Using Transaction Based Parallel Computing to Solve Image Processing and Computational Physics Problems

Harold Trease (het@pnl.gov), Daniel Fraser (fraser@mcs.anl.gov),

Rob Farber (rmfarber@pnl.gov), Stephen Elbert (steve.elbert@pnl.gov)

http://www.cca08.org/

1 Introduction

As HPC (High-Performance Computing) approaches the power, reliability and performance walls for monolithic, standalone architectures the HPC community needs to transition to new parallel computing paradigms. Taking advantage of the increased aggregate computing resources available in cloud computing requires that methods and algorithms (and indeed our whole computing infrastructure) be reformulated. In this paper we describe our initial formulation and implementation for applying transaction based parallel computing for solving image processing and computational physics problems, where a transaction based parallel formulation is used as compared to the tried and true task parallelism and data parallelism paradigms we have used in the past. We describe a computing infrastructure formulated around the Hadoop framework of (Google's) MapReduce for transaction parallelism. Two application areas (image processing and subsurface transport) are used to demonstrate the utility of Hadoop, MapReduce, and transaction based parallel computing to cloud computing. The image processing application demonstrates the use of the Hadoop framework applied to high-throughput, high-performance analysis of archives of video surveillance data. Also, we describe a computational physics application that models subsurface transport of plumes of high-level radioactive waste in complex stratigraphy to demonstrate another application of a transaction based parallel implementation. Algorithms that describe the performance characteristics, redundant computing, failure recovery, and defensive computing will be presented. A goal of this paper is to contrast task and data parallelism vs. transaction parallelism with respect to our two applications. To date the majority of our work as been focused on the mapping our image processing application to a cloud computing infrastructure, but work has begun on mapping the subsurface transport application also.

Section 1 provides an introduction, Section 2 describes the image processing implementation, in Section 3 we describe the subsurface transport implementation, and Section 4 provides conclusions.

KEY WORDS

Transaction Parallelism, Grid Computing, Cloud Computing, Graph Analysis, High-Performance Video Processing, High-Throughput Video Content Analysis

2 Distributed, Hybrid Multi-Core Processing Framework For Video Analysis

In Section 2.1 and 2.2 we describe our current distributed, multi-core image processing framework that supports data-intensive, high-throughput video analysis and in Section 2.3 show how this is mapped into a Hadoop MapReduce framework. As background two aspects of our current framework are highlighted: one is the distributed, hierarchical, hybrid multi-core parallel processing framework and the other are the algorithms used for video processing and video content analysis.

The specific technical areas to be covered in this paper include:

- High-performance, high-throughput processing of video data:
  1) Network framework of interconnected hybrid multi-core processors
  2) Multi-core hardware
  3) Control, data and message framework
  4) MPI (Message Passing Interface), MapReduce and MeDICI network control languages
  5) Programming language for coding video processing application components

- Video content analysis:
  1) Video decoding and frame generation
  2) Object segmentation and feature extraction
  3) High-dimensional signature generation and PCA (Principal Component Analysis) dimensionality reduction and coordinate generation
  4) Graph generation
  5) Social network analysis
  6) Event classification

Fig. 1 show a description of the computational framework that was designed, implemented, and is being used to

support the video processing and content analysis described in this paper. It shows the layout of the hierarchical processor framework as a network of interconnected, distributed machines. The network is both logically and physically distributed, where the nodes of the network are connected through high-speed interconnections. The capabilities of each architecture are exploited in order to optimally perform one of the four principal video processing functions represented by this pipelined workflow. These functions include: 1) ingesting the raw video, 2) decoding the videos, extracting content and generating unique signatures from the content, 3) optimization and data reduction of the signatures using PCA and 4) graph analysis supporting functions such as (human) face recognition, object recognition, video based social network analysis (i.e., who is interacting with whom) and event classification.

