A Heterogeneous Parallel System Running Open MPI on a Broadband Network of Embedded Set-Top Devices

Richard Neill
Dept. of Computer Science
Columbia University
New York, NY
rich@cs.columbia.edu

Alexander Shabarshin
Cablevision Systems
Bethpage, NY 11714
ashabars@cablevision.com

Luca P. Carloni
Dept. of Computer Science
Columbia University
New York, NY
luca@cs.columbia.edu

ABSTRACT

We present a heterogeneous parallel computing system that combines a traditional computer cluster with a broadband network of embedded set-top box (STB) devices. As multiple service operators (MSO) manage millions of these devices across wide geographic areas, the computational power of such a massively-distributed embedded system could be harnessed to realize a centrally-managed, energy-efficient parallel processing platform that supports a variety of application domains which are of interest to MSOs, consumers, and the high-performance computing research community. We investigate the feasibility of this idea by building a prototype system that includes a complete head-end cable system with a DOCSIS-2.0 network combined with an interoperable implementation of a subset of Open MPI *running on the STB embedded operating system. We evaluate the performance and scalability of our system compared to a traditional cluster by solving approximately various instances of the Multiple Sequence Alignment bioinformatics problem, while the STBs continue simultaneously to operate their primary functions: decode MPEG streams for television display and run an interactive user interface. Based on our experimental results and given the technology trends in embedded computing we argue that our approach to leverage a broadband network of embedded devices in a heterogeneous distributed system offers the benefits of both parallel computing clusters and distributed Internet computing.*

Categories and Subject Descriptors

C.5 [**Computer System Implementation**]: Miscellaneous

General Terms

Design

Keywords

Distributed Embedded Systems, MPI, Set-Top Box, Multiple Service Operators, Multiple Sequence Alignment.

CF'10, May 17–19, 2010, Bertinoro, Italy.

1. INTRODUCTION

The explosion of broadband as well as mobile device Internet connectivity has led to rapid increases in the number of consumer embedded devices and applications. Growth of mobile Internet computing has outpaced similar desktop Internet adoption: e.g, during the first two years since launch Apple had acquired over 57 million iPhone and iTouch subscribers, more than eight times the number of AOL users for a similar period [16]. Among the largest providers of consumer embedded devices are traditional *multiple service operators (MSO)*, which in recent years have grown from suppliers of basic subscription TV programming to providers of rich multi-platform interactive media services available over both wireless and high-speed broadband networks. Since 2006, MSOs have accelerated the purchase and deployment of next-generation embedded *set-top box (STB)* devices to support the deployment of digital interactive services and content. According to market research [15], worldwide STB shipments are projected to grow to over 150 million units in 2010, rising to nearly 201 million units by 2013 (Fig. 1).

Growth in mobile computing and broadband STB computing is largely being driven by two trends: 1) rising consumer demand for personalized, easy-to-navigate, content services that are portable between devices and accessible anywhere at any time; 2) emerging social networking applications that offer connected experiences across multiple devices and content formats. MSOs are responding to these trends by developing services such as streamed video-content delivery over multiple consumer devices and by blending together interactive TV and mobile Internet applications. Today, for example, MSOs offer the availability of personalized content search [17], mobile-based video programming of home digital video recorders, and social-networking applications that access Twitter, Facebook, and YouTube through the Internet on a variety of consumer devices. Tomorrow real-time recommendation and personalization systems for millions of users will require the processing of vast quantities of distributed data with sophisticated analytical techniques based on machine learning and parallel data mining. Furthermore, user-generated content within a social computing application that can be accessed on any device will require fast transcoding between multiple formats as well as coordinated distribution of content, two tasks that demand considerable computation and distributed processing.

In order to meet the growing computation and communication demands, MSOs will soon need to assemble complex computing infrastructures which are distributed in nature, highly heterogeneous —as they consist of both computer

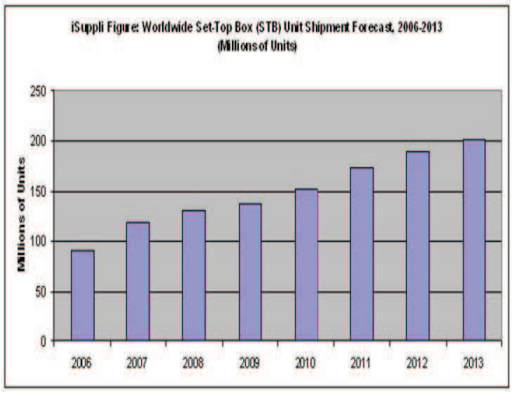


Figure 1: Worldwide set-top box unit shipment forecast, 2006-2013 [Source: iSuppli Corp. Aug 2009].

clusters and networked embedded devices— and must be able to scale to execute concurrent applications across millions of devices.

In this paper we argue that MSOs have the opportunity to build on top of their existing infrastructures while leveraging current trends in both broadband networking and embedded processing to develop a parallel computing system that can support a mix of application workloads spanning from social networking and ubiquitous content access to large-scale distributed processing and even some classes of high-performance computing (Fig. 2). In particular, the value of this opportunity stems from the unique property of this computing system, namely the ability to reach millions of geographically-distributed embedded-consumer devices with a *dedicated and centrally-managed broadband network.*

As the performance of MSO-managed networks continues to grow and STBs start featuring sophisticated multicore processors [5], we propose a heterogeneous system architecture that combines a traditional Unix-based computer cluster with a broadband network of STB devices. In order to evaluate the potential of this idea we implemented a complete prototype system which represents a scaled-down version of the proposed architecture but can fully support a representative MSO streaming-video service (Section 2). We also developed an inter-operable subset of Open MPI which can run on the STB real-time operating system and, therefore, can act as the middleware for the heterogeneous system integration (Sections 3 and 4). We tested our prototype system by porting an MPI-implementation of ClustalW, a computationally-intensive bioinformatics application so that we could run it on both the computer cluster and the network of STBs (Section 5). Our experimental results show that it is possible to execute ClustalW efficiently on our system while the STBs continue simultaneously to operate their primary functions, i.e. decode MPEG streams for monitor display and run an interactive user interface, without any perceived degradation (Section 6). Indeed we observe that content services on the STBs are unaffected by the presence of parallel computation due to the separation of content and application processing functions within their micro-architectures. Major challenges, however, need to be addressed in order to provide a highly-scalable heterogeneous runtime and efficient messaging-passing environment for distributed computation with millions of embedded devices.

