1994-03

Improvement of Janus using 1-meter resolution database with a transputer network

Dundar, Cem Ali

Monterey, California. Naval Postgraduate School

https://hdl.handle.net/10945/30899

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

IMPROVEMENT OF JANUS USING PEGASUS 1-METER RESOLUTION DATABASE WITH A TRANSPUTER NETWORK

by

Cem Ali Dündar

March 1994

Thesis Advisor: Se-Hung Kwak

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE March 1994	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE
Improvement Of Janus Using Pegasus 1-meter Resolution Database With A Transputer Network(U)

5. FUNDING NUMBERS

6. AUTHOR(S)
Dündar, Cem Ali

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
Line-of-sight (LOS) calculation for the Janus combat simulation model is critical to the processes being simulated and impacts the run speed (ratio of game time to real time), since it may be the single most computationally expensive algorithm in simulation.

This thesis presents design and implementation of a transputer network with the purpose of providing an efficient LOS calculation in a distributed memory and computing environment. The approach taken was to use a processor farming method to speed up the LOS calculation. The programs were implemented on a network of 15 transputers using 3L Parallel C++ (version 2.1.1) programming language. A 1-meter resolution terrain database of Fort Hunter Liggett, California was used to get more reliable LOS results.

Expected gain of our system was 3.873 ($\sqrt{15}$). After timing tests, we found that we could speed up the LOS calculation by a factor of 2.581 when comparing the 15 transputer configuration to a conventional processor which is equivalent to a single transputer. The difference between expected gain and our actual gain was found to be the communication overhead in the network of transputers. We stated that further significant improvements can be provided by using our approach with more memory and faster transputers.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Janus, Transputer, Pegasus Database, Parallelism, Line-of-sight	225
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

Approved for public release; distribution is unlimited

IMPROVEMENT OF JANUS USING 1-METER RESOLUTION DATABASE WITH A TRANSPUTER NETWORK

by

Cem Ali Dündar

LTJG. Turkish Navy

BS, Turkish Naval Academy, 1988

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

March 1994

Author: [redacted]

Cem Ali Dündar

Approved By: [redacted]

Se-Hung Kwak, Thesis Advisor

[redacted]

Eugene P. Paulo, Second Reader

[redacted]

Ted Lewis, Chairman,

Department of Computer Science

ABSTRACT

Line-of-sight (LOS) calculation for the Janus combat simulation model is critical to the processes being simulated and impacts the run speed (ratio of game time to real time), since it may be the single most computationally expensive algorithm in simulation.

This thesis presents design and implementation of a transputer network with the purpose of providing an efficient LOS calculation in a distributed memory and computing environment. The approach taken was to use a processor farming method to speed up the LOS calculation. The programs were implemented on a network of 15 transputers using 3L Parallel C++ (version 2.1.1) programming language. A 1-meter resolution terrain database of Fort Hunter Liggett, California was used to get more reliable LOS results.

Expected gain of our system was 3.873 ($\sqrt{15}$). After timing tests, we found that we could speed up the LOS calculation by a factor of 2.581 when comparing the 15 transputer configuration to a conventional processor which is equivalent to a single transputer. The difference between expected gain and actual gain was found to be the communication overhead in the network of transputers. We stated that further significant improvements can be provided by using our approach with more memory and faster transputers.

THESIS DISCLAIMER

Many terms used in this thesis are registered trademarks of commercial products. Rather than attempting to cite each individual occurence of a trademark, all registered trademarks appearing in this thesis are listed below the firm holding the trademark:

<u>INMOS Limited, Bristol, United Kingdom</u>:

inmos

IMS

occam

<u>International Business Machines Corporation:</u>

IBM

<u>3L Ltd.:</u>

3L

<u>Digital Equipment Corporation:</u>

AXP

DECchip

<u>Perihelon Software Ltd.:</u>

Helios

ACKNOWLEDGEMENTS

I would like to thank Dr. Se-Hung Kwak, whose interest in the subject of parallel computing with transputers was the foundation on which this thesis was produced. His continued support, enthusiasm, patience, and guidance were invaluable assets for the completion of this work.

I also would like to thank Major Eugene P. Paulo, for his helps and supports everytime we needed to coordinate with TRAC MTRY group during my thesis work.

TABLE OF CONTENTS

DEDICATION

I dedicate my thesis to my parents Nimet and Ahmet Dündar who were my first teachers and of whom I'm very proud to be their son.

I. INTRODUCTION

A. BACKGROUND

1. Janus

The Janus simulation was fielded in 1978 [Ref. 1]. It was developed as a nuclear effects modeling tool by Lawrence Livermore National Laboratories and became known as Janus(L). TRADOC Analysis Command (TRAC) at White Sands Missile Range (WSMR) modified Janus(L) to meet Army combat development needs. The modified Janus(L) model became known as Janus(T). The Army realized the value of the system for use in the training arena, and tasked TRAC-WSMR with developing a multipurpose system from the best of Janus(L) and Janus(T), which was termed Janus(A). Through enhancements and upgrades, Janus(A) has reached a version level of 4.0 as of January 1994.

The Janus model simulates battle between Blue and Red units. It supports conflict from individual systems and company-sized units through brigade/regimental-sized units. It is an interactive, two-sided, closed, stochastic, ground combat simulation featuring precise color graphics. Janus is "interactive" in that the command and control functions are entered on workstations by military analysts who decide what to do in crucial situations during simulated combat. "Two-sided" refers to the two opposing forces, blue and red, directed simultaneously by two sets of players. "Closed" means that the disposition of opposing forces is largely unknown to the players in control of the other force. "Stochastic" refers to the way the system determines the results of actions such as direct fire engagements; according to the laws of probability and chance. "Ground combat" means that the principal focus is on ground maneuver and artillery units, although Janus also models weather and its effects, day and night visibility, engineer support, minefield employment and breaching, rotary and fixed wing aircraft, resupply and a chemical environment. Janus is an event-driven simulation.

2. The Transputer

The term "transputer" is an acronym for "transistor computer" where it reflects the ability of this device to be used as a system's building block, much like the transistor was in the past [Ref. 2]. The nice feature of the transputer is that it adds a new level of abstraction, which provides a very simple way to design a concurrent system. As a formal definition we could state that the transputer is a single-chip microcomputer that has its own local memory and four communication links. The links may be thought as of as small special purpose processors which steal no cycles from the main CPU, in such a way that we could have all four links and the CPU working at the same time, without degrading the performance of the program's execution.

The transputer is a parallel microprocessor, generally categorized as a Multiple Instruction Multiple Data (MIMD) computer [Ref. 3] [Ref. 4:pp. 498-500]. This means that transputers are used to execute different operations on separate data at the same time. This is somewhat like a football team where individual players execute their own special assignments together during a play. A transputer operates as a stand-alone machine or as a processing element interconnected by their links to form computing arrays and networks. Modular design enables transputers to be used together in arbitrary numbers to support a broad range of applications, and the inherit redundancy of multiprocessing can be utilized for fault tolerance.

B. SCOPE OF THESIS

Line-of-sight (LOS) is a central process in combat simulations that works at item level. The LOS algorithm is critical to the processes being simulated and impacts the run speed (ratio of game time to real time), since it may be the single most computationally expensive algorithm in the simulation.

This study is focused specifically on the following two objectives:

1. To implement an efficient calculation of LOS in a distributed memory environment by using transputers and 1-meter resolution terrain database.

2. To show that the usage of 1-meter resolution terrain database for LOS calculation purposes gives more precise and reliable results than the current 50 or 100-meter resolution terrain databases.

C. THESIS ORGANIZATION

This thesis is presented in six chapters and three appendices.

Chapter I is the introduction to the problem and the background for Janus combat simulation system and the transputer.

Chapter II describes the current issues about parallel computing with transputers.

Chapter III presents a detailed problem statement for this thesis. The current issues about Janus which are PEGASUS terrain database organization and the algorithm for LOS calculation are described in this chapter.

Chapter IV describes the transputer implementation of LOS calculation in both hardware and software aspects.

Chapter V presents the experimental results of the transputer implementation of LOS calculation.

Chapter VI states the conclusions and recommendations for further research.

Appendix A includes the Sun SPARC Station source code.

Appendix B includes the Host Computer (PC) source code.

Appendix C includes the source code for reading terrain data from Pegasus Database.

II. TRANSPUTERS AND PARALLEL COMPUTING

A. PARALLELISM

In the first computing wave, scientific and business computers were more or less identical as they were all big and slow [Ref. 6:p. 1]. Even the early electronic computers were not very fast. This was the "prehistory of computing", where computing had to be employed at any cost.

The second and third waves brought on mainframes, minis, and finally micros. This diversity of computing caused a number of niches to develop which broadened and deepened the computer industry. Scientific and business computing went their separate ways, and there seemed to be a computer in just about everyone's price range.

But the original power users who pioneered computing continued to emphasize speed above all else. Single-processor supercomputers achieved unheard of speeds beyond 100 million instructions per second, and pushed hardware technology to the physical limits of chip building. But soon this trend will come an end, because there are physical and architectural bounds which limit the computational power that can be achieved with a single-processor system.

We are now enjoying the Parallel Wave [Ref. 6:pp. 1-5] of computing, where performance is enhanced by using multiple processors. Parallelism is the process of performing tasks concurrently. It has been touted as a solution to the problem of making computers faster and faster. When the physical limits for single-processor systems are reached, parallelism will be the only course. However, even before the speed limit is reached, there is an economic motivation to use parallel processing in place of faster and more expensive single-processor systems. Indeed, the economic advantage of low-cost, multiple processing systems was realized in the mid-1980s. Hence, the 1990s were poised for the decade of parallelism simply due to economic forces.

Many parallel architectures have been discussed in the past, and there are several superminicomputer parallel systems available today. However, most of these are unable to provide the very wide range of price/performance that parallel processing promises and that transputer-based systems can provide [Ref. 5].

To understand this, it is worth examining the normal approach to parallel systems design. Most parallel systems are constructed by connecting up multiple computers with a single high speed bus. A simplified system can be imagined, consisting of multiple processors sharing a single global memory accessed via a single high performance bus.

This shape of system will provide very disappointing results for obvious reasons; a processor can only access memory when no other processor is accessing memory. With high performance processors, this will provide an upper limit of perhaps two or three processors before performance stops increasing. It is possible to speed the system up, but only by use of memory that is very much faster than the processors. This is expensive.

In more realistic system each processor has some private, local memory in addition to bus access to global memory. The local memory could be organized as either a private address space, or a sufficiently large cache. Now, it is possible to imagine a system where a processor spends perhaps 90% of its time accessing local memory and only 10% accessing the shared store. Then with reasonably-priced memory it should be possible to build a computer which can use perhaps twenty or thirty processors before saturating.

The bottleneck in this system is the shared resource, either the bus or the memory. The bus itself is a poor choice for interconnect in any case; not only does its logical performance degrade as more processors contend for it, the extra electrical loads imposed by adding processors to the bus either slow the system down as more machines are added, or set a much lower bandwidth on the bus for lower processor counts.

Whichever is the bottleneck at present, the apparently inexorable improvement in semiconductor technology will arrange for it to be the bus since affordable memory and processor speeds are increasing faster than improved backplane technologies. As a result, this sort of system is guaranteed non-future proof; as device speeds increase, the system performance flattens out since the maximum number of processors usable before bus saturation reduces with time.

The system architecture can be changed slightly to remove the straitjacket imposed by the bus. An obvious improvement is to use multiple buses, probably arranged in some regular, structured manner, like a hierarchy. Now, clusters of computers, each with its own local memory, share some cluster memory via a cluster bus. Clusters are connected by other buses; these buses themselves can have memory. Then, assuming that 90% of accesses are local, and that 90% of the non-local accesses are to the local cluster shared memory, the earlier arguments suggest that for a well-behaved problem, a twenty cluster system could be built, with each cluster having twenty processors.

This solution should work for a range of applications, but the amount of logic and interconnect needed to implement it makes it expensive. It has another problem, too; while it is an acceptable architecture for a single, centralized computer, shared buses do not seem to be an appropriate paradigm for distributed parallel systems.

These criticisms can be resolved by a small change in attitude to the system architecture, and then a re-implementation. Assume that the system is an actual parallel computing system, rather than just a collection of computers each with access to some shared system resource; then the processors must be interacting with one another. Each will be working on a portion of the problem, and will interchange partial results with other processors as they jointly progress toward completing the program. To do this, each machine will likely provide the equivalent of mailboxes, where the other processors can leave their own results and their requests for information.

But if the processors are cooperating by exchanging messages, then there is no need to use shared memory to implement the communication. Instead, direct interprocessor data transfer channels can be used to Direct Memory Access (DMA) [Ref. 4:pp. 297-301] information from one processor to another. Given such a mechanism, we cure several problems at once: as we add processors, we add interprocessor bandwidth; the processors do not need to be physically located together, and so can be components of a distributed system without necessarily altering the system design or software; and the cost of the interprocessor hardware can be much reduced from bus costs (since, for example, there is no need for an address, we can save by not having address lines; since there is exactly one destination for each driver, the electrical design is simpler).

This is the system architecture chosen for the transputer. Each transputer comes with one or more interprocessor links, each one DMA-driven to ensure that communication can take place in parallel with computation. Transputers further reduce system cost by using serial interconnect; minimizing pin count reduces transputer cost and interconnect cost, eases board layout and minimizes power consumption.

B. THE INMOS TRANSPUTER

The transputer [Ref. 7:pp. 7-30] was developed by INMOS Limited of Bristol, United Kingdom, and has since expanded into a family of very large scale integrated (VLSI) components with different capabilities. Since the transputer is a component designed to exploit the potential of VLSI, that technology allows large numbers of identical devices to be manufactured cheaply. For this reason, it is attractive to implement a concurrent system using a number of identical components, each of which is customized by an appropriate program. The revolutionary architecture of the transputer enables the potential of concurrency to be realized for the first time, making today's applications easier to implement and creating a new dimension for tomorrow's systems.

The transputer uses silicon capability to make programming simpler and to make engineering easier than for any previous microprocessor. The architecture has been optimized to obtain the maximum of functionality for the minimum of silicon. It allows different trade offs between performance and cost, always giving an intrinsic advantage over older architectures. **The architecture is future-proof.** It spans the range of applications from microcontrollers to supercomputers. Transputers will exploit future levels of integration by increasing the amount of processing, memory, communications and concurrency within the same architecture.

A typical member of the transputer family is a single chip containing processor, memory, and communication links which provide point to point connection between transputers. The transputer provides a direct implementation of the process model of computing. A process is an independent computation, with its own program and data, which can communicate with other processes executing at the same time. Communication is by message passing, using explicitly defined channels.

The transputer is designed so that it can implement a set of concurrent processes. Special instructions share the processor time between the concurrent processes and perform interprocess communication.

In addition, the transputer is designed so that its external behavior corresponds to the formal model of a process. As a consequence, it is possible to program systems containing multiple interconnected transputers in which each transputer implements a set of processes. Since a program is defined as a set of processes, it can be mapped onto such a system in a variety of ways, such as minimizing cost, or optimizing throughput, or maximizing the responsiveness to specific events.

The transputer specifically implements the concept of communicating sequential processes (CSP) defined by C.A.R. Hoare [Ref. 8] and to be used as a building block for distributed computing systems. The CSP concept describes the interactions between programs that execute in parallel.

1. Communicating Sequential Processes

Hoare's Communicating Sequential Processes (CSP) is one model for concurrent or parallel programming, and it is central to the design of the transputer. In CSP, a program is a collection of processes which can be combined to execute sequentially on a single processor or in parallel on multiple processors.The data space for any process executing in parallel is disjoint, thus alleviating the need for sharing memory between processors. Although shared memory is not available, processes must still communicate with each other. Therefore, CSP utilizes message passing between any pair of parallel processes via declared communication channels between two processes.

In order for the concurrent processes to communicate, message passing must be synchronized. Such communication occurs when one process names another as destination for output and the second process names the first as source for input. This allows the value to be output by the source process to be copied into the destination process. Note that the synchronization imposes a requirement that an output (input) command must be delayed until the corresponding input (output) command in the other process is ready to be executed.

2. Transputer Architecture

Several versions of the transputer are currently available. This thesis considers transputer types IMS T800 and IMS T805[1]. The following sections describe the features of an IMS T800 20MHz transputer. A complete description of all currently available transputers can be found in [Ref. 7] and [Ref. 9]. A block diagram of an IMS T800 transputer is shown in Figure 2.1.

a. Overall

The IMS T800 is a 64 bit floating point member of a family of transputers, all which are consistent with the INMOS transputer architecture. It integrates a 32 bit microprocessor, a 64 bit floating point unit, four standard transputer communication links, 4Kbytes on-chip RAM for high speed processing, a configurable memory interface and peripheral interfacing on a single chip, using a 1.5 micron CMOS process.

1. T805 is a new version of T800. They are essentially same processors and our lab has a mixture of T800 and T805 transputers.

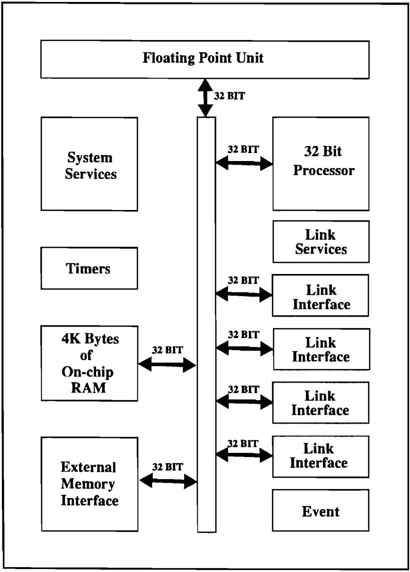

Figure 2.1: IMS T800 Block Diagram of the 32-bit Transputer

b. Central Processor

The 32 bit processor provides 10 MIPs performance. The design achieves compact programs, efficient high level language implementation and provides direct support for the occam (a programming language that will be mentioned later) model of concurrency. Procedure calls, process switching and interrupt latency are all sub-microsecond. The processor shares its time between any number of concurrent processes. A process waiting for communication or a timer does not consume any processor time. Two levels of process priority enable fast interrupt response to be achieved.

c. Floating Point Unit

The 64 bit floating point unit provides single length and double length operation according to the ANSI-IEEE 754-1985 standard for floating point arithmetic and able to perform floating point arithmetic operations concurrently with the processor; sustaining in excess of 1.5 Mega Flops.

The floating point unit (FPU) on the T800 consists of a microcoded computing engine which operates concurrently with and under the control of the Central Processing Unit (CPU). It contains a three deep floating point evaluation stack on which floating point numbers, represented in the IEEE format can be manipulated. All data communication between memory and the floating point unit is done under the control of the CPU.

d. Memory System

The 4Kbytes of on chip static RAM provide a maximum data rate of 80 Mbytes/sec with access for both the processor and links. The IMS T800 can directly access a linear space up to 4 Gbytes. The 32 bit wide external memory interface uses multiplexed data and address lines provides a data rate up to 26.6 Mbytes/sec. A configurable memory controller provides all timing, control and DRAM refresh signals for a wide variety of memory systems. Internal and external memory appear as a single continuous address space.

e. Links

The IMS T800 uses a DMA block transfer mechanism to transfer messages between memory and another transputer product via the INMOS links. The link interfaces and the processor all operate concurrently, allowing processing to continue while data is being transferred on all of the links.

The four standard INMOS serial links on the IMS T800 give a unidirectional transmitted data rate of 1.7 Mbytes/sec and a combined (bidirectional) data rate per link of 2.3 Mbytes/sec, at a link speed of 20 Mbits/sec. Link speeds of 10 Mbits/sec and a 5 Mbits/sec are also available on the IMS T800 making the device compatible with all other INMOS transputer products.

f. Peripheral Interface

The memory controller supports memory mapped peripherals, which may use DMA. Links may be interfaced to peripherals via an INMOS link adaptor. A peripheral can request attention via the event pin.

g. Error Handling

High-level language execution is made secure with array bounds checking, arithmetic overflow detection etc. A flag is set when an error is detected. The error can be handled internally by software or externally by sensing the error pin.

h. Programming IMS T800

The IMS T800 transputer can be programmed in several languages including Occam, C, C++, Ada, Fortran and Pascal.

i. Processes And Concurrency

The transputer provides direct support for concurrency. It has a microcoded scheduler which enables any number of concurrent processes to be executed together, sharing the processing time. This removes the need for a software kernel.

A process starts, performs a number of actions, and then either stops without completing or terminates complete. Typically, a process is a sequence of instructions. A transputer can run several processes concurrently[2]. Processes may be assigned either high or low priority, and there may be any number of each.

At any time, a concurrent process may be

Active	- Being executed
	- On a list waiting to be executed.
Inactive	- Ready to input
	- Ready to output
	- Waiting until a specified time.

The scheduler operates in such a way that inactive processes do not consume any processor time. It allocates a portion of the processor's time to each process in turn. Each process runs until it has completed its action, but is descheduled while waiting for communication from another process or transputer, or for a time delay to complete.

j. Priority

The IMS T800 supports two levels of priority. Priority 1 (low priority) processes are executed whenever there are no active priority 0 (high priority) processes.

High priority processes are expected to execute for a short time. If one or more high priority processes are able to proceed, then one is selected and runs until it has to wait for a communication, a timer input, or until it completes processing. If no process at high priority is able to proceed, but one or more processes at low priority are able to proceed then one is selected. Low priority processes are periodically timesliced to provide an even distribution of processor time between computationally intensive tasks.

Note that the intention of having two priority levels for processes is to allow those high priority tasks, which must be executed when they are invoked, to preempt a currently executing low priority process and execute to completion. It is important that the

2. This is actually a time-sharing for a single CPU system.

high priority tasks have a very short execution time (less than one slicetime period). Otherwise the low priority processes, which should be the computation intensive processes, will not be given fair access to the processor.

k. Communications

Communications between processes is achieved by means of channels. Process communication is point-to-point, synchronized and unbuffered. As a result, a channel needs no process queue, no message queue and no message buffer.

A channel between two processes executing on the same transputer is implemented by a single word in memory; a channel between processes executing on different transputers is implemented by point-to-point links. The processor provides a number of operations to support message passing, the most important being input message and output message.

The input message and output message instructions use the address of the channel to determine whether the channel is internal or external. Thus the same instruction sequence can be used for both, allowing a process to be written and compiled without the knowledge of where its channels are connected.

The process which first becomes ready must wait until the second one is also ready. A process performs an input or output by loading the evaluation stack with a pointer to a message, the address of a channel, and a count of the number of bytes to be transferred, and then executing an input message or output message instruction. Data is transferred if the other process is ready. If the channel is not ready or is an external one the process will deschedule.

3. Programming Languages

There are several languages which can be used to write programs for use on the transputer. Among these are Occam, Alsys-Ada, 3L's Parallel C, C++, Pascal and Fortran. Three of the languages were considered for this thesis. These three languages were Occam [Ref. 10], Alsys-Ada [Ref. 11], and 3L's Parallel C++ [Ref. 12] [Ref. 13].

a. Occam Programming Language

Occam [Ref. 10] is a high level programming language that is designed to run concurrent processes on a network of processing components (e.g. transputers). There are two prime concepts in Occam; they are concurrency and communication. These allow processes to run simultaneously and transfer information, via channels, from process to process. It is based on concepts founded by David May in Experimental Programming Language and Tony Hoare in Communicating Sequential Processes.

It allows processes running on a transputer system to communicate only through channels. These channels are asynchronous, but require the send and receive processes to be ready to send and receive at the same time. This idea of being ready to send and receive simultaneously is known as rendezvous.

Occam has five kinds of constructions that are used to build a process from smaller processes (primitive or other). These constructions are:

- IF: This construction guards a number of processes by a boolean expression.

- CASE: This construction is used to select one of a number of options.

- WHILE: This construction is used for loops.

- PAR: This construction has the effect of allowing the processes within its bounds to execute in parallel.

- ALT: This construction is used to allow a processor to select only one of several guarded processes for execution. The process whose guard is first found to be true is selected.

This language allows the programmer to concentrate on a small, manageable set of processes which can then be connected with other sets of processes. In Occam a set of processes or a set of interconnected processes can be regarded as a single process.

The above features make Occam a powerful and versatile language. It has not gained wide acceptance thus far probably due to the limited use of multiprocessor (transputer) systems and due to the development of parallel versions of other widely used languages.

b. Alsys Ada Programming Language

In October 1989, Alsys produced the first compiler capable of supporting multi-processor programming in Ada [Ref. 11]. Alsys Ada Compilation System consists of the compiler and binder, operating in the Alsys Multi-Library Environment. The compiler generates executable code for transputer for T4 or T8 transputer targets. Multi-Library Environment provides a powerful way of managing Ada development efforts. It allows compilation units to be flexibly shared among libraries, and eliminates the need to copy library units to share them, along with the associated version control problems.

Although it has the features mentioned above, we decided against using it, because the compilation time is too long when compared to the other languages.

c. 3L's Parallel C++ Programming Language

(1) Abstract Model. The treatment of parallel processing in transputer systems is based on the idea of communicating sequential processes which is explained in part B of this chapter. In this model, a computing system is a collection of concurrently active sequential processes which can only communicate with each other over channels. A channel connects exactly one process to exactly one other process and can only carry messages in one direction. Each process can have any number of input and output channels, but note that the channels in a system are fixed; new channels cannot be created during its operation. A process could be a bit of hardware or a software module; in particular it may also be another complex system, itself consisting of a number of communicating processes.

(2) Hardware Model. The transputer was designed to be used as a component in concurrent systems. Each transputer processor has four Inmos links, to connect it with other transputers. Each link has two channels, one in each direction. These hardware channels provide synchronized, unidirectional communication.

Arbitrary networks of transputers can be constructed simply by connecting their links together with ordinary wires, the only limitation being that each processor cannot be directly connected to more than four others. A transputer can therefore be viewed as a single process in a multi-transputer system. However, it is also possible for any number of concurrent processes to be run on an individual transputer. Any word in the transputer's memory may be used as a channel to connect one internal process to another. The address of such a channel word is used to identify it to the transputer instructions (and Parallel C++ functions) which send or receive messages. The contents of the word are used by the hardware to synchronize sending and receiving processes.

From a program's point of view, these internal channels and the hardware link channels are identical. The same instructions (or parallel C++ functions) are used to send and receive messages on both internal channels and the hardware link channels. Hardware link channels are identified by special fixed addresses, but internal channels have addresses allocated by software.

The equivalence of internal channels to hardware link channels means it is possible to develop a parallel system on a single transputer and then move some of its processes onto other transputers without having to recompile any code.

(3) Software Model. Parallel C++ is based on the same abstract model of communicating sequential processes as the transputer hardware.

A complete application is viewed as a collection of one or more concurrently executing tasks. Each task has its own region of memory for code and data, a vector of input ports, and a vector of output ports. The port vectors are passed to the task as arguments to its main function. The code of a task is a single transputer image (*.b4*) file generated by the ordinary linker, *linkt.*

Tasks can be treated as atomic building blocks for parallel systems, to be wired together rather like electronic components. Indeed, several such basic building-block tasks are supplied with the compiler.

Each element in the input and output port vectors is of type "pointer to channel word", (**CHAN*). Ports are bound to real channel addresses by configuration software external to the task itself; the bindings can be changed without recompiling or relinking the task. Extended C++ run-time library functions supplied with the compiler allow C++ programs to send and receive messages over the channels bound to a task's ports.

The configuration software also provides ways of specifying which software tasks are to be run on which hardware processors. Each processor can support any number of tasks, limited only by available memory.

Tasks placed on the same processor can have any number of interconnecting channels. Tasks placed on different processors can only be connected where physical wires connect the processors' links. Each logical connection between two tasks placed on different processors is assigned exclusive use of one the physical link channels connecting the processors. The number of interconnections between tasks on different processors is therefore limited by the number of hardware links each one has.

(4) Parallel Execution Threads. The software features described so far allow us to build parallel systems by connecting together the ports of a number of relatively independent tasks. In particular, all the tasks have separate code and data, and are only allowed to communicate with each other by sending messages over channels.

All of the code of a task can be written in an ordinary sequential language, except for one extra feature needed by languages based on the communicating sequential processes idea. This extra feature is a way of making a process wait until a message is received on any one of a number of input channels. In Parallel C++, it is catered for by the ability to create new concurrent threads of execution within a task. The task creates one thread for each input channel. Each thread executes a sequential message input call and handles messages received on that channel. Each one of Parallel C's threads has its own stack (allocated by its creator), but shares its code, and all of its static and heap data, with any other threads in the same task. Semaphore functions in the run-time library are used to prevent threads to interfering with each other.

(5) Configuring An Application. Once an application has been designed and written as a collection of communicating tasks, it is loaded into physical network of transputers. First, each individual task is built by compiling all its source files with the C++ compiler and using the linker (*linkt*) to combine the resulting binary (*.bin*) files with the Parallel C++ run-time library to produce a task image (*.b4*) file. Then, a bootable application image file must be generated from the component task (*.b4*) files. The program which does this is called the configurer. It is driven by a user-supplied configuration file which specifies:

* the hardware configuration (processors, and the wires connecting them) on which the application is to be run;

* the names of the *.b4* files containing the component tasks of the application;

* the connections between the various tasks' ports;

* the placement of particular tasks onto particular tasks onto particular processors in the physical network.

The output of the configurer is an application file which can booted into the specified hardware network and run using the same *afserver* program used for simple stand-alone programs. The afserver task is an ordinary MS-DOS executable (*.exe*) file that runs on the PC. It loads executable *.b4* files into the transputer and also acts as a file server, handling I/O requests made by the transputer. The afserver and the transputer execute in parallel and communicate via an INMOS link. The messages sent to the afserver are normally generated by the Parallel C++ run-time library. It converts I/O operations into messages requesting the afserver to perform MS-DOS operations and then waits for the afserver to reply.

(6) Processor Farms. The tools described so far allow you to build applications which execute on any transputer network the wiring of which can be specified in advance in a configuration file. For many parallel computations it is useful to be able to create applications which will automatically configure themselves to run on any network of transputers. Such applications will automatically run faster when more transputers are added to a network, without recompilation or reconfiguration.

Parallel C++ allows us to create applications like this, provided the application can be implemented by a processor farm, and provided that there is enough memory on each processor in the network to support the required loading and message handling software.

The processor farm is a method of building applications for the transputer. Many users have found it a useful technique, for the following reasons:

* It takes full advantage of the transputer's parallel processing facilities and the ability of transputers to work together in groups.

* Many existing sequential programs can be converted into processor farms without much difficulty.

* A processor farm is not restricted to a particular network of transputers, but will automatically take advantage of the processors it finds.

A processor farm includes two independent programs, or tasks, written by the user. These are called the master task and the worker task. There is only one copy of the master task, and this is placed on the root transputer, that is, the transputer which is directly connected to the host. A copy of the worker task is placed on every transputer in the network.

The function of the master task is to break up the job which is to be done into a number of small, independent sub-jobs, each of which is performed by one of the copies of the worker task. The master does this by sending details of the sub-job to be done to the worker task. The worker task sends the results of its work back to the master task, which combines it with the results from all the other worker tasks. The worker task is written in such a way that immediately after sending its results back to the master, it is ready to receive details of another sub-job, and so on.

The communication between the master and the workers can be in two ways. Either another task called router can be written by the user, or special procedures which are included in the run-time libraries of the parallel languages and automatically added to the processor farm can be used.

III. DETAILED PROBLEM STATEMENT

A. PEGASUS DATABASE

1. Introduction

The PEGASUS Perspective View Database (PVDB) [Ref. 14] is a geographic database containing elevation data, gray shades taken from aerial photographs, vegetation heights, and other information required for perspective view generation. The PVDB comes in four resolutions: 1-, 4-, 16-and 64-meter.

The Fort Hunter-Liggett (FHL) PVDB covers a rectangular area on the ground measuring 32x28 kilometers. Its southwest corner is at UTM coordinates 43328,63904 and its northeast corner is at UTM 76095,92575. The latitude and longitude of these two points are approximately 35° 48′N, 121° 25′W and 36° 4′ N, 121° 4′W.

2. Database Organization

The PVDB is organized as a collection of tiles, blocks, and posts (see Figure 3.1, Figure 3.2 and Figure 3.3). A post is the smallest element in the database and covers an area on the ground measuring 1x1, 4x4, 16x16, or 64x64 meters for the 1-, 4-, 16-, and 64-meter databases respectively. A post is the only database element for which the area of coverage is resolution dependent.

A block is a collection of posts that always covers an area on the ground measuring 256x256 meters, but the number of posts in a block depends on the resolution. A block in the 1-meter PVDB contains 256x256 posts, a 4-meter block is made up of 64x64 posts, a 16-meter block contains 16x16 posts and a 64-meter block has 4x4 posts.

A tile, the largest element in the database, is a collection of blocks which always covers an area on the ground measuring 4096x4096 meters. A tile contains a 16x16 arrangement of blocks regardless of resolution.