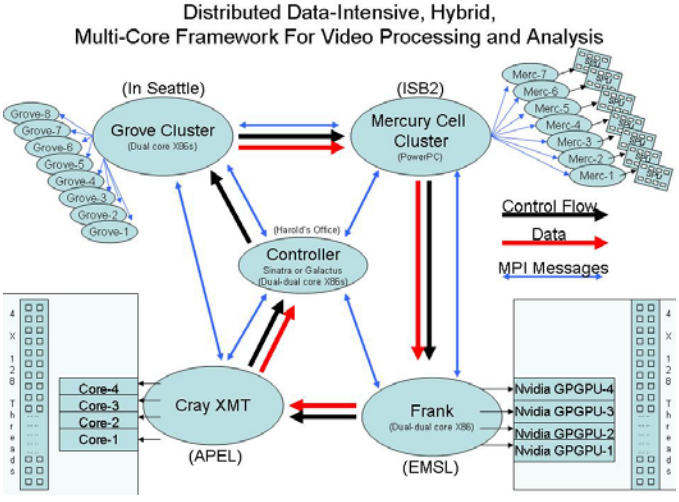

Fig. 1. Presents the distributed, parallel, hybrid multi-core processing framework used for high-throughput video processing and analysis. The hardware components include a Linux cluster (called Grove), an IBM BE Cell (PS3) cluster (called the Mercury cluster), an Nvidia GPGPU cluster (called Frank), and the Cray XMT. Three software networks are shown: 1) Control network (black lines) directs the workflow, 2) Data flow network (red lines) communicates data, and 3) point-to-point communication network (blue lines) communicates messages (both functions and data). Some of the nodes of the network are separated by hundreds of miles which makes the internet communication bandwidth a critical component.

At a very high level the configuration shown in Fig. 1 contains four major elements:

1) A hardware network of multi-core processors connected across a hierarchy of interconnects, where the hardware components are physically separated by hundreds of miles.

2) A control flow network (the black lines) that controls the major workflow components of the network.

3) A data flow network (the red lines) that manages the flow of raw and processed video data throughout the network.

4) A communication network (the blue lines) manages the messages that are passed between the network (master) nodes and the cluster (slave) nodes, where the message may contain both data and functions.

The following sections of this paper will present a brief description of both the hardware architecture, programming models and video processing capability, used within this framework. Section 2.1 describes the hardware architecture used for managing and processing the workflow associated within the framework. Section 2.2 and 2.3 provides a description of the programming models used within the framework. Section 2.4 describes the capabilities and algorithms used for performing the video content analysis.

2.1 Multi-Core Hardware Network

Figure 1 (above) shows a hierarchy of compute nodes that form a network connectivity architecture that supports the data storage, video processing and content analysis framework used to describe the workflow in this paper. There are currently five processing nodes in this hierarchical, heterogeneous network, where each node represents a parallel cluster and the compute nodes of clusters are massively threaded multi-core processors. The controller acts as the "master" process and the other four nodes are "slaves" each of which execute applications that perform the video processing steps 1- 4 described in Section 2.4 of this paper. The purpose of this network is to "map" the video processing workflow onto a set of hybrid, multi-core processors in order to "reduce" the raw unstructured video data to structured data in the form of searchable databases or graphs that can then be used to extract relationships between frames, faces and objects.

The major components of this framework consist of:

1) The Controller (center): The controller is in fact two Linux 64-bit Intel X86 class multi-core workstations with lots of disks and monitors attached; one performs task management and communication; the other acts as a data server. The controller is the "master" process of the control network that orchestrates the parallel processing of the video data across the network.

2) The Grove Cluster (upper left): This is a typical Linux cluster of Intel X86 multi-core processors with a large RAIDed storage system that contains the video data.

3) The Mercury Cell Cluster (upper right): This a cluster of IBM BE Cell [6] processors produced by Mercury Computers Systems, Inc. Cell processors are the high-performance versions of the Sony PlayStation 3 gaming processors. Each Cell processor can execute 16 threads.

4) Frank (lower right): Frank is a dual-dual core X86 workstation with four Nvidia GeForce 8 Series of GPGPUs [7] (General Purpose Graphics Processing

Units) attached. They act as computational accelerators. Each GPGPU executes 128 threads.

5) Cray XMT (lower left): The Cray XMT [8] is a new hardware architecture produced by Cray Computers, Inc. and is a highly multi-threaded, large shared memory machine. Each processor is made up of four cores, where each core is capable of processing 128 independent hyper-threads.

Fig. 2 shows the overall topology of the network (from Fig. 1) with images of the various chips superimposed on each multi-core processor. The front-end processing nodes are all X86 type processors (some are dual core and some are dual-dual core), but the threaded processors for the Cell, Nvidia and Cray XMT are massively (thread) parallel with different types of memory layout and access methods. The X86s use a mixture of shared memory for the multi-core processors and a distributed memory space for the cluster nodes. The Cell processors use a heterogeneous memory architecture with each thread having its own separate memory and a DMA (Direct Memory Access) transfer engine for moving data between the PPE (Power Processing Element) and the SPEs (Synergistic Processing Elements). The Nvidia GPGPUs have a mixture of memory access types that include global, shared and texture memory, where memory transfers between the frontend and the GPGPU accelerators is performed using the DMA transfers accessible as library calls from Nvidia's CUDA compiler language. The Cray XMT uses a large shared memory space.