In summary, the primary goal that we achieved with our experiments has been to determine the feasibility, challenges,

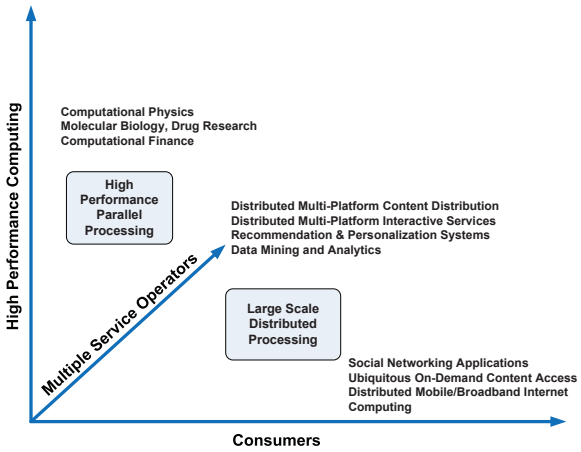


Figure 2: The evolving computational landscape for MSO-managed broadband STB networks.

and opportunities of utilizing embedded STB devices for heterogeneous parallel computing and distributed processing. Second, we gained precious insight into issues associated with heterogeneous parallel system comprising Unix systems, distributed embedded devices, and an embedded messaging-passing framework based on MPI. We believe that the experience that can be acquired with our current prototype system (and the future generations that we plan to develop) can help MSOs in the design and implementation of scalable solutions for distributed computation on their system infrastructures and may also be of benefit to parallel computer architects as it pertains to future scaling of runtime environments for large clusters of multicore systems.

2. SYSTEM ARCHITECTURE

Fig. 3 gives a detailed view of the prototype system that we designed and implemented as a scaled-down, but complete and representative, version of the heterogeneous parallel computing system that we envision. This is a distributed-memory, message-passing parallel computer that combines two functionally-independent clusters: a traditional Unix-based computing cluster with eight 4200 Sun machines (the *Unix Cluster*) and an embedded set-top box cluster with 32 Cisco 4650 devices (the *Set-Top Cluster*). The two clusters share a common control server: the *MPI Master Host*. This server currently consists of a Sun 4200 processor that is connected to the Unix Cluster and Set-Top Cluster through a pair of Cisco Gigabit Ethernet network switches. The MPI Master Host initiates and typically coordinates all MPI processes across the clusters. Parallel applications based on Open MPI can execute on either cluster independently or on the whole system as a single large heterogeneous cluster.

The backbone network of the Unix Cluster is built using Gigabit Ethernet. Instead, the Set-Top Cluster requires a broadband router for converting between the DOCSIS network [12] and the Gigabit Ethernet backbone. DOCSIS is a standard broadband-network technology for TCP/IP over Radio Frequency (RF) cable that is described further below. Notice that each broadband router can support over 10,000 STBs, thus providing large scale fan-out from the Unix Cluster to the Set-Top Cluster.

All 4200 Sun machines of the Unix Cluster are configured with two 2.8 GHz AMD Opteron dual-core processors, 16GB

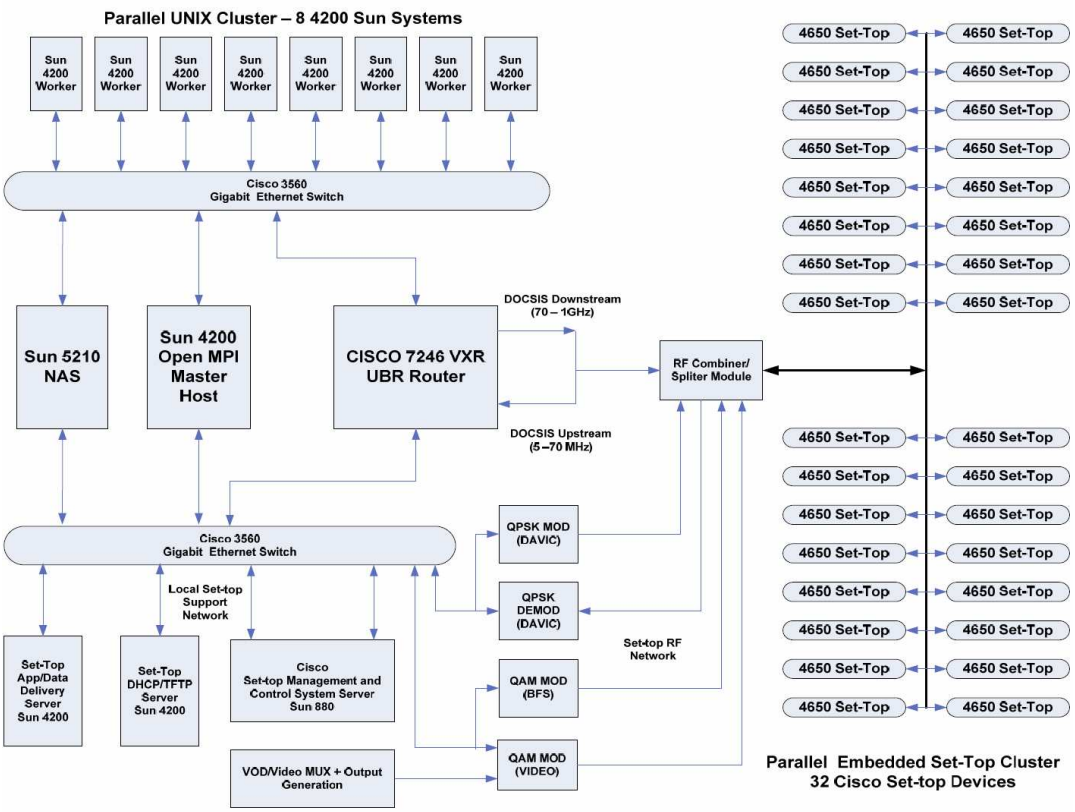


Figure 3: Block diagram of the complete prototype of the proposed heterogeneous parallel computing system.

of system memory and run the Solaris 10 operating system. A Sun 5210 network attached storage array (NAS) provides 750GB of disk space using Sun NFS. The NAS system is also dual-connected using Gigabit Ethernet in the same manner as the MPI Master Host. This allows for a common directory structure and file access model for all processing nodes including the STB devices through a data access proxy system. This is particularly important for the execution of parallel MPI applications because each Open MPI host requires access to a common file system repository.

The Set-Top Cluster consists of 32 Cisco 4650 set-top boxes that are connected using a RF network for data delivery using MPEG and DOCSIS transport mechanisms. The Cisco 4650 is a modern STB that contains a 700MHz MIPS processor, a dedicated video and graphics processor, 128MB of expandable system memory and many network transport interfaces including DOCSIS 2.0, MPEG-2, and DAVIC [13]. Indeed an important architectural feature of modern STBs is the multiple-processor design which allows the MIPS processor, graphics and video processors, as well as network processors to operate in parallel over independent buses. For instance, this makes it possible for user-interface applications to execute efficiently in parallel with any real-time video processing.

