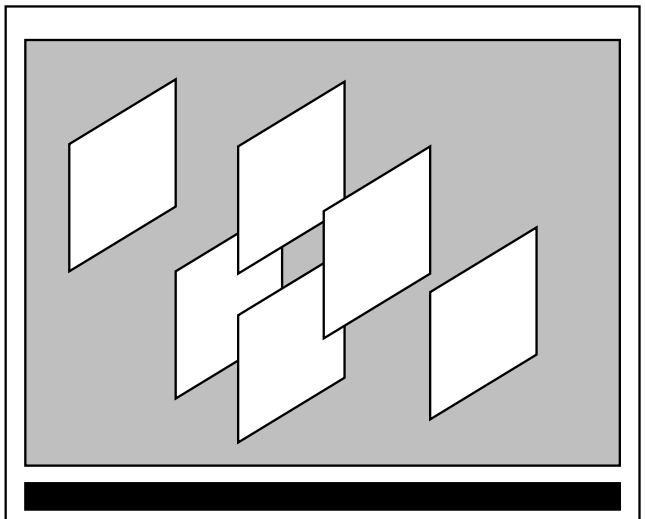

A Concurrent Multi Target Tracker: Benchmarking and Portability [1]

Salim Hariri, Rajesh Yadav, Balaji Thiagarajan,
Sung-Yong Park, Mahesh Subramanyan, Rajashekar Reddy and G. C. Fox

Northeast Parallel Architectures Center
Department of Electrical and Computer Engineering
Syracuse University

SCCS # 659

[1] This research is funded by Rome Laboratory (contract number F-30602-92-C-0063), Rome, NY

Abstract

With the current advances in computing and network technology and software, the gap between parallel and distributed computing environment is gradually becoming narrower. Consequently, parallel programs run on parallel as well as distributed systems. However, programming and porting complex applications to such environment is challenging task and not well understood.

In this paper, we use a concurrent multi target tracker as a running example to analyze and evaluate performance of two different parallel implementations on parallel and distributed systems. We have benchmarked both these implementations on different architectures that vary from a network of workstations(SUN, IBM RS6000) to parallel computers (CM5, iPSC 860) using different parallel/distributed message passing tools(PVM, p4, EXPRESS, HORUS). We have also compared the performance of these tools to provide important communication primitives for high performance computing applications such as send/receive, broadcast and circular communication.

1 Introduction

Current advances in technology is unifying parallel and distributed computing such that the distinction between the two fields is becoming more and more blurred. For example, an application developed based on message passing programming paradigm can run on parallel and distributed environments without any major changes. Such a paradigm can capitalize on existing architectures and on enhanced facilities in computing, networking and communication technology to provide efficient, cost-effective, scalable, and high-performance solutions. Software development for complex application that run on parallel and distributed systems is a non-trivial process and requires a thorough understanding of the application and architectures; currently, most of the well studied applications are small and are not complex enough to help researchers understand the programming, portability and performance issues associated with software tools for parallel and distributed computing environments. The main objective of this paper is to investigate these issues by experimenting with a compute intensive application.

We select the multi target tracking application as the first application in a benchmarking suite being developed at NPAC of Syracuse University to evaluate performance of software tools as well as parallel/distributed platforms. The tracker has been written using different message passing tools [**?**] that include PVM [**?**], p4 [**?**], EXPRESS [**?**] , and PICL [**?**]. The tracker has been ported on the nCUBE, CM5, Intel iPSC 860, IBM-SP1 architectures and a cluster of SUN4 and IBM RS6000 workstations.

To simplify the portability issue in a heterogeneous environment we have developed a uniform structure of the tracker algorithm that can be easily coded using different message passing tools. By studying the performance of the tools and tracker on different parallel and distributed systems, we will be able to identify the best computing environment to run that application.

The remaining sections of the paper are organized as follows: In section 2 we discuss the sequential implementation of the tracker. Section 3 discusses two different implementations of the concurrent multi target tracker. Section 4 details the experimental results of the parallel

and distributed implementations of the tracker. It also compares the performance of message passing tools (like PVM, p4, EXPRESS, HORUS) to provide important communication primitives for high performance computing applications. Section 5 presents a summary and directions of future research.

2 Multi Target Tracking System

The tracker demonstrates the multi target tracking capabilities that is required by a Battle Management Command Control and Communication System. It uses an extended 3 stage Kalman filtering formalism which is the primary "tool" used to provide and sort realistic data. This filtering formalism is general and can be used in problems related to pattern recognition, signal and image processing. The 3 stage filter model has helped the development of a concurrent version of the tracker [**?**].

The multi target tracker, shown in Figure **??**, is designed to provide an estimation of launch vehicle parameters for individual targets/missiles in multi-target scenarios. The system deals with a mass raid scenario and is designed to process situations with varying number of targets and launch sites. The tracker receives input from the Environment Generator and Synthesizer module (see Figure **??**) in terms of sensor scans and target information. The multiple target tracking system has two geostationary sensors which scan specific launch sites for missiles or targets launched from the surface of earth. The launch sites are specified in terms of latitudes and longitudes. The data from these two geostationary sensors are fed to two focal plane tracking (FPT) modules (2 dimensional tracking) at 5 second intervals. The focal plane tracking modules process this data using kinematic filtering algorithms and track pruning and prediction algorithms. The output of this module is an initial prediction of trajectories of launched missiles. This data is then fed to a 3D(three dimensional) tracking system which uses the data from the two focal plane tracking modules to prune duplicate tracks (if any), extend existing tracks, prune bad tracks and initiate new tracks. The output of the system is a list of target trajectories.