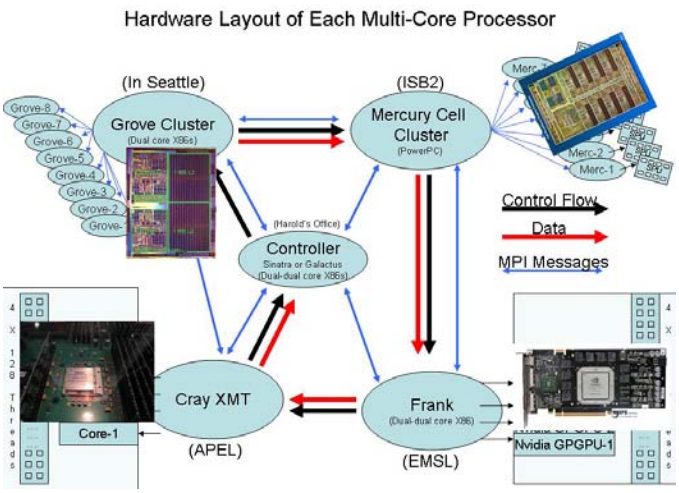

Fig. 2. This figure shows images of the physical layout of the multi-core hardware clips overlaid onto the control and processing network from Fig. 1.

2.2 Parallel Application Programming Models

Figs. 3 and 4 show the programming models used for each multi-core processor. In Section 2.1 (above) we described the hardware processor and memory configuration. These two figures generally show the details of the programming models for each multi-core. For efficiency purposes the applications that run on each hardware take advantage are our best (current) implementation of the best programming language for that platform.

The computer languages and compilers for each platform are:

1) X86s: For the X86s we used C and C++ with MPI for cluster node communication.

2) Cells: Mercury's MCF (MultiCore Framework) and the IBM Cell SDK using macro-assembly language calls from C.

3) GPGPUs: Nvidia's CUDA (Complete Unified Device Architecture) which is basically C with several Nvidia language and library extensions.

4) Cray XMT: Cray's C compiler uses a global shared memory programming style with compiler directives.

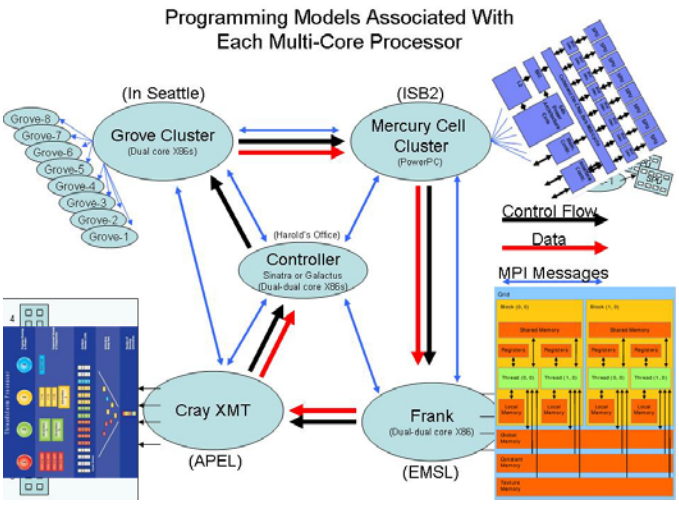

Fig. 3. This figure shows the different programming models applied at the different multi-core processors.

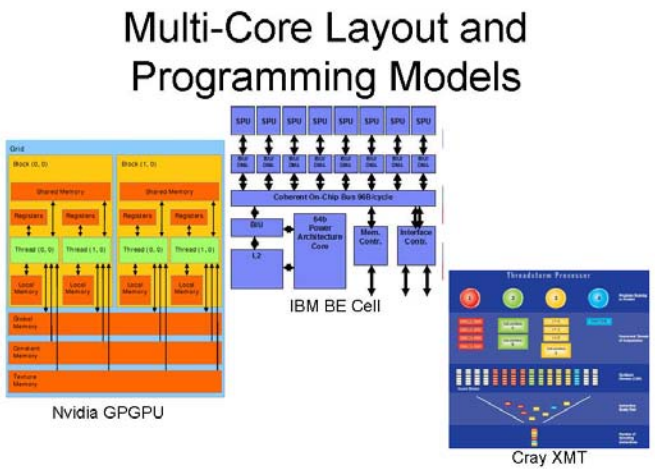

Fig. 4. This figure shows images of the physical layout of the multi-core hardware clips.