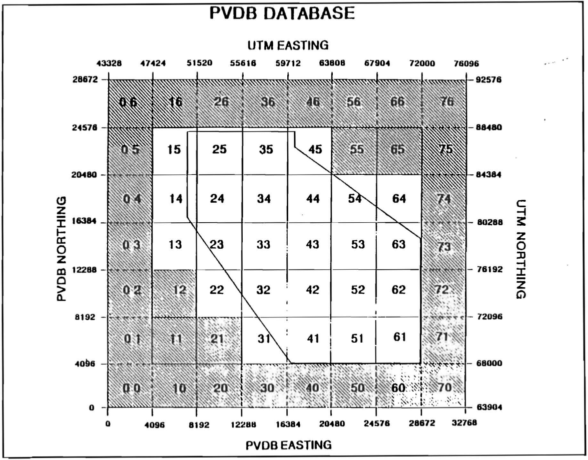

Figure 3.1: Pegasus Perspective Database

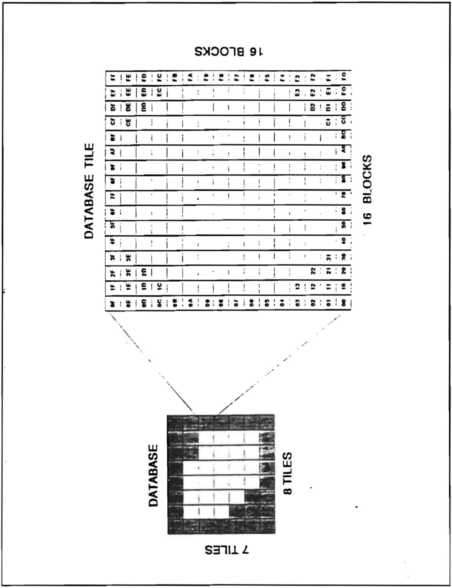

Figure 3.2: PVDB Tile Structure

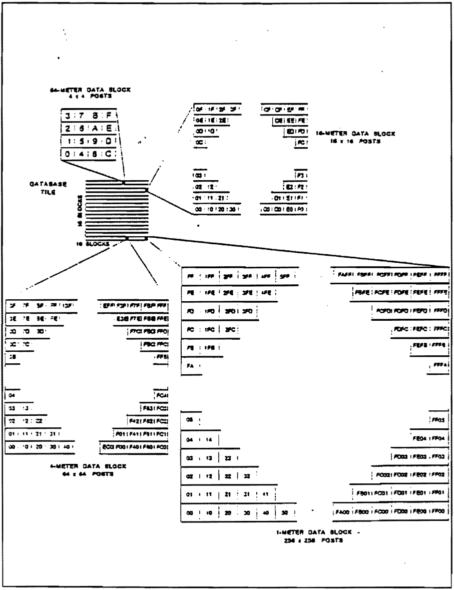

Figure 3.3: PVDB Block Structure

As shown in Figure 3.1, The Fort Hunter-Liggett (FHL) covers a rectangular area which consists of 56 tiles totally. The terrain data for 25 of them (white area in Figure 3.1) forms the actual database. Specifically, it covers 400 km^2 area of FHL. This area is used for training purposes.

Now, we can summarize the size information of a tile, a block and a post for 4 different resolutions as follows:

RESOLUTION	POST SIZE	BLOCK SIZE	TILE SIZE
1 meter	32 bits	256 Kbytes	64 Mbytes
4 meter	32 bits	16 Kbytes	4 Mbytes
16 meter	32 bits	1 Kbyte	256 Kbytes
64 meter	32 bits	64 Bytes	16 Kbytes

3. Post Structure

Figure 3.4 shows how each post in the PVDB is packed and how the 32 bits are distributed among the elements:

3	2			1				
1 0 9 8 7 6 5 4 3 2 1	0	9 8	7 6 5 4	3 2 1 0	9 8	7	6	5 4 3 2 1 0
ELE	EL2	UCI	NOR	VGT	VID	NAT	SSB	GSV

Figure 3.4: PVDB Post Structure

The element information is as follows:

ELEMENT CODE	NUMBER OF BITS	MAXIMUM VALUE	DESCRIPTION
ELE	11	2047	Elevation, in meters
EL2	12	4095	Elevation, in half-meters
UCI	2	3	Under Cover Index
NOR	4	15	Surface Normal Indicator
VGH	4	15	Vegetation Height Index
VID	2	3	Vegetation ID
NAT	1	1	Nature
SSB	1	1	Sun Shade Bit
GSV	6	63	Gray Shade Value

Each element has the following meanings (see Figure 3.5):

ELE: The bald terrain elevation plus the vegetation height (in meters) above the lowest point in the database. At FHL the lowest point is sea level.

EL2: Same as ELE except the units are in half-meters.

UCI: The height, in meters, of a cultural feature above the ground (tree branches, eaves of buildings, etc.).

NOR: A value which serves as an indication of the surface normal.

VGH: Height of the cultural feature. The stored values of 0 to 15 indicate vegetation heights of 0 (water), 0 (grass), 1, 2, 3, 4, 5, 8, 10, 15, 20, 25, 30, 35, 40, and 47 meters.

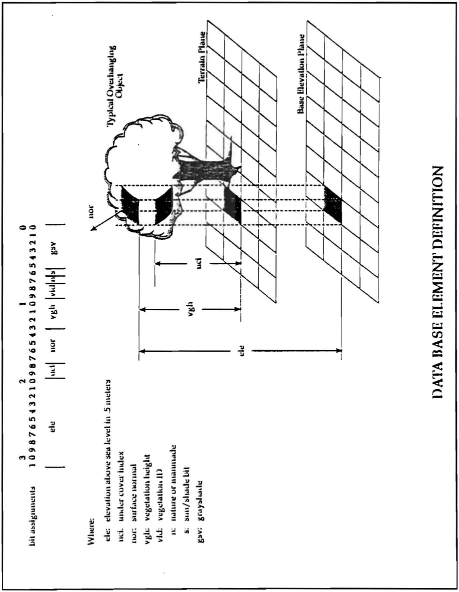

Figure 3.5: Database Element Definition

VID: Indicates the cultural feature. This value is combined with UCI, NOR, VGH, and NAT to determine what a particular object is.

NAT: If set to 1, this value indicates the cultural feature is natural, otherwise it is man-made.

SSB: If set to 0, this post is shaded by another cultural feature. This value is time-dependent.

GSV: A linear set of values ranging from 0 to 63, where 0 indicates black and 63 is white.

B. LINE-OF-SIGHT CALCULATION

Line-of-sight (LOS) is a central process in combat simulations that works at item level [Ref. 1]. The LOS algorithm is critical to the processes being simulated and impacts the run speed (ratio of game time to real time), since it may be the single most computationally expensive algorithm in simulation. Some LOS considerations in Janus have been simplified to increase computational efficiency.

There are three general aspects of LOS processing [Ref. 1:pp. 107-110]:

1. LOS in support of detections.
2. LOS through smoke and/or dust clouds.
3. LOS supporting force deployment.

For this thesis, we implemented the LOS calculation for the first aspect which is LOS in support of detections. A short description will be given for the other two aspects.

1. Line-of-sight for Detection

The first determination to be made is whether or not terrain features block the LOS between the observer and the target (see Figure 3.6). The process is as follows:

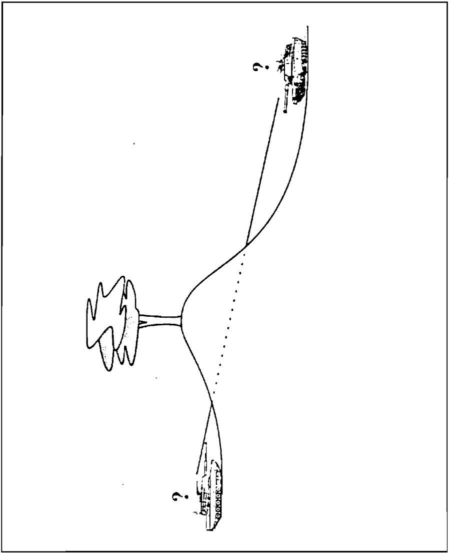

Figure 3.6: Line-of-sight for Detection

- The direct line between the observer and the target is determined, its length calculated and it is divided into equidistant points. Each point is tested to determine if a terrain feature affects the probability of LOS (PLOS).
- The number and the location of points on the line are determined as follows:
    - Compute the distance between the observer and the target (delta(X) and delta(Y)).
    - Determine N(X) and N(Y) by dividing delta(X) and delta(Y), respectively by the terrain grid size. Assign the larger of N(X) or N(Y) to Np, which is the number of points to be tested along the LOS line.
    - Compute dX = delta(X) / Np and Dy = delta(Y) / Np.
- Start at the observer's position + (dx,dy) and determine the terrain height (ground elevation) of the grid in which that point rests. If the ground elevation is greater than that of the observer, LOS is blocked and the process is completed for that observer-target pair.
- If the terrain height at that point is less than or equal to the height of the observer, add the height of trees/urban areas in that grid and recompute the terrain height. If the ground elevation + features height is greater than that of the observer, PLOS is decremented by the LOS degradation factor caused by features in the grid.
- If the resulting PLOS is greater than 0.01, dx and dy are added to the old position and the process continues until LOS is considered blocked or the target position is reached. A random number is drawn and compared with the resultant PLOS to determine if acquisition has taken place.

2. LOS Through Smoke/Dust Clouds

If LOS exists between the target and the observer, the model checks to see if any smoke or dust blocks the LOS line.

3. LOS For Deployment

The LOS for any unit can be displayed by the gamer from the workstation by pucking the LOS block on the menu and then the unit. The parameters of the LOS fan are attached to each unit, depending on its sensor (height, range) and how the orientation and width of the fan have been previously set by the gamer.

C. WHY 1-METER RESOLUTION?

To have reliable data that represents a terrain, there are some concepts that should be considered. First, we will describe these concepts with the help of Figure 3.5 and Figure 3.7.

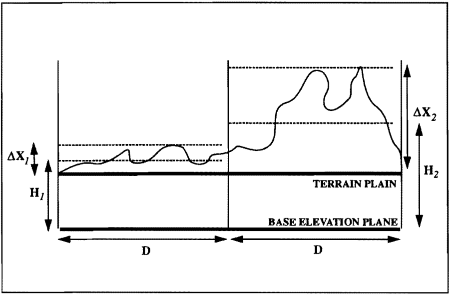

Figure 3.7: General View of A Terrain

The calculation of LOS is based on data stored in a grid of square cells. The elevation, the height of trees or urban buildings are stored as part of the terrain database and they are the factors which cause the unevenness of the terrain.

In Figure 3.7, **D** represents the length of one side of square cells. $\Delta \mathbf{X}_1$ and $\Delta \mathbf{X}_2$ represent the "**absolute variation**" which shows the unevenness of the terrain. H_1 and H_2 represent the height values to be assigned to those square cells.

The real height values are mostly expected to have some decimal digits. So, these values should be rounded by using a resolution value before being assigned to the square cells. We call this resolution value as "**height resolution**" and symbolize it as "ΔH".

The question at this moment is how we can choose the best ΔH. To answer this question, first we consider a flat terrain (see left cell in Figure 3.7)which means that ΔX is small. In this case, a small ΔH can be reasonable. But, when a rough terrain which has a big ΔX is considered (see right cell in Figure 3.7), a small ΔH will not work well. For example, assume we are using 10 centimeter height resolution when dealing with a terrain which has 10 meter of absolute variation. Using such a small height resolution i.e. sensitivity for an absolute variation which is relatively too high for that height resolution value will not give reliable rounded numbers for the real height values for the square cells. So, our first conclusion is as follows:

Conclusion 1: The best idea is to equalize ΔH and ΔX or, to choose ΔH which is bigger than ΔX.

Before applying the first conclusion to our problem, we should first normalize absolute variation and height resolution. Eq 3.1 and Eq 3.2 show this process:

$$Normalized\ Terrain\ Variation = \frac{\Delta X}{D} \qquad \text{(Eq 3.1)}$$

$$Normalized\ Height\ Resolution = \frac{\Delta H}{D} \qquad \text{(Eq 3.2)}$$

After normalizing process, we can approach to our problem more specifically as follows:

We assume the reasonable normalized terrain variation for a man-made flat surface as about 0.5%, for a natural terrain as about 5% and for a rough terrain as about 50%.

Since, The Fort Hunter-Liggett training area can be accepted as a rough terrain, then our second conclusion is as follows:

Conclusion2: The normalized height resolution to be chosen should be around 50%.

Another important factor for our problem is the length of one side of a square cell, namely D. It is obvious that when D increases, ΔX will increase with a high probability since more elevation differences, more trees or more urban buildings will be inside the borders of one square cell. We believe that this situation should be avoided to have reliable height values for each cell. Because, we will use a constant height resolution value and a constant D for our all database and we should not increase the probability of having big values of ΔX by increasing D. So, our third conclusion is as follows:

Conclusion 3: For rough terrain databases the D value should be as small as it can.

When we considered all of the concepts, factors and conclusions, we see that 1-meter resolution database with a 50 centimeter height resolution which has a 50% normalized height resolution is best to apply to our problem, and we believe that it represents The Fort Hunter-Liggett terrain very reliably.

IV. TRANSPUTER IMPLEMENTATION OF LINE-OF-SIGHT CALCULATION

A. HARDWARE

1. General

The designed network of transputer implementation of LOS calculation consist of following elements:

- An IBM PC as a host
- An IMS B004 Evaluation Board inside IBM PC
- An ALTA Remote TRAM Holder
- An ALTA CTRAM-25-4F (with 1 T805 25 MHz transputer)
- A SUN SPARC Station
- An ALTA HSI/SBus inside SUN SPARC Station
- An IMS B012 Evaluation Board
- 16 ALTA CTRAM-25-4F (with 16 T800 20 MHz transputers)

A general view of the network is shown in Figure 4.1. In section 2, each of the network elements will be mentioned in detail. In section 3, the implementation will be described with the modifications made by us towards our design purposes.

2. Background

a. The Transputer/Host Relationship

The transputer is normally employed as an addition to an existing computer, referred to as the host. Through the host, the transputer application can receive the services of a file store, a screen, and a keyboard as shown in Figure 4.2.

When the host is equipped with an add-in transputer interface board and the appropriate software, we call it a transputer development system. Presently, the host computer can be an IBM PC or compatible, a NEC PC, a DEC MicroVAX II, or a Sun

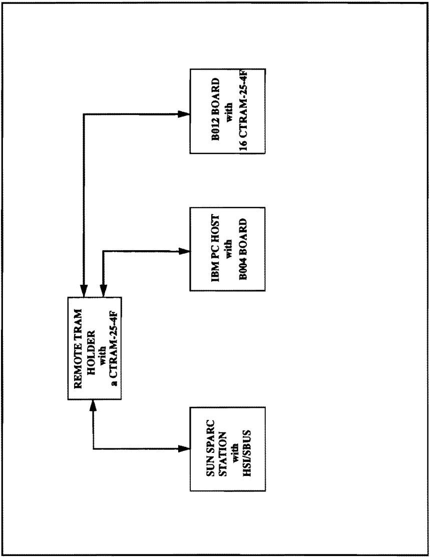

Figure 4.1: General View of the Implementation Network

SPARC Station in transputer development systems. But with the current capacity of our laboratory we are able to use an IBM PC for our implementation.

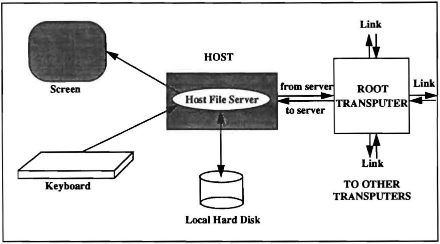

Figure 4.2: The Transputer/Host Relationship

b. IBM PC As A Host

The transputer communicates with the host along a single INMOS link. A program called a server [Ref. 15], executes on the host at the same time as the program on the transputer network runs. The server ensures that the access requirements of the application in terms of keyboard, screen, and filing are fully satisfied. All communications between the application running on the transputer and the host services (like screen, keyboard, and filling resources) take the form of messages. The standard transputer C, C++, Pascal, and Fortran development systems uses a server called **afserver**. The Occam toolset uses a server called **iserver**.

The root transputer in a network is the transputer connecting to the host bus via a link adapter. Any other transputers in the network are connected together using

INMOS links, to the root transputer. A transputer network can contain any size and mix of transputer types.

Transputer components form a unique hardware environment which is not immediately compatible with most existing personal computers (PC) or main frames upon which development work is accomplished. The IMS B004 evaluation board was designed to meet these needs by interfacing a transputer memory with an IBM type PC allowing the software developer to edit, compile and test software using the PC as a host.

c. *The IMS B004 Evaluation Board*

The IMS B004 board is logically divided into three distinct parts [Ref. 16]:

1. The transputer, with buffered links and one or two megabytes of RAM.
2. The PC subsystem logic, which allows a program running on the Personal Computer to reset and analyze systems.
3. The IMS C002 link adaptor, which interface to a parallel address/data bus, such as the one provided on the system expansion slots within an IBM PC. The link adaptor is accessed by a program running on the Personal Computer to transfer data to and from the transputer. This device can convert PC's byte-wide parallel data into serial link data for the transputer links, and visa versa.

These three distinct parts of the board are joined together by jumpers. The "Reset" jumper allows the PC subsystem to respond to addresses from the PC, and connects the transputer's reset, analyze, and error signals to those controlled by the PC. The "Link" jumper connects the link adaptor to one of the transputer's links, and allows the Link Adaptor to respond to addresses from the PC. Figure 4.3 shows a block diagram of the B004 board which fits in a full length eight bit slot of an IBM PC [Ref. 17].

Before any program can be downloaded to a B004 board from a PC, two jumper sockets must be fitted correctly. The use of these jumpers allows more than one

B004 to be present within a PC, but allowing only one of them to respond to the Transputer Development System (TDS).

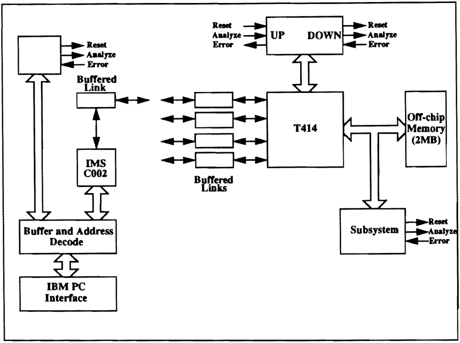

Figure 4.3: IMS B004 Evaluation Board Block Diagram

The board which has the jumpers fitted is designated the Master, and any number of other INMOS evaluation boards can be attached to this one via the links. Figure 4.4 shows the rear edge connectors of the B004, looking from the rear of the board. As can be seen, there are two columns of pins, and these are grouped into sets of five, suitable for the five way sockets which terminate the various cables supplied.

The link sockets are self explanatory. The Up, Down and Subsystem sockets are concerned with system control, initialization and error handling. The simplest way to use them is to connect the DOWN socket of the Master TDS board to the Up socket of the next board with the Reset cable, and then daisy chain the Down from each board to the Up of the next. This method ensures that when the TDS resets the first board, all others in the chain are also reset (see Figure 4.5).

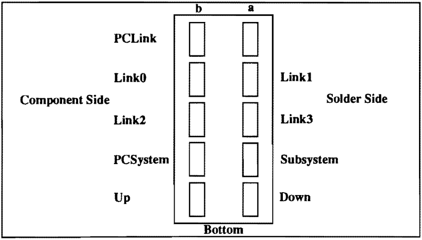

Figure 4.4: The Rear Edge Connectors of the B004

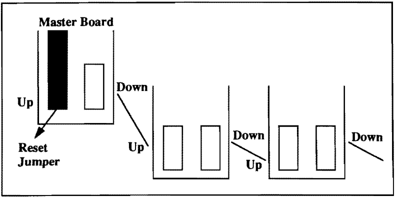

Figure 4.5: Daisy Chaning of the Subsequent Boards

The B004 board uses a group of 5 way connectors, to simplify the location of the various leads for a system (see Figure 4.6).

Pin	b	a
1	GND	NC
2	(missing)	(missing)
3	PCLinkOut	NC
4	PCLinkIn	NC
5	GND	NC
6	NotLink	NC
7	GND	GND
8	(missing)	(missing)
9	LinkOut 0	LinkOut 1
10	LinkIn 0	LinkIn 1
11	GND	GND
12	(gap)	(gap)
13	GND	GND
14	(missing)	(missing)
15	LinkOut 2	LinkOut 3
16	LinkIn 2	LinkIn 3
17	GND	GND
18	(gap)	(gap)
19	(gap)	(gap)
20	(gap)	(gap)
21	(gap)	(gap)
22	PCNotReset	SubsystemNotReset
23	PCNotAnalyse	SubsystemNotAnalyse
24	PCNotError	SubsystemNotError
25	GND	GND(missing)
26	(missing)	(missing)
27	NotSystem	NC
28	UpNotReset	DownNotReset
29	UpNotAnalyse	DownNotAnalyse
30	UpNotError	DownNotError
31	GND	GND(missing)
32	GND(missing)	GND(missing)

Figure 4.6: The B004 Board Edge Connector Pinout

The NotLink (b6) and NotSystem (b27) are used in conjunction with the Link and Reset jumpers described previously. When these signals are at logic 0, they select the functions associated with either reset or link to respond to signals from the PC.

d. ALTA CTRAM (Computation TRAnsputer Module)

The ComputeTRAM (or CTRAM) [Ref. 18] consists of a circuit board with transputer, memory, and connective hardware which is plugged into a TRAM Holder from ALTATechnology or similar boards from INMOS. The CTRAM includes from 1 to 32 Mbytes of DRAM and supports the IMS T80x transputer (with a chip floating point processor) or IMS T425 (integer only) transputers. A variety of processor speeds and memory speeds are available, providing users with a wide range of cost-effective compute modules.

The CTRAM is the basic unit for computation in parallel processing applications. With its range of external memory configurations and processor speeds, the CTRAM is a versatile tool for the system designer or the system integrator. The end-user can find extra value from the CTRAM by matching the configuration of each CTRAM with the needs of his application. This customization results in a tailored, economical mix of processors and memory configurations.

CTRAMs may be connected to other transputer modules via its four transputer links to form a wide variety of topologies.

The module pinouts and descriptions for CTRAM is shown in Table 4.1.

e. ALTA Remote Tram Holder

The Remote TRAM Holder [Ref. 19] may be mounted inside of a disk enclosure, or in a chassis suitable for holding disk drives and/or transputer modules. Figure 4.7 shows the block diagram of an ALTA Remote Tram Holder.

TABLE 4.1: CTRAM MODULE PINOUTS AND DESCRIPTIONS

Pin #	Pin Name	In/ Out	Function
1	Link2out	Out	Link 2 output
2	Link2in	In	Link 2 input
3	VCC		Power (+5V)
4	Link1out	Out	Link 1 output
5	Link1in	In	Link 1 input
6	LinkSpeedA	In	Transputer link speed selection A
7	LinkSpeedB	In	Transputer link speed selection B
8	Clockin	In	5MHz clock signal
9	Analyze	In	Transputer analyze
10	Reset	In	Transputer reset
11	notError	Out	Transputer error indicator (inverted)
12	Link0out	Out	Link 0 output
13	Link0in	In	Link 0 input
14	GND		Ground
15	Link3out	Out	Link 3 output
16	Link3in	In	Link 3 input

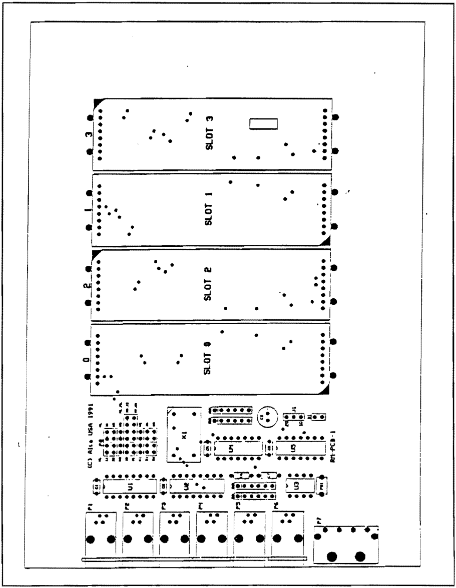

Figure 4.7: The Block Diagram of ALTA Remote Tram Holder

(1) Jumper Options. The jumpers in location P8 are provided to allow a high degree of configuration connects Link 0 of Module 0 with external link 0. The pins are labeled as to module and the link, and contain an arrow pointing out of the LINKOUT signal towards the LINKIN signal. The user may insert jumpers to connect any external links.

Jumper J1 is factory-set to 20 Megabits/Second. The link speed can be changed to 10 Megabits/Second as a second alternative.

(2) External Links. The differentially-driven links on the module are connected via modular plugs and jacks. The modular connectors found at locations P1, P2, P3, and P4 correspond with X0, X1, X2, and X3 of the configuration area (P8). Those links can be connected to any available links in the TRAM SLOTs by jumpers or configuration modules.

(3) TRAM SLOTs and Topology. There are four TRAM SLOTs on the motherboard, labeled SLOT0 to SLOT3. They are arranged such that only a single pair of links (between SLOT1 and SLOT2) is committed (hardwired). All other links are brought out to the P8 configuration area.

(4) System Services. The Remote TRAM may be used without connecting system services (Error, Reset, and Analyze) to the host. The board will assert RESET upon power on. However, in some instances, the user may wish to access system services from the host. Connector P5 contains the equivalent of UP system services and should be connected to the host. Connector P6 contains the equivalent of DOWN services and should be connected towards the next module in the chain. The Error, Reset, and Analyze signals will be propagated UP and DOWN (depending upon the signal) properly to allow daisy-chaining of the system services.

The signals on P5 and P6 are as follows:

PIN	SIGNAL
1	GROUND
2	ERROR
3	RESET
4	ANALAYZE

f. HSI/SBus

The HSI/SBus [Ref. 20] is a single-slot SBus interface between the Sun SPARC Station and transputers. It provides a high-speed interface between the SBus found on a Sun SPARC Station and Transputers.

The HSI/SBus is a 32-bit SBus slave interface for a Sun SPARC Station. The HSI provides system services and four bidirectional transputer links to external transputers, using modular connectors and twisted-pair telephone cables. The links are differentially driven using AT&T 41L/R series of drivers. The HSI/SBus board is a single slot printed circuit board which conforms to Sun Microsystem's published standards for a single slot SBus card. Figure 4.8 shows the layout of the board and the locations of the major board components.

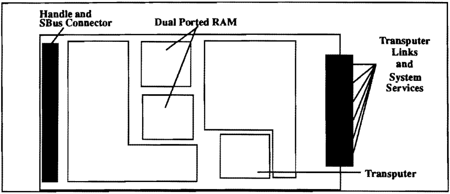

Figure 4.8: The HSI/SBus Board Layout

The SBus interface provides an electrical connection between the host and external transputer modules. It provides four, bi-directional transputer links to external transputers, and provides a set of control signals (Reset, Analyze, and Error) which are controlled by the driver on the SPARC Station host.

When the interface is initialized, transputer boot code is loaded into the dual-ported RAM and the transputer is then booted from that RAM. The transputer then executes the boot code to perform the interface functions.

Connections to external devices are made by using modular telephone handset jacks. Figure 4.9 shows the six jacks on the end of the HSI-card.

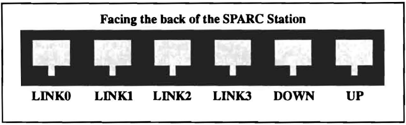

Figure 4.9: HSI-Card Link and Control Connections

The four links from the host interface are designated Link0, Link1, Link2, and Link3.

Reset, Analyze, and Error signals are provided for both DOWN and UP connections. The DOWN connector sends the Reset and Analyze signals to remote transputers.

g. The IMS B012 Evaluation Board

The IMS B012 [Ref. 21] is a eurocard TRAM motherboard which is a member of a family of TRAM motherboards which have a compatible architecture. External signals enable it to control a subsystem of motherboards, or to be a component of such a subsystem.

The smallest TRAM is "size 1". Each of the 16 sites for modules on the IMS B012 board accepts a size 1 module. Each module site, or "slot" has connections for four INMOS links which are designated link 0, link 1, link2, and link 3. TRAMs which are larger than size 1 can be mounted on the B012. A larger module occupies more than one slot and need not use all of the available link connections provided by the slots which it occupies.

The B012 has two IMS C004 link switches. These devices are able to connect together links from the slots and 32 links which are available on an edge connector. The connections can be changed by control data passed to the board down a configuration link, which may come from some master system or from one of the TRAMs on the B012 itself.

The B012 has two DIN41612 96-way edge connectors, P1 and P2. These carry almost all signals and power to/from the board and are easily identified from the board silk screen printing and from Figure 4.10. P2 carries power, pipeline and configuration links and system control signals (reset, analyze, and error).

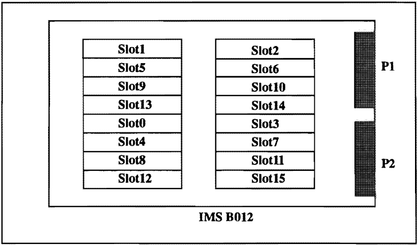

Figure 4.10: IMS B012 Slot Positions

The link connections to the 16 slots are organized as follows:

Two links from each slot (links 1 and 2) are used to connect the 16 slots as a 16-stage pipeline (in a pipeline, multiple processors are connected end-to-end as in Figure 4.11). The pipeline is actually broken by jumper block K1. K1will usually be jumpered in the standard way to give a 16-stage pipeline but can allow other combinations. Figure 4.12 shows the standard jumper configuration for K1 which connects all 16 TRAMs in a pipeline.

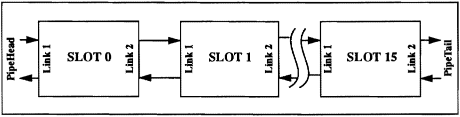

Figure 4.11: A Module Pipeline

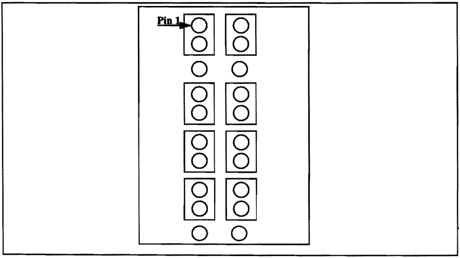

Figure 4.12: K1 Standard Configuration

Link 1 on slot 0 is wired to an edge connector (P2) and is called *PipeHead*. Link 2 on slot 15 is also taken to P2 and is called *PipeTail*. By connecting the pipe heads and tails from multiple boards together, a large, multi-board pipeline is created.

The other two links (links 2 and 3) of each slot are, in general, connected to two IMS C004 programmable link switches. The IMS C004 has 32 input pins and 32 output pins, plus an INMOS link (ConfigLink) used to send configuration information to the IMS C004. Any of the output pins can be "connected" to any of the input pins, so a signal presented on the input pin would be buffered and transmitted on the output pin (with a slight delay). The switch connections are made according to information sent to the IMS C004 down its ConfigLink. The two IMS C004s on the IMS B012 allow 64 link connections to be made under software control.

The Reset, Analyze and Error pins of TRAMs (and transputers) is generally referred to collectively as "system services". The system service signals are used to reset TRAMs and transputers, to place transputers in an analyze state (for debugging) and to carry the fact that an error has occurred in one processor in an array back to some host system which will deal with the error condition.

Some TRAMs and most evaluation boards are capable of generating the system services for other TRAMs and transputers. This is called a subsystem control capability. The IMS B012 can be connected to another board with subsystem control and also accommodate one TRAM with subsystem control. Furthermore, the IMS B012 can generate subsystem control signals for other boards. The system service signals are organized in such a way that, another boards can be daisy-chained by using Up and Down pins on P2. The logic here is same as it is for B004 boards.

The IMS B012 has a six-way DIL switch (SW1) located between P1 and P2. Each of the six switches make up SW1 controls one signal on the board. When a switch is on, the signal is low and when the switch is off, the signal is high. So, the board link speed can be set to either 10 Mbits/s or 20 Mbits/s with these switches.

(1) P1 Connections. Connector P1 has three rows of 32 pins. All the pins in row "a" are connected to the ground. All the pins in row 'b" are link inputs and all the pins in row "c" are link outputs. At each of the 32 positions along P1, the three pins from rows a, b and c carry one link. These signals may be connected to devices with link ports in any way the user desires.

The link connections on connector P1 are intended mainly for communication between the IMS B012 and other boards in a card cage. However, it is also possible to use these P1 links and the IMS C004 link switches to switch link connections for an external system.

