

ARCHITECTURES AND TECHNIQUES FOR USING HIGH PERFORMANCE COMPUTERS TO SUPPORT WORKSTATION BASED DII COE APPLICATIONS

Samip Bhavsar, Wei Hu, and Manish Parashar
Department of Electrical & Computer Engineering
Rutgers, The State University of New Jersey
Piscataway, NJ 08854
{samip, weihu, parashar}@caip.rutgers.edu

Abstract

The primary objective of this paper is to present a software system architecture that will enable the use of network-based High Performance Computing (HPC) environments for a family of DII COE applications. The key innovation of the architecture is an asynchronous interaction model that enables an effective vertical separation of application components (interactions, computation, and database access) and their distribution across the HPC environment. The second objective is to present a series of experiments using a concurrent multi-target tracker application to study its characteristics and to the effect of such an asynchronous architecture on its execution.

Motivation

The Defense Information Infrastructure (DII) Common Operating Environment (COE) encompasses both software and hardware. The current hardware platforms cannot support the kinds of application that require parallel computing, so that the current generation of COE software would little benefit from access to parallel computers. Furthermore, without access to parallel computers, future DII COE applications will be constrained to fit existing DII COE platforms. Readily available parallel processing support for computation intensive applications would be an enabling technology that would usher in new classes of applications desirable for fielding on the DII COE platform, real-time applications such as multi-sensor data fusion, multi-target /multi-asset target assignment, war-gaming simulations, war-gaming simulation support for Course of Action (COA) planning, Forces Modeling, Virtual Prototyping and Virtual Testing, BattleLabs, Battlespace Visualization, and other support for the ongoing Battlefield Digitization. Our objective in this paper is to present an architecture that can make effective use of HPC parallel computing assets to support DII COE applications. We propose a three-tier system consisting of one or more DII COE clients accessing a parallel processing server using specially designed middle-ware (based on active messages) enabling asynchronous interactions. To study the feasibility the architecture, we are using a concurrent implementation of a multi-target tracker required by a Battle Management Command Control. The multi-target tracker tracks in-coming missiles for a mass raid scenario where multiple ground-based missiles are fired in concert against friendly targets. Initial experiments on using asynchronous parallelism for such an application are presented.

Target Application Architecture

Many military applications can be modeled as operations performed on a set of units, such as a brigade or a battalion or a tank, where the operation for each unit or group of units (i.e. a task force) is independent of computations done on the rest of the units. Typical operations include interleaved computations, database stores/fetches, visualization and interactions. The characteristic problem space for such military applications ideal for parallelization. Problems readily scale up based on the number of military units involved. Dependencies amongst units are typically geographic and can be overcome by clustering units based on their geographic

relationships. Although, the relative lack of dependencies between computations supports construction of very nice parallel processing architectures, typical applications require a lot of user interaction, visualization and database accesses interleaved with computations, making conventional parallelization models unsuitable. This is primarily because server subsystems such as the database and visualization applications are stand-alone, workstation based and sequential. Further, these server systems are typically connected to the parallel resource using high latency interconnects such as Ethernet and FDDI. Using conventional models and for parallelization, the performance gains obtained by executing computations on a parallel system and quickly offset by the sequentialization at the data-base and visualization servers, and the latency of the interconnect.

