A Computing Framework for Integrating Interactive Visualization in HPCC Applications

Gang Cheng, Geoffrey C. Fox, Tseng-Hui Lin and Tomasz Haupt

gcheng@npac.syr.edu, (315)443-1118

Northeast Parallel Architectures Center
Syracuse University, Syracuse, NY 13244

Abstract

Network-based concurrent computing and interactive data visualization are two important components in industry applications of high performance computing and communication. We propose an execution framework to build interactive remote visualization systems for real world applications on heterogeneous parallel and distributed computers. Using a dataflow model of a commercial visualization software AVS in three case studies. we demonstrate a simple, effective and modular approach to couple parallel simulation modules into an interactive remote visualization environment. The applications described in this paper are drawn from our industry projects in financial modeling, computational electromagnetics and computational chemistry.

1 Introduction

Scientific visualization has traditionally been carried out interactively on workstations, or in post-processing or batch on super computers. With advances in high performance computing systems and networking technologies, interactive visualization in a distributed environment becomes feasible. In a remote visualization environment, data, I/O, computation and user interaction are physically distributed through high-speed networking to achieve high performance and optimal use of various resources required by the application task. Seamless integration of high performance computing systems with graphics workstations and traditional scientific visualization is not only feasible, but will be a common practice with real-time application systems.

Advances in parallel computing systems and network technology provide new opportunities for implementing computation intensive applications. Applications that integrate modeling and simulation with large scale information processing often require an interactive graphical user interface (GUI) in a real-time computing environment. Most GUIs are event-driven, and serial in nature, making them unsuitable for parallel implementation. On the other hand, visualization tools for parallel systems often require special hardware support, or are developed for a specific hardware/software system and are not portable. Parallel computing environment of the future will likely be based on a heterogeneous networked computing system, and be required to solve applications with various requirements in scientific computation, communication, programming models, memory hierarchies, I/O, database management and visualization. Network-based concurrent computing and advanced data visualization are two important components in real-world applications of high performance computing and communication(HPCC), especially for industrial problems requiring real-time simulation/modeling, information processing and system integration.

During the past two years, Northeast Parallel Architectures Center (NPAC) at Syracuse University has actively engaged in several research and application projects which require interactive data visualization and distributed and parallel processing. We have developed real-time parallel numerical simulation/modeling applications in the areas of financial modeling[17, 20], computational electromagnetics[18], computational chemistry[14] and environmental modeling[19]. These applications use a range of parallel and distributed systems, from MPP machines including CM2,CM5,DECmpp-12000,IBM SP2 to networked heterogeneous workstation clusters such as DEC Alpha Farm and coupled MPPs and workstations. These projects are part of an major HPCC technology transfer program InfoMall[3] developed at NPAC which consists of a set of interactive distributed high performance information systems, collectively called InfoVision [2] (INFOmation, Video, Imagery, and SImulation ON-demand). Targeted at the state-of-the-art applications and enabling technology on NYNET, a regional ATM-based WAN network, and the utilization of supercomputing resource at NPAC, InfoVision focuses on real-time information production, analysis, access and integration on distributed and parallel computing systems – allow geographically located regional end-user to interactively access multimedia information and to remotely perform parallel simulation demonstrations on supercomputers over the high-speed WAN network NYNET.

As a core enabling technology in simulation on-demand applications, interactive software integration environment plays a central role in coupling parallel and distributed computing, interactive remote user-interface, advanced data visualization and transparent networking to support real-time simulations over a heterogeneous computing and networking environment.

In this paper, we describe our experience in building interactive visualization systems for the InfoVision simulation-on-demand projects. Application Visualization System (AVS) is used as an integration framework to facilitate both networking and scientific data visualization in all the three case studies. This paper is organized as follows: in Section 2, we first introduce AVS and propose a distributed visualization and computing framework which is built on AVS's dataflow model and will be elaborated in later sections when applied to specific applications and parallel systems. Section 3 describes an interactive visualization system for a financial option price modeling application originally implemented on a CM5 and a DECmpp-12000 and ported to an IBM SP2 and a DEC Alpha Farm; A remote visualization environment using the AVS model for an electromagnetic scattering simulation on a CM5 and networked workstations is given in Section 4 to demonstrate the effectiveness of this framework. In Section 5, we describe our experience in porting a software package MOPAC to parallel systems and creating a data visualization for computational chemistry applications. Finally we conclude in Section 6.