The concurrent multi target tracker (CMTT) is one of the modules that fits into the battle management command control and communication system(see Figure **??**). This system consists of a set of components which interact with each other by exchanging information for data processing. The main components of the overall system include an Environment Generator and Synthesizer, the Tracker System, Decision Control System and an External Graphics Communication Module, as shown in Figure **??**.

The MTT system gets information from the Environment Generator and Synthesizer. It processes this data and generates the parameters required by the Decision Control System. The Decision Control System uses this information along with data from the Environment Generator and Synthesizer to make decisions on how to manage the existing battle management command and control scenario. This information is fed to an External Graphics Communication module. This module acts as an extension or front-end to the Decision Control Module. For example in a Missile Tracking system the front-end can serves as a visualization tool or data interpreter which shows the trajectory and position of launched missiles. The Fire Control Module performs the actions directed by the Decision Control System.

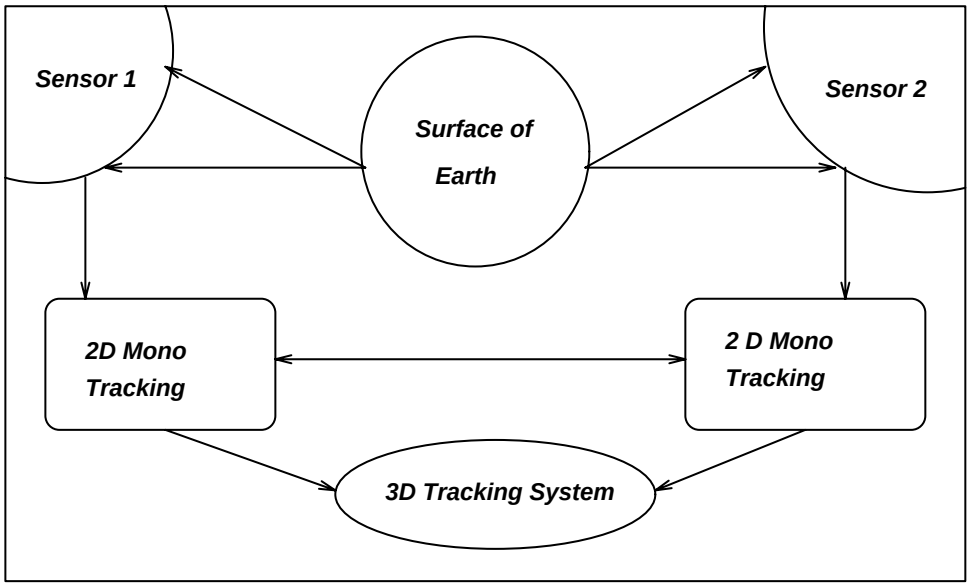

Figure 1: Multi-Target Tracker System

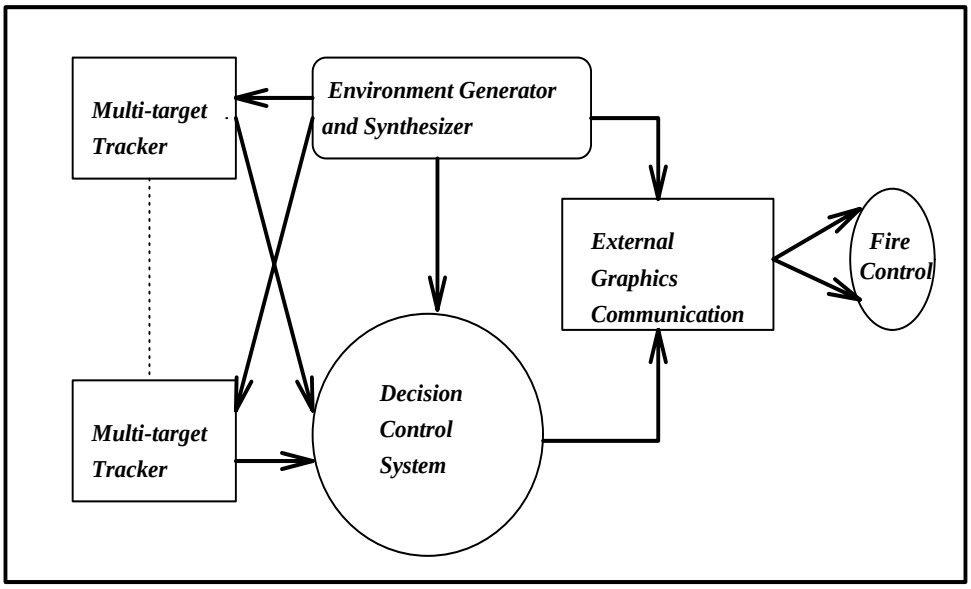

Figure 2: Battle Management Command Control Scenario

In what follows we discuss the issues involved in single target tracking, multi target tracking and concurrent multi target tracking.

2.1 Single Target Tracking (STT)

Tracking of a single target involves three main steps [**?**]:

1. Focal-plane kinematic Kalman filtering
2. Estimation of booster launch parameters from focal-plane state vectors and
3. Combination of individual parameters using a second filter.

The filter in the first step (also called a focal plane filter) processes 2D measurements from the sensor to form estimates of projected kinematic quantities as seen in a sensor focal plane. Position *(x)*, velocity *(v)*, acceleration *(a)* and jerk *($j = da/dt$)* are used for the state variables in the filter. Stochastic contributions are introduced to the jerk to allow the filter to respond to targets travelling along largely unconstrained trajectories. This dynamic uncertainty is the only free parameter of the focal plane filter. Since the filter is linear, all gain and covariance matrices can be computed and tabulated during initialization of the tracking program. This improves the performance of actual filtering. The main output of the focal plane filter is an estimate for the reduced state vector consisting of projected positions and velocities at scan t_k.

$$x = (x, v, a, da/dt) \tag{1}$$

where x specifies the position, v the velocity, a the acceleration and da/dt denotes the jerk introduced in the filter.