(2) P2 Connections. If the IMS B012 is to be used in an INMOS ITEM card cage, the ITEM supplies power and has a built-in back-to-back connector which allows link and reset cables to be connected to P2. Figure 4.13 shows the back-to-back connector pins as viewed from the rear, i.e. looking towards the pins. The boxes represent plugged-in cables. A good 5V power supply must be connected to the appropriate pins on P2.

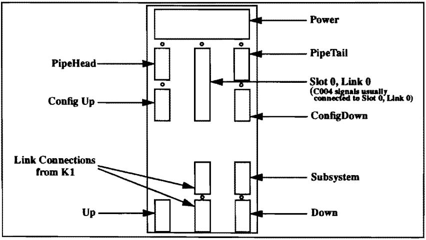

Figure 4.13: View of Back-to-back Connector Pins for B012

(3) IMS B012 as a Slave to a Master Controller. In a standard configuration where the IMS B012 is connected to a master-control system such as an IMS B004, PipeHead and ConfigUp links would be connected to two links on the host system, with "Up" system control port connected to the "Subsystem" port of the host (see Figure 4.14).

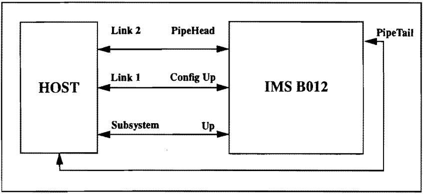

Figure 4.14: The IMS B012 Board as a Slave

(4) IMS B012 as a System Master. If a TRAM with "subsystem" capability is installed in slot 0 then the IMS B012 can act in a stand-alone or master role. With switch 6 (on six-way DIL switch) off, the system control to the other modules on the board and the "Down" system control pins on P2 are driven from the subsystem pins on the TRAM in slot 0.

3. Our Implementation

The steps for our implementation can be summarized as follows:

- To disable T414 transputer on the B004 board inside the PC host.

- To set up a remote tram holder and to place our root transputer on it.

- To connect Sun SPARC Station which has an HSI/SBus to the remote tram holder.

- To place 16 T805 transputers on a B012 board and to connect B012 board to the remote tram holder and B004.

- To set the link speed as 10 Mbits/second.

(1) Disabling the T414 Transputer on the B004 Board. As we have seen in the section which is related with B004 board, only T414 transputer can be used as root transputer on a B004 board and we can have a total of 2Mbytes RAM. But for our application, with a purpose of having more memory and speed, it was decided to use a T805 transputer as root transputer with a total of 4Mbytes RAM, namely an ALTA CTRAM-25-4F. So, the T414 transputer on the board, had to be disabled.

To disable the T414 transputer on the B004 board, two connections were made between two different pin pairs on the edge connector. These connections are shown in Figure 4.15.

(2) Setting Up the ALTA Remote Tram Holder. After disabling the T414 transputer, an ALTA CTRAM-25 4F which is actually a 25Mhz T805 transputer and 4Mbytes DRAM, was placed on slot 0 of the remote tram holder. So, this transputer became the root transputer.

Since a Sun SPARC Station, a B004 board and a B012 board connections were planned for the remote tram holder, each of them had to be taken care of separately because of the different requirements.

The HSI/SBus converts the Sun SPARC Station's parallel data signals to serial data signals for the transputer links. The voltage for the produced signal varies between -15 and +15 AC. But, transputers require 5V DC voltage. This voltage conversion for the signals is normally done by the converter on the remote tram holder if the jumpers are used in the P8 Configuration Area. So, two jumpers were used in the P8 Configuration Area for the link between Sun SPARC Station and remote tram holder to allow the necessary conversion and to assign Link 3 of the root transputer to the Sun SPARC Station (see Figure 4.16).

	Pin	b	a
	1	GND	NC
	2	(missing)	(missing)
	3	PCLinkOut	NC
To Remote Tram Holder Link 0	4	PCLinkIn	NC
	5	GND	NC
	6	NotLink	NC
	7	GND	GND
	8	(missing)	(missing)
	9	LinkOut 0	LinkOut 1
	10	LinkIn 0	LinkIn 1
	11	GND	GND
	12	(gap)	(gap)
	13	GND	GND
	14	(missing)	(missing)
	15	LinkOut 2	LinkOut 3
	16	LinkIn 2	LinkIn 3
	17	GND	GND
	18	(gap)	(gap)
	19	(gap)	(gap)
	20	(gap)	(gap)
	21	(gap)	(gap)
	22	PCNotReset	SubsystemNotReset
	23	PCNotAnalyse	SubsystemNotAnalyse
To Remote Tram Holder Up	24	PCNotError	SubsystemNotError
	25	GND	GND(missing)
	26	(missing)	(missing)
	27	NotSystem	NC
	28	UpNotReset	DownNotReset
	29	UpNotAnalyse	DownNotAnalyse
	30	UpNotError	DownNotError
	31	GND	GND(missing)
	32	GND(missing)	GND(missing)

Figure 4.15: The B004 Board Edge Connector Pinout After Modification

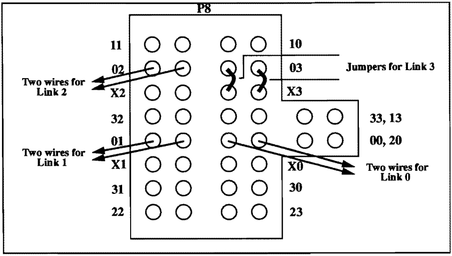

Figure 4.16: Remote Tram Holder P8 Configuration Area After Jumpering

Because the PC's parallel data signals are converted to serial data signals for the transputer links by the C002 Link Adaptor on the B004 board, we didn't need the conversion which was done for the Sun SPARC Station signals. Then, the other 3 links Link 0, Link 1 and Link 2 of the root transputer had to be connected to the PC and B012 board directly, without using jumpers in the P8 Configuration Area. But, the modular connectors P1-P6 (P1-P4 for transputer links, P5 and P6 for system services) have originally AT&T 41L/R series of drivers. So, those three links and UP and DOWN system services were carried to a connector which was located at the back of the remote tram holder and which had drivers for transputer link cables and for system service cables. For carrying links, two wires were used, one for LinkOut and one for LinkIn signal (see Figure 4.16). For carrying system services, three wires were used, one for Analyze, one for Reset and one for Error signal. Figure 4.17 shows the connections made inside the remote tram holder.

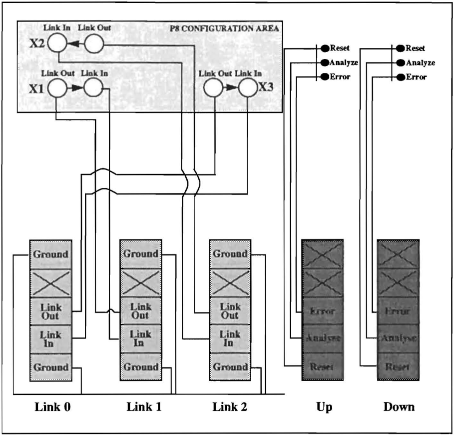

Figure 4.17: The Connections Made Inside the Remote Tram Holder

After the connections were made inside the remote tram holder, the 16 CTRAMs were placed on the B012 board and 16 T805-20 MHz transputers were placed on these CTRAMs.

The fixed hardware configuration for all the transputers in the network can be checked with the program named "check". This program runs in PC Host. Figure 4.18

shows the output of that "check" program for our application[1] and Figure 4.19 shows the physical view of our current fixed hardware configuration that we have for our transputers. We will see how a parallel application is created for a multi-transputer system with a fixed hardware configuration in the software part of this chapter.

Transputer#	LINK 0	LINK 1	LINK 2	LINK 3
0	HOST	1:1	2:2	-
1	-	0:1	3:1	-
2	-	4:2	0:2	-
3	-	1:2	5:1	-
4	-	6:2	2:1	-
5	-	3:2	7:1	-
6	-	8:2	4:1	-
7	-	5:2	9:1	-
8	-	10:2	6:1	-
9	-	7:2	11:1	-
10	-	12:2	8:1	-
11	-	9:2	13:1	-
12	-	14:2	10:1	-
13	-	11:2	15:1	-
14	-	16:2	12:1	-
15	-	13:2	16:1	-
16	-	15:2	14:1	-

Figure 4.18: The Output of "Check" Program for Our Application

1. For example, Figure 4.18 first row shows the following connections for Transputer# 0 (root): Its Link 0 to Host, its Link 1 to Link 1 of Transputer# 1 and its Link 2 to Link 2 of Transputer# 2.

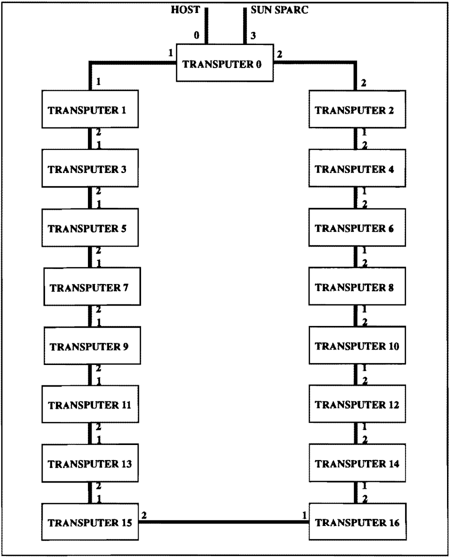

Figure 4.19: The Physical View of the Fixed Hardware Configuration

And finally we made the connections for Sun SPARC Station, B004 board, B012 board and remote tram holder as shown in Figure 4.20 Figure 4.21 and Figure 4.22 (for B004, refer to Figure 4.15).

The slot 0 link 0 on the B012 board usually needs to be connected to IMS C004s. This standard configuration requires a connection to be made via P2. A single connector assembly (termed the "yellow link jumper plug") are used for this purpose. The position of the jumper is shown in Figure 4.22.

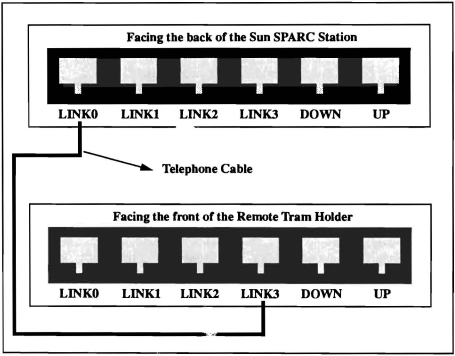

Figure 4.20: The Connection Between Sun SPARC Station and Remote Tram Holder

(3) Setting Up the Link Speed. Because of the B004 board's speed limitation, we set up the link speed as 10 Mbits/sec. To set up link speed for the remote tram holder, we connected the jumper J1 with the center position and the position labelled "10". For the B012 board, we set the DIL switches for links to operate at 10 Mbits/sec.

The link speed set up for the Sun SPARC Station is made by running an independent program, before running the real application program. We will mention about it in the software section of this chapter.

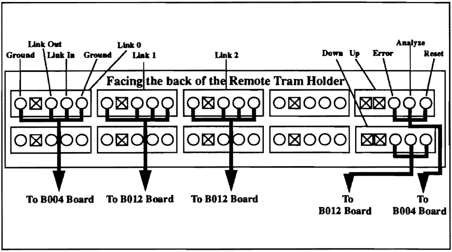

Figure 4.21: The Connections From the Back of Remote Tram Holder

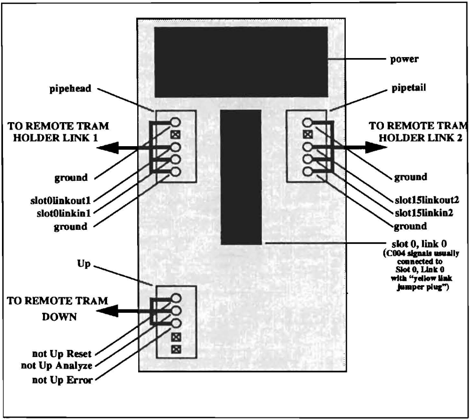

Figure 4.22: The Connections from the Back of B012 Board

B. SOFTWARE

1. General

The elements of the system and their functionalities from the software side of view is shown in Figure 4.23.

The main processes can be summarized in general as follows:

- The link operations between Sun SPARC Station and Remote Tram Holder and setting the link speed as 10 Mbits/sec.

- Loading the height data of the selected terrain from Pegasus Database to the CTRAMs.

- LOS calculation between the start and goal points which are sent to Sun SPARC Station by a server which represents JANUS.

- Sending the result back to the server from which the LOS calculation request is made.

- The afserver task on PC.

a. Installing HSI/Bus and Setting the Link Speed

As we have seen in the hardware part of this chapter, the HSI/Bus is a high-speed interface between the SBus found on a Sun SPARC Station and transputers and it provides link operations between them. [Ref.20] gives all the detailed information for installing and usage.

The program which sets up the link speed between Sun SPARC Station and Remote Tram Holder was supplied by ALTA Technology Corporation upon the request of us. The link speed should be 10 Mbits/sec before executing the main program because of the speed limitation of the PC host.

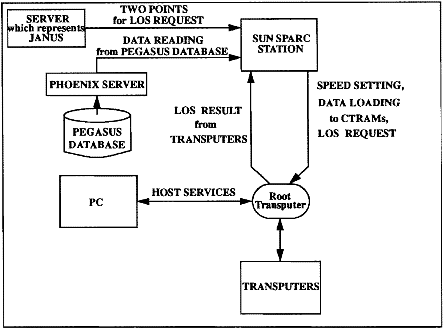

Figure 4.23: The Elements of the System from Software Side of View

b. Our Processor Farm Application

Three things must be written to create a processor farm application [Ref. 12:p. 77]:

1. A master task to split up the job into the independent work packets, i.e. sub-jobs.

2. A worker task, which is automatically copied to each node of the network of transputers.

3. A configuration file, describing the memory requirements and other attributes of the tasks.

(1) Master, Worker and Router Tasks. There is only one copy of the master task, and this is placed on the root transputer. A copy of the worker task is placed on every transputer in the network.

Special procedures are included in the run-time libraries of the Parallel languages to enable the communication between the master and the workers. They work in conjunction with another task, called the router.

Normally, router task is not written by the user, but is automatically added to the processor farm. When the master has a sub-job to be done, it calls a procedure which gives details of the sub-job to the router. The router then finds a worker somewhere in the network which is currently idle, and sends the work packet to it. The worker task then processes the work packet, and when it has finished, it calls a procedure to send the result packet back to the router, which returns it to the master.

For a normal processor farm application:

- A worker task contains three sequences: read a packet, process it, send back a result packet (i.e. input, process, output).

- Every worker should get the same input.

- For every cycle those three sequences start from the beginning.

But, for our application:

- Since we have a big amount of map data, we should divide it to little portions and load them to different CTRAMs at a time. Our map is too big to be loaded to a CTRAM. So every worker has different input.

- If we had used the same three sequences as mentioned above, we would have to load the whole data for every cycle. This would be too time consuming. So, we make first an initialization by loading the map data. Then, we send the point information to workers as input for LOS calculation, they process it and return the LOS result back. And for the second LOS request we don't have to make initialization again. Just the second part that includes input, process and output sequences repeats.

Because of the differences which we just described, routing in our application is done with the programs written by us instead of being done automatically. The source files for master, worker and router tasks are listed in Appendix B.

*(2) **Configuration File.*** The configuration file [Ref. 12:p. 38] describes the system to be built. It lists all the physical processors in the system, the wires connecting them, the tasks to be loaded into the system and their logical interconnections. In this section of the Chapter IV we explained configuration file giving the examples from our actual configuration file "btest180.cfg" which is listed in Appendix B.

The first thing the configuration needs to describe is the hardware configuration between the processors. The following configuration file lines declares the processor in the host PC, the processor in the Sun SPARC station and three transputers including the root transputer and describes the actual physical cables between these processors for our application:

```
processor host
processor sun type=pc
processor root
processor p1
processor p11

wire ?      root[0]     host[0]
wire ?      root[1]     p1[1]
wire ?      root[2]     p2[2]
wire ?      root[3]     sun[0]
wire ?      p1[2]       p11[1]
```

The **PROCESSOR** statement declares a physical processor. Every processor in the physical network must be declared, including the host processor from which the network is to be bootstrapped[2] (normally an IBM PC-type machine). The configurer assumes that the processor named host is the host processor. In the case of an

2. The linker program, linkt, normally produces an executable image file prefixed by a short bootstrap program which allows the the afserver to load the image into an empty transputer: the bootstrap initialises the transputer and reads in the rest of the image file.

IBM PC host processor, the host will usually be executing the afserver program when the network is loaded, simply because that is the program which loads the rest of the network. It is necessary to be able to specify the afserver task to the configurer so that its ports can be connected to ports in user tasks, but without forcing the configurer to attempt to bootstrap the IBM PC. Similarly, some processors in the network might be set to bootstrap from ROM rather than from link. A processor is declared to the configurer as having already been bootstrapped by means of the "**type**" attribute. The default for the host is that it is "type=pc" already. For our application, the Sun SPARC station processor was also described as "type=pc".

The **WIRE** statement declares a physical wire connecting links on two physical processors. Each wire supports two connections, one in either direction. The two link specifiers in the **WIRE** statement may therefore be interchanged without affecting the statement's meaning. Each wire is given a name (or '?' can be used instead of a name if the name will not be referred later). The numbers in the brackets for the **WIRE** statements are the link numbers of those processors which are used for connection. The processor identifiers used in a wire statement must have been declared in a previous **PROCESSOR** statement. This is a general rule: all objects in the configuration language (processors, wires, tasks) must be declared before they are used.

As well as describing the hardware of a system, the configuration file must contain details of all its software tasks and their interconnections. For each concurrently executing task in the system, the configuration file must contain a **TASK** statement. The **TASK** statement declares a task, which may be either a user-supplied task or one of the standard tasks provided with the configurer. The following configuration file lines declares the afserver task, filter task, master task, two router tasks and two worker tasks for our application:

```
task afserver      ins=1 outs=1
task filter        ins=2 outs=2 data=15k
task master        ins=5 outs=5 data=15k file="tr_commt.b4"

task router0       ins=20 outs=20 data=2k file="router.b4"  urgent
task router1       ins=20 outs=20 data=2k file="router.b4"  urgent

task worker00      ins=1 outs=1 data=275k file="worker.b4"
task worker01      ins=1 outs=1 data=275k file="worker.b4"
```

Each task declaration must include an "**ins**" attribute, which specifies the number of elements in the task's vector of input ports and an "**outs**" attribute, which specifies the number of elements in the task's vector of output ports. The "**data**" attribute specifies the amount of memory which a task needs. For example the filter task requires a minimum of 15 KByte of workspace. A user task for which no memory requirement is specified gets all the free memory remaining once any other tasks placed on that processor are loaded. Only one task on each processor can have its memory requirements left unspecified in this way. The configurer would otherwise have to decide how to split the remaining memory between several tasks with unspecified requirements; because an even split is unlikely to be desirable in practice, that is not allowed. The "**urgent**" attribute specifies that the task's initial thread is to be started at the urgent priority level. The default is that the task's initial thread is started at the non-urgent priority level. The "**file**" attribute specifies the file in which the memory image of the task is to be found. Task image files are produced by the linker program. The "file" attribute is ignored for the host processor and for any processor for which the processor attribute "type=pc" has been specified.

The placement of tasks on processors is specified by the **PLACE** statement. It determines which processor a particular task is to execute on. Every task must be placed on some processor. The following configuration file lines describes the placement of the afserver task, filter task, master task, two of the router tasks and two of the worker tasks for our application:

```
place afserver      host
place filter        root
place master        root

place router0       root
place worker00      root

place router1       p1
place worker10      p1
```

The CONNECT statement establishes a channel between two tasks, by connecting an output port to an input port. Because channels (unlike wires) are unidirectional, two CONNECT statements are needed to create channels going in both directions between two tasks. The following configuration file lines describes the channels between the afserver task, filter task, master task, two router tasks and one router-one worker tasks for our application:

```
connect ? afserver[0]     filter[0]
connect ? filter[0]       afserver[0]

connect ? filter[1]       master[1]
connect ? master[1]       filter[1]

connect ? master[2]       router0[0]
connect ? router0[0]      master[2]

connect ? router0[1]      router1[0]
connect ? router1[0]      router0[1]

connect ? router0[4]      worker00[0]
connect ? worker00[0]     router0[4]
```

The CONNECT keyword can be followed by an identifier naming the connection, but all the configuration statements which declare new identifiers allow a question mark to be used in place of the identifier being declared. This is useful when there is no need to refer to an object after it has been declared. After the identifier (or question mark) the output port is coded first, and then the input port is coded.

And, finally the **BIND** statement allows the contents of a port to be explicitly set to some literal value. Normally, ports are only bound by means of the **CONNECT** statement: ports left unbound are pointed at unique transputer channel words so that attempts to send or receive messages through them cause the minimum harm; the thread causing the attempt to communicate over the unbound port simply pauses indefinitely rather than causing failure of possibly all threads running on the processor. One application of the **BIND** statement is to give a task access to the transputer's external event mechanism. This appears as a channel word at a specific address. Another application of the **BIND** statement is to pass an integer parameter to a user task. We used the first application and initialized the "input port 4" and "output port 4" of the master task to point to that channel words at the addresses which are shown in the following configuration file lines:

```
bind input master[4]     value=&8000001C
bind output master[4]    value=&8000000C
```

The configuration files help to create a parallel application for a multi-transputer system with a fixed hardware configuration. For our application, the fixed hardware configuration was shown in Figure 4.19 of the hardware part of this chapter. Our configuration file btest180.cfg is listed in Appendix B and Figure 4.24 shows our multi-transputer system application i.e. current topology for transputers.

c. Loading the Height Data

The Pegasus Database has all the terrain height data, as we detailed in Chapter III. Because of the memory limitations of CTRAMs (each of them has 4Mbyte RAM), we can read and load the height data for a limited area at a time.

In our application program, we use an 5120 x 2304m. terrain which includes the training area whose UTM coordinates are 54000 - 59000 WE and 78000 - 80000 SN and PVDB coordinates are 10692 - 15672 WE and 14096 - 16096 SN. This area was selected because, its vegetation has the desired characteristics for a tank battle training.

The loading process occurs in two basic steps. First, the data is read by the Sun SPARC Station from Pegasus Database and then transferred (loaded) to CTRAMs. Pegasus Database is accessible through the Phoenix Server which is not a member of our department Local Area Network. However, the Pegasus Database was mounted through NFS (Network File System), so the database can be simply accessed by a read function. But, most of the time is still spent during this read function. The source code which we use for this data reading is listed in Appendix C.

For the second part of loading process, if we call all data to be loaded to CTRAMs as map, every CTRAM will have a portion of that map in its own memory after loading. The speed of this transfer is 10 Mbits/sec and the transfer occurs through the links.

The data are loaded to totally 15 CTRAMs. 14 of them are located on the B012 board and one of them is the on the Remote Tram Holder. Each CTRAM in our current system has a 4Mbyte memory. Since the router occupies some memory in each of them, we can load at most 15 blocks (256Kbyte each) to one CTRAM. But, to use as many transputers as we can for efficient calculation and meanwhile to load those CTRAMs equally, we use15 CTRAMs and each of them has 12 blocks. In each CTRAM, 12 blocks are loaded to 12 different workers. These workers are the smallest portions in which an LOS calculation occurs. Figure 4.25 shows the map we load at a time and the distribution of blocks to CTRAMs.

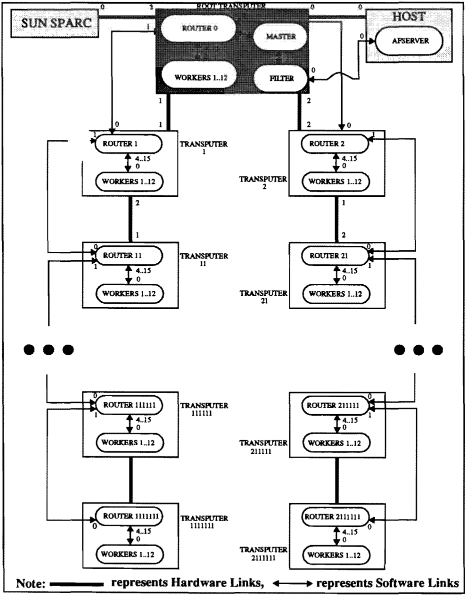

Figure 4.24: Current Topology of the Transputers

2304m					
	TRANSPUTER 2 WITH 12 BLOCKS	TRANSPUTER 1111 WITH 12 BLOCKS	TRANSPUTER 1111111 WITH 12 BLOCKS	TRANSPUTER 2111 WITH 12 BLOCKS	TRANSPUTER 2111111 WITH 12 BLOCKS
1536m					
	TRANSPUTER 1 WITH 12 BLOCKS	TRANSPUTER 111 WITH 12 BLOCKS	TRANSPUTER 111111 WITH 12 BLOCKS	TRANSPUTER 211 WITH 12 BLOCKS	TRANSPUTER 211111 WITH 12 BLOCKS
768m					
	ROOT TRANSPUTER WITH 12 BLOCKS	TRANSPUTER 11 WITH 12 BLOCKS	TRANSPUTER 11111 WITH 12 BLOCKS	TRANSPUTER 21 WITH 12 BLOCKS	TRANSPUTER 21111 WITH 12 BLOCKS
0	1024m.	2048m.	3072m.	4096m.	5120m.

Figure 4.25: The Map Size and the Distribution of Blocks to CTRAMs

d. LOS Calculation

The LOS calculation request between two points is made by a server that represents JANUS system. The information about the start and goal points is sent to Sun SPARC Station using the link communication established between them (the program which is used for this purpose is listed in Appendix A as client_main.C). Then, this information is broadcasted by the Sun SPARC Station to the transputers after receiving the point information.

The LOS calculation is made in each of the transputers. Since each transputer knows the borders of its map portion, the transputers whose map portions don't include the coordinates of those two points and of the line between them returns "0" as an answer automatically. The transputers whose map portions include the coordinates of those two points and of the line between them make LOS calculations for their map portions, and return "0" if LOS exists or "1" otherwise. Then all the answers from transputers are added, and if the total is "0", that means LOS exists between them, but if the total is greater than or equal to "1", that means LOS doesn't exist between them. This answer is sent to the server that represents JANUS by way of Sun SPARC Station.

e. The Afserver Task on Host

The afserver task is an ordinary MS-DOS executable (.exe) file that runs on the PC. It loads executable .b4 files into the transputer and also acts as a file server, handling I/O requests made by the transputer. The afserver and the transputer execute in parallel and communicate via an Inmos link. The messages sent to the afserver are normally generated by the Parallel C++ run-time library. It converts I/O operations into messages requesting the afserver to perform MS-DOS operations and then waits for the afserver to reply.

In principle, the afserver task could be directly connected to the user program. In practice, a filter task is interposed between them. The filter runs in parallel with the afserver and the user task; it simply passes on messages travelling in both directions. The filter is required because sometimes the messages passed between the user program and the afserver are only one byte long and the revision chip cannot handle single-byte message transfers on its hardware links. The filter pads out 1-byte messages to 2 bytes to avoid this problem. The connections for afserver and filter tasks can be seen in btest180.cfg configuration file which is listed in Appendix B.

V. EXPERIMENTAL RESULTS FOR LINE-OF-SIGHT CALCULATION

A. PERFORMANCE ANALYSIS

When a line-of-sight request is received by our system, the information about start and goal points is broadcasted to all transputers in the network. Since each transputer has height data for a different portion of all area, LOS calculations are done only by the transputers along the line between start and goal points. The advantage of parallelism for our application is that each transputer starts doing LOS calculations at the same time. So, when we neglect the time spent for communications between transputers, the total LOS calculation time for all transputers which participate the calculation should be equal to the time spent by the transputer which does maximum LOS calculations.

The most important factor for measuring performance increase with our parallel system is the distance between the two points which are subjects to LOS calculation. If the distance between those two points is too short and only one transputer does the calculation, then this is the worst case and we have no performance gain when we compare with a one processor system. If the distance between those two points is maximum, which is equal to the diagonal of the simulation area, then this is the best case and the performance gain is $\sqrt{n}$ where n represents the number of processors (transputers).

So, ideally the expected average gain after some number of consecutive LOS calculations will be:

$$EXPECTED\ AVERAGE\ GAIN = \frac{\sqrt{n}}{2} \qquad \text{(Eq 5.1)}$$

And the expected average utility of the system will be:

$$EXPECTED\ AVERAGE\ SYSTEM\ UTILITY = \frac{\sqrt{n}}{2}/n = \frac{1}{2\sqrt{n}} \qquad \text{(Eq 5.2)}$$

Since we used 15 transputers in our application, by using Eq 5.1 and Eq 5.2 we can say that the expected average gain of our system is $((\sqrt{15})/2) = 1.936$ and the expected average system utility is $(1/(2\sqrt{15})) = 0.129$.

B. THE RESULTS

In order to test our transputer implementation of line-of-sight calculation, we had to run our program such that all calculations would be done by one transputer. Then we could directly make comparison and see the improvement. But this could be possible only if the points between which the LOS calculation was required were inside the map borders of that transputer module. Since CTRAMs had approximately 4 Mbyte of limited available memory and the total training area required approximately 46 Mbyte memory, it was impossible to do timing testing with one transputer. Then, we decided to use another Sun SPARC station[1] with a large memory to hold all training area data in its memory. We made a modification to our application programs to run them on that Sun station as being a non-transputer or a non-parallel version. So, every LOS calculation was done by a single processor whatever the distance between start and goal points were. Then we could test our implementation by using the scale factor between transputer and that Sun station which will be described below.

We used two different start and goal point pairs for testing. The height values for both pairs were entered as big numbers, so we were sure that there was line-of-sight between start and goal points. This was important to provide a full calculation time. Because, the LOS calculation algorithm stops and returns the answer when a bigger height data is encountered before reaching to the end point. This could take a very short time. But, when there is line-of-sight between two points, this means every data on the line is checked and a full time LOS calculation occurs.

1. The Sun station we used was a SPARCsystem 630MP Model 120 with 128 Mbytes memory and two 40 MHz SPARC2 processors. Its performance was 25 MIPS and 4 MFLOPS for our application. This performance is almost twice of the performance of a SPARCstation1 which features 20 Mhz clock speed, 12 MIPS and 2.5 MFLOPS.

For the first pair, the distance between start and goal points were selected such that the coordinates of the points remained inside the borders of one transputer module. The purpose here was to allow only one transputer to do LOS calculation in our transputer implementation and to get one transputer LOS calculation time. Meanwhile we used the same points to get the Sun station LOS calculation time. These results[2] are shown in Table 5.1 and Table 5.2. The comparison between two calculation times gave us the scale factor between transputer and Sun station:

$$SCALE\ \ FACTOR = \frac{TRTIME1}{SUNTIME1} = 1.117$$

For the second pair, the distance between start and goal points were selected as maximum (as the diagonal of the area). The purpose here was to allow as many transputers as we could to do LOS calculation in our transputer implementation. We also used the same points to get the Sun station LOS calculation time for a maximum distance. These results[3] are shown in Table 5.3 and Table 5.4. Then, we simulated a transputer with enough memory to hold all map data by using the SCALE FACTOR, named that simulated time as SIMTRTIME2 and found the SPEEDUP RATIO for the best case of our implementation:

$$SIMTRTIME2 = SCALE\ \ FACTOR \times SUNTIME2 = 18.956$$

$$SPEEDUP\ \ RATIO = \frac{SIMTRTIME2}{TRTIME2} = 2.581$$

2. These timing results are for 100 consecutive LOS calculations of each point.
3. These timing results are for 100 consecutive LOS calculations of each point.