2 A Distributed Visualization and Computing Model in AVS

2.1 Application Visualization System

AVS[6] is a software development toolkit based on a dataflow model for scientific data visualization and process control. It incorporates visualization, graphics, visual programming, process management and networking into a single comprehensive software development environment.

The AVS dataflow is a model of parallel computation in which a flow network of autonomous processes compute by passing object data along arcs interconnected by input/output ports. Each module, represented as an individual UNIX process, fires autonomously as soon as new input object it requires arrives on its input ports. It then produces output object that flows on its output port thus triggers further computations. A set of base communication objects is pre-defined such as numbers and fields (n-dimensional arrays) and a collection of computational modules is provided that transform sets of input objects into sets of output objects. AVS itself offers an extensive library of ready to use modules mostly for graphics programming and filtering, as well as it specifies the protocol for developing and including user modules into the system. Specific applications are assembled from reusable modules with the aid of visual network editing tools. Modules are visualized as nodes, ports as terminals, and communication objects as links of an interactively constructed dataflow network.

Both process control and data transfer among processes in AVS are in a modular fashion and is completely transparent to the programmer. AVS provides a data-channel abstraction that transparently handles module connectivity and ports type-checking. The module programmer needs only to define the input and output ports in AVS predefined data types by using a set of AVS routines and micros. Message passing occurs at a high level of data abstraction and only through the input and output ports. Each AVS application includes AVS kernel and a set of modules, specified by a dataflow network. AVS kernel executes network specification and instantiates module processes as well as the lower level inter-modular communication links, based on the RPC/XDR UNIX network protocol. The important feature of the AVS execution model is that it supports remote module execution in which various nodes of the computational graph can be executed on distributed machines on a local or wide area network.

2.2 A Dataflow-based Visualization and Computing Model

In a typical remote visualization environment, a simulation cycle is started from a GUI module running on local machine, with simulation parameters are represented as slide buttons and dials, providing real-time instrumentation of the simulation's progress as well as simulation steering (changing simulation parameters before a new simulation cycle). Source data for the simulation can also be read in by the GUI module from databases or data files. The computational task of the simulation is decomposed into several computational independent sub-tasks, which are represented as computing modules distributed to different remote machines. Input data are collected from the GUI by user runtime interaction, from disk files, or both sources, and broadcasted to computing modules on the remote machines. There is no or little data transfer among modules on different remote machines. A remote machine may be a workstation or a supercomputer, whichever architecture and computational power best suited to the decomposed sub-task. The simulation ends with some kind of GUI rendering/viewing modules that run on the local machine and use results generated from remote modules. Fundamentally, this is a data flow (data-driven) programming model, in which activation of a module process is triggered solely by availability of input data from another module process on either same or the different machine.

This general model is well suited to rapid-prototyping for certain simulation and modeling applications that require both scientific data visualization and high performance computing. At the software environment level, this model only requires support of high