In the second step of the 3 stage filter model the state vector in Equation 1 is used to provide an estimate of the launch parameters of the target. An individual trajectory is specified by a 4-component parameter vector as shown below:

$$p = (\theta_l, \Phi_l, \Psi_l, t_l) \tag{2}$$

where θ_l denotes the latitude of the launch site, Φ_l the longitude of the launch site, Ψ_l denotes the initial launch azimuth, and t_l the launch time. The third step simply inverts the relation using the Newton Raphson iteration method [**?**]. The Newton Raphson inversion provides realistic parameter covariance estimates as well as values of the parameters themselves. For the multi-target tracking described in the next section, the task of filtering an individual track is essentially concerning with resolving track-hit ambiguities. The focal plane filter in Single Target Tracking is in fact the primary tool used in sorting out the nature of multiple target threats.

2.2 Multi Target Tracking (MTT)

Given a reasonable filter for processing sensor data from a single target, the essential problem of multi-target tracking is that of associating individual sensor reports with distinct underlying tracks. This objective is accomplished through three additional modules: Track Split Processor, Track Initiator, and Report Function.

TSP Algorithm:

begin

1. Track Extension
   - Extend each track to a predicted position according to the focal plane filter prior to the arrival of new data.
2. Compute Association Region
   - Determine the association region for the predicted position, based on the position uncertainty predicted by the filter.
3. Track Extension
   - Extend/Filter each track using each sensor report found in the association region. If more than one such report is found, the initial track splits into a number of daughter tracks.
4. Track Elimination
   - Eliminate candidate tracks after some number of consecutive scans if no data points are found in the association regions.
5. Track Merging
   - Identify duplicate track entries in the global file and combine according to some merging criteria.

end

Figure 3: Algorithm TSP

2.2.1 Track Split Processor (TSP)

The Track-split processor extends tracks already in a global track file with sensor reports from new data scans. The main steps in the TSP algorithm include Track Extension, Compute Association region, Track Elimination and Merging(see Figure **??**).

2.2.2 Track Initiator (TI)

The track initiator generates new entries in the global track file from previously unused data points. The track initiator is a batch mode processor, meaning that the algorithm uses a number of scans of sensor reports. In the current tracker implementation three scans of data are used. The track initiator investigates all three-hit candidate segments to find potential tracks satisfying two classes of constraints(as shown in Figure **??**). Three-hit candidate segment is identified using the present and previous scan data.

The track split formalism used for multi-target tracking means that the number of can-

TI Algorithm:
For all tracks in the global track file at the end of the present scan:

begin

1. Identify Unique Tracks
   - The 3-hits used in the segment must not coincide with the last three hits of any track already in the global track file.
2. Determine the Consistency of the Track.
   - The track candidate must satisfy a number of simple kinematic cuts for the estimated velocities and accelerations of the 3-hit segment.

end

Figure 4: Algorithm TI

didate tracks maintained by the system will often exceed the number of actual targets. For purposes of integration with the battle management task a third task, "The Report Function" for the multi-target tracker, is introduced as discussed below.

2.2.3 Report Function

The report function summarizes the perceived threat in a concise manner, resolving track-hit ambiguities as well as possible. A standard problem with the tracker is track file "explosion" due to duplicate and superfluous entries. While processing the input data, the tracker builds a database which is used by different algorithms implemented in the multi target tracking system. The report function has various data pruning features to prevent explosion of the track file database. Information processing in the tracker is a complex issue. After each scan of data has been processed, the global report performs the following set of separate tasks:

1. A gross 'threat-portrait' is prepared listing locations of active launch complexes and the number of sensor reports seen for each complex.
2. A single 'best' track candidate is chosen for cases in which multiple entries in the global track file end on a single sensor report.
3. A global report containing parameter estimates, covariance matrices and uniqueness-tags, with precisely one entry for each sensor report association with precision tracks. The file is then sorted according to estimated launch longitudes of the individual threats.
4. The report file for the present scan is correlated with that from the previous scan, so that consistent track identifiers are given to the battle manager throughout the lifetime of the threat.

This report function has been handled by concurrently decomposing the database which is distributed for concurrent processing to various nodes. The processed data is then collected to update the database for further processing. This in effect can be viewed as a distributed database information processing environment.

3 Concurrent Multi Target Tracking (CMTT)

The Multi target tracker was initially developed at California Institute of Technology under Caltech concurrent Computation Project [?]. It was implemented using the CUBIX programming model for embedded architecture (hypercube) viz. Mark III and CrOS III primitives. The CUBIX model is a hostless programming model where there is only one program called a 'node' program which executes on every processor in the hypercube.

We modified the implementation of the tracker so it can be easily ported using existing parallel/distributed computing tools (EXPRESS, PVM, p4, HORUS) on different platforms namely iPSC 860, CM5, IBM-SP1, cluster of SUN SPARC, IBM RS6000 workstations. To achieve this objective we developed a uniform structure of the multi target tracker [?]. In what follows, we discuss two parallel implementations of the CMTT system. In first one the sensors data are processed sequentially while in the second one the sensors data are processed in parallel.

We implemented the CMTT system [?] using host-node programming model because not many parallel/distributed computing tools supports CUBIX programming paradigm. The tasks performed by the host program involves loading and starting the node programs. Because the host tasks are not significant, the computer executing the host program will execute the node program once it has completed the execution of the host program.

As discussed in the previous section, each scan of MTT begins with an existing track file and new set of sensor report. Existing tracks are extended using sensor reports which satisfy the gating criterion [?] of track-split processor.

