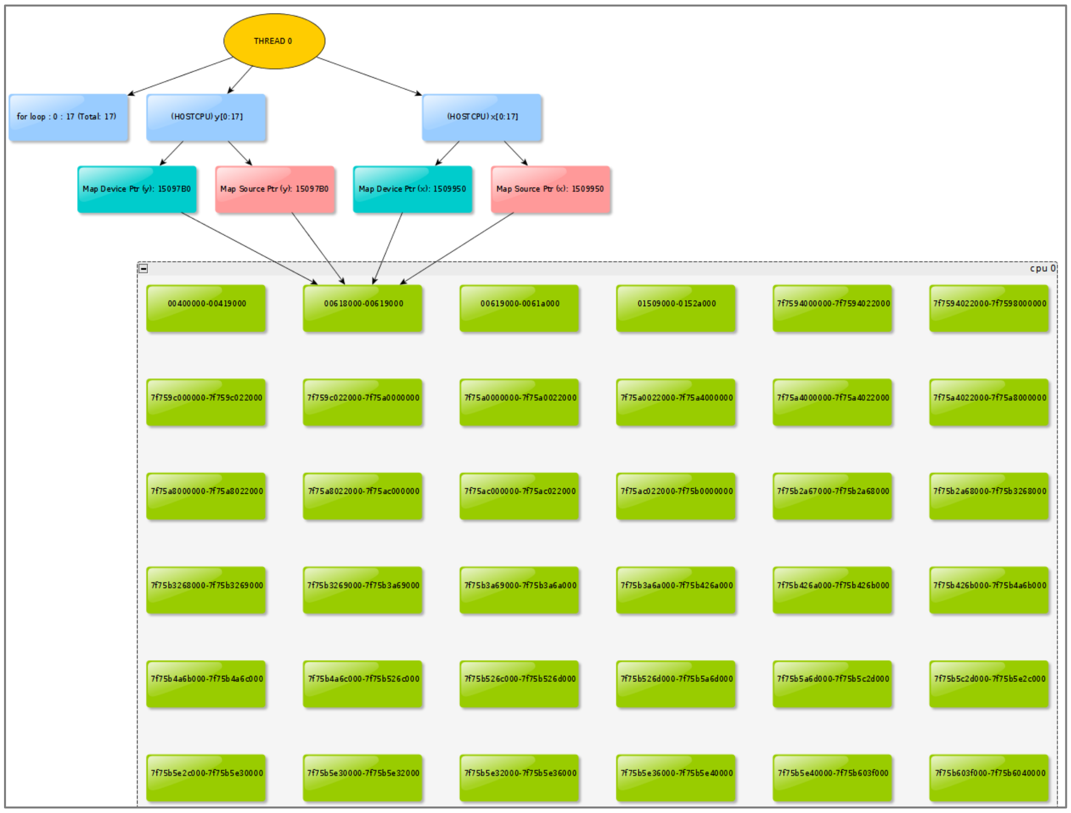
THREAD 0
for loop : 0 : 17 (Total: 17)
(HOSTCPU) y[0:17]
(HOSTCPU) x[0:17]
Map Device Ptr (y): 15097B0
Map Source Ptr (y): 15097B0
Map Device Ptr (x): 1509950
Map Source Ptr (x): 1509950
cpu 0
00400000-00419000
00618000-00619000
00619000-0061a000
01509000-0152a000
7f7594000000-7f7594022000
7f7594022000-7f7598000000
7f759c000000-7f759c022000
7f759c022000-7f75a0000000
7f75a0000000-7f75a0022000
7f75a0022000-7f75a4000000
7f75a4000000-7f75a4022000
7f75a4022000-7f75a8000000
7f75a8000000-7f75a8022000
7f75a8022000-7f75ac000000
7f75ac000000-7f75ac022000
7f75ac022000-7f75b0000000
7f75b2a67000-7f75b2a68000
7f75b2a68000-7f75b3268000
7f75b3268000-7f75b3269000
7f75b3269000-7f75b3a69000
7f75b3a69000-7f75b3a6a000
7f75b3a6a000-7f75b426a000
7f75b426a000-7f75b426b000
7f75b426b000-7f75b4a6b000
7f75b4a6b000-7f75b4a6c000
7f75b4a6c000-7f75b526c000
7f75b526c000-7f75b526d000
7f75b526d000-7f75b5a6d000
7f75b5a6d000-7f75b5c2d000
7f75b5c2d000-7f75b5e2c000
7f75b5e2c000-7f75b5e30000
7f75b5e30000-7f75b5e32000
7f75b5e32000-7f75b5e36000
7f75b5e36000-7f75b5e40000
7f75b5e40000-7f75b603f000
7f75b603f000-7f75b6040000