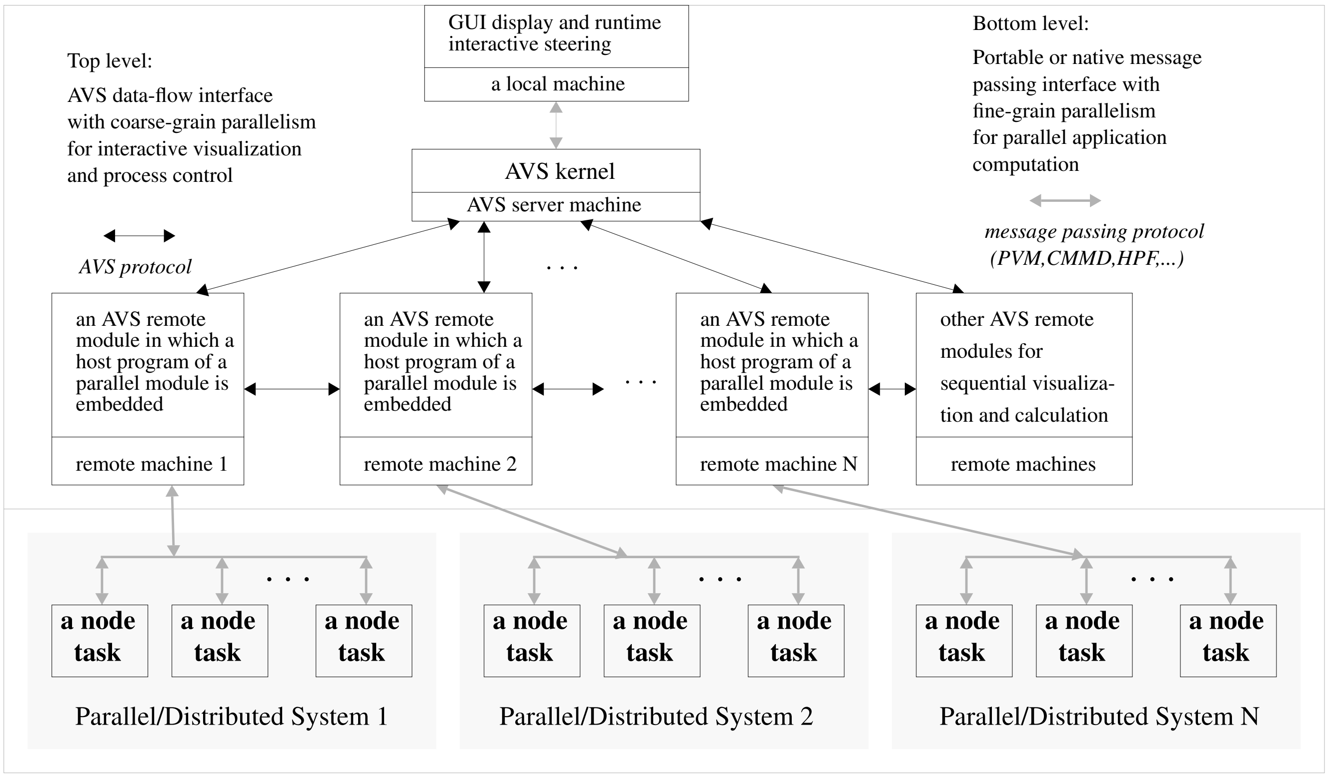

FIG. 1. *A general visualization/computing model*

level data visualization and networking facilities. Most importantly, this data flow based model is well supported by most commercially available data visualization environments, such as AVS[1] from AVS Inc., Explorer [7] from Silicon Graphics Inc. and Data Explorer from IBM.

As shown in Figure 1, using the data-flow programming model in AVS, message passing among modules on the same machine and different machines are identical and completely transparent to module programmer. AVS kernel(protocol) supervises data transfer which is eventually carried out by TCP/IP at a lower level. The module programmer needs only to define module input and output ports in AVS predefined data types. Message passing among AVS modules occurs only through I/O ports and uses AVS's IPC or RPC protocol. A set of routines for initializing and describing modules to AVS, as well as parameter handling, accessing data, error handling and coroutine event handling are provided. Data sources and destinations can be flexibly defined by visually connecting module input and output ports using a network editor.

Data transfers between host and node programs of a parallel module rely on either portable message passing protocols in implementing the parallel module such as PVM and MPI, or native protocols specific to the parallel system such as CMMD, CMFortran on a CM5, MPL on a IBM SP2 or HPF on a DECmpp. They are strictly confined within a parallel module. Message passing between two parallel modules on different machines must go through the dataflow channel between the two sequential AVS modules embedded in the host parts of the parallel programs.

3 A Financial Modeling Application

3.1 Introduction and Problem Description

Financial modeling represents a promising industry application of high performance computing. In previous work, parallel stock option pricing models were developed for the Connection Machine-5 and DECmpp-12000[4, 5], and later were ported on an IBM SP2 and a DEC Alpha cluster. These parallel models run approximately two orders of magnitude faster than sequential models on high-end workstations. To further develop this application in real industry use, a portable, workstation based, interactive visualization environment is developed for a heterogeneous computing environment.

Stock option pricing models are used to calculate a price for an option contract based on a set of market variables, (e.g. exercise price, risk-free rate, time to maturity) and a set of model parameters. Model price estimates are highly sensitive to parameter values for volatility of stock price, variance of the volatility, and correlation between volatility and stock price. These model parameters are not directly observable, and must be estimated from market data. Using optimization techniques for model parameter estimation holds great promise for improving model accuracy.