The concurrency in multi target tracking is achieved by using data parallelism. The data of the global track file, which has the details of processed data obtained from two geostationary sensors, is partitioned among the nodes involved in the CMTT. So, each node executes the same code, but using different data segments of the global track file. Every node has access to full sensor reports file at every scan, and it performs the sequential multi target tracking algorithm on its subset of the global track file.

3.1 Concurrent MTT with sequential sensor data processing (CMTT-SSDP)

Figure ?? highlights the main tasks performed by CMTT-SSPD algorithm.
The most time consuming step in CMTT-SSDP is the redistribution of global track file(step 2.1.3) and it is critical to achieve efficient concurrent implementation. Redistribution must be done such that all tracks ending at a given datum must be assigned to the same node in the next scan. This will reduce the number of duplicate tracks. Because of the irregular transfer of tracks between nodes during redistribution, the transfer of tracks among nodes is done using the *Crystal_Router* communication algorithm [?]. It is an algorithm to redistribute

Algorithm CMTT-SSDP

1. Initialization
   /* initialize the parameters of sensors and targets */
2. **For** I= 1,NO_SCAN **Do**

   begin

   (a) **For** J= 1,NO_SENSORS **Do**

   begin

   2.1 2D tracking

   2.1.1 Compute focal plain data for sensor J.
   2.1.2 Extend existing tracks
   2.1.3 Track Redistribution
   2.1.3.1Compute destination nodes
   /* assign tracks to nodes based on the last data/track */
   2.1.3.2 Redistribute the tracks to the destination nodes
   /* by using Crystal_Router algorithm */
   2.1.4 Compute Focal Plane report
   2.1.5 Initiate new tracks

   end

   (b) 3D tracking (Combine results from both sensors)

   begin

   3.1 Compute 3D tracks
   3.1.1 Generate 3D tracks based on sensor 1 data
   3.1.2 Generate 3D tracks based on sensor 2 data
   3.2 Update 3D tracks
   3.3 Delete poor tracks
   3.3 Associate report to Un-Used data
   3.4 Initiate new 3D tracks
   3.5 Estimate trajectory parameters

   end

   end

3. print results

Figure 5: Algorithm CMTT-SSDP

Crystal_Router Algorithm
/* Standard Crystal_Router algorithm used for distribution of track file /

- **For** each channel I in cube, I = 1, log_2N
  /* N = number of nodes in a cube topology */

  begin

  - **For** each track in the local TF **Do**
    /* TF denotes the track file */

    begin

    if $d_I <> p_I$
    /* d_I (p_I) denotes the I^{th} bit in the address of destination node(current processor) */
    then exchange the tracks between the two nodes connected to channel(I)

    end

  end

Figure 6: Algorithm Crystal_Router

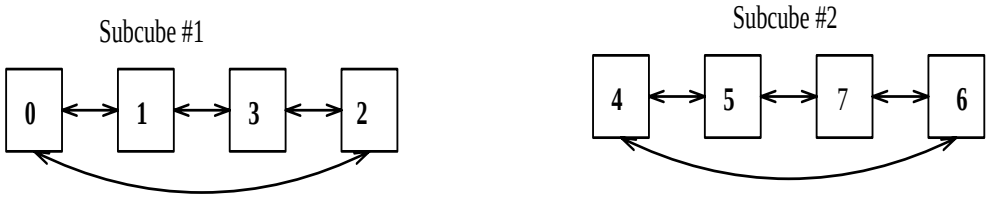

Figure 7: Ring topology in 2D tracking

the track file among all nodes involved in the parallel computation in log_2N steps (where N is number of processors).

3.2 Concurrent MTT with parallel sensor data processing (CMTT-PSDP)

In CMTT-SSDP algorithm, the Do loop (for sensor 1 and sensor 2) in step 2.1 is performed sequentially i.e. first we do 2D tracking for sensor 1 and then perform 2D tracking for sensor 2. In this implementation redistribution of track file (step 2.1.3 in Algorithm CMTT-SSDP) is done between all the nodes in the cube, for both sensor 1 and sensor 2. The performance of CMTT can be improved by overlapping communication and execution. In this case the 2D tracking of the two sensors data is performed concurrently. As a result of processing the sensor data in parallel, the redistribution is done only between half of the nodes working on same sensor data as shown in the Figure **??**. This reduces the redistribution time considerably. However the 3D tracking is done on all nodes/processors.

Figure **??** shows the main step of the CMTT-PSDP algorithm.

Algorithm CMTT-PSDP

1. Initialization
   /* initialize the parameters of sensors and targets */
2. **For** I= 1,NO_SCAN **Do**

   begin

   (a) Partition processors into NO_SENSORS subcubes

   2.2. 2D Tracking

   If processor ID modulo NO_SENSORS = j

   then perform 2D tracking for sensor data J
   /* similar to Algorithm CMTT-SSDP step 2.1 */

   2.3. Exchange the 2D results between processor working on different sensors

   2.4. Initialize the cube of N processors

   2.5. Perform 3D tracking as in Algorithm CMTT-SSDP step 2.2

   end

3. print results

Figure 8: Algorithm CMTT-PSDP

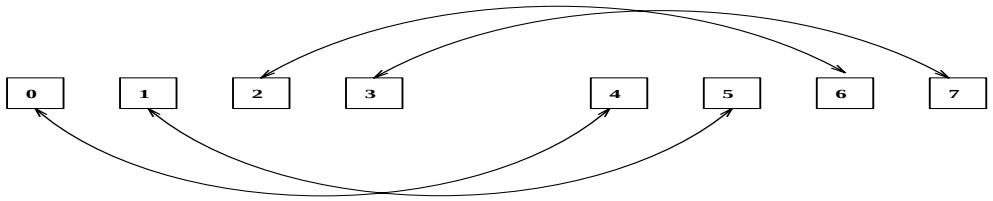

Figure 9: Data exchange after 2D tracking

In Algorithm CMTT-PSDP, after concurrent 2D tracking of both sensors, they must communicate the results with each other before 3D tracking can be initiated. After 2D tracking each node in same subcube has completed track and report file for the sensor data assigned to this subcube. Hence, instead of one processor sending results to every node, the communication occurs only between corresponding nodes in both subcubes (see Figure **??**). This allows to overlap the communication between nodes and thus reduces its overhead. This exchange of results constitute the extra overhead due to our new approach. But this extra overhead is insignificant when compared to the performance gained from overlapping the communication during track file redistribution.