Figure 2. Page mapping on NUMA node 0

7. Related work.

ParaGraph: A Tool for Visualizing Performance of Parallel Programs
Summary: ParaGraph is a graphical display system for visualizing the behavior and performance of parallel programs on message-passing multi-computer architectures. The visual animation is based on execution trace information monitored during an actual run of a parallel program on a message-passing parallel computer. The resulting trace data are replayed pictorially to provide a dynamic depiction of the behavior of the parallel program, as well as graphical summaries of its overall performance. Many different visual perspectives are provided to view the same performance data, in an attempt to gain insights that might be missed by any single view.

Contribution:

- ParaGraph provides twenty five different displays or views, all based on the same underlying execution trace data.
- ParaGraph is based on the X Window System, and thus runs on a wide variety of scientific workstations from many different vendors.
- ParaGraph provides a mechanism for users to add new displays of their own design that can be viewed along with the other displays already provided.
- Utilization Display: It allows to determine the effectiveness with which the processors are used and how evenly the computational work is distributed across the processors.
- Communication display
- Task displays

Limitation:

- Tracefile is a script which is played out, to visually re-enact the original live action of parallel program execution in order to provide insight into the program's dynamic behavior, but it fails to show array and computation distributions from program level view.
- Also, it does not provide mapping of data across multiple cores and between multiple discrete memory spaces such as in accelerators

MemAxes: Interactive Visual Analysis of Memory Access Data
MemAxes is a system for visualizing and analyzing the memory performance of a node within a supercomputing cluster. This system have been developed to collect fine-grained memory access data. By extracting higher-level variables from memory performance data, MemAxes present data in a variety of different contexts, each of which serves to elucidate different aspects of the data. It uses existing techniques as well as new visual metaphors to present each context in an intuitive manner. The fully interactive and tightly linked combination of contextual views allows high-performance computing experts to gain new insight into memory performance data effectively and accurately.

Contribution:

- It explains how well are memory and processor resources being utilized by software and what data access patterns are intended by the HPC program

- It also uncover inefficient access patterns or data structures, improper use of parallelism (memory thrashing, race issues), data-dependent bottlenecks, data objects, lines of code, indices, or time steps associated with performance issues, inefficient hardware resources

Limitations:

- This system currently visualizes the memory accesses of a single node, but HPC systems nearly all involve multiple nodes that may differ in achieved performance
- Although this visualization is designed to help understanding NUMA performance issues, it shows which event occurs and where it occurs, but does not tell directly why it occurs. The user still has to correlate several pieces of information to guess the source of a performance issue.

SCALEA: a performance analysis tool for parallel programs

SCALEA is a performance instrumentation, measurement, analysis, and visualization tool for parallel programs that supports post-mortem performance analysis. SCALEA currently focuses on performance analysis for OpenMP, MPI, HPF, and mixed parallel programs. It computes a variety of performance metrics based on a novel classification of overhead. SCALEA also supports multi-experiment performance analysis that allows one to compare and to evaluate the performance outcome of several experiments. A highly flexible instrumentation and measurement system is provided which can be controlled by command-line options and program directives. SCALEA can be interfaced by external tools through the provision of a full Fortran90 OpenMP/MPI/HPF frontend that allows one to instrument an abstract syntax tree at a very high-level with C-function calls and to generate source code. A graphical user interface is provided to view a large variety of performance metrics at the level of arbitrary code regions, threads, processes, and computational nodes for single and multi-experiment performance analysis. SCALEA divides the program sources into code regions (ranging from entire program units to single statements) and finds out what performance problems occur in those regions.

Contribution:

- It can be used to instrument arbitrary code regions and to enable the programmer to request a large variety of performance metrics ranging from timing information and hardware parameters to performance overheads.
- It contains a performance data repository which holds all relevant information about applications and performance experiments.
- SCALEA is not yet compatible with C/C++, Java programs and other programming paradigms such as the component-based model.

Visualization of Memory Access Behavior on Hierarchical NUMA Architectures

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.

Contribution:

- 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.