We use a set of four option pricing models in this study. Simple models treat stock price volatility as a constant, and price only European (option exercised only at maturity of contract) options. More sophisticated models incorporate stochastic volatility processes, and price American contracts (option exercised at any time in life of contract)[15, 16]. These models are computational intensive and have significant communication requirements. The four pricing models are: BS - the Black-Scholes constant volatility, European model; AMC - the American binomial, constant volatility model; EUS - the European binomial, stochastic volatility model; and AMS - the American binomial, stochastic volatility model. Detailed descriptions about these four models can be found in [15, 4].

Analytic models are useful tools in the financial market, but require expert interpretation. To further evaluate and optimize pricing models to run in a realistic application environment, we combine high performance computing modules for real-time pricing with real-time visualization of model results and market conditions, and a graphical user interface allowing expert interaction with pricing models. We envision a market expert using such a system to start and stop a set of models, adjust model parameters, and call optimization routines according to dynamically changing market conditions.

3.2 System Configuration and Integration

Using the stock option price modeling application as a case study, we demonstrate a simple, effective and modular approach to coupling network-based concurrent modules into an interactive remote visualization environment as proposed in Section 2. Two prototype simulation on-demand systems are developed to demonstrate the portability of this framework, in which parallel option pricing models locally implemented on two system configurations (two meta machines): one with two MPP machines - a 32-node CM5 and a 8K-node DECmpp-12000; the other with two distributed systems - an Ethernet-based IBM SP2 and a cluster of DEC Alpha workstations networked by a FDDI-based GIGA-switch. They are coupled with an interactive graphical user interface over the NYNET ATM-based wide area network. AVS is used to integrate massively parallel processing, workstation based visualization, an interactive system control, and distributed I/O modules.

Configuration 1 - NYNET + CM5 + DECmpp-12000 + Workstations

Figure 1.1 is the system configuration of the first prototype interactive simulation-on-demand system for the option price modeling application, utilizing the network infrastructure and supercomputing facility at NPAC.

The AVS kernel runs on a SUN10 workstation which acts both as an AVS server to coordinate data-flow and top-level concurrent control among remote modules, and as a network gateway which links the NPAC in-house host machines locally networked by an Ethernet to the regional end-user through the NYNET. The ATM-based link is built around two Fore switches that operate at 155 Mbps (OC3c) while the wide area network portion of the network operates at OC48(2400 Mbps) speed.

Our heterogeneous computing system for stock option pricing consists of four compute nodes, a home machine, and two file server machines. All workstations, including the front-ends of the DECmpp-12000 and CM-5, are connected by a 10MBit/second Ethernet based LAN.

The four option pricing models run on remote compute nodes: BS model on a DEC5000, AMC model on a SUN4, EUS model on a CM-5 and AMS on a DECmpp-12000(SX). Each remote compute node has its own I/O capability. Our DECmpp-12000 is a massively parallel SIMD system with 8192 processors. Each RISC-like processor has a control processor, forty 32-bit registers, and 16 KBytes of RAM. All the processor elements are arranged in a rectangular two-dimensional grid and are tightly coupled with a DEC5000 front-end workstation. The theoretical peak performance is 650 Mflops DP. Our CM-5 is a parallel MIMD machine with 32 processing nodes. Each processing node consists of a SPARC processor for control, four proprietary vector units for numerical computation, and 32 MBytes of RAM. The control node of the CM-5 is a SUN4 workstation. The theoretical peak performance is 4 Gflops. Sequential compute nodes include a DEC5000 and a SUN4. The DEC5000 performs at 6.8 Mflops, and has 16 Mbytes memory. The SUN4 runs at 4.3 Mflops and has 32 Mbytes memory.

While displayed on the end-user's home machine, a user interface actually runs on a remote SUN4 which combines user runtime input (model parameters, network configuration) with historical market databases stored on disk, and broadcasts this data to remote compute nodes. Top-level system synchronization occurs with each broadcast.

An IBM RS/6000 is used as a file server for non-graphical output of model data. In this application, model prices calculated at remote compute nodes, along with market data, are written to a database on the output file server for later analysis.

In summary, the heterogeneous computing system illustrated in Figure 4 provides distributed computing, distributed memory, and distributed input/output for the stock option pricing application.

Our heterogeneous computing system integrates diverse functions-computation, visualization, and system control over a diverse set of hardware. We use a mix of programming languages on the remote compute nodes-Fortran77 on the DEC5000, C on the SUN4, CM-Fortran on the CM-5, and MPL (data parallel C) on the DECmpp-12000. At the operating system level, all remote modules are compiled and linked as stand-alone programs. Input and output ports are defined in modules by the programmer using specific library routines provided by AVS. Each module represents a process.

