Abstract—NASA's Mission to Planet Earth (MTPE) is planning to launch the Earth Observing System (EOS) starting in 1998. The large number of planned remote sensing satellites will bring 500 Gigabytes of information per day. The EOS Data and Information System (EOSDIS) is responsible for ingesting and archiving this data. One important component of the EOSDIS system is the data operation, which involves extracting the packets and reconstructing and archiving the original remotely sensed data products. Due to transmission errors, the way data is sampled from the different sensors encoded, packets typically arrive out of order and perhaps with some of them missing or repeated. Many special hardware solutions have been proposed to solve this real-time problem. In this paper, we demonstrate a commercial off the shelf (COTS) solution. The hardware capitalizes on the progress made in the area of network of workstations (NOW), particularly PC-clusters. The software and algorithm exploit the data characteristics and parallelism in the telemetry stream to make use of load balancing and efficient parallel processing. It will be shown that this solution can provide high-performance to cost and programmability.

1. INTRODUCTION

Processing intensive continuous flows of data is one of the challenging problems in many engineering and scientific applications. A prominent example of this problem is represented by NASA's Earth Observing System (EOS) where a number of spacecrafts are expected to send more than one trillion bytes of data per day for 15 years to Earth [1, 2].

The NASA Mission to Planet Earth (MTPE) program plans to launch the Earth Observing System (EOS) starting with the AM-1 satellite in 1998. A large number of planned remote sensing satellites and onboard instruments will send...
a data stream of 500 Gigabytes of information per day. The EOS Data and Information System (EOSDIS) is responsible for ingesting and archiving this data, which is received encoded using the Consultative Committee on Space Data Systems (CCSDS) standard. One important component of the EOSDIS system is the data operation, which involves extracting the CCSDS packets and reconstructing and storing the original remotely sensed data as the so-called level-0 data products. Due to transmission errors, the way data is sampled from the different sensors onboard of the different satellites, and the way packets of data are integrated into Coded Virtual Channel Data Units (CVCDU) packets typically arrive out of order and perhaps with some of them missing or repeated.

The end-to-end transmission and processing of data in a space data system is referred to as telemetry processing. Many special hardware solutions have been proposed to solve this real-time problem. This paper reports on a PC-parallel approach using Beowulf architecture [4] to processing such real-time high-rate and high-volume spacecraft data. The paper describes the parallel software architecture that has been implemented using MPI [8] as the message-passing library, and presents experimental results obtained with a real-life test data set from the X-Ray Timing Explorer (XTE). The XTE data is briefly described in section 5. It is concluded that the architecture can meet the processing requirement of EOS.

Figure 1 shows an overview of PACET and its processes. The CVCDU data stream is fetched at the front-end processor. The rest of the processing is executed in parallel within a cluster of workstations. Our commercial off the shelf (COTS) solution exploits the inherent parallelism in the computational processes in the data operation. The hardware capitalizes on the progress made in the area of network of workstations (NOW), particularly PC-clusters using the cost-efficient PC-marketplace. The software and algorithm exploit the data characteristics in the telemetry stream and take advantage of load balancing and efficient parallel algorithms. It will be shown that this COTS solution can provide the desired high-performance for EOSDIS at a low cost while maintaining programmability.

This paper is organized as follows. Section 2 describes the background of the problem. Section 3 portrays our PACET architecture. In section 4 we detail the parallel algorithm for telemetry processing. Experimental results are given in section 5. Conclusions and future directions are in section 6.

2. BACKGROUND

In NASA's data operation, data are sampled from the different sensors onboard of the different satellites, and packets of data are integrated into the CCSDS data format called Coded Virtual Channel Data Units (CVCDU). Packets typically arrive out of order and perhaps with some of them missing or repeated. This section describes the CCSDS data format, which we used for an input test data in our experiment.

CCSDS Data Format

The techniques necessary to implement the physical and logical components of space data systems are being developed by the Consultative Committee for Space Data Systems (CCSDS) [1], a worldwide cooperative effort of national space agencies. The goal of the committee is to standardize space data communication techniques so that several agencies may cross-support each other's data flow and thus allow complex international missions to be flown. Variable length pieces of telemetry data are encapsulated within standard headers to form delimited data structures
called CCSDS Packets. The CCSDS data format is completely described in [2].