The DOCSIS 2.0 TCP/IP and MPEG-2 transport stream interfaces are based on quadrature amplitude modulation (QAM) protocols for transmitting and receiving signals on North American digital cable systems. DAVIC is a legacy 1.54Mbps interface predating DOCSIS and is used only for start-up signaling during STB initialization. The DOCSIS 2.0 standard provides for an inter-operable RF modem, based on TDMA protocols organized in a star topology connecting the central router and the STB DOCSIS interface. Devices on DOCSIS share access to the network, as arbitrated by the central router, and operate effectively at up to 38Mbps in the downstream direction (towards the STB) and 30Mbps in the upstream direction (towards the cluster). The MPEG-2 interface is primarily used for decoding video programs, but can also receive data delivered via the "broadcast file system" (BFS) service on a dedicated QAM frequency. In our prototype system the BFS data delivery mechanism is used to deliver the embedded runtime environment to the STBs. A number of additional devices are required for STB management, application and data delivery, as well as video-content distribution. Indeed, the Set-Top Cluster is part of a scaled-down version of a complete cable system that consists of two additional subsystems: (1) a subsystem that provides STB management and control and (2) a collection of specialized devices for routing, transport, and distribution that support MPEG video and data delivery and transfer of TCP/IP data between the Gigabit Ethernet network and the RF set-top DOCSIS network.

As shown in Fig. 3, the management and control subsystem consists of a Sun 880 server, which is responsible for the initialization, configuration, and delivery of broadcast applications and data using its BFS carousel. Broadcast applications are STB executables that are simultaneously available to all STB devices connected to the dedicated broadband network. A STB device tunes to a specific channel frequency and receives the broadcast application or data of interest using a proprietary Cisco communications protocol. The BFS data is sent from the central server, or head-end,

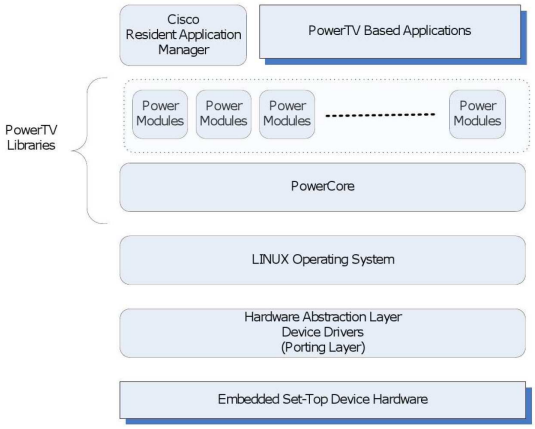


Figure 4: STB embedded software stack.

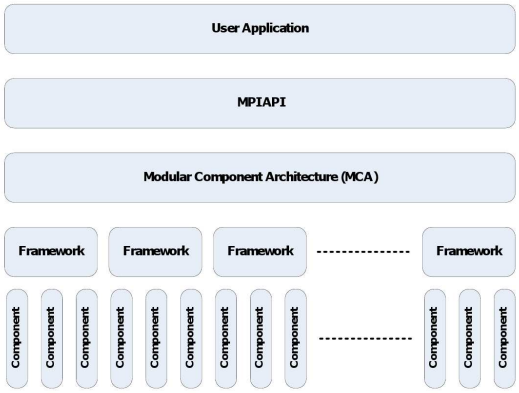


Figure 5: Open MPI component architecture.

at regular cyclical intervals—hence the term carousel—over MPEG-2 directly into a QAM device where it is modulated onto the RF cable plant at a specified frequency for STB reception. For non-broadcast applications, a Sun 4200 is used as an Apache HTTP server that delivers application executables and data in parallel to all requesting STBs through the DOCSIS TCP/IP broadband network. Basic TCP/IP network services are provided by a Sun 4200 running DHCP and TFTP. DHCP is used to assign an Internet address to each STB. TFTP is the primary method for distributing configuration information. A video content channel using a single video source and a MPEG video-output generator is used for video display on all STBs. Interconnect between the Set-Top Cluster and the Unix Cluster as well as RF transport for video and data is supported by a number of specialized devices, including the Cisco 7246 UBR Router. For transmitting and receiving DAVIC, DOCSIS and MPEG-2 data over the RF cable network, a set of QAM/QPSK modulators and demodulators is shown along with a RF combiner/splitter module that connects all devices together.

In summary, our prototype system is representative of a real cable system and allows us to test the execution of high-performance applications on the embedded processors of the set-top boxes under realistic operations scenarios. For instance, we can run target applications such as the MSA program described in Section 5 on the embedded processor, while the rest of the components in the STB, and particularly the MPEG video processing chain, are busy providing streaming-video content. In fact, the experimental results for the MSA application described in Section 6 were obtained while the STBs were simultaneously decoding a test set of MPEG videos for display on a collection of monitors and running an MSO interactive user interface with no perceived degradation of content display.

3. SOFTWARE AND MIDDLEWARE

Set-Top Box Embedded Software. Each Cisco 4650 set-top device runs a hybrid software environment that consists of Cisco's PowerTV middleware layered over a embedded real-time operating system (RTOS) which is based on a Linux kernel (Fig. 4). All applications are written to run on top of the PowerTV application program interface, a proprietary Cisco API, and not on top of the Linux kernel. PowerTV itself consists of PowerCore, and PowerModules, a set of device-independent libraries. The PowerModules libraries provide functionality for communicating on the network, accessing applications or data from the BFS, tuning control such as changing channels, managing MPEG transport streams, encryption services, a widget library called PowerDraw for writing graphically rich applications, and a complete ANSI C library. In our prototype system we used the TCP/IP PowerModule, which is compliant with BSD sockets, to implement the basic MPI send and receive routines as described in the next section.

While the Linux operating system and PowerCore resides in the FLASH memory, all PowerModules are designed to be independent and downloadable over the network at boot time or on-demand through dynamic instantiation much like a dynamic shared library available in Unix. Additionally, there are special user-level processes such as the resident application manager that control all applications life-cycles associated to user-interface applications, communications with the central services, as described in the previous section, and the overall STB user environment.

Open MPI System Software. In the first prototype of our system we use Open MPI version 1.2. Open MPI is an open-source implementation of the Message Passing Interface (MPI) standard for developing parallel applications that execute on distributed memory parallel cluster systems [23]. Open MPI is modular and configurable. It provides an extensible runtime environment called ORTE and an MPI middleware APIs to support robust parallel computation on systems ranging from small mission-critical and embedded systems to future peta-scale supercomputers [8]. Open MPI is based on the *modular component architecture (MCA)*, a lightweight component architecture that allows for on-the-fly loading of frameworks, component modules, and runtime selection of features (including network device, OS, and resource management support), thus enabling the middleware to be highly configurable at runtime. Fig. 5 illustrates the MCA layered component approach [8]. Among all the Open MPI frameworks, the OMPI and ORTE frameworks are of primary interest because they support the point-to-point MPI operations and runtime process execution between the Unix Cluster and the Set-Top Cluster.