TABLE 5.1: THE TIMING RESULTS OF TRANSPUTER VERSION FOR SHORT DISTANCE (LIMITED TO ONE TRANSPUTER)

TEST NO	START POINT PVDB COORDINATE	END POINT PVDB COORDINATE	LOS RESULT	TIME (sec)
1	10672, 14096	11695, 14683	0	5.995
2	10672, 14096	11695, 14683	0	5.983
3	10672, 14096	11695, 14683	0	5.974
AVERAGE TIME = TRTIME1 =5.984				

TABLE 5.2: THE TIMING RESULTS OF NON-PARALLEL VERSION (SUN STATION VERSION) FOR SHORT DISTANCE

TEST NO	START POINT PVDB COORDINATE	END POINT PVDB COORDINATE	LOS RESULT	TIME (sec)
1	10672, 14096	11695, 14683	0	5.250
2	10672, 14096	11695, 14683	0	5.935
3	10672, 14096	11695, 14683	0	4.877
AVERAGE TIME = SUNTIME1 = 5.354				

TABLE 5.3: THE TIMING RESULTS OF TRANSPUTER VERSION FOR MAXIMUM DISTANCE

TEST NO	START POINT PVDB COORDINATE	END POINT PVDB COORDINATE	LOS RESULT	TIME (sec)
1	10672, 14096	15672, 16096	0	7.337
2	10672, 14096	15672, 16096	0	7.356
3	10672, 14096	15672, 16096	0	7.337
AVERAGE TIME = TRTIME2 = 7.343				

TABLE 5.4: THE TIMING RESULTS OF NON-PARALLEL VERSION (SUN STATION VERSION) FOR MAXIMUM DISTANCE

TEST NO	START POINT PVDB COORDINATE	END POINT PVDB COORDINATE	LOS RESULT	TIME (sec)
	10672, 14096	15672, 16096	0	17.028
	10672, 14096	15672, 16096	0	17.226
	10672, 14096	15672, 16096	0	16.661
AVERAGE TIME = SUNTIME2 = 16.971				

The communication overhead slowed down the processing time of transputers. The ratio between the expected best case gain which was $\sqrt{n}$ and the SPEEDUP RATIO showed us the maximum communication overhead between the transputers. We found that we had 33.3 percent of communication overhead as a maximum value for our system:

$$MAXIMUM\ COMMUNICATION\ OVERHEAD = 1 - \left(\frac{SPEEDUP\ RATIO}{\sqrt{15}}\right) = 0.333$$

The next step was to determine the average gain and the average communication overhead for the system. First, we had to find the average LOS calculation times for both transputers and the Sun station to do that. We kept the lower left corner of the map as the start point and used a random number generator to generate 50 different goal points for LOS calculations. We used these 50 pairs of points for our transputer system and for the Sun station. The results[4] were as follows:

$$AVERAGE\ LOS\ CALCULATION\ TIME\ FOR\ TRANSPUTERS = 6.541\,\mathrm{sec}$$

$$AVERAGE\ LOS\ CALCULATION\ FOR\ SUN\ STATION = 8.89\,\mathrm{sec}$$

Then, by using these two average time values and the SCALE FACTOR, we found the AVERAGE GAIN:

$$AVERAGE\ GAIN = \frac{8.89 \times SCALE\ FACTOR}{6.541} = 1.518$$

4. These timing results are for100 consecutive LOS calculations for each 50 points.

And, the comparison of EXPECTED AVERAGE GAIN which was $(\sqrt{n})/2$ and the AVERAGE GAIN gave us the average communication overhead between the transputers. We found that we had about 21.5 percent of communication overhead as an average value for our system:

$$AVERAGE\ COMMUNICATION\ OVERHEAD = 1 - \left(\frac{AVERAGE\ GAIN}{((\sqrt{15})/2)}\right) = 0.215$$

Finally, we calculated the average system utility for our application:

$$AVERAGE\ SYSTEM\ UTILITY = \frac{AVERAGE\ GAIN}{15} = 0.1012$$

VI. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

This thesis was an effort to improve Janus combat simulation model in a distributed memory and computing environment using transputers and PEGASUS 1-meter resolution database. We have shown that line-of-sight (LOS) calculation can be done using a multi transputer system with some modifications in the processor farming idea.

Due to the memory limitations placed on us by the Sun SPARC station[1] that we used in our application, we had to place 12 worker tasks on each transputer in the network. The number of worker tasks could be less only if the Sun SPARC station could keep bigger map data in its memory during each data loading process to the transputers. Because of the big number of worker tasks, we had a high communication overhead which affected the performance of our application.

Although the performance increase is less than the expected values, the timing results have shown that further significant improvements can be provided for LOS calculation time with faster transputers and a Sun SPARC station that has more memory.

B. RECOMMENDATIONS FOR FURTHER RESEARCH

The further research opportunities can be classified under the following main topics:

1. Connection To Janus

In ideal conditions, the line-of-sight calculation requests should be made by Janus system itself and the start and goal point information should be provided to Sun SPARC station. But Janus is not available in NPS Computer Science Department yet. After the

1. The Sun SPARC station in our application (see Figure 4.23) is a SPARCstation IPX with 16 MBytes memory.

completion of setting up the Janus in our department, the future work will be providing the connections between our application and the Janus system and make them work together.

2. INMOS T9000 Transputers

The INMOS T9000 [Ref. 6:p. 351] is the latest member of the transputer family. It is designed to provide far higher performance and greatly improved communication facilities. INMOS has used advanced CMOS technology to integrate a 32-bit integer processor, a 64-bit floating point processor, 16 Kbytes of cache memory, a communications processor and four high bandwidth serial communications links on a single IMS T9000 chip. The IMS T9000 transputer excels in real-time embedded applications, delivering exceptional single processor performance and scalable multiprocessor capability. In addition to executing several instructions each cycle, the number of cycles required to perform many arithmetic and logical operations has been reduced from previous transputers by adding extra hardware. Because of its superior characteristics, IMS T9000 should improve our system performance significantly.

3. ALPHA AXP Farm Programming Environment

Alpha AXP Farms which are produced by Digital Equipment Corporation are another choice for distributed memory parallelism. They also provide tools and libraries for farms. These AXP Farms use DECchip 21064 (Alpha AXP microprocessor) which is the fastest microprocessor in the industry [Ref. 6:p. 351]. DECchip 21064 offers the highest available performance with a 400 peak operations per millisecond, a cache bandwidth of 3.2 GB/s, controls up to 16 MB cache and a 64-bit design. Therefore we believe that the applicability of Alpha AXP Farms to our problem can be a future research area.

4. Parallel Programming Support Environments

A parallel programming environment is a collection of tools for automating part or all of the steps in writing a parallel program [Ref. 6:p. 351]. A variety of environments and tools have been proposed, prototypes constructed, and a few commercially available systems marketed to parallel programmers. Among these EXPRESS [Ref. 6:p. 351] and The HELIOS [Ref. 6:p. 351] are available in our laboratory.

EXPRESS is a collection of routine calls that form a toolbox for writing distributed-memory parallel programs. The toolbox routines are used as built-in functions to distribute data among processors and coordinate processors during parallel program execution. EXPRESS has been implemented on Intel, Mark III, nCUBE, and transputer-based machines [Ref. 6:p. 351].

The HELIOS Parallel Operating System has been designed to run on parallel computers. Such computers contain processing units, and fast communication between the processors. Many such parallel computers are built using transputers, and Helios runs on these machines. However, Helios also runs on parallel computers built using processors other than transputers.

So, another future research area is to check the applicability of these parallel programming support environments to our problem and to investigate how much improvements they can provide for us.

APPENDIX A - SUN SPARC STATION SOURCE CODE

This appendix contains the source listings of the C++ code developed for the Sun SPARC station that is used in this thesis. They are stored in files as listed below:

1. link.h
2. hsilink.h
3. los_com.h
4. los_global.h
5. map.h
6. map_c.h
7. map_s_com.h
8. s_comm.h
9. unix_comm.h
10. vector.h
11. map.C
12. map_c.C
13. map_s_com.C
14. s_comm.C
15. vector.C
16. manager.C
17. client_main.C

```
/************************************************************************************
FILENAME .............: link.h
AUTHOR.................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.......................: September 1993
DESCRIPTION .......: Contains the description of link communication functions which are written in C
                     language.
************************************************************************************/
/* Writes "Count " bytes from "Buffer" to the specified link. "LinkId" is a valid link identifier.
"Timeout" is a non-negative integer representing tenths of a second. A "Timeout" of zero is an infinite
timeout. */
extern "C" int WriteLink(int LinkId, char* Buffer, int Count, int Timeout);

/* Reads "Count" bytes into "Buffer" from the specified link. */
extern "C" int ReadLink(int LinkId, char* Buffer, int Count, int Timeout);

/* Ready the link associated with "Name". */
extern "C" int OpenLink(char* Name);

/* Closes the active link "LinkId". */
extern "C" int CloseLink(int LinkId);
```

```
/*****************************************************************************************
FILENAME.............: hsilink.h
AUTHOR.................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.......................: September 1993
DESCRIPTION.......: Header file which provides the necessary library functions for link
                    communication.
*****************************************************************************************/
/* @(#) Module: hsilink.h, revision 1.0 6/2/92 */
#include <sys/ioccom.h>
#define h 'h'  /* the h actually means nothing as used here */
/*
 * I/O controls
 */
struct HSI_SETF {
  unsigned int   op:16;
  unsigned int   val:16;
};

union HSI_IO {
  struct HSI_SETF set;
};

#define RESET           (1)
#define ANALYSE          (2)
#define SETTIMEOUT        (3)
#define TESTERROR        (4)
#define TESTREAD         (5)
#define TESTWRITE        (6)

/*
 * _IOW write instructions to the kernel within the
 * ioctl command code.
 */

#define SETFLAGS       _IOW(h, 1, union HSI_IO)

/*
 * End of hsilink.h
 */
```

```
/**********************************************************************************
FILENAME.............: los_com.h
AUTHOR.................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.......................: September 1993
DESCRIPTION .......: Header file for two structs. One of them is for information about map and the
                     other is for information about two points in the area.
**********************************************************************************/
#ifndef LOS_COM_H
#define LOS_COM_H

#include "vector.h"

/* Contains the lower left corner coordinates, the size and the grid size of map portion which is sent to
transputers at a time. */
struct MAP_INFO{
 int start_x, start_y, size_x, size_y;
 double grid_size;
};

/* Contains two vectors which have the information of two points between which LOS calculation is
made. */
struct CMD_INFO{
 vector start, goal;
};

#endif LOS_COM_H
```

```
/***************************************************************************************
FILENAME .............: los_global.h
AUTHOR..................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.........................: September 1993
DESCRIPTION .......: Defines three global values used in the program.
***************************************************************************************/
/* Defines that the size of a map portion which is sent to transputers at a time is 256m.x256m. */
#define MAP_SIZE 256

/* Defines that the grid size showing the resolution is 1m. */
#define GRID_SIZE 1.0

/* It is assumed that the beginning and end points of a line in the area are 10m. above the terrain.*/
#define AGENT_HEIGHT 10.0
```

```
/****************************************************************************************
FILENAME .............: map.h
AUTHOR.................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.......................: September 1993
DESCRIPTION .......: Header file for the declarations of the map class and the map class functions.
****************************************************************************************/
#ifndef MAP_H
#define MAP_H
#include "vector.h"

class map {
 public:
  struct map_rep{
   int start_x, start_y, size_x, size_y;
   double grid_size;
   int* data;
   int refs;
   map_rep() {refs = 1;}
  };
  map_rep *p;
  map();                    /* Constructors */
  map(int start_x,int start_y,int size_x,int size_y,double grid_size,int* data);
  map(const map& map); /* Copy constructor */
  map& operator=(const map& map); /* Assignment operator */
  ~map();

  /* Gets the lower left corner coordinates, the size and the grid size information of map. */
  int get_start_x() {return p->start_x;};
  int get_start_y() {return p->start_y;};
  int get_size_x() {return p->size_x;};
  int get_size_y() {return p->size_y;};
  double get_grid_size() {return p->grid_size;};
  int* get_data() {return p->data;};

  vector to_map_coord(vector loc);
  int higher_than(vector& loc);
  int terrain_height(int& grid_x, int& grid_y);
  int map_post(int grid_x, int grid_y);
 };

#endif MAP_H
```

```
/*************************************************************************************
FILENAME .............: map_c.h
AUTHOR.................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.......................: September 1993
DESCRIPTION .......: Header file for the source code which constructs the map portion to be sent to
                     transputers at a time.
*************************************************************************************/
#ifndef MAP_C_H
#define MAP_C_H

#include "map.h"

class map_c: public map {

public:
  map_c( int start_x, int start_y, int size_x, int size_y, double grid_size);
  map map_c_to_map(); /* only x,y are used */
};

#endif MAP_C_H
```

```
/*************************************************************************************
FILENAME ............: map_s_com.h
AUTHOR................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.....................: September 1993
DESCRIPTION .......: Header file for the source code written for sending the map portions to
                     transputers.
*************************************************************************************/
#ifndef MAP_COM
#define MAP_COM

#include "los_com.h"
#include "map_c.h"
#include "s_comm.h"

class map_s_com{
  MAP_INFO map_info;
public:
  map_s_com(){};
  void map_send(int n_tr, int n_pro, map& map, s_comm& s_comm1); /* Sends map portions. */
};

#endif MAP_COM
```

```
/*************************************************************************************
FILENAME .............: s_comm.h
AUTHOR.................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.......................: September 1993
DESCRIPTION .......: Header file for the source code which performs the link communication between
                     SUN station and the transputers.
*************************************************************************************/
#include "link.h"
#include "hsilink.h"
#ifndef S_COMM_H
#define S_COMM_H

const int ROUTER_INIT = 1;
const int SEND = 2;
const int BCAST = 3;
const int LISTEN = 4;
const int TERMINATE = 5;

class s_comm {
 int out_link_num;
 int in_link_num;
 int out_link;
 int in_link;
public:
 s_comm() {};
 s_comm(int out_link_num1, int in_link_num1);
 ~s_comm(){ CloseLink(out_link); CloseLink(in_link); };
 int router_init(int num_trs, int* trs, int* unders, int* prs, int timeout);
 int send(int dst, int nts, int size, char* buf, int timeout);           /* Plain send. */
 int send_i(int dst, int nts, int size, char* buf, int timeout);         /* Send integers. */
 int bcast_d(int size, char* buf, int timeoutf);                     /* Send doubles (byte convert). */
 int listen(int timeout);                                                /* Byte conversion. */
 int terminate(int timeout);
/* Conversion functions for little-indian(transputer) and big-indian(SUN) problem. */
 void convert4(char* buf1, char* buf2);
 void convert_i_array(int* buf1, int* buf2, int size);
 void convert8(char* buf1, char* buf2);
 void convert_d_array(double* buf1, double* buf2, int size);
};

#endif S_COMM_H
```

```
/**************************************************************************************
FILENAME .............: unix_comm.h
AUTHOR.................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.......................: September 1993
DESCRIPTION .......: Header file for the link communication functions between two SUN Stations.
**************************************************************************************/
#define SERVER_PORT_NUMBER 1053
#define CLIENT_PORT_NUMBER 1053

/* Link communication functions from "C library" for sender */
extern "C" int open_stream_s (int port_number);     /* Opens link */
extern "C" int send_buf_s(char* buf, int size);         /* Sends buffer */
extern "C" int receive_buf_s(char* buf, int* sizep);   /* Receives buffer */
extern "C" int close_stream_s (void);                   /* Closes link */

/* Link communication functions from C library"for receiver. */
extern "C" int open_stream_c (char* host_name, int port_number); /* Opens link */
extern "C" int send_buf_c(char* buf, int size);                    /* Sends buffer */
extern "C" int receive_buf_c(char* buf, int* sizep);               /* Receives buffer */
extern "C" int close_stream_c (void);                              /* Closes link */
```

```
/***********************************************************************************
FILENAME .............: vector.h
AUTHOR.................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.......................: September 1993
DESCRIPTION .......: Header file for the description of the vector class and vector class operations.
***********************************************************************************/
#ifndef VECTOR_H
#define VECTOR_H

class vector {
 double x,y,z;
public:
 vector();
 vector(double x1, double y1, double z1);

 double get_x() {return x;};
 double get_y() {return y;};
 double get_z() {return z;};
 friend int operator==(vector v1, vector v2);
 friend vector operator+(vector v1, vector v2);
 friend vector operator-(vector v1, vector v2);
 friend vector operator*(double a, vector v1);
 double dotprod(vector v1);
 double magnitude(void);
 vector normalize(void);
};
#endif VECTOR_H
```

```
/*********************************************************************************
FILENAME .............: map.C
AUTHOR.................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.......................: September 1993
DESCRIPTION .......: This source code defines the map class functions.
*********************************************************************************/
#include "map.h"

map::map()          /* Constructor */
{
  p = new map_rep;
  p->start_x = 0; p->start_y = 0; p->size_x = 0; p->size_y = 0;
  p->grid_size = 0.0;
  p->data = 0;  // null pointer
}

map::map(int start_x,int start_y,int size_x,int size_y,double grid_size,int* data)  /* Constructor */
{
  p = new map_rep;
  p->start_x = start_x; p->start_y = start_y;
  p->size_x = size_x; p->size_y = size_y;
  p->grid_size = grid_size;
  p->data = data;
}

map::map(const map& map)          /* Copy constructor */
{
  map.p->refs++;
  p = map.p;
}

map& map::operator=(const map& map)          /* Assignment operator */
{
  map.p->refs++;
  if (--p->refs == 0) {
    delete[] p->data;
    delete p;
  }
  p = map.p;
  return *this;
}
```

```
map::~map()        /* Destructor */
{
  if (--(p->refs) == 0) {
    delete[] p->data;
    delete p;
  }
}

 vector map::to_map_coord(vector loc)
{
  vector map_offset(((double)p->start_x)*p->grid_size,
       ((double)p->start_y)*p->grid_size,0);
  vector loc_wrt_map = loc - map_offset;
  return (loc_wrt_map);
}

int map::higher_than(vector& loc)
{
  int grid_x = (int) ((loc.get_x() - p->start_x*p->grid_size)/p->grid_size);
  int grid_y = (int) ((loc.get_y() - p->start_y*p->grid_size)/p->grid_size);
  int height = p->data[grid_y*p->size_x+grid_x];
  return ((double)terrain_height(grid_x,grid_y) > loc.get_z());
}

int map::terrain_height(int& grid_x, int& grid_y)
{
  return map_post(grid_x,grid_y);
}

int map::map_post(int grid_x, int grid_y)
{
  int index;
  /* index = size_y*grid_loc.x + grid_loc.y; */
  index = p->size_x*grid_y + grid_x;
  return p->data[index];
}
```

```
/**********************************************************************************
FILENAME ............: map_c.C
AUTHOR................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE......................: September 1993
DESCRIPTION .......: This source code constructs a map portion to be send to transputers at a time.
**********************************************************************************/
#include <iostream.h>
#include <fstream.h>
#include <stdio.h>
#include "PVG_DEC.H"
#include "PVG_DEF.IN"
#include <pvdb.h>
#include "map_c.h"

/* Reads one block of terrain data to a buffer and then loads elevation data to data array of map portion
by using the data in the buffer. */
map_c::map_c(int start_x, int start_y,int size_x, int size_y, double grid_size)
{
  int i;

  p = new map_rep;
  p->start_x = start_x;
  p->start_y = start_y;
  p->size_x = size_x;
  p->size_y = size_y;
  p->grid_size = grid_size;
  p->data = new int[size_x*size_y];
  /* One block of 1m. resolution terrain data is read to a buffer here. */
  get_terr(RESOLUTION_1, start_x, start_y, 1);
  /* 65536 elevation data is loaded to data array of map portion here. */
  for (i=0; i<65536; i++){
    p->data[i]=PVDB_UNPACK_ELE(TERRAIN1[1][i]);
  }
}

/* Converts map_c class to map class. */
map map_c::map_c_to_map()
 {
   map map1(p->start_x,p->start_y ,p->size_x,p->size_y,p->grid_size,p->data);
   return(map1);
 }
```

```
/*************************************************************************************
FILENAME.............: map_s_com.C
AUTHOR.................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE........................: September 1993
DESCRIPTION.......: This source code is for sending one map portion to transputers through the link
                    at a time.
*************************************************************************************/
#include "map_s_com.h"
#include <iostream.h>

void map_s_com::map_send(int n_tr, int n_pro, map& map, s_comm& s_comm1)
{
  MAP_INFO map_info, map_info1;

  map_info.start_x = map.p->start_x;
  map_info.start_y = map.p->start_y;
  map_info.size_x = map.p->size_x;
  map_info.size_y = map.p->size_y;
  /* Converts double,
     solves little_indian(transputer), big_indian(sun) problem,
     sends header,
     converts start_x, start_y, size_x, size_y */
  s_comm1.convert_i_array((int*)&map_info,(int*)&map_info1,4);
  double x = map.p->grid_size;
  double y;
  s_comm1.convert8((char*)&x, (char*)&y);
  map_info1.grid_size = y;
  s_comm1.send(n_tr, n_pro, sizeof(map_info1), (char*)&map_info1,50);
  /* Sends real data (integer is 4 chars) */
  s_comm1.send_i(n_tr, n_pro, map_info.size_x * map_info.size_y * 4,(char*)(map.p->data),50);
};
```

```
/*********************************************************************************
FILENAME ............: s_comm.C
AUTHOR...............: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.......................: September 1993
DESCRIPTION .......: This source code is for performing link communication between SUN station and
                     transputers. It also has conversion functions for solving the little-
                     indian(transputer) and big-indian(SUN) problem.
*********************************************************************************/
#include <iostream.h>
#include "s_comm.h"

/* Opens link. */
s_comm::s_comm(int out_link_num1, int in_link_num1)
{
  out_link_num = out_link_num1;
  in_link_num = in_link_num1;

  char link_str[2];
  link_str[0]= char(out_link_num1);
  link_str[1] = '\0';
  out_link = OpenLink(link_str);

  if (out_link_num1 != in_link_num1) {
    link_str[0]= char(in_link_num1);
    link_str[1] = '\0';
    in_link = OpenLink(link_str); }
  else
    in_link = out_link;
}

/* Does router initialization for transputers. */
int s_comm::router_init(int num_trs, int* trs, int* unders, int* prs, int timeout)
{
  int code = ROUTER_INIT;
  int val;
  convert4((char*)&code, (char*)&val);
  if (WriteLink(out_link, (char*)&val, sizeof(int), timeout) < 0)
    return -1;
  convert4((char*)&num_trs, (char*)&val);
```

```
  if (WriteLink(out_link, (char*)&val, sizeof(int), timeout) < 0)
    return -1;
  int* vals;
  vals = new int[num_trs];
  convert_i_array(trs, vals, num_trs);

  if (WriteLink(out_link, (char*)vals, sizeof(int)*num_trs, timeout) < 0)
    return -1;
  convert_i_array(unders, vals, num_trs);
  if (WriteLink(out_link,(char*)vals, sizeof(int)*num_trs, timeout) < 0)
    return -1;
  convert_i_array(prs, vals, num_trs);
  if (WriteLink(out_link, (char*)vals, sizeof(int)*num_trs, timeout) < 0)
    return -1;
  return 1;
}

/* Plain sending. No conversion. */
int s_comm::send(int dst, int nts, int size, char* buf, int timeout)
{
  int code = SEND;
  int val;
  convert4((char*)&code, (char*)&val);
  if (WriteLink(out_link, (char*)&val, sizeof(int), timeout) < 0)
     return 0;
  convert4((char*)&dst, (char*)&val);
  if (WriteLink(out_link, (char*)&val, sizeof(int), timeout) < 0)
     return 0;
  convert4((char*)&nts, (char*)&val);
  if (WriteLink(out_link, (char*)&val, sizeof(int), timeout) < 0)
     return 0;
  convert4((char*)&size, (char*)&val);
  if (WriteLink(out_link, (char*)&val, sizeof(int), timeout) < 0)
     return 0;
  // No conversion. Send buf directly
  if (WriteLink(out_link, buf, size, timeout) < 0)
     return 0;
  return 1;
}
```

```
/* Sends integers. */
int s_comm::send_i(int dst, int nts, int size, char* buf, int timeout)
{
  int code = SEND;
  int val;
  convert4((char*)&code, (char*)&val);
  if (WriteLink(out_link, (char*)&val, sizeof(int), timeout) < 0)
    return 0;
  convert4((char*)&dst, (char*)&val);
  if (WriteLink(out_link, (char*)&val, sizeof(int), timeout) < 0)
    return 0;
  convert4((char*)&nts, (char*)&val);
  if (WriteLink(out_link, (char*)&val, sizeof(int), timeout) < 0)
    return 0;
  convert4((char*)&size, (char*)&val);
  if (WriteLink(out_link, (char*)&val, sizeof(int), timeout) < 0)
    return 0;
  char* vals;
  vals = new char[size];
  convert_i_array((int*)buf, (int*)vals, size/sizeof(int));
  if (WriteLink(out_link, vals, size, timeout) < 0) {
    delete[] vals;
    return 0;}
  else {
   delete[] vals;
   return 1;};
}
/* Sends doubles. */
int s_comm::bcast_d(int size, char* buf, int timeout)
{
  int code = BCAST;
  int val;
  convert4((char*)&code, (char*)&val);
  if (WriteLink(out_link, (char*)&val, sizeof(int), timeout) < 0)
    return 0;
  convert4((char*)&size, (char*)&val);
  if (WriteLink(out_link, (char*)&val, sizeof(int), timeout) < 0)
    return 0;
  char* vals;
  vals= new char[size];
  convert_d_array((double*)buf, (double*)vals, size/sizeof(double));
```

```
  if (WriteLink(out_link, vals, size, timeout) < 0)
    return 0;
  return 1;
}

/* Reads the value coming from transputers. */
int s_comm::listen(int timeout)
{
  int code = LISTEN;
  int val, result;
  convert4((char*)&code, (char*)&val);
  if (WriteLink(out_link, (char*)&val, sizeof(int), timeout) < 0)
    return 0;

  if (ReadLink(in_link, (char*)&val, sizeof(int), timeout) < 0)
    return 0;
  convert4((char*)&val,(char*)&result);
  return result;
}

int s_comm::terminate(int timeout)
{
  int code = TERMINATE;
  int val;
  convert4((char*)&code, (char*)&val);
  if (WriteLink(out_link, (char*)&val, sizeof(int), timeout) < 0)
    return 0;
  return 1;
}

/* CONVERSION FUNCTIONS FOR LITTLE-INDIAN(TRANSPUTER) AND BIG-INDIAN(SUN)
PROBLEM STARTS HERE. */
void s_comm::convert4(char* buf1, char* buf2)
{
  buf2[3] = buf1[0];
  buf2[2] = buf1[1];
  buf2[1] = buf1[2];
  buf2[0] = buf1[3];
}
```

```
void s_comm::convert_i_array(int* buf1, int* buf2, int size)
{
  for (int i=0; i<size; i++)
    convert4((char*)(&(buf1[i])),(char*)(&(buf2[i])));
}

void s_comm::convert8(char* buf1, char* buf2)
{
  buf2[7] = buf1[0];
  buf2[6] = buf1[1];
  buf2[5] = buf1[2];
  buf2[4] = buf1[3];
  buf2[3] = buf1[4];
  buf2[2] = buf1[5];
  buf2[1] = buf1[6];
  buf2[0] = buf1[7];
}
void s_comm::convert_d_array(double* buf1, double* buf2, int size)
{
  for (int i=0; i<size; i++) {
    convert8((char*)(&(buf1[i])),(char*)(&(buf2[i])));
  }
}
```

```
/*****************************************************************************
FILENAME .............: vector.C
AUTHOR.................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.......................: September 1993
DESCRIPTION .......: This source code defines the vector class operations.
*****************************************************************************/
#include "vector.h"
#include <math.h>

vector::vector() {x=0.0; y=0.0; z=0.0;};
vector::vector(double x1, double y1, double z1) {x=x1; y=y1; z=z1;};
int operator==(vector v1, vector v2)
{
  return((v1.x==v2.x) && (v1.y==v2.y) && (v1.z==v2.z));
}

vector operator+(vector v1, vector v2)
{
  vector v(v1.x+v2.x, v1.y+v2.y, v1.z+v2.z);
  return v;
}

vector operator-(vector v1, vector v2)
{
  vector v(v1.x-v2.x, v1.y-v2.y, v1.z-v2.z);
  return v;
}

vector operator*(double a, vector v1)
{
  vector v(a*v1.x, a*v1.y, a*v1.z);
  return v;
}
double vector::dotprod(vector v2)       /* Dot product */
{
  return(this->x*v2.x + this->y*v2.y + this->z*v2.z);
}
```

```
double vector::magnitude(void)
{
  return(sqrt((*this).dotprod(*this)));
}

vector vector::normalize(void)                /* Vector normalization */
{
  vector result;
  double mag = (*this).magnitude();
  if (mag < 1E-100) {
    result.x = 0.0;
    result.y = 0.0;
    result.z = 0.0;}
  else {
    result = (1.0/mag) * (*this);
  }
  return(result);
}
```

```
/********************************************************************************
FILENAME.............: manager.C
AUTHOR.................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.......................: September 1993
DESCRIPTION.......: This is the main program. The number of transputers, workers and task
                    distribution are defined here. The user is asked to enter the lower left corner
                    coordinates of the 5120m.x2304m. map area first. Then after loading of the
                    whole map to transputers, the information about the two points in the area
                    between which LOS calculation will be made is expected to be entered and sent
                    from another server via the communication link established between them. This
                    information then is sent to the transputers and the result is expected from them.
                    When the result is received, it is sent to the station from which the point
                    information comes. This procedure can be repeated as many as the user wants.
********************************************************************************/
/* THIS VERSION OF MANAGER.C IS FOR 15 TRANSPUTERS, THERE ARE 180 WORKERS. */

#include <iostream.h>
#include "unix_comm.h"
#include <fstream.h>
#include "los_com.h"
#include "map_s_com.h"
#include "los_global.h"
#include "map_c.h"

#define NUM_OF_WORKERS 180    /* Each transputer has 12 workers. */

 int org_x, org_y, org1_x, org1_y;
 int x_counter, y_counter, tr_x, tr_y;
 float info[6];
 int size;
 ifstream source;
 int sum;
 float los_result;
 vector agent(0,0,AGENT_HEIGHT);
 double a,b,c,x,y,z;
 int addr = 0;
```

```
int main(void)
{
 s_comm s_comm1(0,0); // output link and input link
 /* Total number of transputers. */
 const int total_prs = 15;
 /* Total number of workers. */
 const int total_n_pr= 180;
 /* Names of transputers. */
 static int trs[total_prs]     = {
  0,1,2,11,21,111,211,1111,2111,11111,21111,111111,211111,1111111,2111111};
 /* The number of children for each transputer for the current topology. */
 static int unders[total_prs] = {2,1,1,1,1,1,1,1,1,1,1,1,1,0,0};
 /* The number of workers for each transputer. */
 static int prs[total_prs]  = {12,12,12,12,12,12,12,12,12,12,12,12,12,12,12};

 /* The distribution of workers to transputers. */
 static int n_tr[total_n_pr] = {
  0,0,0,0,0,0,0,0,0,0,0,0,
  1,1,1,1,1,1,1,1,1,1,1,1,
  2,2,2,2,2,2,2,2,2,2,2,2,
  11,11,11,11,11,11,11,11,11,11,11,11,
  21,21,21,21,21,21,21,21,21,21,21,21,
  111,111,111,111,111,111,111,111,111,111,111,111,
  211,211,211,211,211,211,211,211,211,211,211,211,
  1111,1111,1111,1111,1111,1111,1111,1111,1111,1111,1111,1111,
  2111,2111,2111,2111,2111,2111,2111,2111,2111,2111,2111,2111,
  11111,11111,11111,11111,11111,11111,11111,11111,11111,11111,11111,11111,
  21111,21111,21111,21111,21111,21111,21111,21111,21111,21111,21111,21111,
  111111,111111,111111,111111,111111,111111,111111,111111,
  111111,111111,111111,111111,
  211111,211111,211111,211111,211111,211111,211111,211111,
  211111,211111,211111,211111,
  1111111,1111111,1111111,1111111,1111111,1111111,1111111,1111111,
  1111111,1111111,1111111,1111111,
  2111111,2111111,2111111,2111111,2111111,2111111,2111111,2111111,
  2111111,2111111,2111111,2111111 };
```

```
/* Names of workers in each transputer. */
 static int n_pr[total_n_pr] = {
   0,1,2,3,4,5,6,7,8,9,10,11,
   0,1,2,3,4,5,6,7,8,9,10,11,
   0,1,2,3,4,5,6,7,8,9,10,11,
   0,1,2,3,4,5,6,7,8,9,10,11,
   0,1,2,3,4,5,6,7,8,9,10,11,
   0,1,2,3,4,5,6,7,8,9,10,11,
   0,1,2,3,4,5,6,7,8,9,10,11,
   0,1,2,3,4,5,6,7,8,9,10,11,
   0,1,2,3,4,5,6,7,8,9,10,11,
   0,1,2,3,4,5,6,7,8,9,10,11,
   0,1,2,3,4,5,6,7,8,9,10,11,
   0,1,2,3,4,5,6,7,8,9,10,11,
   0,1,2,3,4,5,6,7,8,9,10,11,
   0,1,2,3,4,5,6,7,8,9,10,11,
   0,1,2,3,4,5,6,7,8,9,10,11 };

s_comm1.router_init(total_prs,trs,unders,prs,100);

 /* User enters the lower left corner coordinates of the whole map here. */
 cout <<"ENTER X COORDINATE FOR ORIGIN : " <<'\n';
 cin >>org_x;
 cout <<"ENTER Y COORDINATE FOR ORIGIN : " <<'\n';
 cin >>org_y;

 for (tr_x=0; tr_x<5; tr_x++){
  for (tr_y=0; tr_y<3; tr_y++){
   for (x_counter=0; x_counter<4; x_counter++){
    for (y_counter=0; y_counter<3; y_counter++){
     org1_x=org_x+(tr_x*4*256)+x_counter*256;
     org1_y=org_y+(tr_y*3*256)+y_counter*256;

     map_c map1(org1_x, org1_y, MAP_SIZE,MAP_SIZE,GRID_SIZE);
     map c_map;
     map_s_com map_s_com;

 /* Sends map */
 c_map = map1.map_c_to_map();
```

```
/* Conversion of map_c class to map class before sending is done*/
  if (addr < total_n_pr) {
    map_s_com.map_send(n_tr[addr], n_pr[addr], c_map, s_comm1);
  }
  addr++; /* Determines the worker address for map portion to be sent. */
}
     }
     cout<<"12 blocks  of elevation data sent to transputer "
<<addr/12<<'\n';
   }
  }
  cout<<"Each 15 transputer is loaded with 12 blocks  of elevation data "<<'\n';
  CMD_INFO cmd_info;

  cout<<"The server is ready to receive the start and goal point information !"
       <<'\n";

/* The communication link is established between two Sun stations here and the
    information of two points in the area for LOS calculation is received. */

/* Opens socket on server */
  if (open_stream_s(SERVER_PORT_NUMBER) < 0)
    cout <<"Error open \n";

for (;;) {

    if ( receive_buf_s((char*)info,&size) < 0) cout << "Error in receiving \n";

      a=double(info[0]);
      b=double(info[1]);
      c=double(info[2]);
      x=double(info[3]);
      y=double(info[4]);
      z=double(info[5]);

      vector start(a,b,c);
      start = start + agent;

      vector goal(x,y,z);
      goal = goal + agent;
```

```
    cmd_info.start = start;
    cmd_info.goal = goal;
    s_comm1.bcast_d(sizeof(cmd_info),(char*)&cmd_info,50);
    sum = s_comm1.listen(100);

   /* The LOS result will be "0" if LOS exists, or will be "1" if LOS doesn't exist and it will be sent
     to the server which represents Janus. */
    if (sum!=0)
     los_result=float(sum/sum);
    else
     los_result=float(sum);

    cout << "Sum is " << dec << sum << '\n' << flush;
    cout << "LOS Result is " << dec << los_result << '\n' << flush;

    send_buf_s((char*)&los_result, sizeof(float));
}
 s_comm1.terminate(50);

}
```

```
/*********************************************************************************
FILENAME ............: client_main.C
AUTHOR................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.......................: October 1993
DESCRIPTION .......: This program runs in a server other than the one in which the main program runs.
                  The user is asked to enter the information about the two points in the area
                  between which LOS calculation will be made.This information is sent to the
                  main server via the communication link established between them. Ideally the
                  sender is considered to be Janus.After sending the point information, the result
                  is expected from the main server. When the result is received, it is displayed on
                  the screen.This procedure can be repeated as many as the user wants.
*********************************************************************************/

#include <iostream.h>
#include "unix_comm.h"

void main(int argc, char *argv[2])
{
 float a,b,c,x,y,z;
 float buf[6];
 int size;
 float *sum;

 if (open_stream_c(argv[1],CLIENT_PORT_NUMBER) < 0)
  cout << "Error open \n";

  for (;;) {

   cout << "Enter the x-coordinate of start point :"<<'\n";
   cin >>a;
   buf[0]=a;

   cout << "Enter the y-coordinate of start point :"<<'\n";
   cin >>b;
   buf[1]=b ;

   cout << "Enter the height of start point :"<<'\n";
   cin >>c;
   buf[2]=c;

   cout << "Enter the x-coordinate of goal point :"<<'\n";
```

```
    cin >>x;
    buf[3]=x;

    cout << "Enter the y-coordinate of goal point :"<<'\n";
    cin >>y;
    buf[4]=y ;

    cout << "Enter the height of start point :"<<'\n";
    cin >>z;
    buf[5]=z ;

    send_buf_c((char *)buf,sizeof(float)*6);
    cout << "Two  points  sent to server\n";

    receive_buf_c((char *)buf,&size);
    sum = (float *)buf;

    cout << "Result is :" <<  *sum << " \n";

    cout << " If you want to continue, type 'y' \n";
    char ch;
    cin >> ch;
    if (ch == 'n') break;
  }
  close_stream_c();
}
```