Different types of users may share a single physical communication by creating multiple apparently parallel “virtual” channels. At the transmitter the sequence of packets is structured into fixed length blocks of packets known as Virtual Channel Data Units (VCDU), Figure 2. The main purpose of this process is to efficiently utilize the capacity of the virtual channel by allowing multiplexing of packets from multiple sources, thereby forming multiple packet channels within a single physical channel. Demultiplexing at the receiver end restores the individual packets. Up to three optional error control fields may be included in the VCDU, thus forming a Coded VCDU (CVCDU). In the most common format of CVCDU, a block of Reed-Solomon Check Symbols is appended to the end of VCDU. Reed-Solomon coding is a high-performance block-oriented technique, which provides powerful error correction capability. Figure 2 shows the components of and relationship between CCSDS packets and CVCDU’s. Each packet contains a Primary Header and an optional, undefined Secondary Header. The latter, if present, carries ancillary data, which may be required for the interpretation of the user data carried in the User Data field. Secondary Header Flag specifies whether a Secondary Header, which includes supplementary information such as a Time Code, is present. An Application Process Identifier (APID) provides a name for a particular user application. For packet sequence control, fourteen bits provide a Packet Sequence Count and the two Sequence Flag bits identify the packet as the first, last, intermediate, or only packet constructed from a file of user data.

Packets from multiple users are combined into a common data structure, called a Multiplexing Protocol Data Unit (M_PDU), for transmission over a single virtual channel. Normally the M_PDU is of a fixed size for a given virtual channel, which may necessitate the splitting of a packet that does not fit completely within the M_PDU. In such an event, the remaining portion is assigned to the next M_PDU. It may also be necessary to enter fill data in the absence of a sufficient number of packets. To facilitate the demultiplexing of the M_PDU, its header contains a field pointing to the location of the start of the first complete CCSDS Packet contained in the M_PDU.

In VCDU Primary Header, Spacecraft ID (SCID) denotes the logical entity in a spacecraft involved in the data exchange and Virtual Channel ID (VCID) defines the channel over which the data associated with a particular SCID may be exchanged. The VCDU Counter represents a sequential count of the total number of VCDU’s that have been transmitted over the virtual channel identified by the VCID. The Replay Flag discriminates between a VCDU that is being transmitted in real-time or one that has been stored for a time and retrieved for transmission. Telecommunication between a ground terminal and an EOS spacecraft takes place in sessions of nominally 30 minutes duration. There are three modes of operation required in the EOS specifications: real-time processing, quick-look data processing, and production data processing. Real-time processing processes source packets received from selected instruments (e.g. health and safety data) in spacecraft and delivers for use primarily by control centers with minimal delay. Production data processing is the most involving operation and includes the following services:

- Sort packets by SCID and APID and forward time order packets
- Identify missing and redundant packets between multiple sessions, and produce quality and accounting information
- Create a production data set containing merged data from multiple sessions.

Figure 3 Software Architecture for the Parallel Telemetry System

Figure 4 Packet Index Structure.
A quick-look data set, on the other hand, is generated from a single SCID/APID for a single session. Up to 5% of the data received over a 24 hour period may be subject to the above services, except data merging and redundant packet deletion. All these packets are also retained for production data processing [2].

**Parallelism in Multiple CVCDU’s**

CCSDS standard creates data packet streams that are inherently sequential. This is because each packet points at the following packet within the same CVCDU. In sequential processing, packet extraction takes the majority of CPU processing time. However, (by distributing CVCDU’s across the processors) packets can be extracted in parallel by taking into account packets split across CVCDU on different processing nodes. Figure 3 shows our parallel telemetry processing steps. The parallel algorithm maximizes the degree of parallelism in packet extraction, packet index sorting, checking for missing and redundant packets, and storing production data set and index by distributing multiple CVCDU to each processor and balancing the number of packets per processor after packets are extracted. The production data set is saved to the disk and can be located by packet indices stored separately in a packet index file. Figure 4 shows packet index structure. The message passing parallel telemetry processing is described in details in section 4.

