Relating Memory Performance Data to Application Domain Data using an Integration API

Benafsh Husain
Clemson University
bhusain@g.clemson.edu

Alfredo Giménez
University of California, Davis
alfredo.gimenez@gmail.com

Joshua A. Levine
Clemson University
levinej@clemson.edu

Todd Gamblin
Lawrence Livermore National Laboratory
tgamblin@llnl.gov

Peer-Timo Bremer
Lawrence Livermore National Laboratory
bremer5@llnl.gov

ABSTRACT

Understanding performance data, and more specifically memory access pattern is essential in optimizing scientific applications. Among the various factors affecting performance, such as the hardware architecture, the algorithms, or the system software stack, performance is also often related to the applications' physics. While there exists a number of techniques to collect relevant performance metrics, such as number of cache misses, traditional tools almost exclusively present this data relative to the code or as abstract tuples. This can obscure the data dependent nature of performance bottlenecks and make root-cause analysis difficult. Here we take advantage of the fact that a large class of applications are defined over some domain discretized by a mesh. By projecting the performance data directly onto these meshes, we enable developers to explore the performance data in the context of their application resulting in more intuitive visualizations. We introduce a lightweight, general interface to couple a performance visualization tool, MemAxes, to an external visualization tool, VisIt. This allows us to harness the advanced analytic capabilities of MemAxes to drive the exploration while exploiting the capabilities of VisIt to visualize both application and performance data in the application domain.

1. INTRODUCTION

Among the several strategies explored by programmers to optimize their simulation performances, one popular approach is to optimize the layout of the data and reduce memory access times. Data-centric performance analysis is becoming important as bandwidth from the processor to memory is increasingly small relative to processor performance, and data movement is expected to become a performance and power bottleneck on future machines. Understanding the access patterns of data structures for physics simulations, and their relation to performance, provides the opportunity to optimize the layout of the data structure. Most scientific applications are aimed at solving some physical process in a specified domain, for example examining the material behavior in a shock hydrodynamics application implemented over LULESH [1], where LULESH is a typical shock hydrocube. This hydrocube is depicted by a collection of volumetric elements defined by a mesh. This underlying physical domain of the application is referred to as the domain space, and in the case of LULESH is represented by an unstructured mesh comprising of connected elements.

Many modern physics algorithms are adaptive or data dependent, and computation is not uniform across the mesh. For example, an equation of state calculation may use very different interpolation methods depending on the type of material contained by particular mesh cells. Similarly, for an Adaptive Mesh Refinement (AMR) code, a mesh may be adaptively refined at various points as we later discuss in the application Chombo [2]. Also, in higher-order hydrodynamics codes, computation may depend on the order of a particular mesh cell. Therefore, studying the performance of algorithms on progressively higher-order cells becomes relevant.

Hence, it is beneficial to have the ability to view performance based on memory access of the data structure directly related to the domain space mesh. A context is provided to the programmer to relate variability in performance based on the underlying physics simulation. Achieving the projection of memory over the domain mesh can be particularly challenging since data from two different aspects of the simulation performance data and mesh data need to be related. In this paper we propose a lightweight technique to map performance data to the domain space mesh by integrating performance visualization tool to a mesh visualization tool through a lightweight API without requiring the user to modify their original data files.

Most scientific applications write out their application data files or visual data files with mesh information along with physics variables. These files are commonly in a format readable by visualization tools. Adding performance variables to these visual data files, during the run of the simulation can often times be difficult to achieve. Some of the techniques of achieving this projection of performance data over the mesh cells include: (1) Modify the visual data files read in by the visualization tool to include performance data, or (2) Create

VPA2015 November 15-20, 2015, Austin, TX, USA

ACM 978-1-4503-4013-7/15/11 ...$15.00.
DOI: http://dx.doi.org/10.1145/2835238.2835243