Software System Architecture

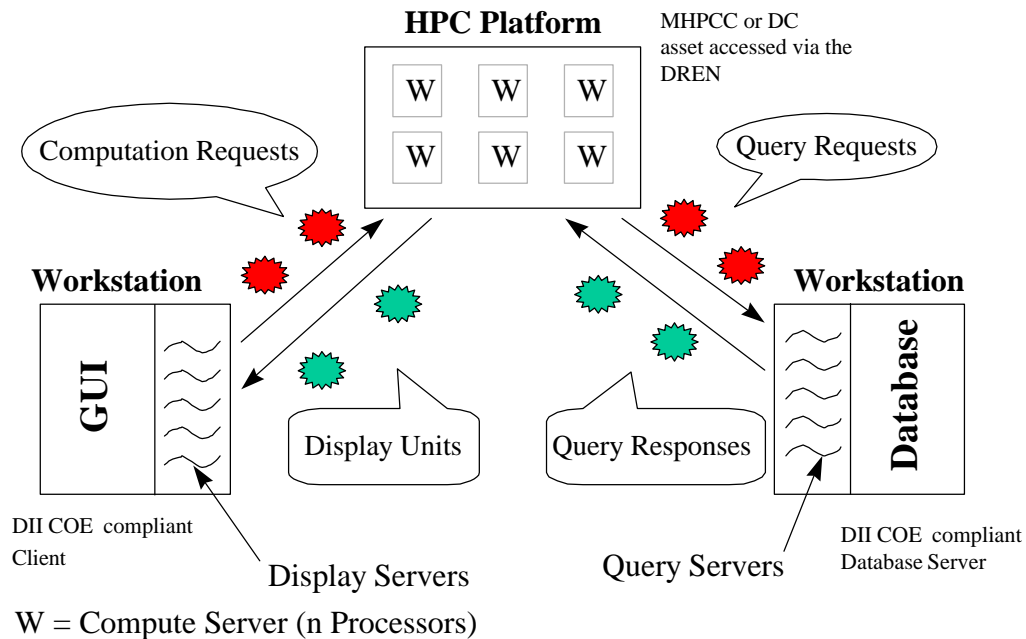


Figure 1 – Three-Tier Software System Architecture

The proposed architecture is a *three-tier* system consisting of front-end services such as the user interface and visualization subsystem, the computational engine and back-end service such as persistent stores. A schematic overview of the architecture is shown in Figure 1. A thread based middle-ware glue, supporting efficient asynchronous interactions and remote function invocations are used to link the three tiers and create a virtual pipeline. Application units are grouped into tasks of sufficient granularity so as to amortize network latency and are piped through the system. The processors of the parallel compute engine are grouped to match the task size to create pool of compute servers.

Consider a three-tier architecture consisting of two DII COE workstations coupled with a server hosted on a SP-2 distributed memory multiprocessor. DII COE Workstation (1) hosts front-end services such as the visual displays and the user interface. DII COE are grouped into a pool of N compute servers, each fulfilling the computational requirements of application tasks composed of m units. An overall compute server executes on a SP2 processor and is responsible for scheduling application tasks to available computer servers. This compute scheduler manages multiple threads of computation. It interleaves computations associated with a cluster of units with the communication latencies, display latencies, and database access latencies associated with

computation at other points in what amounts to a classic data pipeline. Similar servers executing on the front-end and back-end workstations schedule visualization/interaction and database access service requests for application tasks. Several threads of computation in the User Interface on Workstations (1) capture user interactions and display data. Each cluster of computation makes a separate query to the database server on Workstation (2). The architecture does not attempt to combine application parallelization with database parallelization. Parallelization of the database is considered a separate problem. The database server executing however can be designed to exploit the capabilities of a parallelized database.

The interaction middle-ware layer uses active messages to implement asynchronous remote function invocations. Messages are typed to actively specify the service needed and are directly forwarded to appropriate handlers on receipt. The interaction model is optimized to amortize communication and service latencies by overlapping them with computations. As incoming request is directly forwarded to handle routines, all receiver side intermediate data copies are avoided.

The proposed three-tier architecture design is rooted in a fundamental “separation of concerns” so that each part of the application can be parallelized and optimized separately. This greatly simplifies the parallelization process over the complexity inherent in parallelizing monolithic computations and makes it suitable to network-based clustered HPC architectures.