The Open MPI framework (OMPI) includes the actual MPI API layer as well as the underlying components necessary to implement the APIs. Consistent with the MCA, the MPI layer is actually built on top of other management and messaging layers. These, for instance, handle point-to-point messaging over TCP/IP, which supports the primary

MPI APIs used in this work. In Open MPI this is implemented using the point-to-point management layer (PML), the byte transfer layer (BTL) and the BTL management layer (BML) [8]. PML handles the upper-level interface to the MPI API layer as well as message fragmentation, assembly, and re-assembly. BML abstracts the network transport layer by managing one or more BTL modules that support actual movement of data over various network interfaces such as TCP/IP over Gigabit Ethernet, high-performance Infiniband, or even a shared-memory multiprocessor systems executing MPI applications.

The Open Run Time Environment (ORTE) framework is primarily responsible for the MPI environment resource discovery and initialization, process execution, I/O, and runtime control. The execution of a parallel application is started by running the `mpirun` command which activates an ORTE daemon process. This process contacts each MPI host in the cluster to initiate a local peer ORTE process. During startup all ORTE processes participate in a resource discovery and eventually begin execution of the parallel application on all hosts that make up the cluster. ORTE continues to coordinate the MPI application processes until the completion of its execution.

4. PORTING OPEN MPI TO STB DEVICES

In order to execute the target parallel application on our prototype system we developed an interoperable subset of the Open MPI software infrastructure to run on the Set-Top Cluster and interact with the full Open MPI 1.2 implementation running on the Unix Cluster. While trends in embedded computing lead to continue improvements in computational capabilities with each generation of STBs, currently these devices are still limited in terms of memory and processor resources. Hence, an early design consideration was to determine the *minimum subset* of the Open MPI API needed for developing a workload that would enable meaningful experimentation and performance evaluation with some real parallel applications. After analyzing the complete MPI specification and selecting the target application we determined that only the nine API functions reported in Table 1 were necessary for our current purposes. While the first six API functions are often sufficient to support many applications, the `MPI_Pack()` and `MPI_Unpack()` functions were added because they are required for the target application, i.e. the MSA program discussed in Section 5.

Leveraging the modularity and configurability of Open MPI we developed a library and runtime layer that is an inter-operable, compatible implementation of a subset of the OMPI and ORTE frameworks to run on the Cisco 4650 STB embedded software stack illustrated in Fig. 4. We included the following inter-operable components of the OMPI point-to-point framework: the MPI layer supporting the nine APIs listed above, a PML component implementation, and a BTL component for TCP/IP messaging over either Gigabit Ethernet or the DOCSIS broadband network. None of the other OMPI component frameworks are currently supported in our prototype implementation. We combined the implementation of the API of Table 1 with the OMPI and the Open MPI ORTE software frameworks into a single PowerTV application library `cv_mpi` that is loaded on the STB during boot time. Parallel applications written for the PowerTV API running on the STB access this library at runtime in a way similar as MPI developers would load the dynamic shared Open MPI library `libmpi.so` on a Unix system.

API Function	Description
`MPI_Init()`	Initialize the MPI execution environment
`MPI_Finalize()`	Terminate MPI execution environment
`MPI_Send()`	Basic blocking point to point send operation
`MPI_Recv()`	Basic blocking point to point receive operation
`MPI_Wtime()`	Return elapsed time on the calling processor
`MPI_Comm_rank()`	Return the rank of the calling process in the communicator
`MPI_Comm_size()`	Return the size of the group associated with a communicator
`MPI_Pack()`	Pack a datatype into contiguous memory
`MPI_Unpack()`	Unpack a datatype from contiguous memory

Table 1: The set of supported MPI API functions.

The set-top ORTE module supports the minimum ORTE protocol transactions to launch, pass command arguments to, and terminate the given MPI parallel application process. For instance, when a parallel MPI application built using the API functions of Table 1 is started on the master Unix Sun 4200 host of our prototype system by running the command line "`mpirun -np 33 <args> <MPI program>`", the ORTE process running on the MPI Master Host contacts the 32 ORTE-compliant processes running on the corresponding 32 STBs. Each of these processes utilizes the DOCSIS TCP/IP network to download the parallel application on demand. Once the initialization of the runtime environment is completed, the MPI application starts its execution in parallel on the Unix Cluster and the Set-Top Cluster.

Limitations of the Current Implementation. In order to execute the MPI applications launched from the Unix Cluster, the STB devices require an inter-operable implementation of the ORTE framework. By analyzing the standard ORTE framework we identified a number of challenges in terms of protocol overhead and fault tolerance. ORTE is a complex framework: a large portion of the implementation is designed for device discovery, exchange of process environment, and network information, occurring between all host devices. This creates scaling issues as the Set-Top Cluster size increases from thousands to millions of computational devices. As an illustration of the ORTE runtime overhead, testing of our experimental implementation revealed that ORTE requires a minimum of 165 bytes as measured by `tcpdump` per ORTE node-map data structure. This list is sent to all hosts in the system during the ORTE initialization phase as part of a group "allgather" operation. This is acceptable for a typical cluster network of a few hundred or thousand nodes but it would not scale to a network of five million or more STBs because over 1GB of information would be sent to each STB, a quantity that exceeds the memory resources available in today's STBs. Hence, for our prototype system, we implemented a light-weight ORTE framework that relies on a statically-configured environment and consists of the minimal protocol that is sufficient to inter-operate with the full Open MPI environment of the Unix Cluster.

A second challenge to system scalability is the reliable execution of MPI applications. In many cases, the possible failure of a single MPI application would result in the termination of all processes within the process group. A large-scale system with millions of devices will likely have frequent failures. Hence, for future development of large-scale systems based on our architecture we plan to incorporate solutions such as those proposed as part of Open MPI-FT [18].

5. MULTIPLE SEQUENCE ALIGNMENT

To evaluate the feasibility of our heterogeneous parallel system as well as its performance in comparison to a more traditional computer cluster we chose *Multiple Sequence Alignment (MSA)*, one of the most important problems in bioinformatics [21]. Since MSA is an NP-hard problem, in practice most of its instances require the use of approximation algorithms such as the popular ClustalW program [27]. There are various parallel versions of ClustalW, including ClustalW-MPI [19] and Mason (multiple alignment of sequences over a network) [2, 4]. We started from the source code of Mason, which is based on the Mpich implementation of MPI, and we ported it on our system, which uses Open MPI, so that we could run it on both the Unix Cluster and the Set-Top Cluster. This allows us to run multiple experiments to compare the parallel execution of different instances of the MSA problem on different system configurations of each cluster as discussed in Section 6. In all cases, the output of the Mason program produces a final multiple sequence alignment along with detailed timing measurements for each stage of the computation. Next, we describe the approximation MSA algorithm used by ClustalW and its Mason parallel implementation.