APPENDIX B - HOST COMPUTER (PC) SOURCE CODE

This appendix contains the source listings of the C++ code developed for the host computer which is a PC that is used in this thesis. They are stored in files as listed below:

1. line.h
2. los_com.h
3. map.h
4. map_crx.h
5. plane.h
6. rout_cmd.h
7. router.h
8. router2.h
9. router3.h
10. s_los.h
11. tr_comm.h
12. vector.h
13. line.cpp
14. map.cpp
15. map_crx.cpp
16. plane.cpp
17. router.cpp
18. routert.cpp
19. router2.cpp
20. router3.cpp
21. s_los.cpp
22. tr_comm.cpp
23. tr_commt.cpp
24. vector.cpp

25. worker.cpp
26. worker.lnk
27. btest180.cfg

```
/***********************************************************************************
FILENAME.............: line.h
AUTHOR.................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.......................: September 1993
DESCRIPTION .......: Header file for description of line equation class and its functions.
***********************************************************************************/
#ifndef LINE_H
#define LINE_H

#include "vector.h"


class line {                    /* X = Pt + PX0 */

  vector start;
  vector direction;
public:
  line() { };
  line(vector pt1, vector dir);

  vector get_start() {return start;};
  vector get_direction() {return direction;};

};

#endif LINE_H
```

```
/*************************************************************************************
FILENAME ............: los_com.h
AUTHOR.................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.......................: September 1993
DESCRIPTION .......: Header file for two structs. One of them is for information about map and the
                     other is for information about two points in the area.
*************************************************************************************/
#ifndef LOS_COM_H
#define LOS_COM_H

#include "vector.h"

/* Contains the lower left corner coordinates, the size and the grid size of map portion which is sent to
transputers at a time. */
struct MAP_INFO{
  int start_x, start_y, size_x, size_y;
  double grid_size;
};

/* Contains two vectors which have the information of two points between which LOS calculation is
made. */
struct CMD_INFO{
  vector start, goal;
};

#endif LOS_COM_H
```

```
/**************************************************************************************
FILENAME ............: map.h
AUTHOR................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE......................: September 1993
DESCRIPTION .......: Header file for the declarations of the map class and the map class functions.
**************************************************************************************/
#ifndef MAP_H
#define MAP_H
#include "vector.h"
class map {
public:
   struct map_rep{
   int start_x, start_y, size_x, size_y;
   double grid_size;
   int* data;
   int refs;
   map_rep() {refs = 1;}
  };
  map_rep *p;
  map();                    /* Constructors */
  map(int start_x,int start_y,int size_x,int size_y,double grid_size,int* data);
  map(const map& map); /* Copy constructor */
  map& operator=(const map& map); /* Assignment operator*/
  ~map();
  /* Gets the lower left corner coordinates, the size and the grid size information of map. */
   int get_start_x() {return p->start_x;};
  int get_start_y() {return p->start_y;};
  int get_size_x() {return p->size_x;};
  int get_size_y() {return p->size_y;};
  double get_grid_size() {return p->grid_size;};
  int* get_data() {return p->data;};

  vector to_map_coord(vector loc);
  int higher_than(vector& loc);
  int terrain_height(int& grid_x, int& grid_y);
  int map_post(int grid_x, int grid_y);
};

#endif MAP_H
```

```
/*************************************************************************************
FILENAME .............: map_crx.h
AUTHOR.................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.......................: September 1993
DESCRIPTION .......: Header file for the source code which checks whether LOS passes through a map
                     contained in a transputer.
*************************************************************************************/
#ifndef MAP_CRX_H
#define MAP_CRX_H

#include "plane.h"
#include "map.h"

class map_crx {
  double map_x_min, map_y_min, map_x_max, map_y_max;
public:
  map_crx() {};
  map_crx(map map1);

  void set_value(map map1);
  int inside_p(vector pt);
  int map_crossing(vector p1, vector p2, vector& start, vector& end);
  int map_intersect(vector p1, vector p2, vector& start, vector& end);
};

#endif MAP_CRX_H
```

```
/*********************************************************************************
FILENAME .............: plane.h
AUTHOR.................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.......................: September 1993
DESCRIPTION .......: Header file for description of plane class and its functions.
*********************************************************************************/
#ifndef PLANE_H
#define PLANE_H

#include "vector.h"
#include "line.h"

class plane {
  vector unit_normal; /*unit normal vector */
  double distance; /* -distance from origin */
public:
  plane() {};
  plane(vector normal, double dist) {
  unit_normal = normal.normalize();
  distance = dist;
 }

 /* If line is parallel to a plane, then 1e100 is returned */
 /* If line is parallel to a plane and on the plane,  this routine also return 1e100. */
 /* If start of a line touches a plane without being  parallel to the plane, then it will return zero distance */
 double plane_distance(vector velocity, vector position);
 int plane_intersection(line line, vector& pt, double& distance);
 int plane_line_cross(line line1, vector& pt, double& distance);

};

#endif
```

```
/************************************************************************************
FILENAME.............: rout_cmd.h
AUTHOR.................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.......................: September 1993
DESCRIPTION .......: Header file which contains the routing information for use of all routing source
                     codes.
************************************************************************************/
#ifndef ROUT_CMD_H
#define ROUT_CMD_H

#define ROUTER_BUF_SIZE 1024

/* Network definition (actually tree)
master
router0 -- workers
router1router2
router11 router12   router21 router22 router23
      ...   ...     ...  ...   ...
one node can have up to three descendant nodes.
one node can have many workers.
*/

/*
ID number for router12 is  1001
ID number for router123 is  111001
*/

/*
Task number
start from 0!!!! (cf, routers, 0, 1,2, 11,12,13, 21,22 ..)
For example, first task connected router 12 is task120 and
 NTS field in send_map is 0.
*/

/*
Port Numbers
    0: upper
1,2,3 : lower (may none connected)
  4 .. : tasks
*/
```

```
/* init message format (cmd=0 or 1) */
/* 0   cmd  #_of_tasks #_of_lower_router destination current_level*/
/* 1   3     4      4          16          4 bits */
/* 0   CMD     NTS      LOW          DST          CLL   */

/* send_map message format (cmd=2) */
/* 0   cmd  task#  ???  destination  ???*/
/* 1   2   4    4   16     4  bits*/
/* 0   CMD  NTS    ???   DST     ??? */
/*  map-size */
/*   32    */
/*  map data */
/*  variable length */

/* bcast_req message format (cmd=3) */
/* 0   cmd  size  ???*/
/* 1   3   8   20 bits */
/* 0   CMD   BCS  ???   */
/* BCS size message follows */

/* terminate message format (cmd=4) */
/* 0   cmd  ??????  */
/* 1   3   28 bits */
/* 0   CMD  ???    */

/*  cmd 0 : init (start)
1 : terminate init
2 : send map
3 : bcast reqest (los request, automatically replied by workers)
4 : terminate
*/

#define START_INIT 0
#define TERMINATE_INIT 1
#define SEND_MAP 2
#define BCAST_REQ 3
#define TERMINATE 4

#define ROUTE_CMD_MASK  0x70000000
#define ROUTE_NTS_MASK  0x0F000000
#define ROUTE_LOW_MASK  0x00F00000
```

```
#define ROUTE_DST_MASK 0x000FFFF0
#define ROUTE_CLL_MASK 0x0000000F
#define ROUTE_BCS_MASK 0x0FF00000

#define ROUTE_CMD_SHIFT 0x10000000
#define ROUTE_NTS_SHIFT 0x01000000
#define ROUTE_LOW_SHIFT 0x00100000
#define ROUTE_DST_SHIFT 0x00000010
#define ROUTE_CLL_SHIFT 0x00000001
#define ROUTE_BCS_SHIFT 0x00100000

/* Use divides and multiplies instead of shifts for speed */
#define ROUTE_UNPACK_CMD(n) ((n & ROUTE_CMD_MASK) / ROUTE_CMD_SHIFT)
#define ROUTE_UNPACK_NTS(n) ((n & ROUTE_NTS_MASK) / ROUTE_NTS_SHIFT)
#define ROUTE_UNPACK_LOW(n) ((n & ROUTE_LOW_MASK) / ROUTE_LOW_SHIFT)
#define ROUTE_UNPACK_DST(n) ((n & ROUTE_DST_MASK) / ROUTE_DST_SHIFT)
#define ROUTE_UNPACK_CLL(n) ((n & ROUTE_CLL_MASK) / ROUTE_CLL_SHIFT)
#define ROUTE_UNPACK_BCS(n) ((n & ROUTE_BCS_MASK) / ROUTE_BCS_SHIFT)


#define ROUTE_PACK_CMD(p,n) p=(p & (~ROUTE_CMD_MASK)) | (n*ROUTE_CMD_SHIFT)
#define ROUTE_PACK_NTS(p,n) p=(p & (~ROUTE_NTS_MASK)) | (n*ROUTE_NTS_SHIFT)
#define ROUTE_PACK_LOW(p,n) p=(p & (~ROUTE_LOW_MASK)) | (n*ROUTE_LOW_SHIFT)
#define ROUTE_PACK_DST(p,n) p=(p & (~ROUTE_DST_MASK)) | (n*ROUTE_DST_SHIFT)
#define ROUTE_PACK_CLL(p,n) p=(p & (~ROUTE_CLL_MASK)) | (n*ROUTE_CLL_SHIFT)
#define ROUTE_PACK_BCS(p,n) p=(p & (~ROUTE_BCS_MASK)) | (n*ROUTE_BCS_SHIFT)

#endif ROUT_CMD_H
```

```
/*****************************************************************************************
FILENAME ............: router.h
AUTHOR................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE......................: September 1993
DESCRIPTION .......: Header file for the source code which performs the routing for the current
                     topology of transputer network.
*****************************************************************************************/
#ifndef ROUTER_H
#define ROUTER_H

#include <chan.h>
#include "rout_cmd.h"

/*
Port Numbers

 0: upper
1,2,3 : lower (may none connected)
4 .. : tasks
*/
#define UPPER_PORT 0
#define FIRST_LOWER_PORT_NUMBER 1
#define FIRST_TASK_PORT_NUMBER 4

class router {
  int router_id;
  int level;
  int has_leaf_node_p;
  int last_lower_port_number;
  int last_task_port_number;
  CHAN **in_ports;
  int ins;
  CHAN **out_ports;
  int outs;
  int message;
  char router_buf[ROUTER_BUF_SIZE];

public:
  router(CHAN *in_ports[],int ins, CHAN *out_ports[],int outs);
  void init(void);
  int cmd_type(void);
```

```
  void send_map(void);
  void bcast_req(void);
  void terminate(void);
  void answer(void);
  void trans_map(int port_number,int map_size);
};

#endif ROUTER_H
```

```
/*************************************************************************************
FILENAME.............: router2.h
AUTHOR.................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.......................: September 1993
DESCRIPTION .......: Header file for the source code which performs routing between transputers.
*************************************************************************************/
#ifndef ROUTER2_H
#define ROUTER2_H

#include <chan.h>
#include "rout_cmd.h"

class router2 {
  CHAN **in_ports;
  int ins;
  CHAN **out_ports;
  int outs;

public:
  router2(){ };
  router2(CHAN *in_ports[],int ins, CHAN *out_ports[],int outs);

  void router_init(int dts, int low, int nts);
  void router_init_done(void);
  void send(int dst, int nts, int size, char* buf);
  void bcast(int size, char* buf);
  int listen(void);
  void terminate(void);
};

#endif ROUTER2_H
```

```
/*********************************************************************************
FILENAME ............: router3.h
AUTHOR................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE......................: September 1993
DESCRIPTION .......: Header file for the source code which performs routing  in a transputer.
*********************************************************************************/
#ifndef ROUTER3_H
#define ROUTER3_H

#include <chan.h>
#include "rout_cmd.h"

#define SEND SEND_MAP
#define BCAST BCAST_REQ
/*TERMINATE comes from "rout_cmd.h" */

class router3 {
 CHAN **in_ports;
 int ins;
 CHAN **out_ports;
 int outs;
 int message;
 int return_value;

public:
 router3(CHAN *in_ports1[],int ins1, CHAN *out_ports1[],int outs1);
 int cmd_type(int& size); /* return type as well as size of data */
 void receive(int size, char* buf);
 void answer(int value);
 void terminate(void);
};

#endif ROUTER3_H
```

```
/**************************************************************************************
FILENAME.............: s_los.h
AUTHOR.................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.......................: September 1993
DESCRIPTION .......: Header file for the source code which performs LOS calculations between two
                     points.
**************************************************************************************/
#ifndef S_LOS_H
#define S_LOS_H

#include "vector.h"
#include "map.h"

class s_los {

public:
 s_los() { }
 /* Performs LOS calculations. */
 int do_s_los(vector start, vector goal, map& map1);
};

#endif S_LOS_H
```

```
/*************************************************************************************
FILENAME .............: tr_comm.h
AUTHOR.................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE......................: September 1993
DESCRIPTION .......: Header file for the source code which performs the communication between
                     SUN station and transputers.
*************************************************************************************/
#ifndef TR_COMM_H
#define TR_COMM_H

#include <chan.h>
#include "router2.h"

const int ROUTER_INIT_S = 1;
const int SEND_S = 2;
const int BCAST_S = 3;
const int LISTEN_S = 4;
const int TERMINATE_S = 5;

class tr_comm {
  router2 router2a;
  CHAN **in_ports;
  int ins;
  CHAN **out_ports;
  int outs;

public:
  tr_comm(CHAN *in_ports[], int ins, CHAN *out_ports[], int outs);

  int cmd_type();    /* Return type */

  void router_init(void);
  void send(void);
  void bcast(void);
  void listen(void);
  void terminate(void);
};

#endif TR_COMM_H
```

```
/*************************************************************************************
FILENAME ............: vector.h
AUTHOR................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE......................: September 1993
DESCRIPTION .......: Header file for the description of the vector class and vector class operations.
*************************************************************************************/
#ifndef VECTOR_H
#define VECTOR_H

class vector {
 double x,y,z;
public:
 vector();
 vector(double x1, double y1, double z1);

 double get_x() {return x;};
 double get_y() {return y;};
 double get_z() {return z;};

 friend int operator==(vector v1, vector v2);
 friend vector operator+(vector v1, vector v2);
 friend vector operator-(vector v1, vector v2);
 friend vector operator*(double a, vector v1);
 double dotprod(vector v1);
 double magnitude(void);
 vector normalize(void);
};
#endif vector_H
```

```
/*************************************************************************************
FILENAME.............: line.cpp
AUTHOR.................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.......................: September 1993
DESCRIPTION .......: This source code is for a line equation.
*************************************************************************************/
#include "line.h"

line::line(vector pt1, vector dir)
{ start = pt1; direction = dir; }
```

```
/*********************************************************************************
FILENAME ............: map.cpp
AUTHOR................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.......................: September 1993
DESCRIPTION .......: This source code defines the map class functions.
*********************************************************************************/
#include "map.h"

map::map()          /* Constructor */
{
 p = new map_rep;
 p->start_x = 0; p->start_y = 0; p->size_x = 0; p->size_y = 0;
 p->grid_size = 0.0;
 p->data = 0; // null pointer
}

map::map(int start_x,int start_y,int size_x,int size_y,double grid_size,int* data) /* Constructor */
{
 p = new map_rep;
 p->start_x = start_x; p->start_y = start_y;
 p->size_x = size_x; p->size_y = size_y;
 p->grid_size = grid_size;
 p->data = data;
}

map::map(const map& map)          /* Copy constructor */
{
 map.p->refs++;
 p = map.p;
}

map& map::operator=(const map& map)        /* Assignment operator */
{
 map.p->refs++;
 if (--p->refs == 0) {
  delete[] p->data;
 delete p;
 }
 p = map.p;
 return *this;
}
```

```
map::~map()        /* Destructor */
{
 if (--(p->refs) == 0) {
  delete[] p->data;
  delete p;
 }
}

vector map::to_map_coord(vector loc)
{
 vector map_offset(((double)p->start_x)*p->grid_size,
((double)p->start_y)*p->grid_size,0);

 vector loc_wrt_map = loc - map_offset;
 return (loc_wrt_map);
}

int map::higher_than(vector& loc)
{
 int grid_x = (int) ((loc.get_x() - p->start_x*p->grid_size)/p->grid_size);
 int grid_y = (int) ((loc.get_y() - p->start_y*p->grid_size)/p->grid_size);
 return(p->data[grid_y*p->size_x+grid_x] > loc.get_z());
}

int map::terrain_height(int& grid_x, int& grid_y)
{
 return map_post(grid_x,grid_y);
}
 int map::map_post(int grid_x, int grid_y)
{
 int index;
 index = p->size_x*grid_y + grid_x;
 return p->data[index];
}
```

```
/*********************************************************************************
FILENAME ............: map_crx.cpp
AUTHOR.................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.......................: September 1993
DESCRIPTION .......: This source file checks whether LOS passes through a map contained in a
                     transputer or not.
*********************************************************************************/
#include "map_crx.h"

map_crx::map_crx(map map1)
{
  map_x_min = double(map1.get_start_x()) * map1.get_grid_size();
  map_y_min = double(map1.get_start_y()) * map1.get_grid_size();
  map_x_max = map_x_min + double(map1.get_size_x()) * map1.get_grid_size();
  map_y_max = map_y_min + double(map1.get_size_y()) * map1.get_grid_size();
}

void map_crx::set_value(map map1)
{
  map_x_min = double(map1.get_start_x()) * map1.get_grid_size();
  map_y_min = double(map1.get_start_y()) * map1.get_grid_size();
  map_x_max = map_x_min + double(map1.get_size_x()) * map1.get_grid_size();
  map_y_max = map_y_min + double(map1.get_size_y()) * map1.get_grid_size();
}

int map_crx::inside_p(vector pt)
{
 /* inside_p includes boundary too. */
 double delta = 0.00005;
 if ((pt.get_x() > map_x_min-delta) && (pt.get_x() < map_x_max+delta) &&
  (pt.get_y() > map_y_min-delta) && (pt.get_y() < map_y_max+delta))
   return(1);
 else
  return(0);
 }
```

```
int map_crx::map_crossing(vector p1, vector p2,
 vector& start, vector& end)
{
 vector px1,px2;

 if ((inside_p(p1))&&(inside_p(p2))) {
  start = p1;
  end = p2;
  return 1;}
 else {
  if (inside_p(p1)) {
   map_intersect(p1,p2,px1,px2);
   start = p1;
   end = px1;
   return 1;}
  else if (inside_p(p2)) {
   start = p2;
   map_intersect(p1,p2,px1,px2);
   end = px1;
   return 1;}
  else {
   if (map_intersect(p1,p2,px1,px2)) {
    start = px1;
    end = px2;
    return 1;}
   else
    return 0;
   }
  }
 }


int map_crx::map_intersect(vector p1, vector p2, vector& px1, vector& px2)
{
  /* This routine returns two intersection pts: px1, px2 */
  /* If they are identical, then px1 = px2 */
  /* If 3D pts, p1 & p2, are given, then px1 and px2 are 3D pts */

  vector pt, pts[2];
  double dist;
```

```
vector x_normal(1,0,0), y_normal(0,1,0);

plane plane_x1(x_normal, -1.0*map_x_min);
plane plane_x2(x_normal, -1.0*map_x_max);
plane plane_y1(y_normal, -1.0*map_y_min);
plane plane_y2(y_normal, -1.0*map_y_max);
vector delta = p2 - p1;
line line1(p1,p2-p1);

int num = 0;
/* There are two distinct pts */
if (plane_x1.plane_line_cross(line1, pt, dist))
 if (inside_p(pt)) {
   pts[num] = pt;
   num++; }
 if (plane_x2.plane_line_cross(line1, pt, dist))
   if ((inside_p(pt) && num && !(pts[num-1]==pt)) || inside_p(pt)) {
    pts[num] = pt;
    num++; }
 if (plane_y1.plane_line_cross(line1, pt, dist)) {
  if ((inside_p(pt) && num && !(pts[num-1]==pt)) || inside_p(pt)) {
    pts[num] = pt;
    num++; }
 }
 if (plane_y2.plane_line_cross(line1, pt, dist))
  if ((inside_p(pt) && num && !(pts[num-1]==pt)) || inside_p(pt)) {
    pts[num] = pt;
    num++; }
 if (num == 0)
  return 0;
 else if (num==1){
  px1 = pts[0];
  px2 = pts[0];
  return 1;}
 else {
  px1 = pts[0];
  px2 = pts[1];
  return 1;}
 }
```

```
/*********************************************************************************
FILENAME ............: plane.cpp
AUTHOR................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE......................: September 1993
DESCRIPTION .......: This source code is for plane equations and functions.
*********************************************************************************/
#include "plane.h"
#include "vector.h"
#include <math.h>

double plane::plane_distance (vector velocity, vector position)
{
 /* plane (X-Q)N=0, line X=P+tA.
 t = (Q-P)N/(AN), if A is normalized then t is signed distance.
 If t is infinitive, then plane-distance returns NULL.
 otherwise, plane-distance returns distance. */
 vector A = velocity.normalize();
 vector N = unit_normal;
 double dis = -1.0 * distance;
 vector Q = dis * N;
 vector Q_P = Q - position;
 double AN = A.dotprod(N);
 double numerator = Q_P.dotprod(N);

 if (fabs(AN) < 1E-100)
  return(1E100);
 else
  return(numerator/AN);
}

int plane::plane_intersection(line line1, vector& pt, double& distance)
{
  vector velocity = line1.get_direction().normalize();
  distance = (*this).plane_distance(velocity, line1.get_start());
  if (distance < 1E100) {
   pt = line1.get_start() + distance * velocity;
   return 1;}
  else
   return 0;
}
```

```
int plane::plane_line_cross(line line1, vector& pt, double& distance)
{
  vector velocity = line1.get_direction().normalize();
  distance = (*this).plane_distance(velocity, line1.get_start());
  if ((distance >= 0) && (distance < line1.get_direction().magnitude())) {
   pt = line1.get_start() + distance * velocity;
   return 1;}
  else
   return 0;
}
```

```
/*************************************************************************************
FILENAME ............: router.cpp
AUTHOR................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.....................: September 1993
DESCRIPTION .......: This is the main routing source code. It handles routing for the current topology
                     of transputer network.
*************************************************************************************/
#include "router.h"
#include <alt.h>
#include <chan.h>

router::router(CHAN *in_ports1[],int ins1, CHAN *out_ports1[],int outs1)
{
 in_ports= in_ports1;
 ins = ins1;
 out_ports = out_ports1;
 outs = outs1;
}

int next_address(int destination, int current_level)
{
 return((destination >> (current_level * 2)) & 0x00000003);
}

void router::init(void)
{
 /* message format */
 /* 0 cmd #_of_tasks #_of_lower_router destination current_level*/
 /* 1 3 4 4 16 4 bits */
 /* 0 CMD NTS LOW DST CLL */

 /* cmd 1 : init (start)
 2 : terminate init
 */

for (;;) {
 int message;
 chan_in_word(&message,in_ports[0]);

 /* Checks whether to terminate init routine.
    This is detected by the first node. */
```

```
if (ROUTE_UNPACK_CMD(message) == TERMINATE_INIT) {
  for (int i=FIRST_LOWER_PORT_NUMBER; i<=last_lower_port_number; i++)
   chan_out_word(message,out_ports[i]);
   break;
  } /* If there is no lower routers, then it automatically does not
       send anything. */

   int destination = ROUTE_UNPACK_DST(message);
   int current_level = ROUTE_UNPACK_CLL(message);
   int next_chan = next_address(destination, current_level);

   if (!next_chan) {          /* This is the destination. */
     router_id = destination;  /* Destination is ID. */
     level = current_level;
     int num_of_lower_routers = ROUTE_UNPACK_LOW(message);
     last_lower_port_number = num_of_lower_routers + FIRST_LOWER_PORT_NUMBER-1;
     /* 0, 1.. num_of_lower_routers, task_ports.... */
      int num_of_tasks = ROUTE_UNPACK_NTS(message);
     last_task_port_number = num_of_tasks + FIRST_TASK_PORT_NUMBER-1;
     if (num_of_lower_routers != 0)
       has_leaf_node_p = 1;
     else
       has_leaf_node_p = 0;
     }
   else {
     message++;          /* Increments current_level counter. */
    chan_out_word(message, out_ports[next_chan]);
   }
  }
}

int router::cmd_type(void)
{
 chan_in_word(&message,in_ports[0]);
 return(ROUTE_UNPACK_CMD(message));
}

void router::trans_map(int port_number, int map_size)
{
 chan_out_word(message,out_ports[port_number]);         /* Sends header first. */
 int num_of_packets = map_size / ROUTER_BUF_SIZE + 1;
```

```
 int last_packet_size = map_size % ROUTER_BUF_SIZE;
 chan_out_word(map_size,out_ports[port_number]);  /* Sends map size. */
 while (num_of_packets>0) {
  if (num_of_packets==1)
   if (last_packet_size > 0) {
    chan_in_message(last_packet_size,router_buf,in_ports[0]);
    chan_out_message(last_packet_size,router_buf,out_ports[port_number]);}
   else {/* nothing to transfer */}
  else {
   chan_in_message(ROUTER_BUF_SIZE,router_buf,in_ports[0]);
   chan_out_message(ROUTER_BUF_SIZE,router_buf,out_ports[port_number]);
  }
  num_of_packets--;
 }
}

void router::send_map(void)
{
 int map_size;
 chan_in_word(&map_size,in_ports[0]);

 int destination = ROUTE_UNPACK_DST(message);
 /* Two cases: This node's task or pass down */
 int next_chan = next_address(destination, level);

 if (!next_chan) {  /* This is the destination. */
 /* Gets task number. */
  int task_port_number = ROUTE_UNPACK_NTS(message)+FIRST_TASK_PORT_NUMBER;
  trans_map(task_port_number,map_size); }
 else
  trans_map(next_chan,map_size);
 }

void router::bcast_req(void)
{
 int size = ROUTE_UNPACK_BCS(message);
 chan_in_message(size,router_buf,in_ports[0]);
 for (int i=FIRST_LOWER_PORT_NUMBER; i<=last_lower_port_number; i++) {
  chan_out_word(message,out_ports[i]);  /* Sends down */
  chan_out_message(size,router_buf,out_ports[i]);
 }
```

```
  for (i=FIRST_TASK_PORT_NUMBER; i<=last_task_port_number; i++) {
   chan_out_word(message,out_ports[i]);    /* Sends down. */
   chan_out_message(size,router_buf,out_ports[i]);
  }
}

void router::answer(void)
{
 int sum = 0;     /* Should be zero, now just testing mode. */
 int lower_sum, task_sum=0;
 int chan;

 for (int i=FIRST_TASK_PORT_NUMBER; i<=last_task_port_number;i++) {
  chan = alt_wait_vec(ins, in_ports);
  chan_in_word(&task_sum,in_ports[chan]);
  sum = sum + task_sum;
 }
 for (i=FIRST_LOWER_PORT_NUMBER; i<=last_lower_port_number; i++) {
  chan = alt_wait_vec(ins, in_ports);
  chan_in_word(&lower_sum,in_ports[chan]);
  sum = sum + lower_sum;
 }
 chan_out_word(sum,out_ports[0]);
}

void router::terminate(void)
{
 for (int i=FIRST_LOWER_PORT_NUMBER; i<=last_lower_port_number; i++)
   chan_out_word(message,out_ports[i]);
 for (i=FIRST_TASK_PORT_NUMBER; i<=last_task_port_number; i++)
    chan_out_word(message,out_ports[i]);
}
```

```
/*************************************************************************************
FILENAME ............: routert.cpp
AUTHOR.................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.......................: September 1993
DESCRIPTION .......: This source code performs routing  for transputers.
*************************************************************************************/
#include <chan.h>
#include "router.h"

void main(int argc, char *argv[], char *envp[],
   CHAN *in_ports[], int ins, CHAN *out_ports[], int outs)
{
 int exit_flag = 0;
 router router1(in_ports,ins,out_ports,outs);

 router1.init();

 while (!exit_flag)
  switch (router1.cmd_type()) {
  case SEND_MAP :
   router1.send_map();
   break;
  case BCAST_REQ :
   router1.bcast_req();
   router1.answer();
   break;
  case TERMINATE :
   router1.terminate();
   exit_flag = 1;
   break;
  default:
  /*error */
  break;
 }
}
```

```
/*******************************************************************************************
FILENAME ............: router2.cpp
AUTHOR................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE......................: September 1993
DESCRIPTION .......: This source code performs routing between transputers.
*******************************************************************************************/
#include "router2.h"
#include <iostream.h>

const int OUT_PORT_NUM =2;
const int IN_PORT_NUM = 2;

router2::router2(CHAN *in_ports1[],int ins1, CHAN *out_ports1[],int outs1)
{
 in_ports= in_ports1;
 ins = ins1;
 out_ports = out_ports1;
 outs = outs1;
}

int convert_to_dst(int destination)
{
 /* Destination address does not contain zero. */
 int dst=0;
 int digit = destination % 10;
 destination = destination / 10;
 while (digit) {
  dst = (dst << 2) | digit;
  digit = destination % 10;
  destination = destination / 10;
 }
 return dst;
}

void router2::router_init(int destination, int low, int nts)
{
 int message = 0;
 int cll = 0;  /* Current level number */
 ROUTE_PACK_CMD(message,START_INIT);
 ROUTE_PACK_NTS(message,nts);
 ROUTE_PACK_LOW(message,low);
```

```
 ROUTE_PACK_DST(message,convert_to_dst(destination));
 ROUTE_PACK_CLL(message,cll);
 chan_out_word(message,out_ports[OUT_PORT_NUM]);
}

void router2::router_init_done(void)
{
 int message = 0;
 ROUTE_PACK_CMD(message,TERMINATE_INIT);
 ROUTE_PACK_LOW(message,2);
 chan_out_word(message,out_ports[OUT_PORT_NUM]);
}

void router2::terminate(void)
{
 int message = 0;
 ROUTE_PACK_CMD(message,TERMINATE);
 chan_out_word(message,out_ports[OUT_PORT_NUM]);
}

void router2::send(int destination, int nts, int size, char* buf)
{
 int message = 0;
 ROUTE_PACK_CMD(message, SEND_MAP);
 ROUTE_PACK_NTS(message, nts);
 ROUTE_PACK_DST(message, convert_to_dst(destination));
 /* Sends "header" */
 chan_out_word(message,out_ports[OUT_PORT_NUM]);
 /* Sends "size" */
 chan_out_word(size,out_ports[OUT_PORT_NUM]);
 char* bp = buf;

 int num_of_packets = size / ROUTER_BUF_SIZE + 1;
 int last_packet_size = size % ROUTER_BUF_SIZE;
 while (num_of_packets>0) {
  if (num_of_packets==1)
   if (last_packet_size > 0) {
    chan_out_message(last_packet_size,bp,out_ports[OUT_PORT_NUM]);}
   else { /* Nothing to send */ }
```

```
  else {
   chan_out_message(ROUTER_BUF_SIZE,bp,out_ports[OUT_PORT_NUM]);
   bp += ROUTER_BUF_SIZE;
  }
  num_of_packets--;
 }
}

void router2::bcast(int size, char* buf)
{
 int message = 0;
 ROUTE_PACK_CMD(message,BCAST_REQ);
