Title goes here

Servesh Muralidharan
School of Computer Science and Statistics
Trinity College Dublin
Dublin 2, Ireland
Email: muralis@tcd.ie

David Gregg
School of Computer Science and Statistics
Trinity College Dublin
Dublin 2, Ireland
Email: David.Gregg@cs.tcd.ie

***Abstract*—the abstract goes here.**

I. Motivation

In the continued race for improvement of multi core and many core architectures, computer networks are becoming a fundamental area for performance bottleneck especially in systems which are used for distributed processing of parallel applications. In order to handle the requirements posed by these applications the interconnects are becoming more and more complex. Such systems utilize a wide range of custom interconnects ranging from infiniband, myrinet to the more readily available ones such as 1Gbe and 10Gbe Ethernet standards and sophisticated routers and switches to handle the added challenge in routing data. While a small group of the specialized systems employ custom routers and topology configurations the majority of them especially those that use 1Gbe or 10Gbe use the commercial routers. In addition the growing complexity of routing protocols has resulted in advanced switches such as OpenVSwitch [**?**], etc., has resulted in these routers being upgraded frequently to keep up with the growing requirements. Software routers such as click modular router have been gaining popularity due to their flexible nature and cheaper alternative to the more expensive commercial routers and are widely used to prototype newer standards. In using software routers in the above scenario it would be possible to not only benefit from the newer developments in switching and routing protocols but it would require little effort to implement it in existing systems. One might argue that it would be a challenge to handle such complex workloads in software routers but with the presence of multicore and many core architectures and stream processors it is feasible to parallelize the packet processing workloads to improve the overall performance of the system. Several research projects are working on improving the performance by parallelizing the packet processing tasks on these architectures [**?**] [**?**].

In a computing cluster which is used in the execution of parallel applications it is common to dedicate a core or processor in each node for operating system related tasks in order to minimize its effects on the application. Similarly, it should also be possible to utilize an additional core or processor to perform packet processing operations. While this is would be a challenging scenario it does provide the added benefits of a cheaper alternative to commercial routers. Also in bringing the network system into a software controlled domain it should be possible to perform additional optimizations from the application such as prioritized data flow, intelligent routing, etc., that would otherwise be difficult to implement due to the abstraction of the network system.

The focus of this paper is the development of a parallel processing engine that could be integrated with the application and can perform both the actual computations and the packet processing related tasks in parallel. The entire system can be divided into key parts as follows,

- A suitable methodology by which we can access the data in the userspace without interacting with the operating system. To do this (1)We need to read the data from the Network Interface Controller (NIC) before the OS and (2)Using this, establish a efficient manner for sending and receiving the data.(Section **??**)
- Given a graph that represents the workload to be done on the packet flow, we divide it into tasks that can be executed in parallel and determine the regions that could be used to extract data and task parallelism.(Section **??**)
- The parallel processing engine that utilizes the above and does the actual computations on the multi core or many core processor. It also handles load balancing and scheduling of the work across the available computing resources. (Section **??**)

We concentrate more on the framework that can integrate the packet processing and application tasks into a parallel processing engine and the extraction of parallelism from the packet processing related tasks to improve the overall performance. Given an application graph and its partition consisting of parallel tasks we then combine it with the packet processing operations. In addition, Section **??** we show the challenges involved in using a system consisting of threads and process that are used to actually perform the work. The extent to which we extract parallelism ranges not only to the packet tasks but to the low level access of data I/O to the network interface hardware through certain additions to the netmap APIs. Finally in Section **??** we present certain pseudo applications that involves operations such as compression and encryption to evaluate our parallel processing engine.