After communicating the results, we reinitialize the cube environment to form one cube. The 3D tracking proceeds as in Algorithm CMTT-SSDP. We did not attempt to improve the performance of the 3D tracking because its execution time can be ignored when compared to the 2D execution time of the CMTT algorithm.

4 Experimental Results

In this section, we benchmark the implementation of the CMTT using different parallel/distributed tools.

The main objective of this experimentation is to understand the issues related to porting compute intensive applications (with more then 32,000 lines of code) on parallel and distributed systems. Furthermore, we do need to determine the ideal problem size and type of platform (parallel or distributed computing environment). We also need to benchmark the latency and overhead associated with each tool. We benchmark the tools and the CMTT system on two classes of computing environments: Distributed Computing Environment(SUN, IBM RS6000, IBM-SP1[2] and Parallel Computing Environment(CM5, iPSC 860).

4.1 Tool Benchmarking

In this subsection, we evaluate the performance of three important communication primitives:

1. Send/Receive : We measure the elapsed time for a round trip delay associated with sending and receiving a message between two adjacent nodes.

2. Broadcast : We measure the elapsed time between broadcasting a message to all nodes and until the broadcasting node receives reply from all the nodes.

3. Loop : We measure the elapsed time from the instant of sending a message from one node until getting back the same message again after passing through all nodes.

Figures **??**- **??** show the performance results of running these three communication primitives of PVM, p4 and HORUS on a cluster of four SUN SPARC workstations. Figures **??**- **??** show the performance results of these three communication primitives when they run on four IBM RS6000 workstations. The performance results were evaluated by changing the

[2]Current configuration of IBM-SP1 uses dedicated Ethernet for interprocessor communication. The IBM-SP1 high speed switch was not available during this benchmarking

message size between 1K to 100K bytes. For short messages (between 1K and 5K), all tools gave comparable results. However, for larger messages, PVM tool seems to outperform other tools on both platforms (SUN SPARC and IBM RS6000). Tables ?? and ?? summarize the results of tool benchmarking. These results are consistent to those resulted from benchmarking the CMTT applications on parallel and distributed systems. Consequently, tool evaluation can be used as an indication of how complex applications are going to perform on different parallel/distributed platforms.

Order	Send/Receive	Broadcast	Loop
1	PVM	PVM	PVM
2	p4	HORUS	p4
3	HORUS	p4	HORUS

Table 1: Summary of Primitive Performance of Tools (on 4 nodes of SUN SPARC)

Order	Send/Receive	Broadcast	Loop
1	PVM	p4	PVM
2	p4	PVM	p4
3	EXPRESS	EXPRESS	EXPRESS

Table 2: Summary of Primitive Performance of Tools (on 4 nodes of IBM RS6000)

4.2 Benchmarking CMTT on Cluster of Workstations

On a distributed computing environment, the performance of the CMTT has been improved by increasing the number of processors upto a certain threshold, after that the performance starts deteriorating. The optimal system configuration is to use a number of processors that gives minimum execution time. From Figures ??- ??, the execution time deteriorates after two nodes and four nodes for CMTT-SSDP and CMTT-PSDP algorithms, respectively. It is clear from these figures that CMTT-PSDP performs much better than CMTT-SSDP because of reducing the communication time associated with redistribution of the track file.

In terms of platforms, IBM-SP1 out performed other distributed computing environments (SUN SPARC, IBM RS6000 and heterogeneous environment of SUN SPARC and IBM RS6000). Furthermore, the performance of IBM-SP1 will even be better when the high speed interprocessor switch will be in place. For example, CMTT-PSDP implemented using PVM took 23.16 seconds on IBM-SP1 with four processors, whereas it took 65.56 seconds on four SUN SPARC workstations, 60.10 seconds on four IBM RS6000 workstations and 63.51 seconds on heterogeneous environment of two SUN SPARC and two IBM RS6000 workstations. Furthermore, the PVM implementations outperformed other tools. However, for a small number of processors (say 2), the difference between tools is insignificant, while it is large for four or more processors.

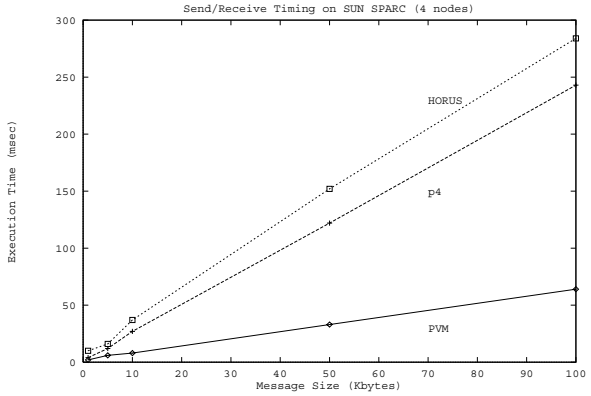

Figure 10: Send/Receive Performance of Each Tool on SUN SPARC (4 nodes)