 ROUTE_PACK_BCS(message,size);
 chan_out_word(message,out_ports[OUT_PORT_NUM]);
 chan_out_message(size,buf,out_ports[OUT_PORT_NUM]);
}

int router2::listen(void)
{
 int message;
 chan_in_word(&message, in_ports[IN_PORT_NUM]);
 return message;
}
```

```
/*******************************************************************************
FILENAME ............: router3.cpp
AUTHOR................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.......................: September 1993
DESCRIPTION .......: This source code performs routing for workers.
*******************************************************************************/
#include "router3.h"

router3::router3(CHAN *in_ports1[],int ins1, CHAN *out_ports1[],int outs1)
{
 in_ports= in_ports1;
 ins = ins1;
 out_ports = out_ports1;
 outs = outs1;
}

int router3::cmd_type(int& size)
{
 chan_in_word(&message,in_ports[0]);
 int cmd = ROUTE_UNPACK_CMD(message);
 if (cmd == SEND)
  chan_in_word(&size,in_ports[0]);
 else
  size = ROUTE_UNPACK_BCS(message);
  return(cmd);
}

void router3::receive(int size, char* buf)
{
 char* bp = buf;

 int num_of_packets = size / ROUTER_BUF_SIZE + 1;
 int last_packet_size = size % ROUTER_BUF_SIZE;

 while (num_of_packets>0) {
  if (num_of_packets==1)
   if (last_packet_size > 0) {
    chan_in_message(last_packet_size,bp,in_ports[0]); }
   else { /* Nothing to send */ }
```

```
  else {
    chan_in_message(ROUTER_BUF_SIZE,bp,in_ports[0]);
     bp += ROUTER_BUF_SIZE;
    }
   num_of_packets--;
  }
}

void router3::answer(int value)
{
 chan_out_word(value,out_ports[0]);
}

void router3::terminate(void) { }
```

```
/*************************************************************************************
FILENAME ............: s_los.cpp
AUTHOR.................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.......................: September 1993
DESCRIPTION .......: This source code performs LOS calculations between two points in the map area.
                     Returns 0 if LOS exists, returns 1 otherwise.
*************************************************************************************/
#include <math.h>

#include "s_los.h"
#include "map.h"

int s_los::do_s_los(vector start, vector goal, map& map1)
{
  int steps,i;
  vector del = goal-start;
  int del_xi, del_yi;
  del_xi = (int) (fabs(del.get_x())/ map1.get_grid_size());
  del_yi = (int) (fabs(del.get_y())/ map1.get_grid_size());
  steps = (del_xi > del_yi) ? del_xi : del_yi;

  /* Steps + 1 is necessary , because without adding 1, the last goal point is not tested. */
  vector delta_step = (1.0/steps)*del;
  vector check_loc = start;

  for (i=0;i<steps;i++){
   if (map1.higher_than(check_loc))
    return 1;
    check_loc = check_loc + delta_step;
   }
  return 0;
}
```

```
/*********************************************************************************
FILENAME ............: tr_comm.cpp
AUTHOR................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.......................: September 1993
DESCRIPTION .......: This source code handles the communication between transputers.
*********************************************************************************/
#include "tr_comm.h"
#include <chan.h>
#include <iostream.h>
#include "los_com.h"

const int IN_PORT_NUM=4;
const int OUT_PORT_NUM=4;

tr_comm::tr_comm(CHAN *in_ports1[], int ins1, CHAN *out_ports1[], int outs1)
{
 router2a = router2(in_ports1, ins1, out_ports1, outs1);
 in_ports= in_ports1;
 ins = ins1;
 out_ports = out_ports1;
 outs = outs1;
}

int tr_comm::cmd_type()
{
 int cmd;
 chan_in_word(&cmd,in_ports[IN_PORT_NUM]);
 return(cmd);
}

void tr_comm::router_init(void)
{
 int num_trs;
 int *trs, *unders, *prs;
 chan_in_word(&num_trs,in_ports[IN_PORT_NUM]);

 trs = new int[num_trs];
 unders = new int[num_trs];
 prs = new int[num_trs];

 int size=num_trs*sizeof(int);
```

```
 chan_in_message(size,(char*)trs,in_ports[IN_PORT_NUM]);
 chan_in_message(size,(char*)unders,in_ports[IN_PORT_NUM]);
 chan_in_message(size,(char*)prs,in_ports[IN_PORT_NUM]);

 for (int i=0; i<num_trs; i++)
  router2a.router_init(trs[i],unders[i],prs[i]);

  /* Terminates initialization. */
  router2a.router_init_done();
}

void tr_comm::send(void)
{
 int dst;
 chan_in_word(&dst,in_ports[IN_PORT_NUM]);

 int nts;
 chan_in_word(&nts,in_ports[IN_PORT_NUM]);

 int size;
 char* buf;
 chan_in_word(&size,in_ports[IN_PORT_NUM]);

 buf = new char[size];
 chan_in_message(size,buf,in_ports[IN_PORT_NUM]);

 router2a.send(dst, nts, size, buf );
}

void tr_comm::bcast(void)
{
 int size;
 chan_in_word(&size,in_ports[IN_PORT_NUM]);

 char* buf;
 buf = new char[size];
 chan_in_message(size,buf,in_ports[IN_PORT_NUM]);

 CMD_INFO *cmd_infop;

 cmd_infop = (CMD_INFO*)buf;
```

```
 router2a.bcast(size, buf);
}

void tr_comm::listen(void)
{
 int value = router2a.listen();
 chan_out_word(value,out_ports[OUT_PORT_NUM]);
}

void tr_comm::terminate(void)
{
 router2a.terminate();
}
```

```
/***********************************************************************************
FILENAME ............: tr_commt.cpp
AUTHOR................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE......................: September 1993
DESCRIPTION .......: This source code handles the communication between SUN and transputers.
***********************************************************************************/
#include "tr_comm.h"
#include <iostream.h>

void main(int argc, char *argv[], char *envp[],
 CHAN *in_ports[], int ins, CHAN *out_ports[], int outs)
{
 int exit_flag = 0;
 tr_comm tr_comm1(in_ports,ins,out_ports,outs);
 while (!exit_flag)
  switch (tr_comm1.cmd_type()) {
  case ROUTER_INIT_S:
   tr_comm1.router_init();
   break;
  case SEND_S:
   tr_comm1.send();
   break;
  case BCAST_S:
   tr_comm1.bcast();
   break;
  case LISTEN_S:
   tr_comm1.listen();
   break;
  case TERMINATE_S:
   tr_comm1.terminate();
   exit_flag = 1;
   break;
  default:  /* Error */
  break;
 }
}
```

```
/*******************************************************************************
FILENAME ............: vector.cpp
AUTHOR................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.......................: September 1993
DESCRIPTION .......: This source code defines the vector class operations.
*******************************************************************************/
#include "vector.h"
#include <math.h>

vector::vector() {x=0.0; y=0.0; z=0.0;};

vector::vector(double x1, double y1, double z1) {x=x1; y=y1; z=z1;};

int operator==(vector v1, vector v2)
{
  return((v1.x==v2.x) && (v1.y==v2.y) && (v1.z==v2.z));
}

vector operator+(vector v1, vector v2)
{
  vector v(v1.x+v2.x, v1.y+v2.y, v1.z+v2.z);
  return v;
}

vector operator-(vector v1, vector v2)
{
  vector v(v1.x-v2.x, v1.y-v2.y, v1.z-v2.z);
  return v;
}

vector operator*(double a, vector v1)
{
  vector v(a*v1.x, a*v1.y, a*v1.z);
  return v;
}

double vector::dotprod(vector v2)       /* Dot product */
{
  return(this->x*v2.x + this->y*v2.y + this->z*v2.z);
}
```

```
double vector::magnitude(void)
{
  return(sqrt((*this).dotprod(*this)));
}

vector vector::normalize(void)              /* Vector normalization */
{
  vector result;
  double mag = (*this).magnitude();

  if (mag < 1E-100) {
   result.x = 0.0;
   result.y = 0.0;
   result.z = 0.0;}
  else {
   result = (1.0/mag) * (*this);
  }
  return(result);
}
```

```
/*****************************************************************************************
FILENAME .............: worker.cpp
AUTHOR.................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.......................: September 1993
DESCRIPTION .......: This source code handles the communication between routers and workers,
                     passes all information to workers and gets the result which they found.
*****************************************************************************************/
#include "router3.h"
#include "los_com.h"
#include "s_los.h"
#include "map_crx.h"

int num_cnt(int num, int* buf, int buf_size)
{
 int cnt = 0;
 for (int i=0; i<buf_size; i++)
   if (buf[i]==num) cnt++;
     return cnt;
}

void main(int argc, char *argv[], char *envp[],
 CHAN *in_ports[], int ins, CHAN *out_ports[], int outs)

{
 /* three cases: get_map
 get_req & return answer
 terminate
 */

 int exit_flag = 0;
 int size = 0;
 int* buf;
 int buf_size;
 CMD_INFO cmd_info;
 MAP_INFO map_info;
 vector test_s, test_g;
 int c_result;
 map c_map;
 router3 router3a(in_ports,ins,out_ports,outs);
 map_crx map_crxer;
 s_los los1;
```

```
 while (!exit_flag)
  switch (router3a.cmd_type(size)) {

  case SEND:
     router3a.receive(size,(char*)&map_info);
     router3a.cmd_type(size);
     buf_size = size / 4;
    buf = new int[buf_size];
    router3a.receive(size,(char*)buf);
    c_map = map(map_info.start_x, map_info.start_y,
    map_info.size_x, map_info.size_y,
    map_info.grid_size, buf);
    break;

  case BCAST:
    router3a.receive(size,(char*)&cmd_info);
    map_crxer.set_value(c_map);
    if (map_crxer.map_crossing(cmd_info.start, cmd_info.goal,  test_s, test_g)) {
     c_result = los1.do_s_los(test_s, test_g, c_map);
    }
    else
     c_result = 0;
     router3a.answer(c_result);
     break;

  case TERMINATE:
    router3a.terminate();
    exit_flag = 1;
    break;
  default:    /* Error */
  break;
 }
}
```

```
/*************************************************************************************
FILENAME ............: worker.lnk
AUTHOR.................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
DATE.......................: September 1993
DESCRIPTION .......: Does the necessary links for workers.
*************************************************************************************/

worker.bin
router3.bin
map.bin
map_crx.bin
s_los.bin
plane.bin
line.bin
vector.bin
```

```
!**********************************************************************************
!   FILENAME...............: btest180.cfg
!   AUTHOR..................: Dr. Se-Hung KWAK & Cem Ali DUNDAR
!   DATE.........................: September 1993
!   DESCRIPTION ........: This configuration file are for 15 transputers, one Sun SPARC Station and one
PC Host. !                       There are one router and 12 worker tasks for each transputers.
!**********************************************************************************

processor host
processor sun type=pc
processor root
processor p1
processor p2
processor p11
processor p21
processor p111
processor p211
processor p1111
processor p2111
processor p11111
processor p21111
processor p111111
processor p211111
processor p1111111
processor p2111111

wire ?      root[0]       host[0]
wire ?      root[1]       p1[1]
wire ?      root[2]       p2[2]
wire ?      root[3]       sun[0]
wire ?      p1[2]         p11[1]
wire ?      p2[1]         p21[2]
wire ?      p11[2]        p111[1]
wire ?      p21[1]        p211[2]
wire ?      p111[2]       p1111[1]
wire ?      p211[1]       p2111[2]
wire ?      p1111[2]      p11111[1]
wire ?      p2111[1]      p21111[2]
wire ?      p11111[2]     p111111[1]
wire ?      p21111[1]     p211111[2]
wire ?      p111111[2]    p1111111[1]
wire ?      p211111[1]    p2111111[2]

! Task connected to filter cannot use 0 channel of task therefore, master has to have 5 ins & outs
! Also a channel to filter has to be lowest number.

task afserver          ins=1 outs=1
task filter            ins=2 outs=2 data=15k
task master            ins=5 outs=5 data=15k file="tr_commt.b4"
```

```
task router0            ins=20 outs=20 data=2k file="router.b4"  urgent
task router1            ins=20 outs=20 data=2k file="router.b4"  urgent
task router2            ins=20 outs=20 data=2k file="router.b4"  urgent
task router11           ins=20 outs=20 data=2k file="router.b4"  urgent
task router21           ins=20 outs=20 data=2k file="router.b4"  urgent
task router111          ins=20 outs=20 data=2k file="router.b4"  urgent
task router211          ins=20 outs=20 data=2k file="router.b4"  urgent
task router1111         ins=20 outs=20 data=2k file="router.b4"  urgent
task router2111         ins=20 outs=20 data=2k file="router.b4"  urgent
task router11111        ins=20 outs=20 data=2k file="router.b4"  urgent
task router21111        ins=20 outs=20 data=2k file="router.b4"  urgent
task router111111       ins=20 outs=20 data=2k file="router.b4"  urgent
task router211111       ins=20 outs=20 data=2k file="router.b4"  urgent
task router1111111      ins=20 outs=20 data=2k file="router.b4"  urgent
task router2111111      ins=20 outs=20 data=2k file="router.b4"  urgent

task worker00           ins=1 outs=1 data=275k file="worker.b4"
task worker01           ins=1 outs=1 data=275k file="worker.b4"
task worker02           ins=1 outs=1 data=275k file="worker.b4"
task worker03           ins=1 outs=1 data=275k file="worker.b4"
task worker04           ins=1 outs=1 data=275k file="worker.b4"
task worker05           ins=1 outs=1 data=275k file="worker.b4"
task worker06           ins=1 outs=1 data=275k file="worker.b4"
task worker07           ins=1 outs=1 data=275k file="worker.b4"
task worker08           ins=1 outs=1 data=275k file="worker.b4"
task worker09           ins=1 outs=1 data=275k file="worker.b4"
task worker0100         ins=1 outs=1 data=275k file="worker.b4"
task worker0101         ins=1 outs=1 data=275k file="worker.b4"

task worker10           ins=1 outs=1 data=275k file="worker.b4"
task worker11           ins=1 outs=1 data=275k file="worker.b4"
task worker12           ins=1 outs=1 data=275k file="worker.b4"
task worker13           ins=1 outs=1 data=275k file="worker.b4"
task worker14           ins=1 outs=1 data=275k file="worker.b4"
task worker15           ins=1 outs=1 data=275k file="worker.b4"
task worker16           ins=1 outs=1 data=275k file="worker.b4"
task worker17           ins=1 outs=1 data=275k file="worker.b4"
task worker18           ins=1 outs=1 data=275k file="worker.b4"
task worker19           ins=1 outs=1 data=275k file="worker.b4"
task worker1100         ins=1 outs=1 data=275k file="worker.b4"
task worker1101         ins=1 outs=1 data=275k file="worker.b4"

task worker20           ins=1 outs=1 data=275k file="worker.b4"
task worker21           ins=1 outs=1 data=275k file="worker.b4"
task worker22           ins=1 outs=1 data=275k file="worker.b4"
task worker23           ins=1 outs=1 data=275k file="worker.b4"
task worker24           ins=1 outs=1 data=275k file="worker.b4"
task worker25           ins=1 outs=1 data=275k file="worker.b4"
task worker26           ins=1 outs=1 data=275k file="worker.b4"
task worker27           ins=1 outs=1 data=275k file="worker.b4"
```

```
task worker28          ins=1 outs=1 data=275k file="worker.b4"
task worker29          ins=1 outs=1 data=275k file="worker.b4"
task worker2100        ins=1 outs=1 data=275k file="worker.b4"
task worker2101        ins=1 outs=1 data=275k file="worker.b4"
task worker110         ins=1 outs=1 data=275k file="worker.b4"
task worker111         ins=1 outs=1 data=275k file="worker.b4"
task worker112         ins=1 outs=1 data=275k file="worker.b4"
task worker113         ins=1 outs=1 data=275k file="worker.b4"
task worker114         ins=1 outs=1 data=275k file="worker.b4"
task worker115         ins=1 outs=1 data=275k file="worker.b4"
task worker116         ins=1 outs=1 data=275k file="worker.b4"
task worker117         ins=1 outs=1 data=275k file="worker.b4"
task worker118         ins=1 outs=1 data=275k file="worker.b4"
task worker119         ins=1 outs=1 data=275k file="worker.b4"
task worker11100       ins=1 outs=1 data=275k file="worker.b4"
task worker11101       ins=1 outs=1 data=275k file="worker.b4"

task worker1110        ins=1 outs=1 data=275k file="worker.b4"
task worker1111        ins=1 outs=1 data=275k file="worker.b4"
task worker1112        ins=1 outs=1 data=275k file="worker.b4"
task worker1113        ins=1 outs=1 data=275k file="worker.b4"
task worker1114        ins=1 outs=1 data=275k file="worker.b4"
task worker1115        ins=1 outs=1 data=275k file="worker.b4"
task worker1116        ins=1 outs=1 data=275k file="worker.b4"
task worker1117        ins=1 outs=1 data=275k file="worker.b4"
task worker1118        ins=1 outs=1 data=275k file="worker.b4"
task worker1119        ins=1 outs=1 data=275k file="worker.b4"
task worker111100      ins=1 outs=1 data=275k file="worker.b4"
task worker111101      ins=1 outs=1 data=275k file="worker.b4"

task worker11110       ins=1 outs=1 data=275k file="worker.b4"
task worker11111       ins=1 outs=1 data=275k file="worker.b4"
task worker11112       ins=1 outs=1 data=275k file="worker.b4"
task worker11113       ins=1 outs=1 data=275k file="worker.b4"
task worker11114       ins=1 outs=1 data=275k file="worker.b4"
task worker11115       ins=1 outs=1 data=275k file="worker.b4"
task worker11116       ins=1 outs=1 data=275k file="worker.b4"
task worker11117       ins=1 outs=1 data=275k file="worker.b4"
task worker11118       ins=1 outs=1 data=275k file="worker.b4"
task worker11119       ins=1 outs=1 data=275k file="worker.b4"
task worker1111100     ins=1 outs=1 data=275k file="worker.b4"
task worker1111101     ins=1 outs=1 data=275k file="worker.b4"

task worker111110      ins=1 outs=1 data=275k file="worker.b4"
task worker111111      ins=1 outs=1 data=275k file="worker.b4"
task worker111112      ins=1 outs=1 data=275k file="worker.b4"
task worker111113      ins=1 outs=1 data=275k file="worker.b4"
task worker111114      ins=1 outs=1 data=275k file="worker.b4"
task worker111115      ins=1 outs=1 data=275k file="worker.b4"
task worker111116      ins=1 outs=1 data=275k file="worker.b4"
task worker111117      ins=1 outs=1 data=275k file="worker.b4"
```

```
task worker111118          ins=1 outs=1 data=275k file="worker.b4"
task worker111119          ins=1 outs=1 data=275k file="worker.b4"
task worker11111100        ins=1 outs=1 data=275k file="worker.b4"
task worker11111101        ins=1 outs=1 data=275k file="worker.b4"


task worker1111110         ins=1 outs=1 data=275k file="worker.b4"
task worker1111111         ins=1 outs=1 data=275k file="worker.b4"
task worker1111112         ins=1 outs=1 data=275k file="worker.b4"
task worker1111113         ins=1 outs=1 data=275k file="worker.b4"
task worker1111114         ins=1 outs=1 data=275k file="worker.b4"
task worker1111115         ins=1 outs=1 data=275k file="worker.b4"
task worker1111116         ins=1 outs=1 data=275k file="worker.b4"
task worker1111117         ins=1 outs=1 data=275k file="worker.b4"
task worker1111118         ins=1 outs=1 data=275k file="worker.b4"
task worker1111119         ins=1 outs=1 data=275k file="worker.b4"
task worker111111100       ins=1 outs=1 data=275k file="worker.b4"
task worker111111101       ins=1 outs=1 data=275k file="worker.b4"

task worker11111110        ins=1 outs=1 data=275k file="worker.b4"
task worker11111111        ins=1 outs=1 data=275k file="worker.b4"
task worker11111112        ins=1 outs=1 data=275k file="worker.b4"
task worker11111113        ins=1 outs=1 data=275k file="worker.b4"
task worker11111114        ins=1 outs=1 data=275k file="worker.b4"
task worker11111115        ins=1 outs=1 data=275k file="worker.b4"
task worker11111116        ins=1 outs=1 data=275k file="worker.b4"
task worker11111117        ins=1 outs=1 data=275k file="worker.b4"
task worker11111118        ins=1 outs=1 data=275k file="worker.b4"
task worker11111119        ins=1 outs=1 data=275k file="worker.b4"
task worker1111111100      ins=1 outs=1 data=275k file="worker.b4"
task worker1111111101      ins=1 outs=1 data=275k file="worker.b4"


task worker210             ins=1 outs=1 data=275k file="worker.b4"
task worker211             ins=1 outs=1 data=275k file="worker.b4"
task worker212             ins=1 outs=1 data=275k file="worker.b4"
task worker213             ins=1 outs=1 data=275k file="worker.b4"
task worker214             ins=1 outs=1 data=275k file="worker.b4"
task worker215             ins=1 outs=1 data=275k file="worker.b4"
task worker216             ins=1 outs=1 data=275k file="worker.b4"
task worker217             ins=1 outs=1 data=275k file="worker.b4"
task worker218             ins=1 outs=1 data=275k file="worker.b4"
task worker219             ins=1 outs=1 data=275k file="worker.b4"
task worker21100           ins=1 outs=1 data=275k file="worker.b4"
task worker21101           ins=1 outs=1 data=275k file="worker.b4"

task worker2110            ins=1 outs=1 data=275k file="worker.b4"
task worker2111            ins=1 outs=1 data=275k file="worker.b4"
task worker2112            ins=1 outs=1 data=275k file="worker.b4"
task worker2113            ins=1 outs=1 data=275k file="worker.b4"
```

```
task worker2114            ins=1 outs=1 data=275k file="worker.b4"
task worker2115            ins=1 outs=1 data=275k file="worker.b4"
task worker2116            ins=1 outs=1 data=275k file="worker.b4"
task worker2117            ins=1 outs=1 data=275k file="worker.b4"
task worker2118            ins=1 outs=1 data=275k file="worker.b4"
task worker2119            ins=1 outs=1 data=275k file="worker.b4"
task worker211100          ins=1 outs=1 data=275k file="worker.b4"
task worker211101          ins=1 outs=1 data=275k file="worker.b4"

task worker21110           ins=1 outs=1 data=275k file="worker.b4"
task worker21111           ins=1 outs=1 data=275k file="worker.b4"
task worker21112           ins=1 outs=1 data=275k file="worker.b4"
task worker21113           ins=1 outs=1 data=275k file="worker.b4"
task worker21114           ins=1 outs=1 data=275k file="worker.b4"
task worker21115           ins=1 outs=1 data=275k file="worker.b4"
task worker21116           ins=1 outs=1 data=275k file="worker.b4"
task worker21117           ins=1 outs=1 data=275k file="worker.b4"
task worker21118           ins=1 outs=1 data=275k file="worker.b4"
task worker21119           ins=1 outs=1 data=275k file="worker.b4"
task worker2111100         ins=1 outs=1 data=275k file="worker.b4"
task worker2111101         ins=1 outs=1 data=275k file="worker.b4"

task worker211110          ins=1 outs=1 data=275k file="worker.b4"
task worker211111          ins=1 outs=1 data=275k file="worker.b4"
task worker211112          ins=1 outs=1 data=275k file="worker.b4"
task worker211113          ins=1 outs=1 data=275k file="worker.b4"
task worker211114          ins=1 outs=1 data=275k file="worker.b4"
task worker211115          ins=1 outs=1 data=275k file="worker.b4"
task worker211116          ins=1 outs=1 data=275k file="worker.b4"
task worker211117          ins=1 outs=1 data=275k file="worker.b4"
task worker211118          ins=1 outs=1 data=275k file="worker.b4"
task worker211119          ins=1 outs=1 data=275k file="worker.b4"
task worker21111100        ins=1 outs=1 data=275k file="worker.b4"
task worker21111101        ins=1 outs=1 data=275k file="worker.b4"

task worker2111110         ins=1 outs=1 data=275k file="worker.b4"
task worker2111111         ins=1 outs=1 data=275k file="worker.b4"
task worker2111112         ins=1 outs=1 data=275k file="worker.b4"
task worker2111113         ins=1 outs=1 data=275k file="worker.b4"
task worker2111114         ins=1 outs=1 data=275k file="worker.b4"
task worker2111115         ins=1 outs=1 data=275k file="worker.b4"
task worker2111116         ins=1 outs=1 data=275k file="worker.b4"
task worker2111117         ins=1 outs=1 data=275k file="worker.b4"
task worker2111118         ins=1 outs=1 data=275k file="worker.b4"
task worker2111119         ins=1 outs=1 data=275k file="worker.b4"
task worker211111100       ins=1 outs=1 data=275k file="worker.b4"
task worker211111101       ins=1 outs=1 data=275k file="worker.b4"

task worker21111110        ins=1 outs=1 data=275k file="worker.b4"
task worker21111111        ins=1 outs=1 data=275k file="worker.b4"
task worker21111112        ins=1 outs=1 data=275k file="worker.b4"
```

```
task worker21111113          ins=1 outs=1 data=275k file="worker.b4"
task worker21111114          ins=1 outs=1 data=275k file="worker.b4"
task worker21111115          ins=1 outs=1 data=275k file="worker.b4"
task worker21111116          ins=1 outs=1 data=275k file="worker.b4"
task worker21111117          ins=1 outs=1 data=275k file="worker.b4"
task worker21111118          ins=1 outs=1 data=275k file="worker.b4"
task worker21111119          ins=1 outs=1 data=275k file="worker.b4"
task worker2111111100        ins=1 outs=1 data=275k file="worker.b4"
task worker2111111101        ins=1 outs=1 data=275k file="worker.b4"


!Port numbers 0 ... 3 for routers.
!Port numbers 4 ...  for tasks(workers).

place afserver               host
place filter                 root
place master                 root

place router0                root
place worker00               root
place worker01               root
place worker02               root
place worker03               root
place worker04               root
place worker05               root
place worker06               root
place worker07               root
place worker08               root
place worker09               root
place worker0100             root
place worker0101             root

place router1                p1
place worker10               p1
place worker11               p1
place worker12               p1
place worker13               p1
place worker14               p1
place worker15               p1
place worker16               p1
place worker17               p1
place worker18               p1
place worker19               p1
place worker1100             p1
place worker1101             p1

place router11               p11
place worker110              p11
place worker111              p11
place worker112              p11
```

place worker113	p11
place worker114	p11
place worker115	p11
place worker116	p11
place worker117	p11
place worker118	p11
place worker119	p11
place worker11100	p11
place worker11101	p11

place router111	p111
place worker1110	p111
place worker1111	p111
place worker1112	p111
place worker1113	p111
place worker1114	p111
place worker1115	p111
place worker1116	p111
place worker1117	p111
place worker1118	p111
place worker1119	p111
place worker111100	p111
place worker111101	p111

place router1111	p1111
place worker11110	p1111
place worker11111	p1111
place worker11112	p1111
place worker11113	p1111
place worker11114	p1111
place worker11115	p1111
place worker11116	p1111
place worker11117	p1111
place worker11118	p1111
place worker11119	p1111
place worker1111100	p1111
place worker1111101	p1111

place router11111	p11111
place worker111110	p11111
place worker111111	p11111
place worker111112	p11111
place worker111113	p11111
place worker111114	p11111
place worker111115	p11111
place worker111116	p11111
place worker111117	p11111
place worker111118	p11111
place worker111119	p11111

place worker11111100	p11111
place worker11111101	p11111
place router111111	p111111
place worker1111110	p111111
place worker1111111	p111111
place worker1111112	p111111
place worker1111113	p111111
place worker1111114	p111111
place worker1111115	p111111
place worker1111116	p111111
place worker1111117	p111111
place worker1111118	p111111
place worker1111119	p111111
place worker111111100	p111111
place worker111111101	p111111
place router1111111	p1111111
place worker11111110	p1111111
place worker11111111	p1111111
place worker11111112	p1111111
place worker11111113	p1111111
place worker11111114	p1111111
place worker11111115	p1111111
place worker11111116	p1111111
place worker11111117	p1111111
place worker11111118	p1111111
place worker11111119	p1111111
place worker1111111100	p1111111
place worker1111111101	p1111111
place router2	p2
place worker20	p2
place worker21	p2
place worker22	p2
place worker23	p2
place worker24	p2
place worker25	p2
place worker26	p2
place worker27	p2
place worker28	p2
place worker29	p2
place worker2100	p2
place worker2101	p2
place router21	p21
place worker210	p21
place worker211	p21
place worker212	p21
place worker213	p21

place worker214	p21
place worker215	p21
place worker216	p21
place worker217	p21
place worker218	p21
place worker219	p21
place worker21100	p21
place worker21101	p21
place router211	p211
place worker2110	p211
place worker2111	p211
place worker2112	p211
place worker2113	p211
place worker2114	p211
place worker2115	p211
place worker2116	p211
place worker2117	p211
place worker2118	p211
place worker2119	p211
place worker211100	p211
place worker211101	p211
place router2111	p2111
place worker21110	p2111
place worker21111	p2111
place worker21112	p2111
place worker21113	p2111
place worker21114	p2111
place worker21115	p2111
place worker21116	p2111
place worker21117	p2111
place worker21118	p2111
place worker21119	p2111
place worker2111100	p2111
place worker2111101	p2111
place router21111	p21111
place worker211110	p21111
place worker211111	p21111
place worker211112	p21111
place worker211113	p21111
place worker211114	p21111
place worker211115	p21111
place worker211116	p21111
place worker211117	p21111
place worker211118	p21111
place worker211119	p21111
place worker21111100	p21111
place worker21111101	p21111

```
place router211111            p211111
place worker2111110           p211111
place worker2111111           p211111
place worker2111112           p211111
place worker2111113           p211111
place worker2111114           p211111
place worker2111115           p211111
place worker2111116           p211111
place worker2111117           p211111
place worker2111118           p211111
place worker2111119           p211111
place worker211111100         p211111
place worker211111101         p211111

place router2111111           p2111111
place worker21111110          p2111111
place worker21111111          p2111111
place worker21111112          p2111111
place worker21111113          p2111111
place worker21111114          p2111111
place worker21111115          p2111111
place worker21111116          p2111111
place worker21111117          p2111111
place worker21111118          p2111111
place worker21111119          p2111111
place worker2111111100        p2111111
place worker2111111101        p2111111


connect ? afserver[0]         filter[0]
connect ? filter[0]           afserver[0]

connect ? filter[1]           master[1]
connect ? master[1]           filter[1]

connect ? master[2]           router0[0]
connect ? router0[0]          master[2]


connect ? router0[1]          router1[0]
connect ? router1[0]          router0[1]

connect ? router0[2]          router2[0]
connect ? router2[0]          router0[2]


connect ? router0[4]          worker00[0]
connect ? worker00[0]         router0[4]

connect ? router0[5]          worker01[0]
connect ? worker01[0]         router0[5]
```