2.3 Cloud Computing Framework For Image Processing

Fig. 1 shows the hierarchical framework for processing video data. The center node is called the "controller". The other four nodes (Grove, Cell, Frank, and the Cray XMT) are the slave nodes for the framework, but also act as master nodes that spawn off other parallel jobs to their respective clusters. The communication between the nodes of this network is accomplished using a standard MPI (Message Passing Interface) point-to-point message passing protocol. MPI is implemented as a set of library callable functions that are embedded into the master and slave applications, where the master communicates with the slaves and the slave communicate with each other. Both functions and data are communicated through the MPI interface.

In a recent re-implementation of this framework the MPI controller is being replaced with two experimental controller paradigms to implement this video processing workflow. One is shown in Fig. 5 that shows the use of Apache Hadoop[9] to implement a version of Google's MapReduce[10]. The other uses a version of the PNNL MeDICI[11] software architecture. Both Hadoop and MeDICI are based on Java implementations.

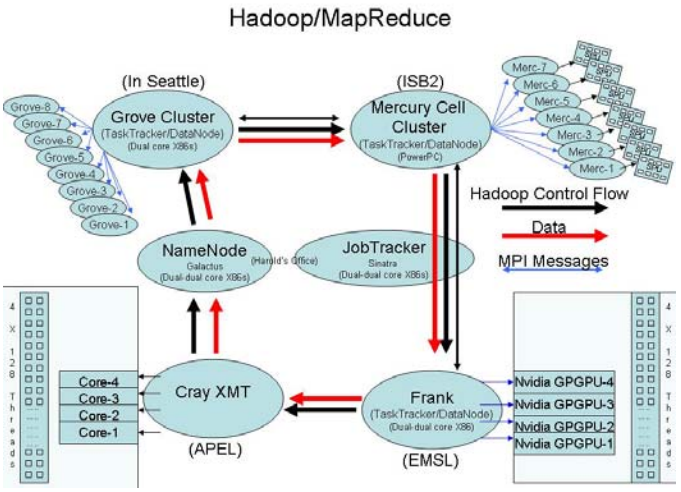

Fig. 5. This figure shows the control, data and message layout of the network that uses the Hadoop open source implementation of Google's MapReduce.

3 Transaction Based Parallel Computing Applied to Subsurface Transport

Our second application demonstrates a new generation of "edge" based computational physics solver that uses an unstructured mesh, subsurface transport model, scalable to very large numbers of processor cores and fault tolerant to hardware failures of processors, communication links, and memory/disk storage devices. This demonstrates the use of transaction based parallel computing as the parallel programming model to enable scalability, fault tolerant, and defensive computing capabilities while solving subsurface transport using adaptive, unstructured computational physics meshes and solvers. This transaction based parallel implementation uses a MapReduce programming model, where a transaction is defined to be a mesh edge connection (i.e., element-to-element or node-to-node connection). The high-resolution unstructured, boundary-fitted/volume-filling subsurface mesh is composed of many (i.e. many billions) of edges that serve as transactions that are mapped to the processors followed by a reduction of the results, thus integrating the subsurface transport equations. In this subsurface transport case, each transaction edge represents a connection between two element centers with a conserved, areal, fluxing face centered between them, along the edge. The result of the Map operation is the amount of the conserved quantity that is transport from one element to the other. The result of the Reduce operation is the update to the elements reflecting how much of the conserved quantity each element gained or lost based on the fluxes between it an all of its neighboring elements. Another way of looking at this transaction edge is that it represents a row or column entry in a sparse matrix, where the rows represent the mesh elements, the columns represent the neighboring elements connected to a given row-element, and the value at a specific row/column

intersection represents the coupling coefficient. The result of the MapReduce operation, in this case, is a matrix/vector multiply. The subsurface transport model equations and solvers will be based on PNNL's Subsurface Transport Over Multiple Phases (STOMP) code, but implemented on PNNL's P3D NWGrid/NWPhys adaptive, unstructured 3-D meshes using an edge-based finite-volume integration scheme. The Metis option in Zoltan is used for domain decomposition for defining the partitioning of the mesh edges. This application provides a valuable demonstration of the functionality, utility, and efficiency of transaction based parallel computing for executing complex applications for solving subsurface and carbon sequestration transport problems.