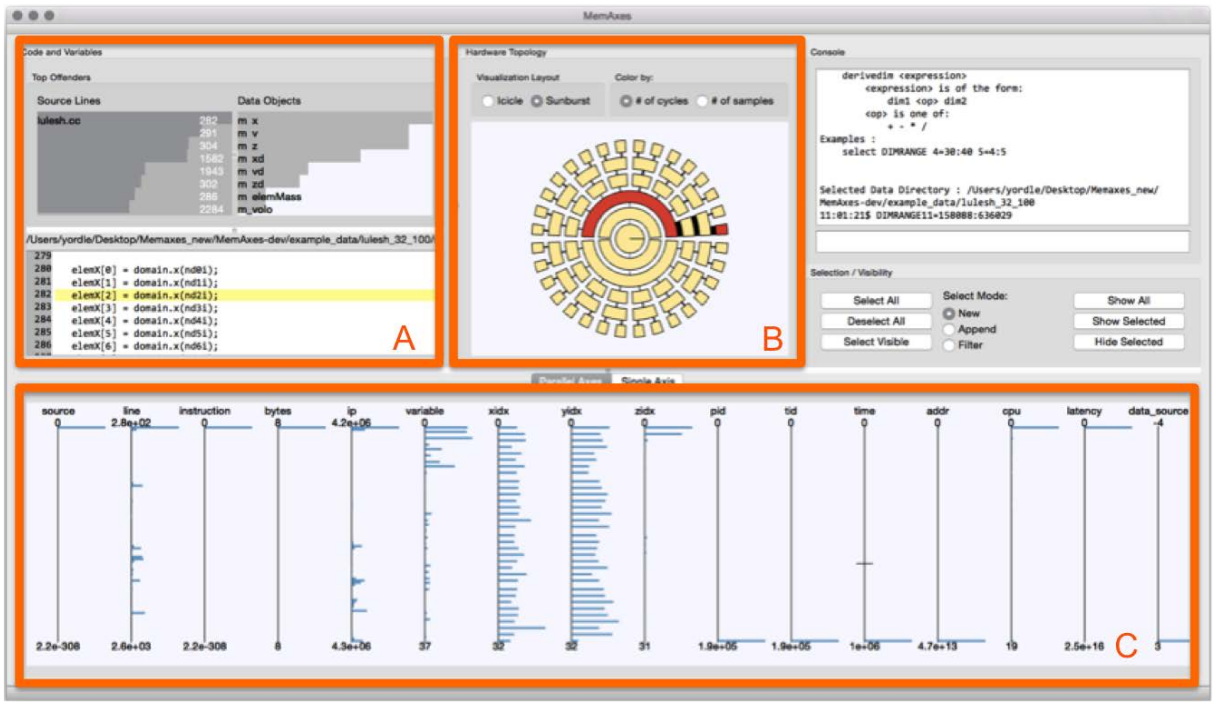

Figure 1: MemAxes application window upon loading the memory access data. (A) is the source code view, which visualizes the source files (not utilized for VisIt integration), (B) is the memory topology view, which shows accesses mapped onto a hardware layout, consisting of Sunburst and Icicle plots (Sunburst shown in the figure), (C), is the parallel coordinate view, depicting all attributes of the data as a high dimensional visualization. All these views have capabilities for interactive linking and brushing.

a visualization tool specifically for your performance visualization application. Option 1 can be challenging if the user is trying to add performance variables to the visual data files on the fly during simulation run. Post processing of the visual data files might be easier than during simulation run, but it requires the user to be familiar with several vis data formats. Both these techniques can prove to be cumbersome, and since performance visualization tools with a built in spatial domain representation are rare, the onus now lies on the programmer to construct one. Given that there are a wide variety of mesh formats, such as ALE, structured, unstructured, low and high order, AMR, non-AMR, etc. we reuse existing visualization tools and leverage their capabilities.

The contributions of this paper are as follows:

1. Provide a tool to record information required for the mapping, such as mesh coordinate information from visual data files.
2. Add multiple performance variables to the mesh on the fly without requiring modifications to the original visual data files
3. A general API and communication protocol that allows performance tools to integrate with scientific vis tools.

Our API does not require the programmer to decipher complicated visual file formats or implement a mesh viewer from scratch. To demonstrate this API we utilize the performance vis tool, MemAxes [3], and the visualization tool, VisIt [4]. MemAxes provides us with performance data including several data fields and an association to hardware topology to investigate the ease of integration, and effective data analysis, with a performance visualization tool. Using an existing visualization tool such as VisIt saves time and effort in rebuilding an existing vis tool, and user can effectively utilize all the interaction and analysis capabilities available in the vis tool.

2. RELATED WORKS

There are several applications catering towards visualizing performance data by itself. Rutar et al. [5] takes an approach similar to the data collected by Mitos [3] in our paper to focus on data centric mapping instead of being code centric. Their visualization techniques thus provides no context to data structure or to the application space. We try to mitigate their limitations by using a profiler collecting data for both code centric and data centric mapping.

Fewer efforts have been focused towards directly mapping performance over spatial or logical domain. Having said that, our work is not unique in concept, since past works have achieved similar results, but unique in its implementation methodology. We attempt to build on previously implemented ideas to mitigate their shortcomings. Schulz et al. [6] demonstrated mapping of performance hardware counters to the hardware and communication domains. They also utilize VisIt to map performance parameters directly over the application space, which is similar to the basis of our implementation. Once we achieve direct mapping over

the application domain we extend the implementation to further provide context with an external performance visualization tool.

Bohme's thesis [7] presents methods to extract performance problems generated by MPI that are load and communication imbalance related. They project wait-time propagation of Community Earth System Model (CESM) ice-sea model onto a 2D process grid using Cube 3D. Results published in the thesis iterate that mapping performance over a physical domain can lead to interesting insight. Although it lacks information required to extend similar mapping to a more general scenario. Huck et al. [8] has an approach similar to Bohme's thesis, where they use TAU [9] as a performance profiling toolkit and VisIt to visualize performance and metadata of The Model for Prediction Across Scales (MPAS) [10] application in the application domain space. They achieve this by creating an external performance data file collected by TAU and creating a new VisIt reader to augment reading the application data. Their implementation could re-create results similar to the ones described in this paper, but are limited to using TAU as a profiler. Switching to another profiler would essentially require re-writing of a VisIt reader to decipher the performance data and augment to the application data.