Solving MSA with ClustalW. MSA is the problem of aligning biological sequences, typically DNA sequences or protein sequences, in an optimal way such that the highest possible number of sequence elements is matched. Given a scoring scheme to evaluate the matching of sequence elements and to penalize the presence of sequence gaps, solving the MSA problem consists in placing gaps in each sequence such that the alignment score is maximized [2]. The ClustalW approximation algorithm consists of three phases. Given an input of N sequences, Phase 1 computes the *distance matrix* by aligning and scoring all possible pairs of sequences. For N input sequences, there are $\frac{N \cdot (N-1)}{2}$ possible optimal pair-wise alignments that can be derived with the dynamic programming algorithm of Needleman and Wunsch [22] as modified by Gotoh [9] to achieve a $O(n^2)$ performance, where n is the length of the longest sequence. The resulting distance matrix is simply a tabulation of score metrics between every pair of optimally-aligned sequences. Phase 2 of the algorithm processes the distance matrix to build a *guide tree*, which expresses the evolutionary relationship between all sequences. The guide tree can be derived with the *neighbor-joining clustering algorithm* by Saitou and Nei [24]. Phase 3 produces the final multiple sequence alignment of all N original sequences by incrementally performing $(N-1)$ pair-wise alignments in the order specified by the guide tree (*progressive alignment algorithm*) [7, 27].

Parallel MPI Implementation of MSA. Mason is a parallel implementation of ClustalW for execution on distributed-memory parallel systems using MPI [2, 4]. It proceeds through nine steps (Fig. 6):

1. The master host processor reads the input file containing N sequences and allocates an $N \times N$ distance matrix structure that is used to hold all optimal alignment scores between any two sequences. The master also partitions the distance matrix by dividing up the sequences among the P worker processors to distribute the workload, network, and resource requirements during the alignment step.
2. The master sends all required sequences to the P worker processors based on the distance matrix partitioning scheme computed in Step 1. Each processor has a fraction N_f of the total number of N sequences to align.
3. The P worker processors compute their $\frac{N_f \cdot (N_f - 1)}{2}$ alignments in parallel using the pairwise alignment dynamic programming algorithm.
4. All worker processors send their resulting alignment scores back to the master in parallel.
5. After receiving all scores and completing the distance matrix the master builds the guide tree.
6. The master computes and sends an alignment order along with the respective sequences required for progressive alignment to all worker processors.
7. All worker processors perform progressive alignment in parallel.
8. All worker processors send their partial multiple alignments back to the master in parallel.
9. The master progressively aligns the remaining multiple alignments to produce the final result.

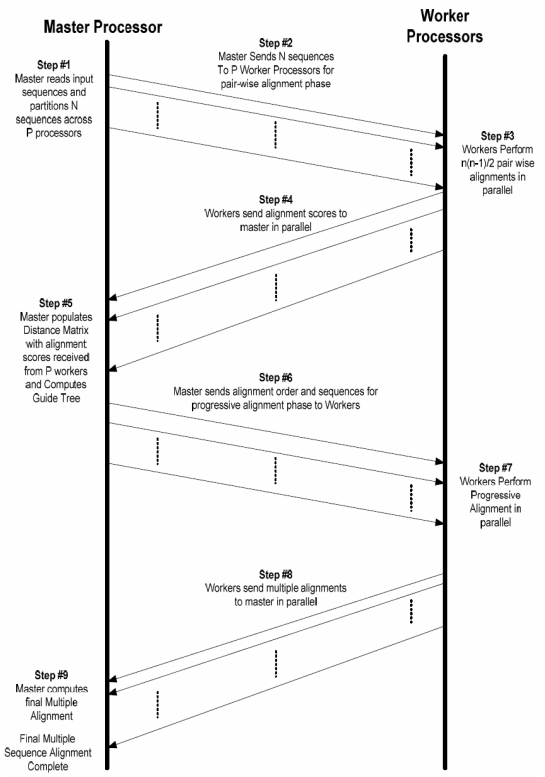


Figure 6: Parallel execution of ClustalW algorithm.

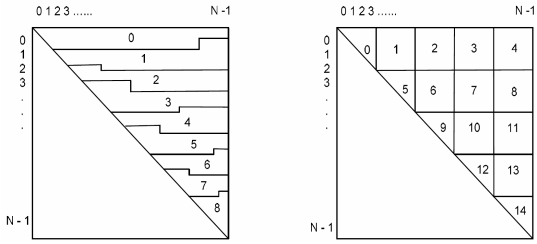
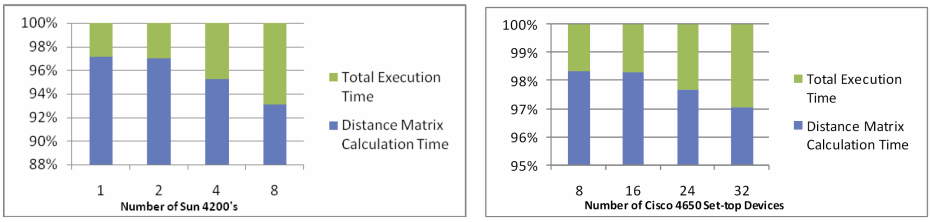


Figure 7: Alternative partitions of distance matrix.

Experimental results by Datta and Ebedes show that 96% of computational time is spent in the derivation of the distance matrix, i.e. in the first three steps of Fig. 6, while the remaining time is split between the other two phases [2].



(a) Unix Cluster configurations (b) Set-Top Cluster configurations

Figure 8: Execution time breakdown for each of the four configurations of the two clusters.

Processor Type	#	Overall Execution Time (Sec) - Floating Point S500-L1100	S100-L1500	S500-L200	S200-L300	Avg.	Overall Execution Time (Sec) - Fixed Point S500-L1100	S100-L1500	S500-L200	S200-L300	Avg.
Sun 4200	1	5129	830	426	162		4666	757	383	146	
	8	1580	245	157	55		1427	228	146	51	
	Speedup	**3.2**	**3.4**	**2.7**	**2.9**	**3.1**	**3.3**	**3.3**	**2.6**	**2.9**	**3.0**
Cisco 4650	8	51989	8185	4256	1769		29732	4924	2575	1090	
	32	16617	2651	1485	714		9507	1629	908	441	
	Speedup	**3.1**	**3.1**	**2.9**	**2.5**	**2.9**	**3.1**	**3.0**	**2.8**	**2.5**	**2.9**

Table 2: Overall execution times and speedups for two different system configurations of each cluster.