II. Background

Let us consider a simple file storage system that is based in a data center. The application handles sensitive information by performing encryption on the data that matches predefined rules like file names, access level of user, etc., otherwise the default operation is to perform compression and store the information in the location specified. For the purpose of this example let us assume that the filesystem is located on a NFS server and it is responsible for concurrent operations on the data. A group of servers act as the interface to the storage system and the requests are evenly distributed over these nodes. The important operations that these nodes perform on the data are represented in the stream graph shown in figure **??**. In the data retrieval part the information fetched from the filesystem is checked to see if it matches a set of predefined rules to check if it has to be encrypted otherwise passes it to the compression operation. During data storage the input data is checked whether it has to be decrypted or decompressed and then stored on to the filesystem.

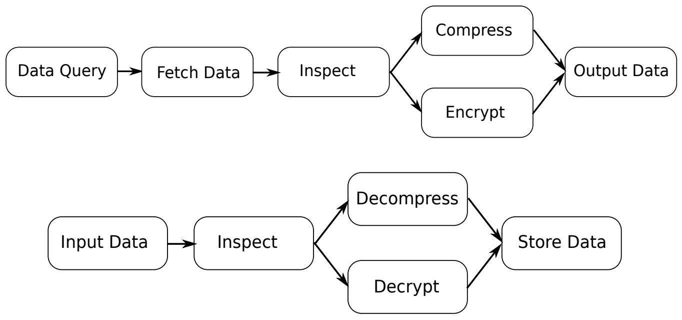

Fig. 1. Operations performed on the data stored and retrieved

The nodes that are connected in the cluster are linked through a switch or router. In such a scenario the number of routers increase with the size of the cluster and the bandwidth required by the application. Due to this the performance of the application is directly dependent on the network especially the functionality of the routers. Moreover with a fixed infrastructure such as this it is not only difficult to upgrade the routers but it would also be a expensive proposition. In such a situation it makes more sense to use software routers which are much more flexible and cheaper and could prove to be a more effective than or in addition to commercial switches or routers.

In implementing the software routers if one were to dedicate several nodes for this purpose it might prove to be inefficient use of resources since the network traffic generated is highly dependent on the applications being executed in such a closed environment. Moreover such systems might act as a communication bottleneck and the routing operations are focused on these nodes. A more effective approach would be to bring the software routers into the nodes that are used for executing the applications itself. In doing this we dedicate a specific set of resources of each node for the packet processing operations and in essence make each node act as a router. All the nodes in such a cluster are connected with each other based on network topology suitable for its requirements. This would make it possible to divide the workload generated due to the network operations across all the nodes in the cluster. In this paper we propose such a system that not only brings the software routers as separate entity into the actual application nodes but takes it a step further and integrates the network operations with the application itself.

There are several challenges arises due to this mainly how do we tackle such large number of computations both from the application and the network effectively. Two important constraints would enable us to satisfy the requirements posed by such a system. First, the operations of both the application and the network is parallel in nature and second, the presence of co-processors[], network processors[], Graphics Processing Units (GPUs) & General Purpose GPUs (GPGPUs) has dramatically improved the capability of commodity hardware. With this theoretically, we should be able to meet the computational demand but if such a system were to succeed an efficient parallel processing engine is crucial.

write a para forward referencing other sections

An important part of our system is the tasks related to packets such as routing, forwarding, etc., Since our design focuses more on the aspects of the parallel framework we utilize the click modular router for the packet processing tasks. Click modular router [**?**] is a commonly used software router that includes a library of elements which is used in building flexible and reconfigurable routers. It offers a platform that is flexible for rapid development of newer network protocols and achieves good performance. An improved version known as SMP Click [**?**] also supports thread based parallel processing of network related tasks. Until recently click's performance in the user space was not as good as its kernel space equivalent. This is mainly due to its extensive use of polling driver in the kernel space. Netmap I/O [**?**] provides a set of APIs that could directly access the packets from the NIC and it helped in the recent improvement of click's user space performance. We found the simpler construction of netmap easier to adapt to our framework compared to other user space packet access APIs.