connect ? router0[6]	worker02[0]
connect ? worker02[0]	router0[6]
connect ? router0[7]	worker03[0]
connect ? worker03[0]	router0[7]
connect ? router0[8]	worker04[0]
connect ? worker04[0]	router0[8]
connect ? router0[9]	worker05[0]
connect ? worker05[0]	router0[9]
connect ? router0[10]	worker06[0]
connect ? worker06[0]	router0[10]
connect ? router0[11]	worker07[0]
connect ? worker07[0]	router0[11]
connect ? router0[12]	worker08[0]
connect ? worker08[0]	router0[12]
connect ? router0[13]	worker09[0]
connect ? worker09[0]	router0[13]
connect ? router0[14]	worker0100[0]
connect ? worker0100[0]	router0[14]
connect ? router0[15]	worker0101[0]
connect ? worker0101[0]	router0[15]
connect ? router1[1]	router11[0]
connect ? router11[0]	router1[1]
connect ? router1[4]	worker10[0]
connect ? worker10[0]	router1[4]
connect ? router1[5]	worker11[0]
connect ? worker11[0]	router1[5]
connect ? router1[6]	worker12[0]
connect ? worker12[0]	router1[6]
connect ? router1[7]	worker13[0]
connect ? worker13[0]	router1[7]
connect ? router1[8]	worker14[0]
connect ? worker14[0]	router1[8]

```
connect ? router1[9]          worker15[0]
connect ? worker15[0]         router1[9]

connect ? router1[10]         worker16[0]
connect ? worker16[0]         router1[10]

connect ? router1[11]         worker17[0]
connect ? worker17[0]         router1[11]

connect ? router1[12]         worker18[0]
connect ? worker18[0]         router1[12]

connect ? router1[13]         worker19[0]
connect ? worker19[0]         router1[13]

connect ? router1[14]         worker1100[0]
connect ? worker1100[0]       router1[14]

connect ? router1[15]         worker1101[0]
connect ? worker1101[0]       router1[15]


connect ? router11[1]         router111[0]
connect ? router111[0]        router11[1]

connect ? router11[4]         worker110[0]
connect ? worker110[0]        router11[4]

connect ? router11[5]         worker111[0]
connect ? worker111[0]        router11[5]

connect ? router11[6]         worker112[0]
connect ? worker112[0]        router11[6]

connect ? router11[7]         worker113[0]
connect ? worker113[0]        router11[7]

connect ? router11[8]         worker114[0]
connect ? worker114[0]        router11[8]

connect ? router11[9]         worker115[0]
connect ? worker115[0]        router11[9]

connect ? router11[10]        worker116[0]
connect ? worker116[0]        router11[10]


connect ? router11[11]        worker117[0]
connect ? worker117[0]        router11[11]
```

```
connect ? router11[12]          worker118[0]
connect ? worker118[0]          router11[12]

connect ? router11[13]          worker119[0]
connect ? worker119[0]          router11[13]

connect ? router11[14]          worker11100[0]
connect ? worker11100[0]        router11[14]

connect ? router11[15]          worker11101[0]
connect ? worker11101[0]        router11[15]


connect ? router111[1]          router1111[0]
connect ? router1111[0]         router111[1]

connect ? router111[4]          worker1110[0]
connect ? worker1110[0]         router111[4]

connect ? router111[5]          worker1111[0]
connect ? worker1111[0]         router111[5]

connect ? router111[6]          worker1112[0]
connect ? worker1112[0]         router111[6]

connect ? router111[7]          worker1113[0]
connect ? worker1113[0]         router111[7]

connect ? router111[8]          worker1114[0]
connect ? worker1114[0]         router111[8]

connect ? router111[9]          worker1115[0]
connect ? worker1115[0]         router111[9]

connect ? router111[10]         worker1116[0]
connect ? worker1116[0]         router111[10]

connect ? router111[11]         worker1117[0]
connect ? worker1117[0]         router111[11]

connect ? router111[12]         worker1118[0]
connect ? worker1118[0]         router111[12]

connect ? router111[13]         worker1119[0]
connect ? worker1119[0]         router111[13]

connect ? router111[14]         worker111100[0]
connect ? worker111100[0]       router111[14]

connect ? router111[15]         worker111101[0]
connect ? worker111101[0]       router111[15]
```

connect ? router1111[1]	router11111[0]
connect ? router11111[0]	router1111[1]
connect ? router1111[4]	worker11110[0]
connect ? worker11110[0]	router1111[4]
connect ? router1111[5]	worker11111[0]
connect ? worker11111[0]	router1111[5]
connect ? router1111[6]	worker11112[0]
connect ? worker11112[0]	router1111[6]
connect ? router1111[7]	worker11113[0]
connect ? worker11113[0]	router1111[7]
connect ? router1111[8]	worker11114[0]
connect ? worker11114[0]	router1111[8]
connect ? router1111[9]	worker11115[0]
connect ? worker11115[0]	router1111[9]
connect ? router1111[10]	worker11116[0]
connect ? worker11116[0]	router1111[10]
connect ? router1111[11]	worker11117[0]
connect ? worker11117[0]	router1111[11]
connect ? router1111[12]	worker11118[0]
connect ? worker11118[0]	router1111[12]
connect ? router1111[13]	worker11119[0]
connect ? worker11119[0]	router1111[13]
connect ? router1111[14]	worker1111100[0]
connect ? worker1111100[0]	router1111[14]
connect ? router1111[15]	worker1111101[0]
connect ? worker1111101[0]	router1111[15]
connect ? router11111[1]	router111111[0]
connect ? router111111[0]	router11111[1]
connect ? router11111[4]	worker111110[0]
connect ? worker111110[0]	router11111[4]
connect ? router11111[5]	worker111111[0]
connect ? worker111111[0]	router11111[5]

```
connect ? router11111[6]       worker111112[0]
connect ? worker111112[0]      router11111[6]

connect ? router11111[7]       worker111113[0]
connect ? worker111113[0]      router11111[7]

connect ? router11111[8]       worker111114[0]
connect ? worker111114[0]      router11111[8]

connect ? router11111[9]       worker111115[0]
connect ? worker111115[0]      router11111[9]

connect ? router11111[10]      worker111116[0]
connect ? worker111116[0]      router11111[10]

connect ? router11111[11]      worker111117[0]
connect ? worker111117[0]      router11111[11]

connect ? router11111[12]      worker111118[0]
connect ? worker111118[0]      router11111[12]

connect ? router11111[13]      worker111119[0]
connect ? worker111119[0]      router11111[13]

connect ? router11111[14]      worker11111100[0]
connect ? worker11111100[0]    router11111[14]

connect ? router11111[15]      worker11111101[0]
connect ? worker11111101[0]    router11111[15]


connect ? router111111[1]      router1111111[0]
connect ? router1111111[0]     router111111[1]

connect ? router111111[4]      worker1111110[0]
connect ? worker1111110[0]     router111111[4]

connect ? router111111[5]      worker1111111[0]
connect ? worker1111111[0]     router111111[5]


connect ? router111111[6]      worker1111112[0]
connect ? worker1111112[0]     router111111[6]

connect ? router111111[7]      worker1111113[0]
connect ? worker1111113[0]     router111111[7]

connect ? router111111[8]      worker1111114[0]
connect ? worker1111114[0]     router111111[8]
```

```
connect ? router111111[9]        worker1111115[0]
connect ? worker1111115[0]       router111111[9]

connect ? router111111[10]       worker1111116[0]
connect ? worker1111116[0]       router111111[10]

connect ? router111111[11]       worker1111117[0]
connect ? worker1111117[0]       router111111[11]

connect ? router111111[12]       worker1111118[0]
connect ? worker1111118[0]       router111111[12]

connect ? router111111[13]       worker1111119[0]
connect ? worker1111119[0]       router111111[13]

connect ? router111111[14]       worker111111100[0]
connect ? worker111111100[0]     router111111[14]

connect ? router111111[15]       worker111111101[0]
connect ? worker111111101[0]     router111111[15]


connect ? router1111111[4]       worker11111110[0]
connect ? worker11111110[0]      router1111111[4]

connect ? router1111111[5]       worker11111111[0]
connect ? worker11111111[0]      router1111111[5]

connect ? router1111111[6]       worker11111112[0]
connect ? worker11111112[0]      router1111111[6]

connect ? router1111111[7]       worker11111113[0]
connect ? worker11111113[0]      router1111111[7]

connect ? router1111111[8]       worker11111114[0]
connect ? worker11111114[0]      router1111111[8]

connect ? router1111111[9]       worker11111115[0]
connect ? worker11111115[0]      router1111111[9]

connect ? router1111111[10]      worker11111116[0]
connect ? worker11111116[0]      router1111111[10]

connect ? router1111111[11]      worker11111117[0]
connect ? worker11111117[0]      router1111111[11]

connect ? router1111111[12]      worker11111118[0]
connect ? worker11111118[0]      router1111111[12]

connect ? router1111111[13]      worker11111119[0]
connect ? worker11111119[0]      router1111111[13]
```

```
connect ? router1111111[14]         worker1111111100[0]
connect ? worker1111111100[0]       router1111111[14]

connect ? router1111111[15]         worker1111111101[0]
connect ? worker1111111101[0]       router1111111[15]


connect ? router2[1]                router21[0]
connect ? router21[0]               router2[1]
connect ? router2[4]                worker20[0]
connect ? worker20[0]               router2[4]

connect ? router2[5]                worker21[0]
connect ? worker21[0]               router2[5]

connect ? router2[6]                worker22[0]
connect ? worker22[0]               router2[6]

connect ? router2[7]                worker23[0]
connect ? worker23[0]               router2[7]

connect ? router2[8]                worker24[0]
connect ? worker24[0]               router2[8]

connect ? router2[9]                worker25[0]
connect ? worker25[0]               router2[9]

connect ? router2[10]               worker26[0]
connect ? worker26[0]               router2[10]

connect ? router2[11]               worker27[0]
connect ? worker27[0]               outer2[11]

connect ? router2[12]               worker28[0]
connect ? worker28[0]               router2[12]

connect ? router2[13]               worker29[0]
connect ? worker29[0]               router2[13]

connect ? router2[14]               worker2100[0]
connect ? worker2100[0]             router2[14]

connect ? router2[15]               worker2101[0]
connect ? worker2101[0]             router2[15]


connect ? router21[1]               router211[0]
connect ? router211[0]              router21[1]
```

```
connect ? router21[4]         worker210[0]
connect ? worker210[0]        router21[4]

connect ? router21[5]         worker211[0]
connect ? worker211[0]        router21[5]

connect ? router21[6]         worker212[0]
connect ? worker212[0]        router21[6]

connect ? router21[7]         worker213[0]
connect ? worker213[0]        router21[7]

connect ? router21[8]         worker214[0]
connect ? worker214[0]        router21[8]

connect ? router21[9]         worker215[0]
connect ? worker215[0]        router21[9]

connect ? router21[10]        worker216[0]
connect ? worker216[0]        router21[10]

connect ? router21[11]        worker217[0]
connect ? worker217[0]        router21[11]

connect ? router21[12]        worker218[0]
connect ? worker218[0]        router21[12]

connect ? router21[13]        worker219[0]
connect ? worker219[0]        router21[13]

connect ? router21[14]        worker21100[0]
connect ? worker21100[0]      router21[14]

connect ? router21[15]        worker21101[0]
connect ? worker21101[0]      router21[15]


connect ? router211[1]        router2111[0]
connect ? router2111[0]       router211[1]

connect ? router211[4]        worker2110[0]
connect ? worker2110[0]       router211[4]

connect ? router211[5]        worker2111[0]
connect ? worker2111[0]       router211[5]

connect ? router211[6]        worker2112[0]
connect ? worker2112[0]       router211[6]

connect ? router211[7]        worker2113[0]
connect ? worker2113[0]       router211[7]
```

connect ? router211[8]	worker2114[0]
connect ? worker2114[0]	router211[8]
connect ? router211[9]	worker2115[0]
connect ? worker2115[0]	router211[9]
connect ? router211[10]	worker2116[0]
connect ? worker2116[0]	router211[10]
connect ? router211[11]	worker2117[0]
connect ? worker2117[0]	router211[11]
connect ? router211[12]	worker2118[0]
connect ? worker2118[0]	router211[12]
connect ? router211[13]	worker2119[0]
connect ? worker2119[0]	router211[13]
connect ? router211[14]	worker211100[0]
connect ? worker211100[0]	router211[14]
connect ? router211[15]	worker211101[0]
connect ? worker211101[0]	router211[15]
connect ? router2111[1]	router21111[0]
connect ? router21111[0]	router2111[1]
connect ? router2111[4]	worker21110[0]
connect ? worker21110[0]	router2111[4]
connect ? router2111[5]	worker21111[0]
connect ? worker21111[0]	router2111[5]
connect ? router2111[6]	worker21112[0]
connect ? worker21112[0]	router2111[6]
connect ? router2111[7]	worker21113[0]
connect ? worker21113[0]	router2111[7]
connect ? router2111[8]	worker21114[0]
connect ? worker21114[0]	router2111[8]
connect ? router2111[9]	worker21115[0]
connect ? worker21115[0]	router2111[9]
connect ? router2111[10]	worker21116[0]
connect ? worker21116[0]	router2111[10]

```
connect ? router2111[11]        worker21117[0]
connect ? worker21117[0]        router2111[11]

connect ? router2111[12]        worker21118[0]
connect ? worker21118[0]        router2111[12]
connect ? router2111[13]        worker21119[0]
connect ? worker21119[0]        router2111[13]

connect ? router2111[14]        worker2111100[0]
connect ? worker2111100[0]      router2111[14]

connect ? router2111[15]        worker2111101[0]
connect ? worker2111101[0]      router2111[15]


connect ? router21111[1]        router211111[0]
connect ? router211111[0]       router21111[1]

connect ? router21111[4]        worker211110[0]
connect ? worker211110[0]       router21111[4]

connect ? router21111[5]        worker211111[0]
connect ? worker211111[0]       router21111[5]

connect ? router21111[6]        worker211112[0]
connect ? worker211112[0]       router21111[6]

connect ? router21111[7]        worker211113[0]
connect ? worker211113[0]       router21111[7]

connect ? router21111[8]        worker211114[0]
connect ? worker211114[0]       router21111[8]

connect ? router21111[9]        worker211115[0]
connect ? worker211115[0]       router21111[9]

connect ? router21111[10]       worker211116[0]
connect ? worker211116[0]       router21111[10]

connect ? router21111[11]       worker211117[0]
connect ? worker211117[0]       router21111[11]

connect ? router21111[12]       worker211118[0]
connect ? worker211118[0]       router21111[12]

connect ? router21111[13]       worker211119[0]
connect ? worker211119[0]       router21111[13]

connect ? router21111[14]       worker21111100[0]
connect ? worker21111100[0]     router21111[14]
```

```
connect ? router21111[15]        worker21111101[0]
connect ? worker21111101[0]      router21111[15]


connect ? router211111[1]        router2111111[0]
connect ? router2111111[0]       router211111[1]


connect ? router211111[4]        worker2111110[0]
connect ? worker2111110[0]       router211111[4]

connect ? router211111[5]        worker2111111[0]
connect ? worker2111111[0]       router211111[5]

connect ? router211111[6]        worker2111112[0]
connect ? worker2111112[0]       router211111[6]

connect ? router211111[7]        worker2111113[0]
connect ? worker2111113[0]       router211111[7]

connect ? router211111[8]        worker2111114[0]
connect ? worker2111114[0]       router211111[8]

connect ? router211111[9]        worker2111115[0]
connect ? worker2111115[0]       router211111[9]

connect ? router211111[10]       worker2111116[0]
connect ? worker2111116[0]       router211111[10]

connect ? router211111[11]       worker2111117[0]
connect ? worker2111117[0]       router211111[11]

connect ? router211111[12]       worker2111118[0]
connect ? worker2111118[0]       router211111[12]

connect ? router211111[13]       worker2111119[0]
connect ? worker2111119[0]       router211111[13]

connect ? router211111[14]       worker211111100[0]
connect ? worker211111100[0]     router211111[14]

connect ? router211111[15]       worker211111101[0]
connect ? worker211111101[0]     router211111[15]


connect ? router2111111[4]       worker21111110[0]
connect ? worker21111110[0]      router2111111[4]

connect ? router2111111[5]       worker21111111[0]
connect ? worker21111111[0]      router2111111[5]
```

```
connect ? router2111111[6]        worker21111112[0]
connect ? worker21111112[0]       router2111111[6]

connect ? router2111111[7]        worker21111113[0]
connect ? worker21111113[0]       router2111111[7]

connect ? router2111111[8]        worker21111114[0]
connect ? worker21111114[0]       router2111111[8]

connect ? router2111111[9]        worker21111115[0]
connect ? worker21111115[0]       router2111111[9]

connect ? router2111111[10]       worker21111116[0]
connect ? worker21111116[0]       router2111111[10]

connect ? router2111111[11]       worker21111117[0]
connect ? worker21111117[0]       router2111111[11]

connect ? router2111111[12]       worker21111118[0]
connect ? worker21111118[0]       router2111111[12]

connect ? router2111111[13]       worker21111119[0]
connect ? worker21111119[0]       router2111111[13]

connect ? router2111111[14]       worker2111111100[0]
connect ? worker2111111100[0]     router2111111[14]

connect ? router2111111[15]       worker2111111101[0]
connect ? worker2111111101[0] router2111111[15]

bind input master[4]              value=&8000001C  !Link3
bind output master[4]             value=&8000000C
```

APPENDIX C - SOURCE CODE FOR READING TERRAIN DATA

This appendix contains the source listings of the C code developed for reading a block of terrain data from PEGASUS database into a specified buffer location which is stored in SUN memory. The source code is stored in files as listed below:

1. PVG_DEC.H
2. PVG_DEC.IN
3. PVG_DEF.IN
4. get_terr.c

```
#ifndef PVG_INCLUDED
#define PVG_INCLUDED

/*****************************************************************************
FILENAME: PVG_DEC.H

PURPOSE: GLOBAL PARAMETER DECLARATION FILE FOR PVG ALGORITHMS

DESCRIPTION: The PVG_DEC.H include file includes all global variables
required for sharing data between major software components of
PVG software.
Parameters are divided into major categories using asteric lines.
All global variables shall be ALL CAPITAL letters.

USE EXAMPLE:
#include "PVG_DEC.H"
*****************************CODE START**********************************/
#include "PVG_DEF.IN"
/***************** COLOR PARAMETERS DECLARATIONS***********************/

/*******************TERRAIN DATA BASE DECLARATIONS********************/

/* Sun main memory terrain storage buffers*/
 u_int TERRAIN1[MAX_BLOCK1][BLOCK1_SIZE];/*one meter
 terrain buffer*/

/* Terrain data bit assignments valid for all resolutions:

 3 2 1
 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
| ELE | SPARE | NOR |S| VEG | GSV |

 where:

ELE = elevation from sea level to top of vegetation in meters
SPARE = not used
NOR = 4 bit surface normal
S = sun shade bit
GSV = gray shade value
*/
```

```
int TERRAINMAP[MAX_EAST_BLOCK][MAX_NORTH_BLOCK];/* terrain map
 contains pointers to terrain data blocks*/
 int HAVEMAP[MAX_EAST_BLOCK][MAX_NORTH_BLOCK];/* Terrain resolution
map tells what resolution blocks are in memory*/


/* range resolution parameters */
 int SRMIN[RES_RANGE_NUM];/*minimum resolution in meters */
 int SRMAX[RES_RANGE_NUM];/*maximum resolution in meters*/
 int SRSTEP[RES_RANGE_NUM];/* step size in meters*/


/* HSPVG terrain data management */
 int IFOVGRID[MAX_EAST_BLOCK][MAX_NORTH_BLOCK];/* Terrain grid
 to image map. Specifies the image location of
 ground points.
EX: IFOVGRID[E][N]= image i coordinate
 in upper word
 = image j (row) coordinate
 in lower word
 = -1 if terrain point is not
in the IFOV image*/

 /* HSPVG terrain data communication variables*/
 u_short TER_PROC_HAS[MAX_EAST_BLOCK][MAX_NORTH_BLOCK]
[RAY_PROC_MAX][RES_RANGE_NUM];/*Map of terrain
data in the HSPVG ray trace processors.
Dimensions are:
-easting quarter kilometer block numbers
-northing quarter kilometer block numbers
-ray processor number
-resolution ranges
 for each resolution range a 16 bit value is stored
with the following meaning
bit 15 Bits0to14 description
 0 0no data no need
 1 0no data but needs it
 0 block#has data no need
 1 block# has data and needs it
*/
```

```
u_short TER_PROC_SEND[RAY_PROC_MAX+1][MAX_BLOCK64][4];
/* Terrain processor send list. Dimensions are
-HSPVG processor number were data is to come from
0= SUN processor
-list entry index sized to allow a full
of every quarter kilometer
[][][0]-destination processor number
if -1 means delete this terrain data
[][][1]-easting block number of data to be sent
[][][2]-northing block number of data to be sent
[][][3]-resolution of data to be sent*/

 u_short SEND[RAY_PROC_MAX+1];/* list entry pointer for
 TER_PROC_SEND contains the number of blocks
 each processor needs to send.



/*******************TARGET DATA BASE DECLARATIONS*********************/
 int TARGETLIST[MAX_TARGETS][10];/* target information list
[0]=target type ID
[1]=easting position of target in meters
[2]=northing position of target in meters
[3]=altitude position of target in meters
[4]= target heading in millirads clockwise from northing
[5]= target pitch in millirads positive up
[6]= target roll in millirads clockwise positive
[7]= speed in millimeters/sec
[8]= status
[9]= spare configuration parameter*/

 struct HAVELISTEL {
 unsigned char *TAR_PTR; /* pointer to target data in memory */
 int RES; /* resolution index of target data */
 } HAVELIST[MAX_TARGETS]; /* list of data in SUN memory */

/*SUN resident target file buffers. These buffers are sized to hold
entire target file for a binary write*/
 unsigned char TARBUF1[MAX_TAR1][TAR1_SIZE];
```

```
/* 1's resolution SUN resident target buffer */
 unsigned char TARBUF2[MAX_TAR2][TAR2_SIZE];
/* 2'nd resolution SUN resident target buffer */
 unsigned char TARBUF3[MAX_TAR3][TAR3_SIZE];
/* 3'd resolution SUN resident target buffer */
 unsigned char TARBUF4[MAX_TAR4][TAR4_SIZE];
/* 4'th resolution SUN resident target buffer */

 int TAR_SEND_LIST[MAX_TARGETS][4];/* list of data to be sent to the
target processor
[0]=source of data processor ID
[1]=destination of data processor ID
[2]=source data start address of data packet (UNUSED)
[3]=number of data elements to send*/
 unsigned char *TAR_SEND_LIST_PTR[MAX_TARGETS];
/* replaces [2] of above */

 int TAR_HAVE_LIST[MAX_TARGETS][20];/* information block buffer
used to collect and store information about what data
the target processor has.
[0]=target type ID
>0 target type ID
<0 player not needed or not in field of view
[1]=easting position of target in meters
[2]=northing position of target in meters
[3]=altitude position of target in meters
[4]= target heading in milliradians clockwise from northing
[5]= target pitch in milliradians positive up
[6]= target roll in milliradians clockwise positive
[7]= speed in meters/sec
[8]= status
[9]= spare configuration parameter*/
/*[10]=data transfer instruction parameter
=0 no change
=1 delete old view data
=2 delete old view data and add new data
[11]=view resolution
[12]=resolution linear array dimension
[13]=view heading milliradians
[14]=view pitch milliradians
[15]=view roll milliradians
```

```
[16]= image center in column pixels, i
[17]= image center in row pixels, j
[18]= image scale in pixels per millimeter
[19]= spare view parameter*/

 int NUM_TAR_TRIAL; /* number of targets in trial */


/******************CAMERA AND FLIGHT DECLARATIONS********************/

 int FLIGHT_CHAR[10];/* Missile flight characteristics
[0]= flight speed in meters per second
[1]=turn rate in degrees per second
[2]=launch acceleration in meters/sec/sec
[3] to [9] = undefined*/

 int IFOVNOW[10];/* instantaneous field of view vector
[0]=easting position of camera in meters
[1]=northing position of camera in meters
[2]=altitude position of camera relative to sea level
[3]=boresight direction heading clockwise from
northing axis(milliradians)
[4]=boresight direction pitch positive up from
horizontal plane(milliradians)
[5]=field of view roll about boresight vector
clockwise positive looking out(millirads)
[6]=zoom factor in milliradians
[7]=curser location, x pixels in upper word
 y pixels in lower word
[8]=auto pilot control status,
0=pre launch
1=launch under auto pilot control
2=flight under auto pilot control
3=flight no autopilot
4=flight lock on target
5=crash no signal
[9]= spare*/
```

```
 int IFOV_PREDICT[PREDICT_INT_MAX][8];/* IFOV predict matrix
[0]=easting position of missile in meters
[1]=northing position of missile in meters
[2]=altitude position of missile in meters
[3]= easting velocity direction cosine
[4]= northing velocity direction cosine
[5]= vertical velocity direction cosine
[6]= speed in meters/sec
[7]= autopilot control status
0=pre launch
1=launch under auto pilot control
2=flight under auto pilot control
3=flight no autopilot
4=flight lock on target
5=crash no signal*/

 int PREDICT_INT[PREDICT_INT_MAX];/* Predict interval
array in seconds. */

 int WAYPOINTS[WAYPOINT_MAX][3];/* point coordinate vectors*/

 int LOCK_POS_IMAGE[3]; /* target lock position and status in
 image coordinates returned to PVG from flyout model
[0] = pixel row count
[1] = pixel column count
[2] = lock status <0 not locked, >0 locked*/

 int LOCK_POS_UTM[4]; /* target or terrain position and status
of locked on pixel location sent to flyout model
 from PVG in UTM coordinates
[0]= easting in meters
[1] = northing in meters
[2] = altitude in meters from sea level
[3] = miss distance from closest target if zero lock
on identified target otherwise it is locked on
a terrain feature*/
```

```
/*******************OUTPUT IMAGE PARAMETERS DECLARATIONS**************/

 int OUTPUT_IMAGE[PVG_HEIGHT][PVG_WIDTH];/* output image buffer
bits 0 to 7 red
bits 8 to 15 green
bits 16 to 23 blue
bits 24 to 31 alpha*/
/* THIS WILL NOT WORK!! YOU PLOT A COLOR INDEX, NOT AN RGB VALUE. */

 short RAY_SEG[RAY_PROC_MAX][4];/* ray trace calculation image
 window definitions the first dimension is the
 processor number, the four parameters represent
0= lower left row
1= lower left column
2= upper right row
3= upper right column*/

 u_short TAR_OUT[PVG_HEIGHT][PVG_WIDTH][2];
/* Target PVG output array
[0]= gray shade
[1]= slant range*/

 int TER_OUT[PVG_HEIGHT][PVG_WIDTH][2];
/* Terrain PVG ray trace output array
[0]= terrain data base element
[1]= slant range*/

 u_char RLUT[RLUT_BYTES];/* Rendering lookup table converts terrain
data base and environmental parameters to gray shade*/

 u_char ATTLUT[ATTLUT_BYTES];/* Attmospheric attenuation lookup*/

/********************ADMINISTRATIVE SOFTWARE DECLARATIONS*************/
/********************MEMORY MANAGEMENT DECLARATIONS*******************/

/*********************TAAC BOARD PARAMETERS DECLARATIONS**************/

/*********************HSPVG HARDWARE PARAMETERS DECLARATIONS**********/

 #endif
```

```
#ifndef PVG_INCLUDED
#define PVG_INCLUDED

/******************************************************************************
FILENAME: PVG_DEC.IN

PURPOSE: GLOBAL PARAMETER DECLARATION FILE FOR PVG ALGORITHMS

DESCRIPTION: The PVG_DEC.IN include file includes all global variables
required for sharing data between major software components of
PVG software.
Parameters are divided into major categories using asteric lines.
All global variables shall be ALL CAPITAL letters.

USE EXAMPLE:
#include <PVG_DEC.IN>
****************************CODE START**********************************/
#include "PVG_DEF.IN"
/***************** COLOR PARAMETERS DECLARATIONS***********************/

/*******************TERRAIN DATA BASE DECLARATIONS********************/

/* Sun main memory terrain storage buffers*/
extern u_int TERRAIN1[MAX_BLOCK1][BLOCK1_SIZE];/*one meter
 terrain buffer*/

/* Terrain data bit assignments valid for all resolutions:

 3 2 1
 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
| ELE | SPARE | NOR |S| VEG | GSV |

 where:

ELE = elevation from sea level to top of vegetation in meters
SPARE = not used
NOR = 4 bit surface normal
S = sun shade bit
GSV = gray shade value
*/
```

```
extern int TERRAINMAP[MAX_EAST_BLOCK][MAX_NORTH_BLOCK];/* terrain map
 contains pointers to terrain data blocks*/
extern int HAVEMAP[MAX_EAST_BLOCK][MAX_NORTH_BLOCK];/* Terrain resolution
map tells what resolution blocks are in memory*/


/* range resolution parameters */
extern int SRMIN[RES_RANGE_NUM];/*minimum resolution in meters */
extern int SRMAX[RES_RANGE_NUM];/*maximum resolution in meters*/
extern int SRSTEP[RES_RANGE_NUM];/* step size in meters*/


/* HSPVG terrain data management */
extern int IFOVGRID[MAX_EAST_BLOCK][MAX_NORTH_BLOCK];/* Terrain grid
 to image map. Specifies the image location of
 ground points.
EX: IFOVGRID[E][N]= image i coordinate
 in upper word
 = image j (row) coordinate
 in lower word
 = -1 if terrain point is not
in the IFOV image*/


/* HSPVG terrain data communication variables*/
extern u_short TER_PROC_HAS[MAX_EAST_BLOCK][MAX_NORTH_BLOCK]
[RAY_PROC_MAX][RES_RANGE_NUM];/*Map of terrain
data in the HSPVG ray trace processors.
Dimensions are:
-easting quarter kilometer block numbers
-northing quarter kilometer block numbers
-ray processor number
-resolution ranges
 for each resolution range a 16 bit value is stored
with the following meaning
bit 15 Bits0to14 description
 0 0no data no need
 1 0no data but needs it
 0 block#has data no need
 1 block# has data and needs it
```

```
*/
extern u_short TER_PROC_SEND[RAY_PROC_MAX+1][MAX_BLOCK64][4];
/* Terrain processor send list. Dimensions are
-HSPVG processor number were data is to come from
0= SUN processor
-list entry index sized to allow a full
of every quarter kilometer
[][][0]-destination processor number
if -1 means delete this terrain data
[][][1]-easting block number of data to be sent
[][][2]-northing block number of data to be sent
[][][3]-resolution of data to be sent*/

extern u_short SEND[RAY_PROC_MAX+1];/* list entry pointer for
 TER_PROC_SEND contains the number of blocks
 each processor needs to send.



/*******************TARGET DATA BASE DECLARATIONS*********************/
extern int TARGETLIST[MAX_TARGETS][10];/* target information list
[0]=target type ID
[1]=easting position of target in meters
[2]=northing position of target in meters
[3]=altitude position of target in meters
[4]= target heading in millirads clockwise from northing
[5]= target pitch in millirads positive up
[6]= target roll in millirads clockwise positive
[7]= speed in millimeters/sec
[8]= status
[9]= spare configuration parameter*/

extern struct HAVELISTEL {
 unsigned char *TAR_PTR; /* pointer to target data in memory */
 int RES; /* resolution index of target data */
} HAVELIST[MAX_TARGETS]; /* list of data in SUN memory */

/*SUN resident target file buffers. These buffers are sized to hold
entire target file for a binary write*/
extern unsigned char TARBUF1[MAX_TAR1][TAR1_SIZE];
```

```
/* 1's resolution SUN resident target buffer */
extern unsigned char TARBUF2[MAX_TAR2][TAR2_SIZE];
/* 2'nd resolution SUN resident target buffer */
extern unsigned char TARBUF3[MAX_TAR3][TAR3_SIZE];
/* 3'd resolution SUN resident target buffer */
extern unsigned char TARBUF4[MAX_TAR4][TAR4_SIZE];
/* 4'th resolution SUN resident target buffer */

extern int TAR_SEND_LIST[MAX_TARGETS][4];/* list of data to be sent to the
target processor
[0]=source of data processor ID
[1]=destination of data processor ID
[2]=source data start address of data packet (UNUSED)
[3]=number of data elements to send*/
extern unsigned char *TAR_SEND_LIST_PTR[MAX_TARGETS];
/* replaces [2] of above */

extern int TAR_HAVE_LIST[MAX_TARGETS][20];/* information block buffer
used to collect and store information about what data
the target processor has.
[0]=target type ID
>0 target type ID
<0 player not needed or not in field of view
[1]=easting position of target in meters
[2]=northing position of target in meters
[3]=altitude position of target in meters
[4]= target heading in milliradians clockwise from northing
[5]= target pitch in milliradians positive up
[6]= target roll in milliradians clockwise positive
[7]= speed in meters/sec
[8]= status
[9]= spare configuration parameter*/
/*[10]=data transfer instruction parameter
=0 no change
=1 delete old view data
=2 delete old view data and add new data
[11]=view resolution
[12]=resolution linear array dimension
[13]=view heading milliradians
[14]=view pitch milliradians
[15]=view roll milliradians
```