While the distance-matrix computation is the main target for parallelization, maximizing the achievable speedup depends on the strategy that is used to partition the matrix among the P worker processors. A possible approach consists in sending all N sequences to all processors so that each processor computes exactly $\frac{N\cdot(N-1)}{2P}$ pairwise alignments. This approach, which is illustrated in Fig. 7(left), distributes evenly the workload among the processors, but has the highest message-passing cost since all processors receive all N sequences, whether they are used or not. Alternative partitioning strategies have been proposed to reduce the communication cost [2]. The strategy shown in Fig. 7(right) assigns a square section of the distance matrix to most of processors, while some processors receive one of the smaller sections along the diagonal. This method reduces the amount of message passing in exchange for an unequal workload distribution. For small input sizes the first approach outperforms the second because the communication costs are still relatively low and all processors are fully utilized [2]. However, for aligning large sets of long DNA or protein sequences, which consist of perhaps thousands or even tens of thousands of sequences, the second approach may be more convenient because the resulting distance matrix can be partitioned such that communication costs and processor memory resources are minimized.

6. EXPERIMENTAL RESULTS

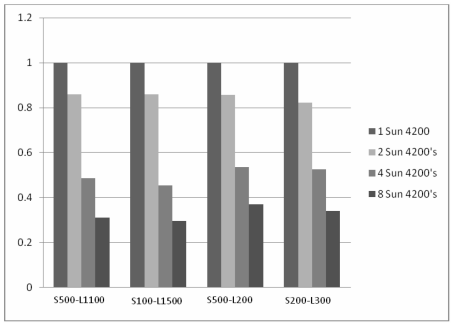
Using the MSA problem as our target application, we completed a set of experiments on the prototype system of Fig. 3. While our system allows us to analyze various combinations of devices, in these experiments we focused on comparing different configurations of the Unix Cluster, which consists only of Sun 4200 nodes, with different configurations of the Set-Top Cluster, which consists only of Cisco 4650 STBs. Specifically, we considered four Unix Cluster configurations with $1, 2, 4$ and 8 Sun 4200 acting as worker processors and four Set-Top Cluster configurations with $8, 16, 24$ and 32 STBs acting as worker processors. Every configuration uses the same Sun 4200 processor as MPI master host. In each experiment we run MASON on a particular input data set that consists of a given number of DNA sequences.

Generation of Data Sets. Given an input file that specifies a set of transition probabilities, the ROSE sequence generator tool returns sets of either protein or DNA sequences that follow an evolutionary model and, therefore, are more realistic than purely random-generated sequences [25]. Using ROSE's standard input settings we generated four sets of DNA sequences, which present different size and complexity:

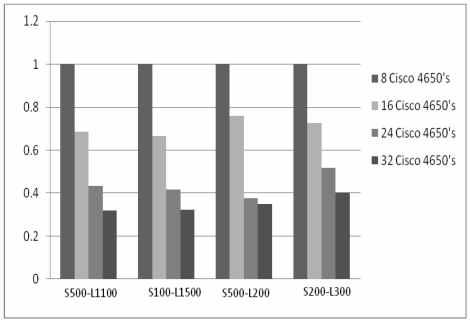
- *S500-L1100*: 500 DNA Sequences with 1100 base pairs.
- *S100-L1500*: 100 DNA Sequences with 1500 base pairs.
- *S500-L200*: 500 DNA Sequences with 200 base pairs.
- *S200-L300*: 200 DNA Sequences with 300 base pairs.

In general, larger sets of longer sequences are computationally more complex than smaller sets of shorter sequences. The algorithm partitions N sequences into $\frac{N_{part}\cdot(N_{part}-1)}{2}$ tasks, each requiring the pairwise alignment of M_{part} sequences. Hence, an upper bound on the computational complexity of MSA is given by $O((N_{part})^2\cdot(M_{part\_max})^2)$, where M_{part_max} denotes the longest sequence in the set of M_{part} sequences making up any given sequence partition N_{part}. In our collection the *S500-L1100* set is the most computational complex while the *S200-L300* set is the least complex. The other cases lie somewhere in between.

Floating-Point vs. Fixed-Point. As part of our experiments, we analyzed also the performance impact of converting floating-point operations in MASON to fixed-point operations for each possible configuration of both clusters. This analysis is important because current STBs do not feature a floating-point hardware unit and only support floating point operations through emulation in software. Hence, to estimate how much of the current performance gap is due to the lack of this unit allows us to better extrapolate the performance that could be obtained when running other algorithms, which necessarily require floating-point precision, on future clusters of next-generation of embedded devices, which are expected to contain floating-point units. We focused our effort on converting to fixed-point operations only the code in the first phase of the algorithm (distance-matrix computation) because it accounts for over 90% of the overall computation time. The conversion was achieved by replacing float variables with long integers and multiplying by a constant factor sufficient to maintain five digits of precision in all scoring routines comprising the pairwise-alignment algorithm. Results were converted back to floating point from fixed-point values by dividing all fixed-point score values by a constant factor prior to populating the distance matrix. In terms of accuracy, we found negligible difference in results beyond five digits of precision.

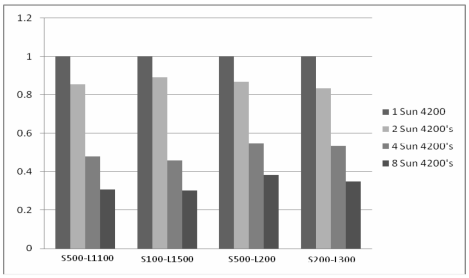


(a) Unix Cluster (float)

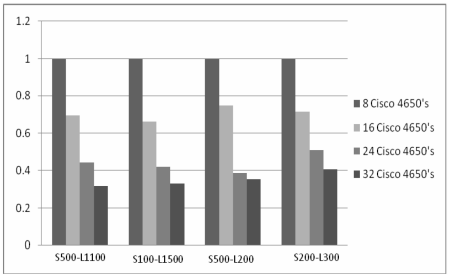


(b) Set-Top Cluster (float)

Figure 9: Relative parallelization speedup: Unix Cluster vs. Set-Top Cluster (floating-point computation).



(a) Unix Cluster (fixed)



(b) Set-Top Cluster (fixed)

Figure 10: Relative parallelization speedup: Unix Cluster vs. Set-Top Cluster (fixed-point computation).

Execution Time Breakdown. Figures 8(a) and 8(b) report the execution time breakdown of the floating-point version of Mason with the S500-L1100 input data set for each of the four configurations of the Unix Cluster and Set-Top Cluster, respectively. The results for the fixed-point version as well as for the other data sets are similar. In both cases, as expected, the distance-matrix computation accounts for well over 90% of the overall execution time regardless of the number of processors. This number, we recall, varies from 1 to 8 for the Unix Cluster and from 8 to 32 for the Set-Top Cluster. These results are in line with similar results presented in the literature for other parallel implementations of ClustalW [4, 2, 19] and confirm the validity of the parallelization efforts discussed in Section 5.