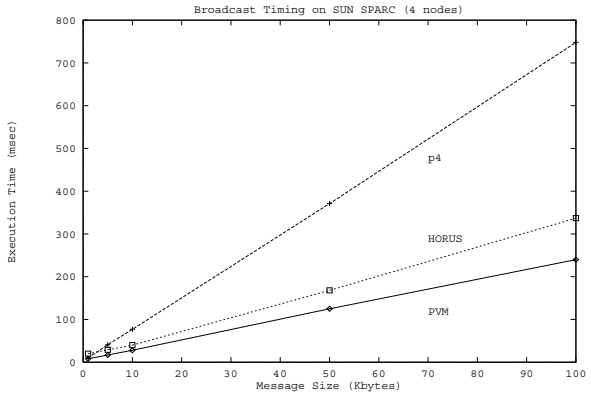

Figure 11: Broadcast Performance of Each Tool on SUN SPARC (4 nodes)

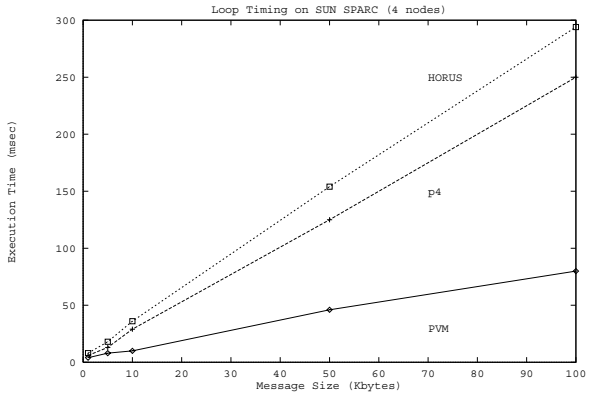

Figure 12: Loop Performance of Each Tool on SUN SPARC (4 nodes)

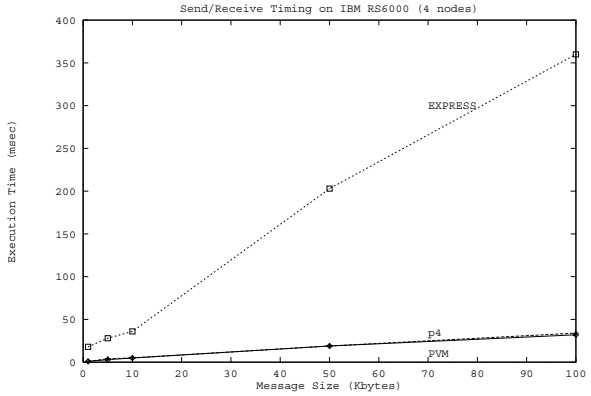

Figure 13: Send/Receive Performance of Each Tool on IBM RS6000 (4 nodes)

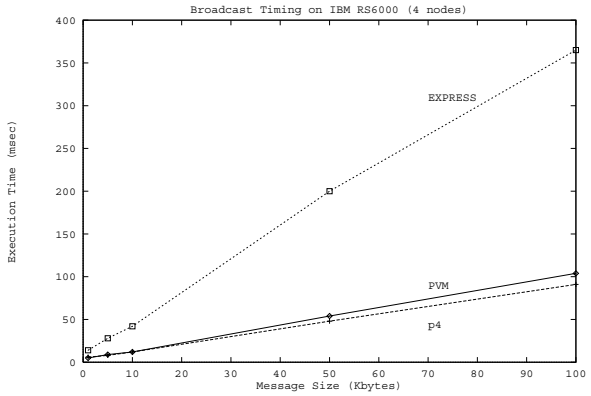

Figure 14: Broadcast Performance of Each Tool on IBM RS6000 (4 nodes)

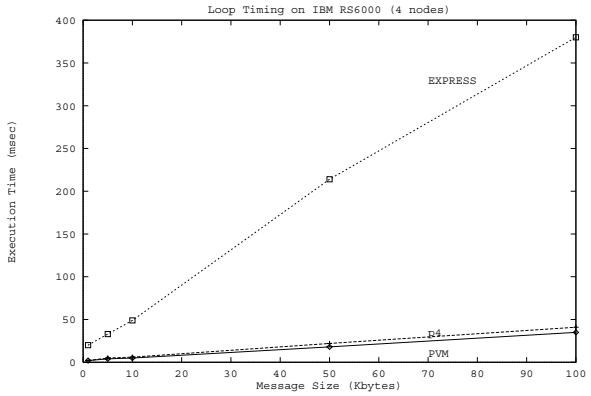

Figure 15: Loop Performance of Each Tool on IBM RS6000 (4 nodes)

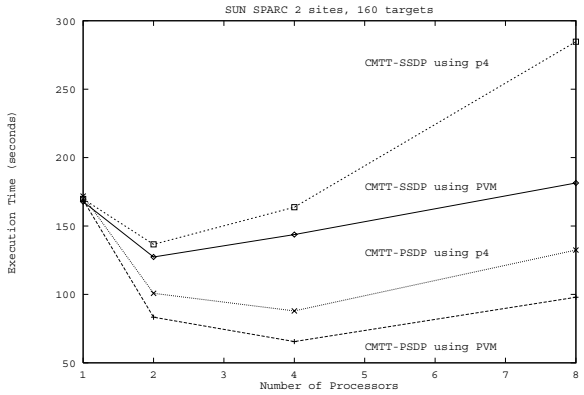

Figure 16: Performance Result of CMTT on SUN SPARC implemented with p4 and PVM

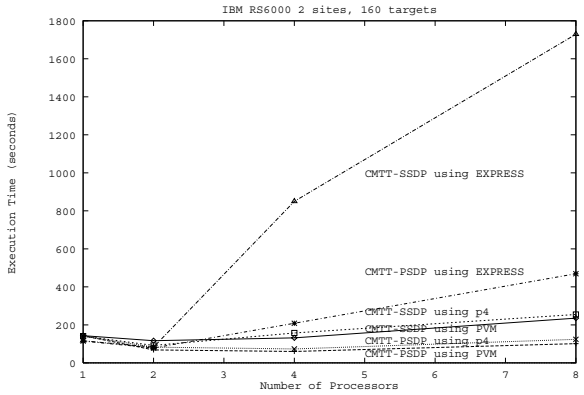

Figure 17: Performance Result of CMTT on IBM RS6000 implemented with p4, PVM and EXPRESS