Click dynamically constructs routers in which the configuration file is passed as a parameter for the kernel module or user space binary. As a result most of the actual components are dynamically allocated and connected at run time. While this may be an advantage considering the alternate is to recompile the linux kernel, it does act as significant bottleneck in the case of user space. Moreover some of the restrictions on how the configuration can be parallelized limits the actual performance that could be achieved in combination with netmap. By integrating the ability to access click's elements within our parallel processing engine we were able to scale significantly better. This does require a limited amount of modifications to some of click's libraries and elements. In generating the actual code statically, compilers such as gnu c, etc., can perform optimizations such as loop fusions, in-lining, dead code optimization etc., which were not possible previously.

III. Old Background

The constant advancement in server hardware has made systems with large number of hardware threads[] readily available. In addition to this the presence of co-processors[], network processors[], Graphics Processing Units (GPUs) & General Purpose GPUs (GPGPUs) has dramatically improved the capability of commodity hardware. These developments has paved way to a platform with immense computing potential which existing applications already take advantage of. Along with the improvements in the Multi-Core & Many-Core processor architecture the memory and I/O speeds have also seen significant improvement. In particular the network subsystem has seen significant growth, in combination with the improvements to optical communication it has lead to 10GbE interfaces being quite common in enterprise servers. While 40GbE and 100GbE are available for backbone infrastructure but would eventually follow into commodity hardware.

Network applications represent those applications that operate on any of the seven network layers described in the OSI Model. {describe more here}

Several previous work have discussed the process of parallelizing network applications[][][] to enable it to run faster in such multi threaded systems. The development of Software routers[][][] is encouraged further because it costs only a fraction of the commodity hardware routers. Moreover the advancements in the network interfaces has made faster interfaces readily in combination with the multi core servers the performance It has been shown that such routers can be used to process packets at line rate [][][] which has been a challenge on its own. Network applications including those that operate on few or all of the network layers have been designed with sequential processing as the target environment. The shift to multi core architectures in recent times has induced such applications to utilize parallel processing as a alternate means of computing. While this could lead to significant improvement in performance it also suffers from the same problems as any parallel application. These include scalability, load-balancing, data dependency and the presence of non parallel regions in the program apart from those that might arise due to that specific application. The scaling of software routers has been discussed extensively in RouteBricks[] which shows the problems suffered by software routers when they are scaled beyond a single system. One could argue that complexity of network applications such as Deep Packet Inspection (DPI), Packet Forwarding, etc., could be handled by a single many socket Multi-Core SMP system. However, as the complexity of network applications develops it would be inevitable but to move to a multi node environment in order to perform the computations and to handle more network bandwidth.

Large scale computing clusters has been used in high-performance scientific application to solve problems and applications of very large size. It uses a cluster of compute nodes and distributes the workload across them in order to solve it faster. These systems utilize parallel programs consisting of both threads and MPI to break massive problems into smaller workload that can be executed in parallel. In addition, the presence of GPUs and GPGPUs requires a mixture of parallelization constructs such as OpenCL, CUDA, etc., in order to achieve the best performance in the system. In this paper we discuss how some of the techniques used by these applications could be used to parallelize network applications.

Click Modular Software Router[] consists of extensive elements which are used to construct software routers. Click primarily targets packet processing applications especially those that operate on L4 - L7. {elaborate more here}

Discuss about what each sections contains here.....

IV. Design

This section describes the design principles related to our methodology. The objective is to determine the requirements necessary to parallelize network applications. Our methods focus on ways to improve the click system in some cases even extend its functionality.

A. *Emphasize on user level execution environment*

The existing software routers are predominantly implemented in the kernel space for the ease of access to resources and can bypass the OS overhead incurred when data is accessed through the network subsystem. While this alternative provides better performance it also leads to concerns regarding stability and security of the system. Newer techniques such as Netmap[] allows direct access to network buffers which makes it possible to write high performance packet processing applications in userspace itself. Typically network applications which operate on L4 - L7 are developed in the kernel space in order to intercept the network packets before it flows through the OS. Whereas those applications that utilize L1 - L3 are present in the user space. The objective here is to completely switch to user space to solve any of the stability or security concerns. In addition, scenarios where the applications that operate across most or all the layers could be improved by combining the operations across the different layers in user space itself.