Application Description: Multi-Target Tracker (MTT)

MTT implements multi-target tracking capabilities required by Battle Management Command Control and Communication Systems. The application receives input from the Environment Generator and Synthesizer module in the form of sensor scans and target information. The overall multiple target tracking system has two geostationary sensors which scan specific launch sites (specified in terms of latitudes and longitudes) for missiles or targets launched from the surface of the earth. Data from these two geostationary sensors is fed to two focal plane tracking modules at 5 second intervals. The focal plane tracking modules process this data using kinematic filtering and track pruning and prediction algorithms. The four key tasks performed by these modules are focal plane track extension, redistribution of tracks (load balancing), report construction for 3D-tracking, and track initiation. Outputs produced by the focal plane tracking modules are two focal plane reports (for the two sensors) containing an initial prediction of the launch trajectories. The sensor data is then fed to the 3D tracking system, which uses the two focal plane reports to prune duplicate and bad tracks, extend existing tracks, and initiate new tracks. The output of the MTT application is a list of target trajectories.

Figure 2 shows the overall flow for one scan of the MTT application. Sequential and parallel regions inherent to the implementation are shown. The concurrent MTT uses coarse-grained SPMD (single program multiple data) parallelism. Data is transformed through a system of data-structures which are typically global to all the processors and are synchronized at regular intervals. The boxes in the figure indicate the structures through which the data passes while the labels on the arrows indicate transformations. The concurrent implementation of MTT application described in this section (and used in our experiments) was developed by Dr. Salim Hariri and his group at Syracuse University. Further information on the concurrent multi-target tracker application can be obtained from <http://www.npac.syr.edu/users/ryadav/papers/ICPP/Main.html>.