3. PACET ARCHITECTURE

**The Hardware Architecture**

Our hardware architecture is based on the concept used in the Beowulf project [4,5,6] that attempts to exploit parallelism using a high performance PC network of workstations (NOW). The PACET system consists of a Sun Sparc-20 workstation front-end and a set of computing nodes connected through a high-speed network of 100Mbps bandwidth. Our current cluster consists of five 150 MHz Pentium workstations, Figure 6. Each workstation, called a computing node, has its own local memory of 16 MB and a fast 1.6 GB disk drive as well as a 100 Mbps high speed PCI network interface card. We plan to add four more nodes to complete the system with nine nodes. The interconnection network will also be upgraded to 1 Gbps. This will help cut the communication time by a factor of ten, theoretically. As to be able to tolerate the resulting higher I/O rate, the disk subsystem will be upgraded to SCSI system and data will be striped across all disks to overlap the I/O overhead as much as possible. The hardware architecture works under the assumption that the CVCDU data stream is already on the disk of the master node. The pieces of production data set are saved in local disks while the master has the global index table.

**The Software Architecture and Operation**

The Beowulf-based PACET software includes 1) Linux operating system; 2) MPI [8] and PVM [9] message passing library; and 3) GNU gcc compiler. All the software used in PACET is freely available in many anonymous ftp sites. Linux is a Unix operating system developed originally for Intel microprocessor-based PC systems. It is a fully functional and stable Unix operating system. The communication libraries used are MPI and PVM. MPI is a portable de-facto standard message-passing interface available on most parallel machines as well as NOWs. PVM is developed at Oak Ridge National Laboratory (ORNL). The parallel telemetry processing in this work is implemented using MPI. The cross-platform compiler GNU “gcc” is also freely distributed under GNU conditions. Based on this software base we have developed the message passing parallel telemetry processing on our PACET system. The parallel telemetry processing is described in the next section.

<table>
<thead>
<tr>
<th>Processing Fractions</th>
<th>100Mbps</th>
<th>1Gbps</th>
<th>100Mbps</th>
<th>1Gbps+MD switch</th>
<th>1Gbps+MD switch+9procs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Scattering CVCDU</td>
<td>79.39</td>
<td>10.09</td>
<td>77.60</td>
<td>2.52</td>
<td>2.52</td>
</tr>
<tr>
<td>2. Extracting the packet from CVCDU</td>
<td>5.36</td>
<td>5.36</td>
<td>5.36</td>
<td>5.36</td>
<td>4.17</td>
</tr>
<tr>
<td>3. Sequential sorting at each node</td>
<td>3.18</td>
<td>3.18</td>
<td>2.70</td>
<td>2.70</td>
<td>2.10</td>
</tr>
<tr>
<td>4. find the global pivot list</td>
<td>4.27</td>
<td>2.47</td>
<td>2.57</td>
<td>0.62</td>
<td>0.62</td>
</tr>
<tr>
<td>5. Exchanging the packet among processors</td>
<td>10.71</td>
<td>1.23</td>
<td>10.58</td>
<td>0.31</td>
<td>0.31</td>
</tr>
<tr>
<td>6. writing index to disks</td>
<td>0.34</td>
<td>0.34</td>
<td>0.09</td>
<td>0.09</td>
<td>0.09</td>
</tr>
<tr>
<td>7. Gathering the packet to the host</td>
<td>15.28</td>
<td>7.18</td>
<td>10.57</td>
<td>1.80</td>
<td>1.80</td>
</tr>
<tr>
<td><strong>Total time (seconds)</strong></td>
<td><strong>103.25</strong></td>
<td><strong>22.66</strong></td>
<td><strong>98.90</strong></td>
<td><strong>11.60</strong></td>
<td><strong>9.80</strong></td>
</tr>
<tr>
<td><strong>Speed (packets/second)</strong></td>
<td><strong>7854.86</strong></td>
<td><strong>35787.47</strong></td>
<td><strong>8200.26</strong></td>
<td><strong>69931.38</strong></td>
<td><strong>82717.12</strong></td>
</tr>
<tr>
<td><strong>Improvement</strong></td>
<td><strong>456%</strong></td>
<td><strong>104%</strong></td>
<td><strong>890%</strong></td>
<td><strong>1053%</strong></td>
<td></td>
</tr>
</tbody>
</table>

**Note**: Processing fraction 1, 3, 4, 5, and 7 compose of all I/O, computation and communication. Fraction 2 has only computation. And, Fraction 6 has only I/O processing.
4. THE PARALLEL TELEMETRY PROCESSING