Parallelization Speedup. Table 2 reports the overall execution times in seconds obtained running both the floating-point version and the fixed-point version of Mason with each input data set on two configurations of the Unix Cluster (consisting of 1 and 8 Sun 4200 processors respectively) and two configurations of the Set-Top Cluster (consisting of 8 and 32 Cisco 4650 STBs respectively). For instance, for the case of the floating-point version of Mason, the data set for the hardest problem (S500-L1100) is processed in $5129s$ by a single Sun 4200 while a cluster with eight Sun 4200 processors takes only $1580s$ (a speedup of 3.2). Meanwhile, for this problem, a cluster of eight Cisco 4650 devices need $51989s$ but this time is reduced to $16617s$ for a cluster of 32 devices (a speedup of 3.1). The speedup values are similar across the four data sets and regardless of the program version (floating-point or fixed-point) with an approximate average value of $3X$ for the Unix Cluster and $2.9X$ for the Set-Top Cluster. Figures 9 and 10 illustrate the relative speedups due to parallelizations normalized to the slowest computation time for each data set as we increase the number of nodes for each of the four configurations of the Unix Cluster and Set-Top Cluster for both types of computation, respectively. Notice that while the execution time does not decrease linearly as we increase the number of Sun processors (or Cisco devices), an important result is that *across all the various data sets both the Unix Cluster and Set-Top Cluster exhibit similar scaling and performance improvements, with comparable speedup due to parallelization in both platforms.*

Data Set	32 STB vs. 1 Sun 4200		
	(float)	(fixed)	Gain (%)
S500-L1100	3.24	2.04	37
S100-L1500	3.19	2.15	33
S500-L200	3.49	2.37	32
S200-L300	4.41	3.02	31

Table 3: Comparing a Set-Top Cluster with 32 boxes to 1 Sun 4200: gain due to fixed-point computation.

How Many Set-Top Boxes to Make a High-Performance Processor? The first row of Table 3 reports the ratio of the execution time of the fastest Set-Top Cluster configuration (32 Cisco 4650) over the execution time of the slowest Unix Cluster configuration (1 Sun 4200) for the various data sets. For instance, this ratio is $16617/5129 = 3.24$ when the two clusters run the floating-point version of Mason with the S500-L1100 data set. In other words, 32 STB devices take 3.24 as much time as a single Sun processor to perform the same task. Or, again, we could say that one high-performance processor is equivalent to 104 STB devices. On the other hand, this ratio drops to 2.04 when the fixed-point version of Mason is used *for both clusters* [1],

[1] We recall that using the fixed-point computation for the MSA problem is faster for both clusters, but is relatively better for the Set-Top Cluster because the STBs do not have a floating-point hardware unit. Naturally, there are other important scientific problems that necessarily require floating-point operations. Our approach to parallel computing will be applicable to these problems only when STBs will feature a floating-point unit, which is expected to happen soon [5].

an improvement of 37% as shown in the remaining rows of Table 3. This translates in a new *equivalence gap factor* of 66 STBs per high-performance processor. Finally, notice that the frequency of the Sun 4200 processor is four times as fast as the frequency of the embedded processor in the Cisco 4650 STB. Hence, accounting for this frequency ratio, the equivalence gap factor would become 16.5.

7. LESSON LEARNT AND NEXT STEPS

Impact of Communication. In comparing the relative platform speedups we observe that the communication cost or overhead due to MPI is not a factor on either the Unix Cluster and Set-Top Cluster for our experimental system. This is due to the small size of the experimental broadband network and limited number of embedded devices. In a larger broadband network with many STBs, a key consideration will be assuring ample network bandwidth and minimal latency to each embedded device. MSOs are currently supporting millions of embedded devices over DOCSIS networks and plan to continue improving overall network performance through ongoing protocol enhancements that will include the following: 1) quality-of-service (QoS) enforcement on a per-application basis assuring minimum bandwidth allocations; and 2) increased network performance in both directions to achieve over 300Mb/s downstream and 120Mb/s upstream through channel bonding. Channel bonding is part of the DOCSIS 3.0 standard which increases network performance by concatenating multiple channels into one larger virtual channel [10]. Additionally, MSOs are improving the DOCSIS broadband network by continuing to reduce the number of embedded devices per DOCSIS channel. With fewer devices per channel, additional access slots within the TDMA scheme are available, thus increasing the throughput for all devices on that channel.

Trends in STB Hardware and Software. An interesting extrapolation is to project the future potential of our proposed heterogeneous system when using next-generation embedded STBs by comparing the currently available Cisco 4650 devices with the previous generation Cisco 4200 devices [13]. As illustrated in Fig. 11, during the last two years the clock frequency of the embedded processor has almost tripled while the DRAM capacity has grown by a factor of at least four. Moreover, these improvements in computational power happened without any cost increase. STBs in the near future are expected to host even faster, multicore processors in the range of 1.5 to 2 GHz with FPU support, larger DRAM memory, and the addition of 3-D graphics units. For instance, this trend is confirmed by the recent announcement that STMicroelectronics and ARM have teamed up to deliver high performance ARM Cortex-A9 MPCore processors to the STB industry [5]. The ARM Cortex-A9 features from 1 to 4 cores, L1 and L2 caches, with each core also containing either a media processing unit or FPU with support for both single and double precision floating-point operations [26]. Further, these systems will run non-proprietary operating systems such as Linux and support Java as a ubiquitous programming environment. Assuming similar technology trends in the embedded space and considering the power constraints limiting the frequency of today's highest performance processors, it is reasonable to expect that *in the near future the next generations of STB devices, along with other embedded systems, will further narrow the performance gap with traditional cluster-blade technology.* Consequently, we believe that, as this gap continues to close, heterogeneous systems that take advantage of high-speed broadband networks and geographically-distributed embedded processors for distributed computation will become an interesting proposition for MSOs in the realization of cost-effective, energy-efficient, high-performance computing infrastructures.

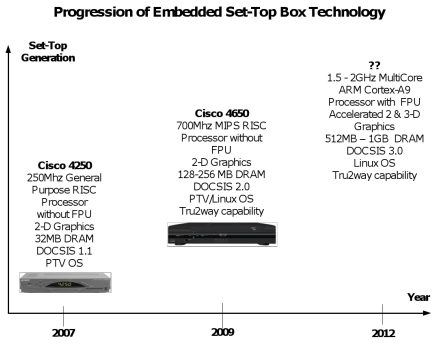


Figure 11: Progressive evolution of STB technology.

8. RELATED WORK

While the utilization of large scale networks of embedded devices for heterogeneous computing within a managed, dedicated system cloud raises new challenges and opportunities in system scalability and performance, the idea of harnessing distributed embedded systems, particularly over the Internet, is not new. A number of initiatives have focused on utilizing volunteer PC hosts or game consoles for solving difficult problems such as those found in Computational Biology. For instance, Folding@Home [11] and GridRepublic [14] were formed to leverage the enormous number of idle processor cycles available on the Internet. In these projects, specialized software is downloaded to participating PCs or game consoles, such as the PlayStation 3 featuring the IBM Cell multi-core processor, with units of computation dynamically offered by subscribers typically through a screensaver application or background agent running on the host device. In this model when the agent is available it receives tasks assigned by a master server to be processed on a best-effort basis. Results are sent back to the master server when the task is completed. This system offers scalability on the order of the number of PCs or game consoles active on the Internet at any give time. Its success shows the potential of harnessing distributed embedded devices that are widely deployed by the tens of millions units today. Still the fact that the "device participation" is not predictable ultimately limits throughput guarantees, maximum task concurrency, and service-level agreement between actors.