3. MEMAXES

We use MemAxes [3] as our performance visualization tool to integrate with VisIt. MemAxes is an interactive system aimed at visualizing and analyzing fine-grained memory access data on complex many-core architectures to support software and hardware developers in improving memory performance.

3.1 MemAxes Data

The data represented by MemAxes is collected via the application Mitos [3], which acquires fine-grained memory access samples during the simulation. MemAxes visualizes high-dimensional, multivariate instruction data with the ability to associate memory problems on aspects such as physical machine, the source code, and even within application data structures. The profiler involves the programmer specifying a set of data objects for which the attributes will be recorded. On a higher level, each memory access resolves to an event containing information such as: instruction pointer, data address, cache level accessed, processor id, number of clock cycles (latency) etc. On a finer level, the attributes include: 1) Source code line, 2) Source code file, 3) Data Symbol, 4) Buffer Size, 5) Element Size, 6) Access Index, 7) Accessed Value, 8) Resources Used, along with additional attributes defined by the user. A combination of high level information and finer level attributes form the tuple of data collected by Mitos.

The Access index attribute refers to a 1-dimensional index, obtained by the element size, start address, and data address. The 1-dimensional index is then resolved into spatial coordinates such as xid, yid, and zid. The xid, yid, and zid values correspond to the x, y, and z coordinates of the mesh in the visual data files read in by VisIt. Thus an instruction is resolved into several attributes of performance data, and when applicable correspond to mesh cell ids.

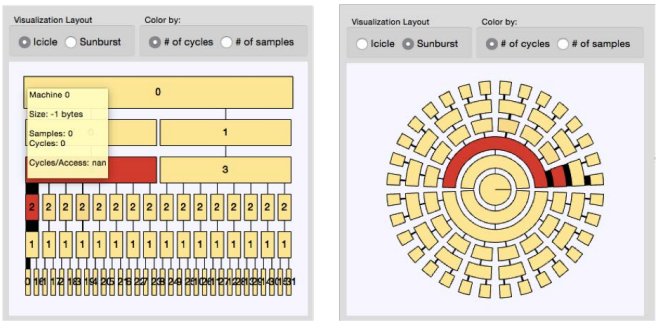

Figure 2: Memory topology, represented by the Icicle view and Sunburst view, which shows accesses mapped onto a hardware layout.

3.2 MemAxes Views

There are several views which are depicted in the MemAxes application such as source code view or selection statistics view. In this paper we focus on views which are integral in mapping data onto the domain space. We refer the reader to [**?**] for an in-depth insight into navigating the MemAxes tool along with interpreting the represented data.

The primary view representing all the contextual data for a particular application is represented by the parallel coordinates depicted in the bottom half of Figure 1. This view displays all the data types and their ranges and can be viewed for our purposes as a extensive selection tool. The parallel coordinates view depicts all the attributes of the Mitos output file and presents them at each coordinate and displays the min and max values. The user can use this view to navigate through MemAxes to obtain the hardware context of the performance data. Other relevant views of MemAxes include the Sunburst and Icicle views depicted by Figure 2, which are used to map the performance data onto a topological mapping of the hardware. The Icicle layout represents the CPUs, L1, L2, L3 caches and main node connectivity along with latency (or number of samples) data mapped onto it in an hierarchical manner. It also represents the allocations to CPU to their corresponding cache levels, i.e. which CPUs share a common L1 cache and so forth. The Sunburst plot has a similar representation in concentric circles. The upper semi-circle represents the Node 0, and the lower representing Node 1. We focus on these views since they provide a selection criteria for our API.

4. VISIT

VisIt [4] is an open source, turnkey application for large scale simulated and experimental data sets. The application is an infrastructure for parallelized, general post-processing of extremely massive data sets. VisIt can read several data formats, therefore the scientific application under consideration can generate visual data files for any pre-defined visual data formats. VisIt contains a rich set of visualization features to enable users to view a wide variety of data including scalar and vector fields defined on two- and three-dimensional (2D and 3D) structured, adaptive and unstructured meshes. Due to the implementations of VisIt plugins, it permits several external linking abilities, such as an extensive Python interface. We utilize this Python interface to implement our API. The combined advantages of using VisIt with its several in-

herent capabilities along with a performance tool results in integrating two different aspects of the data rarely viewed in conjunction.

The domain space representation in VisIt can display the physics data associated to the scientific application such as pressure, temperature, velocity vector, etc, but additionally, we provide the ability of mapping memory access data to the cells of the domain. In this paper we have mapped the latencies in clock cycles associated to each instruction if that instruction is resolved on a domain cell. Instead of latencies, other parameters such as number of samples, or average latency per sample can be mapped with minor modifications.