B. *Eliminate the overhead associated with creation of dynamic objects*

Click constructs application graphs based on dynamic creation of objects and passing the data packets across the virtual functions. Though this is a very elegant approach in creating the software router it does incur the penalty of the virtual function tables. The click-tool[] addresses this issue but it does not solve the problem entirely. The construction of the software router at run time is the primary reason behind it. This mainly arises due to the presence of kernel module which installs the router as a kernel module based on the input configuration. However in the user level it would not be necessary for a separate process to initialize the router instead we could generate the entire program statically including the necessary flow associated with the specific application. This not only eliminates the overhead associated but also opens up means to optimize the graph further.

C. Optimize based on application graph

Network applications especially that operate on the lower layers are very sensitive to the overhead associated with operating on the data separately on each layer. Instead given an application graph if we were to provide a sequential flow of the different functions at the various layers then it would be possible for the compiler to optimize the regions of program into a single and optimized block of code. This would also improve the cache locality of the data accessed by consecutive functions. This also presents us with the ability to perform other types of stream graph optimizations[] which would not be possible if the specific graph was constructed dynamically at run time.

D. Automated data parallelism

Network traffic is a classic example by which data parallelism can be utilized to achieve scalable parallelism. Click's support for parallelism in user level is described as experimental, it is implemented by using a task graph consisting of each work associated with each element which is executed in the threads. The support for parallelism in the kernel mode is better in comparison to that of user mode. Since our target environment is in the user space the proposed design involves a different approach to which we can extract data parallelism. We first split the data across multiple queues and then replicate the application graph across to process the data the queues in parallel. An important condition here is to make sure packets of the same flow should pass through the same queue. Since the application graph can be replicated it also makes it possible to dynamically load balance several applications based on the workload.

E. Efficient implementation of task parallelism

In generating the application graph as a sequence of operations it also makes it possible to process different stages of the operation in a pipelined manner. This when combined with stream optimizations and could be used to determine which steps of the graph is best suited to form the stages of a pipeline. The condition to balance here is the weight associated with computation in that stage with that of the overhead incurred in computing the data in two separate stages. It would be necessary to determine whether the additional over head involved in executing the task in parallel provides any significant improvement in performance.

V. Implementation

Click was designed as a means to easily implement a software router on top of a Linux OS. This is implemented in two different ways one is by installing a kernel module and the other is a program that executes in the user level. Both of them accept a configuration file described in click's language specification. This is then used to construct the router by linking the virtual functions for the push and pull operations of the different elements in accordance with the specified graph.

A. Modifications to click system

1) Static Generation Of ???: The first step in the design was to overcome the limitation of using click's infrastructure to construct the software router which consists of dynamically initialized objects and virtual functions. This was necessary due to two important conditions, one was to eliminate the overhead associated with the dynamic construction of the objects and virtual functions and the other was to provide the compiler a chance to optimize the program during the compilation phase. This would however result in not being able to construct the software router using click's infrastructure but would require the application specific code to be recompiled. This could be a significant hurdle in the case of running the software router in kernel mode where the router configuration is installed by the click kernel module in which case the entire kernel module should be recompiled and reinitialized. Whereas in user space this would not be an issue since routers of different configurations are basically treated as separate applications and would just have to go through an additional compilation phase before being used. The added advantage of this would be to enable network applications that operate on L1 - L3 layers to use the packet processing (L4 - L7) infrastructure provided by click elements.

2) Implementation of data parallelism: –Talk about MPI & threads

–Our goal here is to focus on the network application as a workload and characterize it as such without involving any kind of overhead that might occur due to the network.

VI. Experimental Evaluation

VII. Related Work

VIII. Conclusions

Acknowledgment

This work is funded by the IRCSET Enterprise Partnership Scheme along with IBM Research, Ireland.