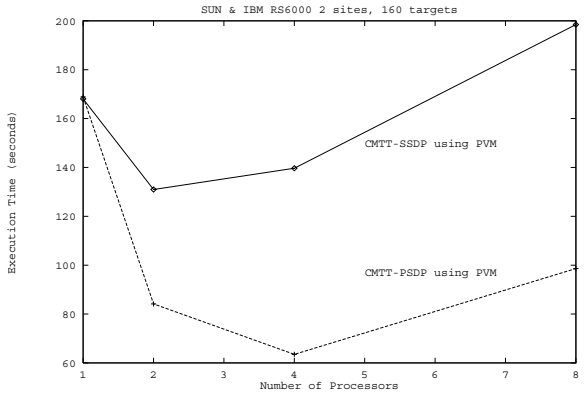

Figure 18: Performance Result of CMTT on SUN SPARC and IBM RS6000 implemented with PVM

Order	1 Node	2 Node	4 Node	8 Node
1	IBM-SP1 (PVM)	IBM-SP1 (PVM)	IBM-SP1 (PVM)	iPSC 860 (EXPRESS)
2	IBM-SP1 (p4)	IBM-SP1 (p4)	IBM-SP1 (p4)	IBM-SP1 (PVM)
3	iPSC 860 (EXPRESS)	iPSC 860 (EXPRESS)	iPSC 860 (EXPRESS)	IBM-SP1 (p4)
4	IBM RS6K (EXPRESS)	IBM RS6K (PVM)	CM5 (PVM)	CM5 (PVM)
5	IBM RS6K (p4)	IBM RS6K (EXPRESS)	IBM RS6K (PVM)	SUN SPARC (PVM)
6	IBM RS6K (PVM)	IBM RS6K (p4)	SUN&RS6K (PVM)	SUN&RS6K (PVM)
7	SUN&RS6K (PVM)	SUN SPARC (PVM)	SUN SPARC (PVM)	IBM RS6K (PVM)
8	SUN SPARC (PVM)	SUN&RS6K (PVM)	IBM RS6K (p4)	IBM RS6K (p4)
9	SUN SPARC (p4)	CM5 (PVM)	SUN SPARC (p4)	SUN SPARC (p4)
10	CM5 (PVM)	SUN SPARC (p4)	IBM RS6K (EXPRESS)	IBM RS6K (EXPRESS)

Table 3: Summary of Performance Result of CMTT-PSDP Algorithm

4.3 Benchmarking CMTT on Parallel Computers

When we implemented the CMTT system on parallel computers, we obtained consistent results with those of distributed computing environment; CMTT-PSDP version outperforms CMTT-SSDP version. Also the execution time reduces up to four nodes in CMTT-SSDP version and up to eight nodes in CMTT-PSDP version. Thus, parallel computing environment works fine for larger number of processors because the communication latency is less than that of Ethernet. For example, CMTT-PSDP version using EXPRESS took 34.41 seconds on eight processors of iPSC 860, whereas CMTT-PSDP version using PVM took 37.57 seconds on eight processors of IBM-SP1. When we compare the performance of the tracker on iPSC 860 and CM5, we found that iPSC 860 implementation using EXPRESS performs better than CM5 using PVM.

4.4 Discussion

We have shown the benchmarking results of the tracker implementations on distributed computing and parallel computing environments. For a small number of processors, distributed computing environment works better because of the higher execution rate of each workstation, but can not outperform parallel computing environment for a large number of processors because of the network latency associated with distributed computing environment. Table ?? orders the different implementations of the tracker on parallel/distributed systems based on their performance and for different number of nodes. The Overall performance of CMTT shows the viability of distributed computing environment to achieve supercomputer performance (e.g. IBM-SP1 outperformed both iPSC 860 & CM5 for 2-4 nodes). Even higher performance can be achieved if faster networks are available. We are currently benchmarking the implementation of the tracker on a cluster of workstations interconnected by FDDI and