5. INTEGRATION API

In this paper we provide an integration API for the user to link performance data to the domain space. We demonstrate this by utilizing the two applications described above, MemAxes and VisIt. Figure 3 represents a flow diagram which describes the lightweight API, where modules in blue depict the existing application data file requirements and their inherent application capabilities, and modules in red represent the implementation of the API. As shown by the figure, MemAxes reads in the hardware topology along with memory access data, and is capable of displaying and analyzing memory access patterns in the context of the hardware architecture. As one of the views in MemAxes is the parallel coordinates, it serves as an elaborate list of collected data fields and their available ranges. Selecting a variable and a particular range will affect the corresponding display in VisIt. MemAxes thus provides a selection tool for the data to be displayed in VisIt, specifically in context to performance data correlated to the hardware topology. VisIt tools also provide selection techniques, but the key difference in utilizing MemAxes is that VisIt tools alone will lose any context to hardware topology or the source code.

VisIt also reads in the same Mitos output file (memory access data), along with visual data files representing underlying domain structure. Visual data files with data formats such as .vtk or .silo can be generated during the application simulation independent of collection of performance data. Visual file dumps for an application can be also collected intermittently during the run such as at a particular time interval or for every iteration. These visual data files depict the domain mesh structure of the application along with representing the data of the physical attributes such as pressure, temperature, etc, evolution during the course of the simulation run.

The output file of Mitos is generally a list of tuples and their corresponding values for every recorded memory access. The xid, yid, and zid are a part of the tuples, which refer to mesh cell coordinates. Visual data files read by VisIt generally do not contain any performance information themselves, but contain the mesh cell structure and physical variable data for each cell. By reading in the Mitos output file in VisIt, we provide VisIt with the information regarding the cell ids which have latencies associated to them. We can now replace physical variables displayed on the mesh by a performance variable for these cell ids, details for which are discussed in detail below. We essentially construct a data table of the Mitos data in VisIt. Therefore, when the user selects a data field and its ranges in MemAxes, VisIt correspondingly does a table look up for that data field and its range to determine the cell ids which fall under this selection. It then displays performance data (e.g.,latency) for those selected cells. Our technique thus also allows addition of variables associated to each cell of the mesh without the user having to modify the visual data files. Multiple performance variables can now be associated to the mesh cells, which is difficult to achieve while collecting visual data files during the simulation run.

Our lightweight API does not disrupt or reduce any of the builtin capabilities of the two applications, and therefore, MemAxes and VisIt are launched and run independently in the regular manner. All communication between MemAxes and VisIt goes through a local socket. A significant advantage of exchanging messages via sockets is the ability of the user to split the two applications over two machines instead of restricting viewing both the applications on the same machine. Since the primary aim is to link the two applications without affecting any of their inherent capabilities, we have minimized the modifications required to the actual source codes to both MemAxes and VisIt. Hence, permitting eventual developments of an API between any two performance visualization and mesh visualization tools.

5.1 MemAxes modifications

The modifications implemented to the original MemAxes application source code are limited to ensure that original capabilities of the application remain unchanged. The API intercepts any selection made by the user in the performance visualization tool, which is beneficial to conceptually integrate any performance tool into VisIt for performance data projection on domain space.

We implement the integration with the MemAxes application as a client. The MemAxes application keeps track of most of the selection and de-selection within its tool so as to enable interactivity between its various views. Thus, the modifications include tracking the current selection, storing the selection in a specified data structure, and sending the data structure message as a client through its port to the server. For example, in the Parallel Coordinate making a selection would create a client message of ***dimension number***, ***min value***, and ***max value***. Since VisIt has also read in the same Mitos output file, the dimension numbers and min and max values are meaningful to VisIt. Similar messages are created on making a selection in the other views of MemAxes. Our API adds a new context to the users on the basis of memory access data projected over the simulation mesh. This allows users to not only evaluate the performance data based on the distribution over hardware but forms a basis of evaluation on the physics of the scientific application.

5.2 VisIt API

The visual file dumps of the scientific application run are directly read into VisIt and can be displayed and manipulated through the known VisIt interactions as represented by Figure 3. The Mitos output file on the other hand can be read into VisIt using the ***PythonExpression Editor*** interface, which is a Python interface for VisIt to manipulate visual