Parallel Telemetry Processing on PACET

For the purpose of data set production, packets have to be sorted with respect to their SCID, APID, and Sequence Number fields. Therefore, the first step must be to extract packets from the incoming data stream, which can be sorted using a high-performance system. Packets within the same CVCDU cannot be extracted in parallel because of the sequential nature of the packets residing in that CVCDU. On the other hand, if the packet extraction is done sequentially it would present a bottleneck for the data set production. Parallel packet extraction, however, can be exploited at CVCDU level under the provision that split packets over CVCDUs on different processors are merged together. The key idea is to distribute the chunks of CVCDUs to all available processors and extract the packets in parallel. Load balancing problem arises when all processors attempt to sort packets after extracting them from the CVCDU stream. This is because each CVCDU may contain different number of packets. The number of packets in each processor ranges from 80,000 to 120,000 packets and determines the intensity of the processing needed.

The parallel algorithm consists of five main parts: 1) scattering the CVCDU data from the master node to the slaves; 2) extracting the packet data from each CVCDU data and combining the split packets; 3) load balancing; 4) sorting the packet and checking for the redundant and missing packets; and 5) gathering the packet data. The first step, scattering, distributes data from the master to other slave nodes. Then, on every node, sequential algorithm for extracting the packet is applied on CVCDUs that reside locally. All packet indexes are sorted in parallel across the machine by using parallel sorting by regular sampling (PSRS) [7]. Finally, the sorted data are gathered back to the master node for storing as the production data set.

The PSRS is one of the best algorithms that are appropriate for low-bandwidth network [7]. The PSRS has four main processing steps. At first, the local data is sorted at each node. Secondly, each node finds the list of values (pivots) that separate the local sorted data into equal partitions and send this list to the master node. The master sorts all the pivots to find the global pivot list, which it then broadcasts to every node. All slave nodes use the global pivot list to re-sort their local data. The third step is to exchange the partitioned data among the processors. In the last step, all sorted packets must be sent back to and merged at the master.

5. EXPERIMENTAL RESULTS

Experimental Setup

The XTE data (described in the next subsection) is used in our experiment. The total XTE data file size is approximately 80 Mbytes. It contains about 64K CVCDUs. As to observe the feasibility of sustaining real-time rate of incoming data we also extrapolated the projection of the system with higher network bandwidth and I/O speed.

The XTE Data

The system performance has been tested using real-life test data primarily to observe I/O and data processing rates. The test data has been obtained from X-ray Timing Explorer (XTE) spacecraft, which was launched in August 1995 for measuring astrophysical X-ray source characteristics [10]. Three instruments studying a variety of sources from white dwarfs to active galactic nuclei transmit real-time or playback data at rates of up to 1M bits per second. XTE telemetry data is composed of user telemetry data packets that contain individual spacecraft and instrument housekeeping, engineering, and science data parameters. These packets are encapsulated in fixed CVCDUs of 1264 bytes and assembled into 13 dedicated virtual channels.

Performance Measurements

Individual experimental results are obtained for seven activities: 1) Scattering CVCDU data; 2) Extracting the packet data from the CVCDUs; 3) Sorting local data on every node; 4) Finding the global pivot list; 5) Exchanging the packet among the processors; 6) Writing indices to disks; and 7) Gathering the indices back to the master. In the first part, called scattering, the master reads the CVCDUs data from disks and distributes them to all other node. This part requires long time for both reading the disk and distributing the CVCDU. In packet extraction, each processor extracts the packets concurrently before passing to the third step, which is to sort the local data on each node. In the second and third steps, most of the processing time is spent in computation. In the fourth step, each node finds the pivot values of its data, send them to the master, and then wait for receiving the global pivot list back from the master. After all nodes have global pivot list, in the fifth part, every processor begins exchanging the data among other processors. Most of the time spent in this part is for communication. The sixth step, data are written to the local disks in parallel before they are gathered back to the master node.