[16]= image center in column pixels, i
[17]= image center in row pixels, j
[18]= image scale in pixels per millimeter
[19]= spare view parameter*/

extern int NUM_TAR_TRIAL; /* number of targets in trial */

/*******************CAMERA AND FLIGHT DECLARATIONS********************/

extern int FLIGHT_CHAR[10];/* Missile flight characteristics
[0]= flight speed in meters per second
[1]=turn rate in degrees per second
[2]=launch acceleration in meters/sec/sec
[3] to [9] = undefined*/

extern int IFOVNOW[10];/* instantaneous field of view vector
[0]=easting position of camera in meters
[1]=northing position of camera in meters
[2]=altitude position of camera relative to sea level
[3]=boresight direction heading clockwise from
northing axis(milliradians)
[4]=boresight direction pitch positive up from
horizontal plane(milliradians)
[5]=field of view roll about boresight vector
clockwise positive looking out(millirads)
[6]=zoom factor in milliradians
[7]=curser location, x pixels in upper word
y pixels in lower word
[8]=auto pilot control status,
0=pre launch
1=launch under auto pilot control
2=flight under auto pilot control
3=flight no autopilot
4=flight lock on target
5=crash no signal
[9]= spare*/

```
extern int IFOV_PREDICT[PREDICT_INT_MAX][8];/* IFOV predict matrix
[0]=easting position of missile in meters
[1]=northing position of missile in meters
[2]=altitude position of missile in meters
[3]= easting velocity direction cosine
[4]= northing velocity direction cosine
[5]= vertical velocity direction cosine
[6]= speed in meters/sec
[7]= autopilot control status
0=pre launch
1=launch under auto pilot control
2=flight under auto pilot control
3=flight no autopilot
4=flight lock on target
5=crash no signal*/

extern int PREDICT_INT[PREDICT_INT_MAX];/* Predict interval
array in seconds. */

extern int WAYPOINTS[WAYPOINT_MAX][3];/* point coordinate vectors*/

extern int LOCK_POS_IMAGE[3]; /* target lock position and status in
image coordinates returned to PVG from flyout model
[0] = pixel row count
[1] = pixel column count
[2] = lock status <0 not locked, >0 locked*/

extern int LOCK_POS_UTM[4]; /* target or terrain position and status
of locked on pixel location sent to flyout model
from PVG in UTM coordinates
[0]= easting in meters
[1] = northing in meters
[2] = altitude in meters from sea level
[3] = miss distance from closest target if zero lock
on identified target otherwise it is locked on
a terrain feature*/
```

```
/********************OUTPUT IMAGE PARAMETERS DECLARATIONS**************/

extern int OUTPUT_IMAGE[PVG_HEIGHT][PVG_WIDTH];/* output image buffer
bits 0 to 7 red
bits 8 to 15 green
bits 16 to 23 blue
bits 24 to 31 alpha*/
/* THIS WILL NOT WORK!! YOU PLOT A COLOR INDEX, NOT AN RGB VALUE. */

extern short RAY_SEG[RAY_PROC_MAX][4];/* ray trace calculation image
 window definitions the first dimension is the
 processor number, the four parameters represent
0= lower left row
1= lower left column
2= upper right row
3= upper right column*/

extern u_short TAR_OUT[PVG_HEIGHT][PVG_WIDTH][2];
/* Target PVG output array
[0]= gray shade
[1]= slant range*/

extern int TER_OUT[PVG_HEIGHT][PVG_WIDTH][2];
/* Terrain PVG ray trace output array
[0]= terrain data base element
[1]= slant range*/

extern u_char RLUT[RLUT_BYTES];/* Rendering lookup table converts terrain
data base and environmental parameters to gray shade*/

extern u_char ATTLUT[ATTLUT_BYTES];/* Attmospheric attenuation lookup*/

/********************ADMINISTRATIVE SOFTWARE DECLARATIONS*************/
/********************MEMORY MANAGEMENT DECLARATIONS*******************/

/*********************TAAC BOARD PARAMETERS DECLARATIONS**************/

/*********************HSPVG HARDWARE PARAMETERS DECLARATIONS**********/

#endif
```

```
#ifndef PVG_DEF_INCLUDED
#define PVG_DEF_INCLUDED

/*******************************************************************************

FILENAME: PVG_DEF.LARGE

PURPOSE: GLOBAL PARAMETER DEFINITION FILE FOR PVG ALGORITHMS

DESCRIPTION: The PVG_DEF.IN include file includes all global constants
 required for defining global constants used by PVG software
 components.
 Parameters are divided into major categories using asteric
 lines.
 All global constants shall be ALL CAPITAL letters.

USE EXAMPLE:

 #include "PVG_DEF.IN"
 in your directory link to /home/fogm/include/PVG_DEF.IN

*****************************CODE START**********************************/
#include <FOGM/stdef.h> /*standard definitions */
/*#include "stdef.h"*/
#include <sys/types.h>

/***************** COLOR PARAMETERS DECLARATIONS************************/

/*******************TERRAIN DATA BASE DECLARATIONS********************/

/* Sun main memory terrain storage buffers*/
/* terrain data blocks all cover a 256meterx256 meter area*/

#define MAX_BLOCK1 4 /* # one meter terrain BLOCK1_SIZE*4byte blocks*/
#define BLOCK1_SIZE 65536 /* # elements in 1 meter block */

#define MAX_BLOCK4 1024 /* # 4 meter terrain BLOCK4_SIZE*4byte blocks*/
#define BLOCK4_SIZE 4096 /* # elements in 4 meter block */

#define MAX_BLOCK16 14336 /* # 16 meter terrain BLOCK16_SIZE*4byte blocks*/
```

```
#define BLOCK16_SIZE 256 /* # elements in 16 meter block */

#define MAX_BLOCK64 14336 /* # 64 meter terrain BLOCK64_SIZE*4byte blocks*/
#define BLOCK64_SIZE 16 /* # elements in 64 meter block */

#define MAX_BLOCK256 14336 /* # 256 meter terrain 4byteblocks*/

#define MAX_EAST_BLOCK 128 /* # 256 meter blocks in east direction*/
#define MAX_NORTH_BLOCK 112 /* # 256 meter blocks in north direction*/

#define MIN_EAST_UTM 43328 /* lower left hand corner of data base UTM east*/
#define MIN_NORTH_UTM 63904 /* lower left hand corner of data base UTM north*/

/* Range resolution parameters */

#define RES_RANGE_NUM 4 /* # resolution ranges */

/* Resolution codes */

#define RESOLUTION_1 0
#define RESOLUTION_4 1
#define RESOLUTION_16 2
#define RESOLUTION_64 3

/*******************TARGET DATA BASE DECLARATIONS********************/
/* SUN target data buffers*/
#define MAX_TARGETS 256/* Maximum number of targets in PVG*/
#define MAX_TAR_TYPE 32/*Maximum number of different targets*/
#define MAX_TAR1 8/* Maximum number of target types in 1'st
 resolution level*/
#define MAX_TAR2 8/* Maximum number of target types in 2'nd
 resolution level*/
#define MAX_TAR3 8/* Maximum number of target types in 3'd
 resolution level*/
#define MAX_TAR4 8/* Maximum number of target types in 4'th
 resolution level*/
#define TAR1_SIZE 1069056/* Buffer size in bytes for 64pictures of 1'st
 resolution level*/
#define TAR2_SIZE 282624/* Buffer size in bytes for 64 pictures of 2'nd
 resolution level*/
#define TAR3_SIZE 86016/* Buffer size in bytes for 64 pictures of 3'd
```

```
resolution level*/
#define TAR4_SIZE 36864/* Buffer size in bytes for 64 pictures of 4'th
 resolution level*/

/*******************CAMERA AND FLIGHT DECLARATIONS********************/

#define PREDICT_INT_MAX 4 /* Number of IFOV predict intervals*/
#define WAYPOINT_MAX 20 /* Maximum # way point coordinate vectors*/

/*******************OUTPUT IMAGE PARAMETERS DECLARATIONS**************/

#define PVG_HEIGHT 256 /* output image # pixel rows*/
#define PVG_WIDTH 256 /* output image #pixel columns*/

#define PVG_PIX_SIZE 65536 /* output image size in pixels*/

#define RLUT_BYTES 2097152 /* # bytes in the RLUT*/
#define RLUT_BIT_SIZE 21 /* # bits in RLUT input addess*/

#define VIEW_INDEX_SIZE 3 /*# bits in view vector of the RLUT input address*/
#define NORM_INDEX_SIZE 4 /*# bits in surface normal of the RLUT input address*/

#define ATTLUT_BYTES 2097152 /*# bytes in attenuation table ATTLUT*/
#define ATTLUT_BIT_SIZE 21 /* # bits in ATTLUT input address*/

#define TAR_VIS_MASK 255 /*if target gray shade is 255 let background through */

/********************ADMINISTRATIVE SOFTWARE DECLARATIONS*************/

#define D2R 0.0174532 /* degrees to radians */
#define D2MR 17.4532 /* degrees to milliradians */
#define R2D 57.295827 /* radians to degrees */
#define MR2D 0.057295827 /* milliradians to degrees */

/********************MEMORY MANAGEMENT DECLARATIONS*******************/
#define MAX_SEND 14336 /* maximum number of messages in
TER_PROC_SEND */



/*********************TAAC BOARD PARAMETERS DECLARATIONS**************/
```

```
/**********************HSPVG HARDWARE PARAMETERS DECLARATIONS**********/

#define RAY_PROC_MAX 1 /* maximum # ray trace processors */
#define TAR_PROC_MAX 1 /* maximum # target processors */


#endif
```

```
/*
 * (C) Copyright Nascent Systems Development Inc. 1991
 * Developed under contract DABT62-90-C-0006, Subcontract CSC/ATD-WR-FO-0101
*/
/******************************************************************************

FILENAME: get_terr

AUTHOR: J.R. Akin, August 1989

PURPOSE: Read a block of terrain data into a specified buffer location
 which is stored in SUN main memory. The block needed has a
 lower-left corner at DB coordinates x, y.

DESCRIPTION:

 Opening and closing files eats up a lot of time. Ideally, this function
 should open up every file at start-up but it can't because the number of
 files that can be opened at one time is 60 and the number of PVDB files
 is 83. Since other functions will open up who-knows-how-many files, I've
 set the open file limit to MAX_FILE_HANDLES, an arbitrary value. Files
 are opened and usage statistics maintained until the file handle list is
 exhausted, at which point the least often used file is closed, and a new
 file is opened to take its place.

 Instead of using time-wasting string comparisons for file opens, a list
 of hash values for file names is maintained. The hash value for a file
 is computed as:

 hash_value = (resolution_code << 8) | tile_number

 This way, time can be saved by not using nested "if" statements to
 determine disk partition numbers.

 Processing Steps:

 make sure the x,y coordinates are in bounds

 based upon the resolution code
```

```
{
compute the block length
set the data block pointer to the starting address of the
appropriate (global) TERRAIN buffer
compute the tile number
compute the block number
}

compute the hash value based upon the tile and block number

if the file with this hash value is already open
{
get the file handle for it
increment the number of times this file handle has been used
}
else
{
find least used file handle

if there is already an open file associated with it
close the file

open a new file for this file handle index

set number of times file handle has been used to 1
}

compute file offset for block number

seek to that file offset

compute the number of bytes that need to be read

read the block into the array referenced by bufindex

return number of elements successfully read

}
```

RETURN VARIABLE: Returns number of elements successfully loaded, else ERROR.

REQUIRED INCLUDE FILES:

PVG_DEC.IN
PVG_DEF.IN

LIBRARIES REQUIRED:

INPUT/OUTPUT FILES: use name description

EXTERNAL PARAMETERS: use name include description

Sun main memory terrain storage buffers (declared in PVG_DEC.IN) updated by get_terr() and called by tstwter.c. MAX_BLOCK sizes are declared in PVG_DEF.IN.

IO TERRAIN1 [MAX_BLOCK1] [BLOCK1_SIZE]
IO TERRAIN4 [MAX_BLOCK4] [BLOCK4_SIZE]
IO TERRAIN16 [MAX_BLOCK16] [BLOCK16_SIZE]
IO TERRAIN64 [MAX_BLOCK64] [BLOCK64_SIZE]

FUNCTIONS/SUBROUTINES CALLED: None.

USAGE EXAMPLE:

get_terr_stat = get_terr(RESOLUTION_1, x, y, bufindex);

This call reads a quarter kilometer (block) of 1-meter data at location index x,y and puts it into TERRAINn[bufindex].

```
****************************CODE START********************************/

#include "PVG_DEF.IN"
#include "PVG_DEC.IN"

#include <stdio.h>
#include <fcntl.h> /* for binary I/O */

#define MAX_FILE_HANDLES 32

#define MAX_PVDB_FILES 83
#define MAX_FILE_NAME_LEN 32

#define MAX_PVDB_EAST 32767
#define MAX_PVDB_NORTH 28671

#define UNUSED -1

int get_terr( res_code, x, y, bufindex )

int res_code; /* Resolution: 1, 4, 16, or 64 */
int x, y; /* terrain map coordinate indices */
int bufindex; /* buffer index of block to be read */

{
/**************************************************************
 extern unsigned int TERRAIN1 [MAX_BLOCK1] [BLOCK1_SIZE];
 extern unsigned int TERRAIN4 [MAX_BLOCK4] [BLOCK4_SIZE];
 extern unsigned int TERRAIN16 [MAX_BLOCK16] [BLOCK16_SIZE];
 extern unsigned int TERRAIN64 [MAX_BLOCK64] [BLOCK64_SIZE];
**************************************************************/

 /* Actual file names, only used for open() */

 static char file_name [MAX_PVDB_FILES] [MAX_FILE_NAME_LEN] =
 {
 "/pvdb_data/pvdb.64", "/pvdb_data/pvdb.16",

 "/pvdb_data/pvdb.4.00", "/pvdb_data/pvdb.4.01", "/pvdb_data/pvdb.4.02",
 "/pvdb_data/pvdb.4.03", "/pvdb_data/pvdb.4.04", "/pvdb_data/pvdb.4.05",
 "/pvdb_data/pvdb.4.06", "/pvdb_data/pvdb.4.10", "/pvdb_data/pvdb.4.11",
```

```
"/pvdb_data/pvdb.4.12", "/pvdb_data/pvdb.4.13", "/pvdb_data/pvdb.4.14",
"/pvdb_data/pvdb.4.15", "/pvdb_data/pvdb.4.16", "/pvdb_data/pvdb.4.20",
"/pvdb_data/pvdb.4.21", "/pvdb_data/pvdb.4.22", "/pvdb_data/pvdb.4.23",
"/pvdb_data/pvdb.4.24", "/pvdb_data/pvdb.4.25", "/pvdb_data/pvdb.4.26",
"/pvdb_data/pvdb.4.30", "/pvdb_data/pvdb.4.31", "/pvdb_data/pvdb.4.32",
"/pvdb_data/pvdb.4.33", "/pvdb_data/pvdb.4.34", "/pvdb_data/pvdb.4.35",
"/pvdb_data/pvdb.4.36", "/pvdb_data/pvdb.4.40", "/pvdb_data/pvdb.4.41",
"/pvdb_data/pvdb.4.42", "/pvdb_data/pvdb.4.43", "/pvdb_data/pvdb.4.44",
"/pvdb_data/pvdb.4.45", "/pvdb_data/pvdb.4.46", "/pvdb_data/pvdb.4.50",
"/pvdb_data/pvdb.4.51", "/pvdb_data/pvdb.4.52", "/pvdb_data/pvdb.4.53",
"/pvdb_data/pvdb.4.54", "/pvdb_data/pvdb.4.55", "/pvdb_data/pvdb.4.56",
"/pvdb_data/pvdb.4.60", "/pvdb_data/pvdb.4.61", "/pvdb_data/pvdb.4.62",
"/pvdb_data/pvdb.4.63", "/pvdb_data/pvdb.4.64", "/pvdb_data/pvdb.4.65",
"/pvdb_data/pvdb.4.66", "/pvdb_data/pvdb.4.70", "/pvdb_data/pvdb.4.71",
"/pvdb_data/pvdb.4.72", "/pvdb_data/pvdb.4.73", "/pvdb_data/pvdb.4.74",
"/pvdb_data/pvdb.4.75", "/pvdb_data/pvdb.4.76",

"/pvdb_data/pvdb.1.13", "/pvdb_data/pvdb.1.14", "/pvdb_data/pvdb.1.15",
"/pvdb_data/pvdb.1.22", "/pvdb_data/pvdb.1.23", "/pvdb_data/pvdb.1.24",
"/pvdb_data/pvdb.1.25", "/pvdb_data/pvdb.1.31", "/pvdb_data/pvdb.1.32",
"/pvdb_data/pvdb.1.33", "/pvdb_data/pvdb.1.34", "/pvdb_data/pvdb.1.35",
"/pvdb_data/pvdb.1.41", "/pvdb_data/pvdb.1.42", "/pvdb_data/pvdb.1.43",
"/pvdb_data/pvdb.1.44", "/pvdb_data/pvdb.1.45", "/pvdb_data/pvdb.1.51",
"/pvdb_data/pvdb.1.52", "/pvdb_data/pvdb.1.53", "/pvdb_data/pvdb.1.54",
"/pvdb_data/pvdb.1.61", "/pvdb_data/pvdb.1.62", "/pvdb_data/pvdb.1.63",
"/pvdb_data/pvdb.1.64"
};

/* Hash values for file names */

static int file_name_hash_value [MAX_PVDB_FILES] =
{
(RESOLUTION_64<<8)|0x00,
(RESOLUTION_16<<8)|0x00,

/* 4-meter files */

(RESOLUTION_4<<8)|0x00, (RESOLUTION_4<<8)|0x01, (RESOLUTION_4<<8)|0x02,
(RESOLUTION_4<<8)|0x03, (RESOLUTION_4<<8)|0x04, (RESOLUTION_4<<8)|0x05,
(RESOLUTION_4<<8)|0x06,
```

```
(RESOLUTION_4<<8)|0x10, (RESOLUTION_4<<8)|0x11, (RESOLUTION_4<<8)|0x12,
(RESOLUTION_4<<8)|0x13, (RESOLUTION_4<<8)|0x14, (RESOLUTION_4<<8)|0x15,
(RESOLUTION_4<<8)|0x16,

(RESOLUTION_4<<8)|0x20, (RESOLUTION_4<<8)|0x21, (RESOLUTION_4<<8)|0x22,
(RESOLUTION_4<<8)|0x23, (RESOLUTION_4<<8)|0x24, (RESOLUTION_4<<8)|0x25,
(RESOLUTION_4<<8)|0x26,

(RESOLUTION_4<<8)|0x30, (RESOLUTION_4<<8)|0x31, (RESOLUTION_4<<8)|0x32,
(RESOLUTION_4<<8)|0x33, (RESOLUTION_4<<8)|0x34, (RESOLUTION_4<<8)|0x35,
(RESOLUTION_4<<8)|0x36,

(RESOLUTION_4<<8)|0x40, (RESOLUTION_4<<8)|0x41, (RESOLUTION_4<<8)|0x42,
(RESOLUTION_4<<8)|0x43, (RESOLUTION_4<<8)|0x44, (RESOLUTION_4<<8)|0x45,
(RESOLUTION_4<<8)|0x46,

(RESOLUTION_4<<8)|0x50, (RESOLUTION_4<<8)|0x51, (RESOLUTION_4<<8)|0x52,
(RESOLUTION_4<<8)|0x53, (RESOLUTION_4<<8)|0x54, (RESOLUTION_4<<8)|0x55,
(RESOLUTION_4<<8)|0x56,

(RESOLUTION_4<<8)|0x60, (RESOLUTION_4<<8)|0x61, (RESOLUTION_4<<8)|0x62,
(RESOLUTION_4<<8)|0x63, (RESOLUTION_4<<8)|0x64, (RESOLUTION_4<<8)|0x65,
(RESOLUTION_4<<8)|0x66,

(RESOLUTION_4<<8)|0x70, (RESOLUTION_4<<8)|0x71, (RESOLUTION_4<<8)|0x72,
(RESOLUTION_4<<8)|0x73, (RESOLUTION_4<<8)|0x74, (RESOLUTION_4<<8)|0x75,
(RESOLUTION_4<<8)|0x76,

/* 1-meter files */

(RESOLUTION_1<<8)|0x13, (RESOLUTION_1<<8)|0x14, (RESOLUTION_1<<8)|0x15,

(RESOLUTION_1<<8)|0x22, (RESOLUTION_1<<8)|0x23, (RESOLUTION_1<<8)|0x24,
(RESOLUTION_1<<8)|0x25,

(RESOLUTION_1<<8)|0x31, (RESOLUTION_1<<8)|0x32, (RESOLUTION_1<<8)|0x33,
(RESOLUTION_1<<8)|0x34, (RESOLUTION_1<<8)|0x35,

(RESOLUTION_1<<8)|0x41, (RESOLUTION_1<<8)|0x42, (RESOLUTION_1<<8)|0x43,
(RESOLUTION_1<<8)|0x44, (RESOLUTION_1<<8)|0x45,
```

```
(RESOLUTION_1<<8)|0x51, (RESOLUTION_1<<8)|0x52, (RESOLUTION_1<<8)|0x53,
(RESOLUTION_1<<8)|0x54,

(RESOLUTION_1<<8)|0x61, (RESOLUTION_1<<8)|0x62, (RESOLUTION_1<<8)|0x63,
(RESOLUTION_1<<8)|0x64
};

/* file_opened[n] tells whether a file has been opened. Static storage */
/* without initilization garauntees that elements will be set to 0 (NO) */

static int file_opened [MAX_PVDB_FILES] =
{
UNUSED, UNUSED,

UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED,
UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED,
UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED,
UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED,
UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED,
UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED,
UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED,
UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED,

UNUSED, UNUSED, UNUSED,
UNUSED, UNUSED, UNUSED, UNUSED,
UNUSED, UNUSED, UNUSED, UNUSED, UNUSED,
UNUSED, UNUSED, UNUSED, UNUSED, UNUSED,
UNUSED, UNUSED, UNUSED, UNUSED,
UNUSED, UNUSED, UNUSED, UNUSED
};

/* List of file handles used for I/O */

static int fh [MAX_FILE_HANDLES] =
{
UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED,
UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED,
UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED,
UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED
};
```

```
/* List of hash value indices for opened files */

static int fh_hash_value_index [MAX_FILE_HANDLES] =
{
UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED,
UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED,
UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED,
UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED
};

static int file_usage [MAX_FILE_HANDLES] =
{
UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED,
UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED,
UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED,
UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED, UNUSED
};

static int fh_index_to_use = 0; /* file handle index to use */

int tile_num, block_num; /* tile and block numbers */

int hash_value;
int hash_index; /* hash index */
int usage;
int min_usage;
int n;

int block_length;
unsigned int *bp; /* block pointer */
unsigned int file_offset;
unsigned int bytes_to_read, bytes_read;
int elements_read;
```

```
/************************* BEGIN EXECUTION ***************************/

if( x < 0 || x > MAX_PVDB_EAST )
{
fprintf( stdout, "get_ter: X coordinate (%d) OOB\n", x );
return( ERROR );
}

if( y < 0 || y > MAX_PVDB_NORTH )
{
fprintf( stdout, "get_ter: Y coordinate (%d) OOB\n", y );
return( ERROR );
}

if( (x % 256) != 0 )
{
fprintf( stdout, "get_ter: X (%d) not an even multiple of 256\n", x );
return( ERROR );
}

if( (y % 256) != 0 )
{
fprintf( stdout, "get_ter: Y (%d) not an even multiple of 256\n", y );
return( ERROR );
}

switch( res_code )
{
case RESOLUTION_1:
block_length = BLOCK1_SIZE;
bp = &TERRAIN1[bufindex][0];
break;

default:
fprintf( stdout, "get_ter: Invalid res_code (%d)\n", res_code );
return( ERROR ); /* invalid resolution */
}

/* Tile and block numbers are the same regardless of resolution */
tile_num = ((x>>8) & 0xF0) | (y>>12);
block_num = ((x>>4) & 0xF0) | ((y>>8) & 0xF );
```

```
/* ... but the hash values aren't. The 16- and 64-meter databases ARE */
/* broken into tile numbers but they aren't stored in multiple files. */

if( res_code < RESOLUTION_16 )
hash_value = (res_code << 8) | tile_num;
else
hash_value = (res_code << 8);

/* Find the index to the file name's hash value */

for( hash_index=0; hash_index < MAX_PVDB_FILES; hash_index++ )
{
if( hash_value == file_name_hash_value[hash_index] )
break;
}

if( hash_index == MAX_PVDB_FILES ) /* no match was found */
{
fprintf( stdout, "No data available at %d,%d, for resolution %d\n",
x, y, res_code );
return( ERROR );
}

if( file_opened[hash_index] != UNUSED ) /* file is already open */
{
/* Get the proper file handle and increment the number of times used */

fh_index_to_use = file_opened[hash_index];
file_usage[fh_index_to_use]++;
}
else /* this file needs to be opened */
{
/* Open a new file. Find the least used file handle; */
/* if it is used (open), close it first. */

fh_index_to_use = 0;
min_usage = file_usage[fh_index_to_use];

for( n=0; n < MAX_FILE_HANDLES; n++ )
{
usage = file_usage[n];
```

```
if( usage < min_usage )
{
min_usage = usage;
fh_index_to_use = n;
}
}

if( fh[fh_index_to_use] != UNUSED ) /* close it first */
{
close( fh[fh_index_to_use] );
fh[fh_index_to_use] = UNUSED;
file_opened[ fh_hash_value_index[fh_index_to_use] ] = UNUSED;
}

/* Open the new file */

fh[fh_index_to_use] = open( file_name[hash_index], O_RDONLY );

if( fh[fh_index_to_use] == ERROR )
{
fprintf( stdout, "get_ter: Can't open file\n" );
return( ERROR );
}
else
{
file_opened[hash_index] = fh_index_to_use;
file_usage[fh_index_to_use] = 1;
fh_hash_value_index[fh_index_to_use] = hash_index;
}
}

/* Compute file offset for this block_num, based upon the resolution */
/* and seek to that location. */

switch( res_code )
{
case RESOLUTION_1:
case RESOLUTION_4:
file_offset = block_num * block_length * sizeof( int );
break;
```

```
/* For 16- and 64-meter resolutions compute the tile sequence number */
/* number (Tile_x*7+Tile_y), multiply it by the number of elements in */
/* a tile, add the block offset and multiply by the number of bytes */
/* in an int. */

case RESOLUTION_16:
case RESOLUTION_64:
file_offset = ( ((tile_num>>4)*7+(tile_num&0xF))
* (256*block_length) + (block_num*block_length) )
* sizeof( int );
break;
}

if( lseek( fh[fh_index_to_use], file_offset, 0 ) == ERROR )
{
fprintf( stdout, "Can't seek on file\n" );
return( ERROR );
}

/* Read block into address at bp */

bytes_to_read = block_length * sizeof( int );

bytes_read = read( fh[fh_index_to_use], (char *)bp, bytes_to_read );

if( bytes_read != bytes_to_read )
{
fprintf( stdout, "Bad read, X:%d Y:%d res:%d\n", x, y, res_code );
fprintf( stdout, "%d bytes read instead of %d\n",
bytes_read, bytes_to_read );
return( ERROR );
}

elements_read = bytes_read / sizeof( int );

/*************************************************************************

fprintf( stdout, "X:%d, Y:%5d, res_code:%d bufindex:%d address:%08X\n",
x, y, res_code, bufindex, bp );

*************************************************************************/
```

```
return( elements_read );

}

#undef MAX_FILE_HANDLES
#undef MAX_PVDB_FILES
#undef MAX_FILE_NAME_LEN
#undef MAX_PVDB_EAST
#undef MAX_PVDB_NORTH
#undef UNUSED
```

LIST OF REFERENCES

[Ref. 1] Titan Tactical Applications, *JANUS (A) 2.1 Software Design Manual*, 1992.

[Ref. 2] INMOS Limited, *The Transputer Family 1987*, p. 4, April 1987.

[Ref. 3] INMOS Limited, *Transputer Handbook*, p. 1, October 1989.

[Ref. 4] Shiva, S.G., *Computer Design & Architecture*, 2nd ed., Harper Collins Publishers Inc., 1991.

[Ref. 5] INMOS Limited, *An Introduction To Transputers*, Draft 2.0, pp. 5-6, January 1988.

[Ref. 6] Lewis, T.G., El-Rewini, H., *Introduction To Parallel Computing*, Prentice-Hall Inc., 1992.

[Ref. 7] INMOS Limited, *The Transputer Databook*, 2nd ed., 1989.

[Ref. 8] Hoare, C.A.R., *"Communicating Sequential Processes"*, *"Communications of the ACM"*, v. 21, n. 8, pp. 666-667, August 1978.

[Ref. 9] INMOS Limited, *T9000 Transputer Products Overview Manual*, 1991.

[Ref. 10] INMOS Limited, *OCCAM 2 Reference Manual*, Prentice-Hall Inc., 1988.

[Ref. 11] Alsys Inc., *Alsys Ada Compilation System User Manual*, 1989.

[Ref. 12] 3L Ltd., *Parallel C User Guide*, 1988.

[Ref. 13] 3L Ltd., *Parallel C++ User Guide*, 1991.

[Ref. 14] NASCENT Systems Development Inc., *The Pegasus Documentation Package Book-1*, December 1992.

[Ref. 15] Inmos Ltd., *Inmos Technical Note 53 - Some Issues in Scientific Language Application Portion and Farming Using Transputers*, by A. Hamilton, pp. 7-8, July 1989.

[Ref. 16] Inmos Ltd., *IMS B004 Evaluation Board User Manual*, pp. 1-18, 1985.

[Ref. 17] Inmos Ltd., *Inmos Technical Note 11 - IMS B004 IBM PC Add-In Board*, by S. Ghee, 1989.

[Ref. 18] Alta Technology Corporation, *CTRAM Computation Transputer Module Data Sheet*, 1993.

[Ref. 19] Alta Technology Corporation, *Remote Tram Holder Installation Guide and User Manual (Version 1.0)*, September 1991.

[Ref. 20] Alta Technology Corporation, *HSI/SBUS Installation Guide and User Reference (Version 1.1)*, October 1992.

[Ref. 21] Inmos Ltd., *IMS B012 User Guide and Reference Manual*, 1988.

[Ref. 22] Digital Equipment Corporation, *Alpha AXP Systems Handbook*, 1993.

[Ref. 23] ParaSoft Corporation, *EXPRESS 3.0 Introductory Guide*, 1990.

[Ref. 24] Perihelion Software Ltd., *The HELIOS Parallel Operating System*, Prentice Hall International (UK) Ltd., 1991.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexanderia, VA 22304-6145	2
2.	Dudley Knox Library Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Dr. Ted Lewis Code CS/Lt Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	1
4.	Dr. Se-Hung Kwak Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	2
5.	Dr. Se-Hung Kwak 75 Adams Avenue West Newton, MA 02165	1
6.	Maj. Eugene Paulo TRAC-MTRY Naval Postgraduate School Monterey, CA 93943	1
7.	Dr. Amr M. Zaky Code CS/Za Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	1
8.	Dr. Wolfgang Baer Code CS/Ba Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	1
9.	Deniz Kuvvetleri Komutanligi Personel Daire Baskanligi Bakanliklar, Ankara / TURKEY	1

10. Golcuk Tersanesi Komutanligi 1
Golcuk, Kocaeli / TURKEY

11. Deniz Harp Okulu Komutanligi 1
Tuzla, Istanbul / TURKEY 81704

12. Taskizak Tersanesi Komutanligi 1
Kasimpasa, Istanbul / TURKEY

13. LTJG Cem Ali Dündar 1
Ziya Bey Cad. Etibank Sitesi No:14
Balgat, Ankara / TURKEY