4 Conclusion

In this paper we have presented a hierarchical network of multi-core processing nodes used for the high-performance, high-throughput processing and video content analysis of streaming and archived video data. The work presented in this paper shows an implementation of a video processing and content analysis workflow mapped to a hetrogenous, distributed network of multi-core processors. Each component of the workflow represents an optimal implementation on a given architecture. We also described the implementation and application of hierarchical parallel programming models. For the control network two models are presented based on MPI (for task parallelism) and MapReduce (for transaction parallelism). For the parallel processing on the respective processing nodes the applications use a combination of shared/distributed SMP parallelism, SIMD data, vector, and hyper-threaded parallelism.

The goal of this work is to maximize the number of frames-per-second of streaming or archived video data that can be processed. Because each multi-core processor has its own programming language (or language extensions) and optimization strategies, that are different from other architectures, the code development process must be highly organized to keep track of everything. The applications that can be supported by the work presented in the paper include: 1) processing data from many (100s, 1000s, 10,000s+) surveillance cameras, and 2) processing archived video data from repositories or databases. The rate of speed of this system ranges from one DVD per second (5 Gigabytes per second) on Linux clusters[12] to 500 Gigabytes per second (for the PCA analysis on a modern Nvidia GPGPU multi-threaded cluster using only 4 GPGPUs). These numbers demonstrate a much greater that real-time processing rate for the framework.

The workflow and algorithms described in this paper have not yet been implemented on a TeraGrid architecture, but that is the next step and the TACC Ranger system is the first target.

Acknowledgment

This work was performed at the US DOE Pacific Northwest National Laboratory in Richland, Washington and used facilities provided by EMSL (the Environmental Molecular Sciences Laboratory). Equipment was provided by Nvidia, Mercury Computer Systems Inc., and the Cray: The Supercomputer Company.

The authors wish to thank Tim Carlson (PNNL), Lynn Trease (PNNL), and Ryan Mooney (now with Google.com) for their contributions to this project. In addition, we wish to thank Pacific Northwest National Laboratory Data Intensive Computing Initiative and the Environmental Molecular Sciences Laboratory (EMSL) at PNNL for making the resources available to do this work. We wish to thank the TeraGrid and TACC organizations for providing us the opportunity to implement and test the processing algorithms and control network described in the paper on their computing infrastructure.

References

[1] Trease, HE, "Advanced Data Processing: Parallel, High-Performance Data Analysis Using Scalable, Adaptive and Predictive Information Processing Agents", 2007. PNNL-17177.

[2] MPlayer: http://www.mplayerhq.hu/design7/news.html

[3] FFmpeg: http://ffmpeg.mplayerhq.hu/

[4] Trease HE, T Carlson, R Mooney, R Farber, and LL Trease. 2007. "Unstructured Data Analysis of Streaming Video Using Parallel, High-Throughput Algorithms ." SIP (Signal and Image Processing), 2007:300-305. PNNL-SA-57622, Pacific Northwest National Laboratory, Richland, WA.

[5] Trease, H.E. (1981), "A Two-Dimensional Free Lagrangian Hydrodynamics Model," Ph.D. Thesis, University of Illinois, Urbana-Champaign. "The Free-Lagrange Method", Lecture Notes in Physics, Vol. 238, 1985, Edited by M.J. Fritts, W.P.Crowley, and H.E.Trease. "Advances in the Free-Lagrange Method", Lecture Notes in Physics, Vol. 395, 1991, Edited by H.E.Trease, M.J.Fritts and W.P.Crowley.

[6] IBM BE Cell: http://en.wikipedia.org/wiki/Cell_microprocessor

[7] Nvidia GPGPU: http://www.reghardware.co.uk/2007/06/21/nvidia_launches_tesla/

[8] Cray XMT: http://www.cray.com/products/xmt/

[9] Apache Hadoop: http://hadoop.apache.org/

[10] Google's MapReduce: http://en.wikipedia.org/wiki/MapReduce

[11] PNNL'a MeDICI: http://dicomputing.pnl.gov/capabilities/softwarearchitectures/

[12] SC05: http://www.homeandoffice.hp.com/HPC/cache/281629-0-0-34-61.html

[13] Trease HE, RM Farber, AS Wynne, and LL Trease. "High-Performance Video Content Analysis Using Hybrid, Multi-Core Processors", submitted to IASTED SIP (Signal and Image Processing) for publication April 2008.

[14] Farber RM, HE Trease, "Linearly Scalable Algorithms for Unstructured Video Analysis", submitted to the TeraGrid'08 conference.