Most related works in the area of integrating message-passing middleware into embedded devices have focused on reducing the size of the MPI stack. Lightweight MPI (LMPI) is based on a thin-client model where the MPI API layer is implemented on top of a thin-client-message protocol which communicates with a proxy server that supports a full MPI stack [1]: client requests are routed through the proxy server, which acts in behalf of one or more MPI thin-client user processes. A key benefit of the LMPI approach is the elimination of the need for an operating system on the embedded device. Other approaches attempt to reduce the size of the MPI stack through refactoring or are based on a bottom-up

implementation of a minimal MPI stack, as in the case of Embedded MPI (eMPI) [20]. Similarly to eMPI we use a bottom-up approach to implement the minimal MPI API set. But in our proposed system each Cisco STB contains a modern real-time operating system that can support a native MPI implementation. Also, while previous work in executing MPI on embedded devices has focused on small test kernels, from an application-viewpoint our work is closer to the work of Datta [4, 2] and Li [19] because we evaluate our parallel system with a real workload.

Google designed and deployed a massively-parallel system comprised of commodity dual-core PCs running Linux combined with the Google-custom Map-Reduce framework for parallel computing [3]. The Google platform is distributed across many data-centers and was estimated in size at over 450,000 systems [6]. A possible future large-scale version of our proposed architecture, which will require to address the scaling issues discussed in Section 4, would have important differences with the Google platform, including the use of a broadband network of embedded devices instead of a network of clusters of PCs and the MPI programming model instead of the Map-Reduce application development platform. On the other hand, an open research area is to explore the combination of the Map-Reduce paradigm with a heterogeneous parallel computing system featuring a large number of distributed embedded devices, which may offer interesting opportunities for such applications as data mining.

9. CONCLUSIONS

We proposed a heterogeneous platform architecture for distributed computing that leverages traditional Unix cluster technologies in combination with a broadband network of embedded set-top boxes (STB). We implemented a complete prototype system that fully represents a scaled-down version of the proposed architecture and developed an interoperable subset of Open MPI to integrate and run it on the heterogeneous devices. We ported a parallel version of the ClustalW bioinformatics application on our system by completing the necessary optimizations to reduce the memory requirements for execution on the STBs and improve parallel workload data distribution. We established that it is possible to execute ClustalW efficiently on our prototype system while the STBs continue simultaneously to operate their primary functions, i.e. decoding MPEG streams for monitor display and running an interactive user interface, without any perceived degradation. Further, our experimental results show that scaling up the system by adding more STBs to the embedded cluster gives equivalent performance gains as scaling up the number of processors in a traditional Unix cluster. While the proposed platform architecture has the potential of scaling to millions of units, we identified critical challenges in the area of protocol overhead and fault tolerance when implementing the MPI ORTE for embedded devices. To address these challenges is our next goal. Indeed we think that the present work opens new important avenues of research in the area of distributed computing systems. Given the technology trends in MSO broadband networks and in hardware/software solutions for STBs, our study demonstrates that to leverage a broadband network of embedded devices is an interesting proposition to build a low-cost and energy-efficient computing platform that can support both computationally-intensive service-provider workloads and emerging consumer-driven applications.

Acknowledgments

This work was partially supported by Cablevision Systems. The authors gratefully acknowledge the help of Serguei Tcherepanov for the porting of Mason and Satish Marada for the building of the system prototype.

10. REFERENCES

[1] A. Agbaria et al. LMPI: MPI for heterogeneous embedded distributed systems. In *12th Intl. Conf. on Parallel and Distributed Systems*, pages 79–86, July 2006.

[2] A. Datta and J. Ebedes. Multiple sequence alignment in parallel on a workstation cluster. In A. Zomaya, editor, *Parallel Computing for Bioinformatics and Computational Biology*, pages 193–210. J. Wiley & Sons, Oct. 2006.

[3] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, Jan. 2008.

[4] J. Ebedes and A. Datta. Multiple sequence alignment in parallel on a workstation cluster. *Bioinformatics*, 20(7):1193–1195, May 2004.

[5] *STMicroelectronics and ARM Team Up to Power Next-Generation Home Entertainment*. Press release: `www.arm.com/news/26284.html`, Oct. 2009.

[6] C. Evans. *Future of Google Earth*. Booksurge Llc., 2008.

[7] D. Feng and R. Doolittle. Progressive sequence alignment as a prerequisite to correct phylogentic trees. *J. Mol. Evol.*, 25(4):351–360, Aug. 1987.

[8] E. Gabriel et al. Open MPI: goals, concept, and design of a next generation MPI implementation. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface, 11th European PVM/MPI Users Group Meeting*, pages 97–104, Sept. 2004.

[9] O. Gotoh. An improved algorithm for matching biological sequences. *J. Mol. Biol.*, 162(3):705–708, Dec. 1982.

[10] `http://en.wikipedia.org/wiki/DOCSIS`.

[11] `http://folding.stanford.edu`.

[12] `http://www.cablemodem.com`.

[13] `http://www.cisco.com`.

[14] `http://www.gridrepublic.org`.

[15] `http://www.isuppli.com`.

[16] `http://www.morganstanley.com`.

[17] `http://www.multichannel.com`.

[18] J.Hursey et al. The design and implementation of checkpoint/restart process tolerance for Open MPI. In *Intl. Symp. on Parallel and Distributed Processing*, pages 415–422, 2007.

[19] K. Li. ClustalW-MPI: ClustalW analysis using distributed and parallel computing. *Bioinformatics*, 19(12):1585–1586, Aug. 2003.

[20] T. McMahon and A. Skjellum. eMPI: Embedded MPI. In *MPI Developers Conference*, pages 180–184, July 1996.

[21] D. W. Mount. *Bioinformatics: Sequence and Genome Analysis*. CSHL Press, New York, 2001.

[22] S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequences of two proteins. *J. Mol. Biol.*, 48(3):443–453, June 1970.

[23] Open MPI. Available at `http://www.open-mpi.org`.

[24] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogentic trees. *Mol. Biol. Evol.*, 4(4):406–425, July 1987.

[25] J. Stoye, D. Evers, and F. Meyer. Rose: generating sequence families. *Bioinformatics*, 14(2):157–163, Mar. 1998.

[26] The ARM Cortex-A9 Processors. White paper available at: `www.arm.com/pdfs/ARMCortexA-9Processors.pdf`.

[27] J. Thompson et al. ClustalW: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res.*, 22(22):4673–4680, Nov. 1994.