Projection of the results

The bottleneck of the PACET is the network bandwidth and I/O performance. After running the experiment, we project the result by assuming that we could have faster interconnection, more machines, and/or disk speed. The results are projected from the testbed system to another that uses 9 nodes, Multiple 4-striped disks (MD), and 1Gbps switch. We assume that processing time on the sequential computing fractions would be gained 9/7 times, disk-processing fractions would be gained 4 times, and communication fractions would be gained around 40 times. For communication fractions, we estimate the projection from the basis that we would have 10-time faster network and 4 parallel disks at each machine.
Experimental Results
The results of the experiments and expected projection are shown in Table 1. The first column is for a seven 150 MHz Pentium PC cluster connected via 100 Mbps Fast Ethernet Hub. All other columns, they are the projected results from the first column. Table 1 shows that up to 82,717 packets per second can be processed, using a 9-node system, which would meet our goal of 70,000 packets per second. Figure 5 shows the execution budget charts for the experimental results compared with the projected ones. Note that, for our current configuration of PACET, the parallel algorithm spent about 67% for communication. However, with the faster network and high-speed disks (1Gb+MD), this overhead can be reduced to only about 18% of the total time.

6. CONCLUSIONS AND FUTURE DIRECTIONS
In this paper, we demonstrated a commercial off the shelf (COTS) solution for telemetry processing, using our Beowulf-based PACET system. It was shown that networks of workstations, particularly PC-clusters, can achieve NASA's goals for telemetry processing and potentially many other applications. The key to achieving such results is understanding and exploiting the data characteristics. In the case of telemetry processing, the common belief that packet indices should be distributed after packets are extracted sequentially is quite limiting. On the other hand, distributing CVCDUs helps parallelism, but introduces load imbalances. A load-balancing step, however, is easy to incorporate and improves the overall performance.

REFERENCES


[10] Rossi X-ray Timing Explorer (RXTE) 
http://heasarc.gsfc.nasa.gov/docs/xte/XTE.html

Tarek El-Ghazawi received his B.S. degree from the University of Helwan, Helwan, Egypt, in 1980, and his M.S. and Ph.D. degrees from New Mexico State University, Las Cruces, NM, in 1984 and 1988 respectively, all in Electrical and Computer Engineering. Dr. El-Ghazawi has been on the faculty of Department of Electrical Engineering and Computer Science at George Washington University since 1990. Prior to joining GWU, Dr. El-Ghazawi also taught at the John Hopkins University and Frostburg State University. His research interests include high-performance parallel computing, high-performance I/O systems, parallel and experimental computer architecture, and performance analysis.

Dr. El-Ghazawi's research work has been supported by NASA Goddard Space Flight Center, NASA HPCC and CESDIS/USRA. He has served as a program co-chair for the International Conference on Parallel and Distributed Computing and Systems, 1991, and is the workshop chair for the Frontiers of the Massively Parallel Computations, 1995. Dr. El-Ghazawi is a senior member of IEEE, and a member of the ACM, ISCA, and Phi Kappa Phi national honor society.

Prachya Chalermwat is received his B.S. degree in Electrical engineering from Chulachomklao Royal Military Academy (CRMA), Thailand, in 1985, and his M.S. in Computer Science from the George Washington University (GWU) in 1991. He is now a Ph.D. student and a graduate research assistant in Department of Electrical Engineering and Computer Science at GWU. He received a scholarship for his M.S. and Ph.D. degree from CRMA where he has been a faculty in Department of Electrical and Computer Engineering. His research interests include parallel and distributed computing on cluster of workstations, heterogeneous computing, and parallel image processing. He is a student member of IEEE.

Punpiti Piamsa-nga received his B.Eng. and M.Eng. (Electrical Engineering) from Kasetsart University (KU), Thailand in 1988 and 1991, respectively. He has been a faculty at Department of Computer Engineering, KU since 1991. He is now a Ph.D. student at Department of Electrical Engineering and Computer Science at George Washington University, with a scholarship from Ministry of Science, Technology, and Environment, Thailand. His current research interest is parallel and distributed modeling on multimedia systems. He is a student member of ACM.

Armagan Ozkaya received his BSEE and MSEE degrees in Electronics and Telecommunication Engineering from Istanbul Yildiz Technical University, Turkey in 1985 and 1987, respectively. Between 1990 and 1995, he held teaching and research assistantship positions at NASA-GSFC and the George Washington University where he earned a doctoral degree in Computer Science in 1996. He is currently a Technical Director in Health Administration Systems, Inc. Dr. Ozkaya's interests lie in the areas of
parallel and supercomputing systems, algorithms and advanced data structures, and imaging and OCR.

*Nick Speciale* is the microelectronics section head for the mission operations and data systems directorate, Code 500, at NASA GSFC.

*Donald Wilson* is the technology manager for the mission operations and data systems directorate, Code 500, at NASA GSFC.