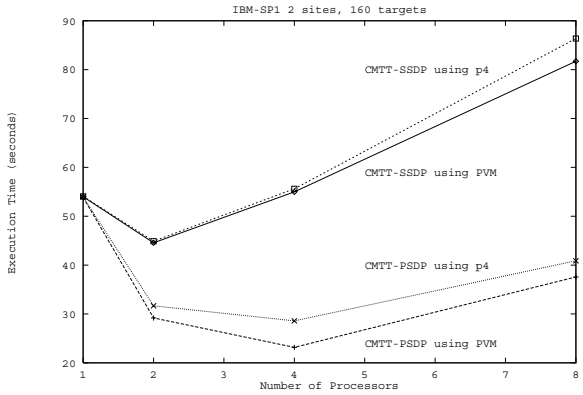

Figure 19: Performance Result of CMTT on IBM-SP1 implemented with p4 and PVM

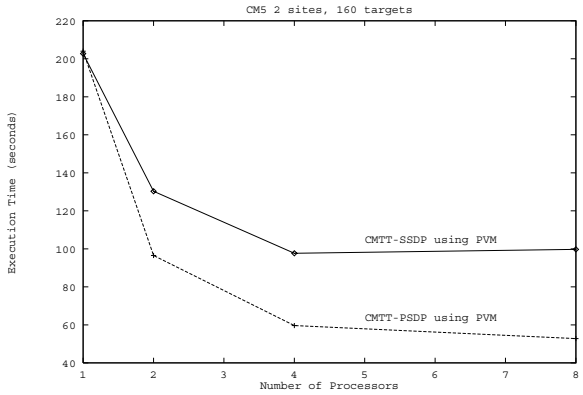

Figure 20: Performance Result of CMTT on CM5 implemented with PVM

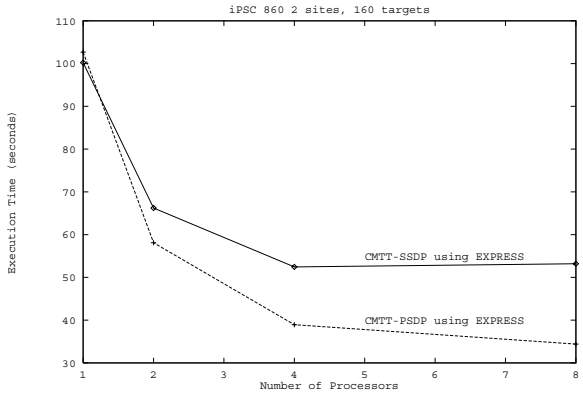

Figure 21: Performance Result of CMTT on iPSC 860 implemented with EXPRESS

an ATM network.

5 Conclusions

In this paper we have benchmarked different implementations of the concurrent multi target tracker on parallel and distributed systems. A uniform structure of tracker was developed to make it easily portable to different message passing tools like Express, PVM, PICL, EXPRESS and p4. The tracker was implemented on nCUBE, iPSC 860, CM5, IBM-SP1, cluster of SUN SPARC workstations and cluster of IBM RS6000 workstations. We also developed a more efficient algorithm for concurrent multi target tracker by overlapping the processing and the communication of sensors data.

In this paper, the distributed computing environment used Ethernet for interprocessor communication, which is quite slow. We are now studying the performance of the tracker on a cluster of Alpha workstations connected by an FDDI network and on another cluster of workstations interconnected by ATM network. These networks were not up running when the paper was written. Furthermore, the results of our performance analysis, will be used to quantify the performance metrics of software tools that will be used in the evaluation methodology currently being developed at Syracuse University.

References

[1] T. D. Gottschalk., "CALTRAX The Tracking Program for Simulation 87", California Institute of Technology, Pasadena, California 91125, Caltech Report C3P-478.

[2] Paul Messina., "Performance Study of Missile Tracking Algorithm on Selected Computer Architectures", California Institute of Technology, Pasadena, California 91125, Clatech Report C3P-668. 87", California Institute of Technology, pasadena, California 91125, Caltech Report C3P-478. Institute of Technology, Pasadena, CA 91125, July 1989. Caltech Report.

[3] T. D. Gottschalk., "Concurrent Multi-Target Tracking", California Institute of Technology, Pasadena, CA 91125, July 1989. Caltech Report.

[4] T. D. Gottschalk, "Precision Filters For Boost Phase Tracking", Caltech report C^3P-479(1987).

[5] Geoffrey C. Fox., David W. Walker., "A Portable Programming Environment for Multiprocessors", California Institute of Technology, Pasadena, CA 91125, Caltech Report C3P-496.

[6] Paul Messina., Arnold Alagar., Clive Ballie., Edward Felten., Paul Hipes., ANke Kamrath., Robert Leary., Wayne Pfeiffer., Jack Rogers., David Walker., Roy Williams., "Benchmarking Advanced Architecture Computers", Caltech Supercomputing Facility, San Diego Super Computing Center, Department of Mathematics, University of South Carolina, Caltech Report, C3P712.

[7] G. C. Fox, W. Furmanski, "Communications Algorithms for Regular Convolutions on the Hypercube", Caltech report C^3P-329(1986).

[8] Geoffrey C. Fox., Mark A. Johnson., Gregory A. lyzenga., Steve W. Otto., John K. Salmon., David W. Walker., "Solving Problems on Concurrent Processors", New Jersey : Prentice Hall, 1988.

[9] Salim Hariri, Geoffrey C. Fox, Balaji Thiagarajan, Manish Parashar, "Parallel Software Benchmark for BMC^3/IS Systems", Northeast Parallel Architectures Center, 111 College Place, Syracuse University, Syracuse, NY 13244-4100.

[10] Adam Beguelin., Jack Dongara., Al Geist., Robert Manchek., Vaidy Sunderam., "User Guide to PVM", Oak Ridge National Laboratory, Oak Ridge TN 37831-6367 and Department of Mathematics and Computer Science, Emory Unive rsity, February 1993.

[11] R.Olson.,"Parallel Processing in a Message Based Opearting System", IEEE Software, July 1985.

[12] D.Reed and D.Grunwald, "The performance of multicomputer interconnection network", IEEE Computer, June 1987.

[13] Adam Beguelin, Jack Dongara, Al Geist, Robert Manchek , and Vaidy Sunderam, "User Guide to PVM", Oak Ridge National Laboratory, Oak Ridge TN 378 31-6367 and Department of Mathematics and Computer Science, Emory University, February 1993.

[14] Ralph Butler, and Ewing Lusk, "User's Guide to the p4 Programming System", Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439-4801

[15] Parasoft Corporation, "Express 3.0 Documentation", Parasoft Corporation, 2500, E.Foothill Blvd. Pasadena, CA 91107.

[16] G. A. Geist, M.T. Heath, B.W. Peyton, and P.H. Worley, "User Guide to PICL", Mathematical Sciences Section, P.O. Box 2009, Bldg. 9207-A, Oak Ridge National Laboratory, Oak Ridge, TN 37831-8083, August 1990.