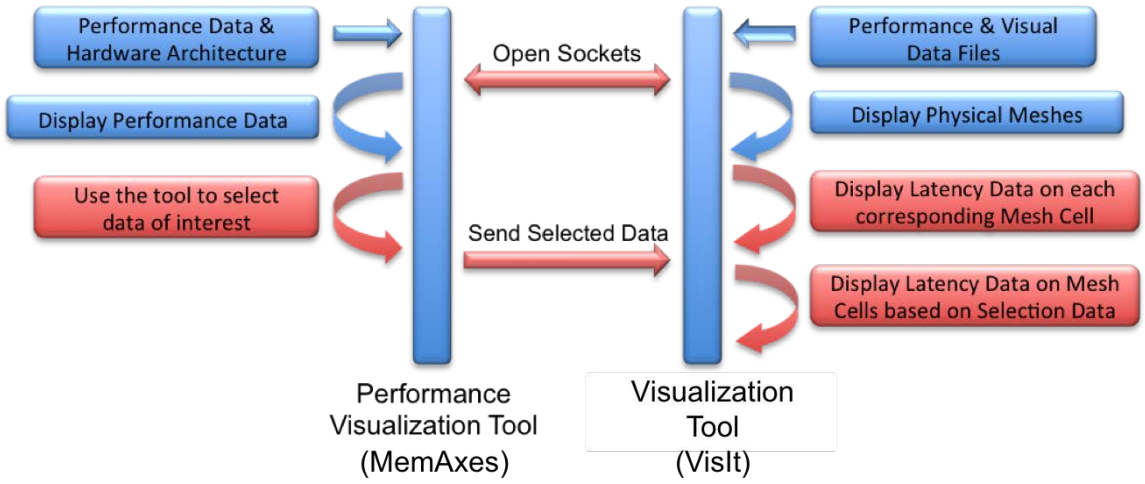

Figure 3: Control flow diagram representing the Integration API.

data files by external parameters. A manner of describing the expression editor is the ability to manipulate the physical data (e.g. pressure) in some mathematical form and displaying that manipulated result over the domain mesh.

The ***PythonExpression Editor*** interface is implemented as a server, which receives messages from the client (MemAxes), in the form of a data structure with information resembling: dimension and minimum and maximum values. The default values on the initialization of the data structure are the entire range of that data types, for example for data type CPU number, the default min would be 0 and default max, 31 (in the case of 32 CPU processes). We prove the capability to the user of sub-selection, that is if the user wants to select a CPU between CPU 1 and CPU 10, and within that selection observe only latencies above certain clock cycles, it is possible, and that every message received by the server does not reset all the other selections. The API on the VisIt interface recognizes the messages received from which of the corresponding views on the MemAxes side, and accordingly pushes those messages on the data structure.

One of the capabilities implemented in VisIt's ***PythonExpression Editor*** is the ability to extract information from the visual data files loaded into VisIt in a fine grained manner. This refers to accessing the desired cell coordinate for a particular iteration and their corresponding physical variables. This capability is interesting since data fields captured in the Mitos output files is the mapping of memory access data to a mesh cell location whenever applicable. This allows for representation of memory data directly over each cell of the mesh. We utilize the capability of VisIt to obtain per cell information to extract cell coordinate information and create a new variable alongside the physical variable, namely ***PerfVar***. In this case ***PerfVar*** corresponds to latencies in clock cycles. This implies that during the displaying of the domain space of the application, instead of displaying physical variables over the mesh, we assign a latency value to each of the cell coordinate.

Determining the latency values which are needed to be assigned to ***PerfVar*** is based on the selections sent through the client side of MemAxes. For example, if the selections from the memory topology view restricts the CPU number to CPU 1, the mesh displays only the cells which have accessed CPU 1 and their corresponding latencies. The selections can be further restricted, for example within CPU 1, only the latencies of cells of *x coordinate* value less than 15. We utilize the MemAxes views to achieve these selections. Based on the user analysis of interesting memory levels, selection of any memory levels in the Sunburst or the Icicle plot will restrict the display on VisIt to only that memory level. For the parallel coordinates view, any combination of variable ranges displayed by the parallel coordinates in MemAxes is correspondingly represented in VisIt. Since the parallel coordinates display all data fields, the user can restrict the data projected onto the mesh, for e.g., selecting xid values between 0 - 15, will display performance data on mesh cells only with xid values less than 15.

Although we are demonstrating this API by projecting latency information, it can be extended to any performance factor, such as number of memory accesses per cell, or the average latency per access for each cell. A new variable over the mesh cells can be created for every performance factor. Since VisIt now considers the performance variables the same as the physical variables in the original visual data files, all of the capabilities available in VisIt can now be utilized to further analyze performance data in context to the physical mesh. These capabilities may include determining the cells with maximum latency, or number of accesses, or the iteration with high memory accesses etc. This forms an important basis to utilize our API for memory−mesh integration, rather than the user developing a mock up of a mesh visualization tool. Existing mesh visualization tools are well developed with user supportive, and ample analysis capabilities.

Therefore, the output file of Mitos, which contains memory access performance data along with xid, yid, and zid (for each applicable event) is associating performance data to mesh cells ids. Utilizing these mesh cell ids in VisIt and replacing the physics variables with performance data allows us to project performance over the domain mesh. The API

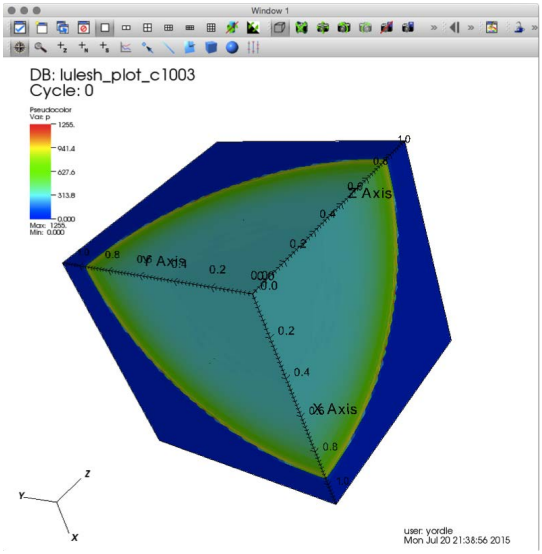

Figure 4: Hydrocube mesh of LULESH displayed in VisIt representing the *speed* variable.