Limitation:

- This paper does not throw light on data distribution and data movement among the multiple cores.

A Visual Approach to Investigating System Behaviour of NUMA Systems and Job Scheduling Processes in HPC Clusters

This paper present the design of the visualization tools and demonstrate how these tools allow users to gain valuable insights into the systems they are studying. The proposed visualization tools run on the targeted iOS platform (iPads), offering users mobility as well as an unique experience of monitoring and analyzing system behaviour via an intuitive, multi-touch interface.

Limitations:

- 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.

Visualizing the Memory Access Behavior of Shared Memory Applications on NUMA Architectures

This paper presents a visualization tool displaying the monitored data in a user understandable way thereby showing the memory access behavior of shared memory applications. In addition, it projects the physical addresses in the memory transactions back to the data structures within the source code. This increases a programmer’s ability to effectively understand, develop, and optimize programs.

Limitations:

- The “Access histogram” reflects the memory accesses to the whole global virtual memory, while the “Memory region” shows a region of it. But falls short to show the mapping between virtual and physical pages.
- Although all these tools can help developers understand the kind of performance issues they are facing, they never give the reason why a particular issue is happening, for instance by showing the distribution of memory accesses within data structures.
- This paper is more oriented to visualization of the interconnect traffic

MemProf: A Memory Pro_ler for NUMA Multicore Systems
MemProf is a profiler that allows programmers to choose and implement efficient application-level optimizations for NUMA systems. MemProf builds temporal flows of interactions between threads and objects, which help programmers understand why and which memory objects are accessed remotely.

Limitation:

- It relies on programmers to establish a diagnosis and devise a solution
- It provide only textual output
- The developer might face a huge amount of information and not be able to differentiate normal behaviors from problematic ones.

A Tool to Analyze the Performance of Multithreaded Programs on NUMA Architectures
This paper presents a visualization tool displaying the monitored data in a user understandable way thereby showing the memory access behavior of shared memory applications. In addition, it projects the physical addresses in the memory transactions back to the data structures within the source code.

Limitation:

- It provide an address centric visualization, which shows how much each thread accesses a data structure. Such a visualization is a bit closer to providing the source of the performance issue, but it does not show how the accesses are distributed inside a structure, and how the structure is shared between the threads.

The Paradyn Parallel Performance Measurement Tool
Paradyn is a tool for measuring the performance of large-scale parallel programs. The goal of this new performance tool is to provide detailed, flexible performance information without incurring the space (and time) overhead typically associated with trace-based tools. Paradyn achieves this goal by dynamically instrumenting the application and automatically controlling this instrumentation in search of performance problems. Dynamic instrumentation lets us defer insertion until the moment it is needed (and remove it when it is no longer needed).

Limitations:

- Paradyn's Performance Consultant decides when and where to insert instrumentation. Although Paradyn enables dynamic insertion of probes into a running code, its analysis is limited to procedures and procedure calls whereas the proposed system can instrument at source level arbitrary code regions including single statements.

Ad Hoc Visualization of Distributed Arrays
This paper have incorporated data visualization in a debugger for parallel programs written with PVM. It use features of the debugger to write the requested data to a file for each PVM task,

provide a simple menu for the user to describe the data distribution, and create a single file with the data in natural order. The user then invokes any existing visualizer to look at the data.

Visualization and Analysis of Parallel Dataflow Execution with Smart Traces
This work investigates the requirements to create visualizations of execution traces of parallel programs modeled as dataflows. It proposes the Smart Trace (ST) concept, to encode the structure of the data, and guide the construction of specialized visualizations. A visualization tool can then leverage the relationships in the data to automate a given analysis task. The paper show with examples the power and flexibility of visualizations one can create to address specific questions formulated about the analysis of the data, with emphasis in parallel dataflow traces.

TABARNAC: Tools for Analyzing Behavior of Applications Running on NUMA Architecture
TABARNAC provides tools to trace and visualize the memory access behavior of parallel applications. More precisely, it helps to understand why performance issues occur by providing information on how data structures are accessed and shared by the different threads. Since it is based on memory accesses traces, TABARNAC has a very high accuracy while maintaining a reasonable overhead that enables the analysis of large applications. In an evaluation with several parallel applications, it is seen that relatively small code changes suggested by TABARNAC can substantially improve application performance. Providing a deep understanding of the memory access behavior, it enables the user to find the best mapping policy.