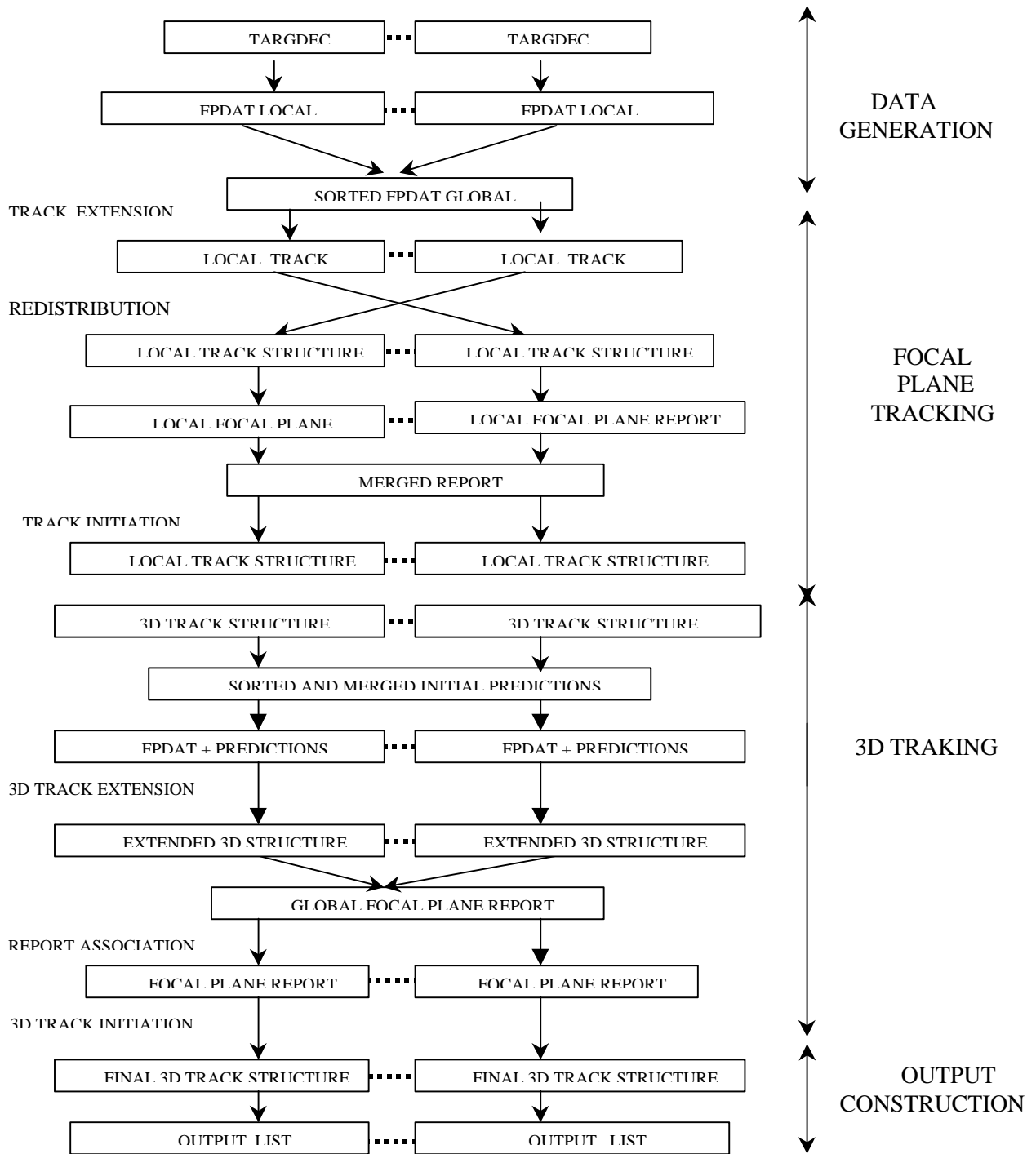


Figure 2 – Control Structure of the Multi-Target Tracker

Experimentation and Analysis

The primary goal of the experiments presented in this section is to analyze the behavior of the concurrent MTT application and to study the effects of asynchronous parallelism on its performance. The concurrent MTT is a non-trivial CPU and memory intensive application consisting of about 34,000 lines of C source code. Inter-processor communications are performed using the Message Passing Interface (MPI)¹ library. Experimental results presented in this section were obtained using the IBM SP2 distributed memory multiprocessor system at the CAIP Center at Rutgers University. This system has 8 eight “thin” nodes with a 66.7 MHz clock and 266 Mflops peak performance. The nodes are connected by a high-speed crossbar switch. The original implementation was based on a synchronous communication model and targeted to hypercube architectures. Its performance on the SP2 is shown in Figure 3.

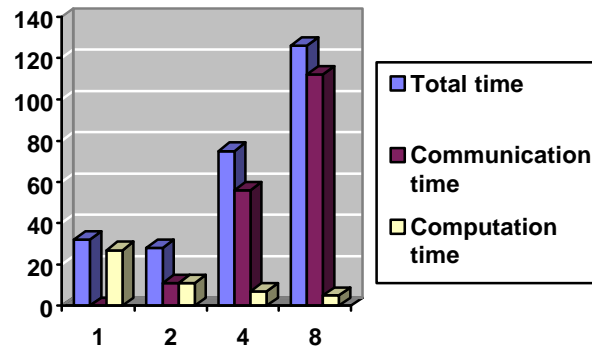


Figure 3 – Original MTT Performance

Experiment 1: Synchronous Vs Asynchronous Communication

Inter-processor communications in the original implementation were performed by a “cshift” algorithm designed for hypercube architectures. In this algorithm each processor sends and receives information using channels along each dimension of the cube. The communications are synchronous with no overlap between computation and communication, and are tailored to the specific architecture. It can be seen from the graph in Figure 3 that communications dominate execution time as the number of processors increased and scaling suffers in this implementation. Our first experiment was to replace the original synchronous cshifts with asynchronous communication where the processors posts a send or receive and returns immediately without waiting for completion. Due to the large amount of communications, this simple change brings significant performance improvements. However, the structure of the implementation and the distribution of data caused the program to continue to be dominated by communication and scaling still suffers for larger number of processors.