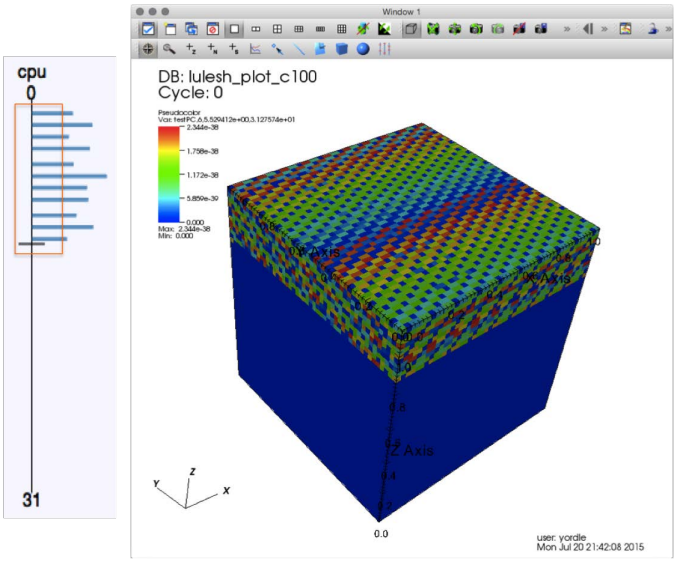

Figure 5: MemAxes Data field: CPU number selected from the parallel coordinate view used to restrict the latencies between CPU 0 to CPU 10. Latencies are projected over the LULESH shock hydrocube.

demonstrates the capability of projecting performance data of any performance profiler which is capable of linking data to mesh information. The mesh information can be in various levels of granularity, such as cell level or domain level mapping. We have designed the API in a lightweight manner to make the concept easily migrated between any performance visualization to visualization tools. Our API also permits easy modifications of the performance variables based on the requirements and specifications of the user.

6. APPLICATIONS

To demonstrate the integration of a performance visualization tool, MemAxes with physical simulation data, VisIt, we utilize two applications, LULESH and Chombo. Both these application are based on modeling hydrodynamics. These applications have an underlying domain space, which have memory accesses associated to them.

6.1 LULESH

Computer simulations of a wide variety of science and engineering problems require modeling hydrodynamics, which describes the motion of materials relative to each other when subject to forces. LULESH [1] represents a typical shock hydrocode, depicted by Figure 4 that approximates the hydrodynamics equations discretely by partitioning the spatial problem domain into a collection of volumetric elements defined by a mesh. LULESH is a highly simplified application, hard-coded to only solve a simple Sedov blast problem with analytic answers, but represents the numerical algorithms, data motion, and programming style typical in scientific C or C++ based applications.

6.1.1 Application View

On launching the VisIt application, the LULESH mesh represented by Figure 4 is displayed with physical variable *speed*. Our API does not restrict any capabilities of VisIt, and the user can independently maneuver through the displays. On any interaction with MemAxes, the API is initiated, where each cell of the mesh takes on the value of latencies associated with it throughout the simulation. Selecting a particular range of values in MemAxes, correspondingly limits the range of data displayed over the mesh.

Since LULESH represents an evolving mesh, each iteration of the simulation creates a separate visual data file. Visualization tools represent these groups of files as incremental time steps, which depicts the evolution of the mesh through the steps of the simulation. This is relevant to our API since we want the capability to map performance data onto the mesh at every iteration ('time step' in the case of VisIt) of the simulation. This API provides the capability to view latencies on the mesh cells (1) irrespective to the iteration of the visual data file, i.e. latencies are aggregated for the entire simulation run and projected over the mesh (2) aggregation only up to the iteration of the mesh currently displayed, i.e. at every time step of the iteration the latencies associated to each cell increases, (3) latencies per cell displayed only for that iteration. This assists the programmer in utilizing both a global view of the latency distribution or to identify a particularly offending iteration cycle.

6.1.2 Case Study

One approach to analyzing performance data is to determine the distribution of memory accesses between CPU cores. In Figure 5, on selecting the CPU ranges from 0 to 10 we observe that a contiguous numbers of cores relate to a contiguous array of mesh cells. By the principle of spatial locality, caches are more efficiently utilized when they are used to access data elements which are grouped closely together. Therefore, ideally these contiguous cores should be in close proximity to each other to access closely grouped data. Therefore, by placing colossally grouped data on lo-

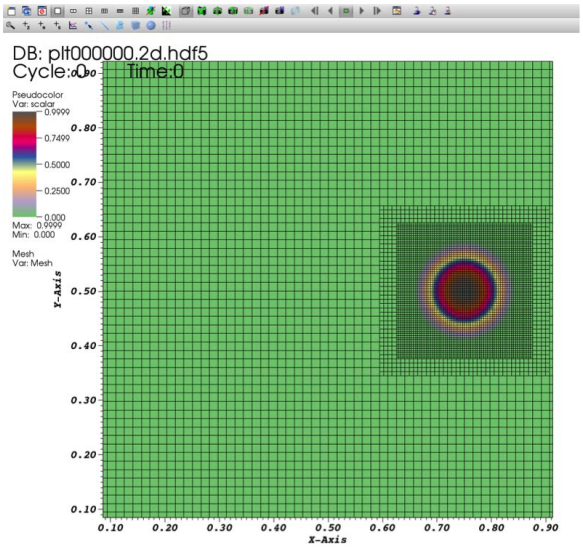

Figure 6: AMRGogunov example visual data file using Chombo library, representing a mesh with three domains of varying granularity.

cally spatial cores, one might observe improve in performance.

6.2 Chombo

In scientific applications, partial differential equation problems such as multiple length scales and strong spatial localizations are commonly occurring problems. Examples include Nonlinear systems of hyperbolic PDEs in combustion problems. Finite difference calculation using block-structured adaptive mesh refinement (AMR) is a powerful tool for computing solutions to partial differential equations involving such multiple scales. In one particular approach, the domain space is converted into a rectangular grid by discretization, and the solution is then composed on the regular grid. It identifies regions requiring additional resolution by a local measure of original error. A new solution is then calculated on the composite grid structure. Chombo [2] utilizes this approach of computing time-dependent solutions to hyperbolic partial differential equations in multiple dimensions with an implementation of C++ classes designed to support block structured AMR applications. Chombo is based in part on the BoxLib toolkit.

Applications involving AMR are interesting from a performance stand point since a particular cell can be associated to memory accesses at various granularity levels. Hence, optimized data-structure layout becomes essential. From the extensive library of Chombo we select the AMRGodunov example to demonstrate the relation between adaptive mesh calculation and performance. The AMRGodunov example uses the unsplit, second-order Godunov method for integrating systems of conservation laws, such as the Euler equations of gas dynamics, on an AMR grid hierarchy. We observe the output of the application in Figure 6, where there are three granularities of the mesh overlaid on top of each other. Each granularity is referred to as the domain of the mesh, and each domain contain their individual cell ids and values. On annotating the AMRGodunov example with the Mitos API, we obtain the Mitos output file with tuples similar to the one described for LULESH. One major difference being that ever xid, yid recorded also has the domain level information associated to it.

6.2.1 Case Study

In our API we have implemented the capability to extract domain level information for the mesh. VisIt, on reading the data associate the cell ids recorded in the Mitos file to the corresponding domain level mesh id in the visual data files. In Figure 7 we show distribution of latencies over the three domains of the AMR mesh. This allows the user to relate latency distribution over the physics simulation of a high resolution grid. The figure represented is displayed as a logarithmic color scale to allow the users to identify lower latency cells. We utilize our API to define a new variable of average latency per access which is depicted in Figure 8. Since all VisIt capabilities are available to the user, they can select their desired display specifications. Figure 8 is obtained by restricting the display of latencies to CPU 0-14. Similar to the case displayed in LULESH, the domain of the finest granularity has latencies projected in a more concentrated manner over the bottom left of the domain. This refers to neighboring cell ids in this region perform multiple memory accesses for the 0-14 CPUs. The CPUs can be so adjusted to achieve spatial locality.

7. DISCUSSION AND FUTURE WORK

There are several directions intended to improve the integration API. Firstly, we intend to explore more applications that would benefit from performance mapping on to the mesh structure. To achieve this we aim to modify the API to a more general implementation using Conduit [11]. Conduit is an intuitive model for describing hierarchical scientific data in C, C++, python, Fortran etc. It creates *nodes* consisting of the memory data structure defined by the user. The novelty of Conduit lies in the sharing of nodes between different applications, written in different programming languages, where each application with the same node would resolve the same data structure. Hence, Conduit will permit the VisIt API to receive the node in the same data structure, as when it is created by MemAxes. The node permits generalization to the degree where our API does not need to have prior knowledge of the data fields or the data structure.

Another interesting modification can include two way communication between the MemAxes and VisIt. In the current implementation, although MemAxes is able to receive messages from VisIt, we are not sending any meaningful data. Interesting messages to send to MemAxes could include, selecting clusters of cells or domains based on memory data and correspondingly the views in MemAxes get updated. Similarly, utilizing builtin VisIt (or any mesh visualization tool) operations, such as histograms or correlation and projecting the results in MemAxes over hardware topology. Part of our research efforts are also focused towards determining if there exists a meaningful correlation between performance data and physical variables. Although this correlation metric is a challenging issue in itself, our API provides a strong basis for representing the correlation visually.