¹ For more information on MPI see <http://www.mcs.anl.gov/mpi/>

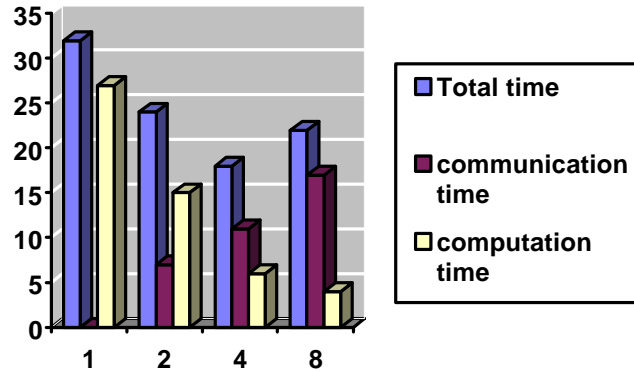


Figure 4 – Experiment 1 – Asynchronous Communication

Experiment 2: A Separation of Concerns

The second set of experiments was aimed at separating out independent concerns in the implementation, which could then be executed asynchronously.

1. The focal plane-tracking module sequentially processed sensor data from the two sensors. This processing was separated into two modules that can run independently and asynchronously.
2. The original implementation was based on a series of global structures that were synchronized at regular intervals (see Figure 2) and were used by the computation throughout each scan. For example, in the focal plane tracking module, the focal plane data structure (FPDAT) containing track data is first synchronized (globally sorted and merged) to form a global structure and then used for focal plane track extension. The extended tracks are then redistributed among the processors to balance load. Applying a separation of concerns we first extracted synchronization and distribution of the FPDAT structure from the track extension computation. This enabled us to distribute FPDAT so that all tracks using a particular focal plane data were mapped to the same processor before track extension. As a result track redistribution was no longer required and the processors could compute the tracks locally with minimum communication. Such distribution also maximized the number of matches among tracks during focal plane report construction that follows track extension.

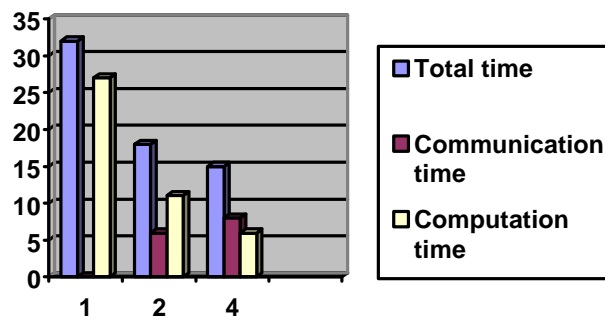


Figure 5 - Experiment 2: Separation of Concerns

Conclusions

This paper presented an architecture for a software system that will enable the use of network-based High Performance Computing (HPC) environments for a family of DII COE applications. The architecture was based on using multi-threaded asynchronous interactions to enable effective vertical separation of application components (interactions, computation, and database access) and their distribution across the HPC environment. The paper also presented initial experimental results illustrating the benefits of asynchronous communications and separation of concerns to an existing concurrent multi-target tracking application. The goal of this work is to use the proposed software system architecture to port this application to a network based HPC system where the parallel system will process incoming sensor information in real-time and display the computed tracks on a graphics workstation.

Acknowledgements

This work was done under contract with the U. S. Army CECOM as part of the C3TE&I program under Delivery Order 079 High Performance Computing for AGCCS. This work was funded by the Department of Defense (DoD) High Performance Computing Modernization Program (HPCMP) initiative through the Department of Defense's High Performance Computing Modernization Office (HPCMO) under its Common High Performance Computing Software Support Initiative (CHSSI) for the Computational Technology Area (CTA): Forces Modeling and Simulation / C4I (FMS) as a portion of work done in support of the FMS-1: Scalable HPC Environment for C4I (SHEC) project at the U. S. Navy, SPAWAR Systems Center San Diego. The authors would like to thank Dr. Salim Hariri and his group at Syracuse University for making the source of the concurrent multi-target tracker application available. The authors would also like to acknowledge the contributions of Dr. Israel Mayk and Dr. David Usechak at U.S. Army CECOM, and Joe Stowers at Booz-Allen and Hamilton, for their invaluable contributions to this work.