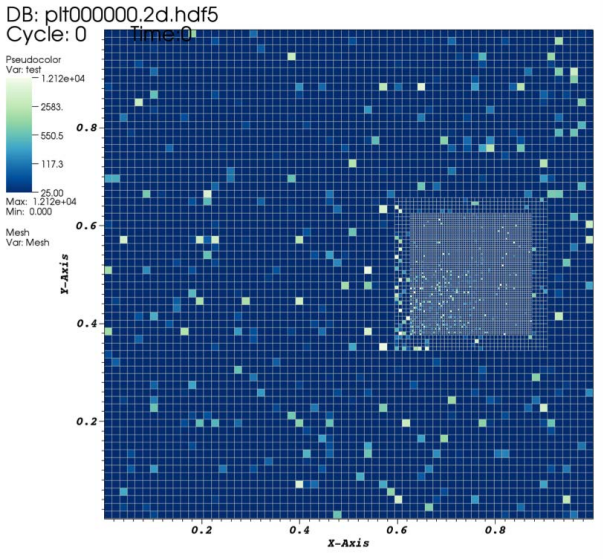

Figure 7: Latency data represented over an AMR mesh consisting of three domains.

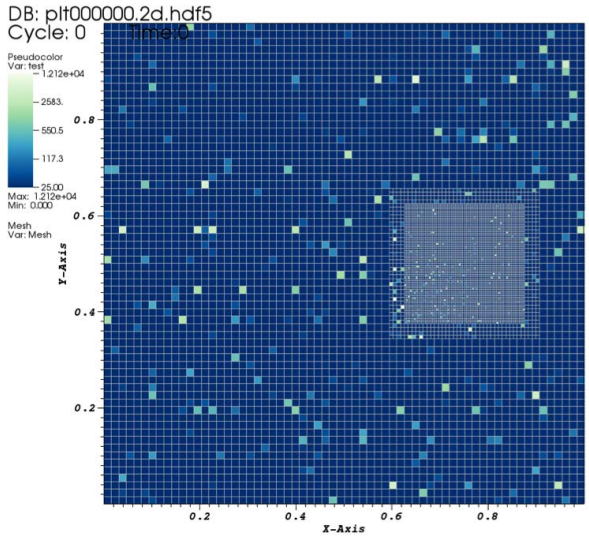

Figure 8: Defining a new variable: Latency per sample for the AMR mesh.

8. CONCLUSION

Application performance are dependent on the layout of data-structures and their memory access pattern. Relating the data-structures to the application physics provides insight to the programmer for optimizations, which can be achieved by projecting performance data onto the application domain mesh. These projections can be achieved by either building a user specific visualization tool or modifying visual data files to include performance data. Both these techniques can be cumbersome, and future extensions may prove to be difficult. In this paper we have demonstrated a lightweight API which integrates the memory performance visualization tool, MemAxes, to an existing visualization tool, VisIt. Using the Python interface of VisIt, we utilize local sockets to map performance data to the cells of the domain mesh. This API allows users to use performance tools as a navigation mechanism to view performance over the interesting mesh cells.

9. ACKNOWLEDGEMENT

This material is based upon work supported by the National Science Foundation under Grant No. IIS-1314757.

This work was also performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory (LLNL-PROC-678035).

References

[1] Hydrodynamics Challenge Problem, Lawrence Livermore National Laboratory. Technical Report LLNL-TR-490254.

[2] M. Adams et al. *Package for AMR Applications - Design Document, Lawrence Berkeley National Laboratory Technical Report LBNL-6616E.*

[3] Alfredo Giménez, Todd Gamblin, Barry Rountree, Abhinav Bhatele, Ilir Jusufi, Peer-Timo Bremer, and Bernd Hamann. Dissecting on-node memory access performance: A semantic approach. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, pages 166–176, Piscataway, NJ, USA, 2014. IEEE Press.

[4] Hank Childs. Visit: An end-user tool for visualizing and analyzing very large data. 2013.

[5] Nick Rutar and Jeffrey K. Hollingsworth. Data centric techniques for mapping performance data to program variables. *Parallel Computing*, 38(1âĂŞ2):2 – 14, 2012. Extensions for Next-Generation Parallel Programming Models.

[6] M. Schulz, J.A. Levine, P.-T. Bremer, T. Gamblin, and V. Pascucci. Interpreting performance data across intuitive domains. In *Parallel Processing (ICPP), 2011 International Conference on*, pages 206–215, Sept 2011.

[7] David Böhme. *Characterizing load and communication imbalance in parallel applications*, volume 23. Forschungszentrum Jülich, 2014.

[8] Kevin A. Huck, Kristin Potter, Doug W. Jacobsen, Hank Childs, and Allen D. Malony. Linking performance data into scientific visualization tools. In *Proceedings of the First Workshop on Visual Performance Analysis*, VPA '14, pages 50–57, Piscataway, NJ, USA, 2014. IEEE Press.

[9] S. Shende and A. D. Malony. The tau parallel performance system. In *International Journal of High Performance Computing Applications*, vol. 20, page 287?311, 2006.

[10] Lanl and ncar. MPAS, http://mpas-dev.github.io.

[11] Conduit. Technical Report LLNL-CODE-666778.