There are two source of input data: historical market data read from disk files, and runtime input of model parameters by the user through a GUI on the home machine. Output from all four models is rendered in a graphics window of the GUI and displayed numerically in a shell window, both on the home machine, and written to a database on

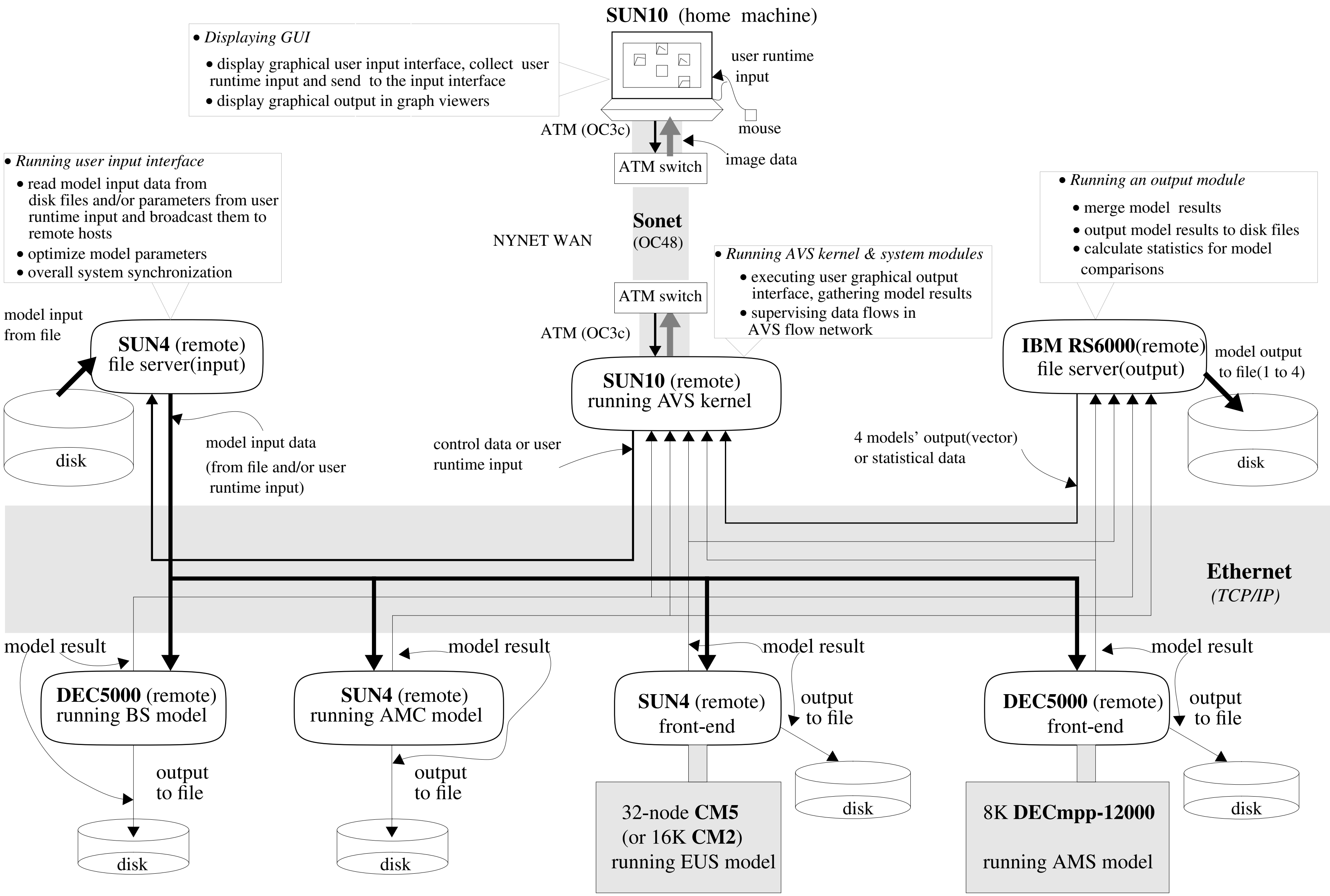

Figure 3.1: System Configuration 1 for the Financial Modeling on CM-5 and DECmpp-12000

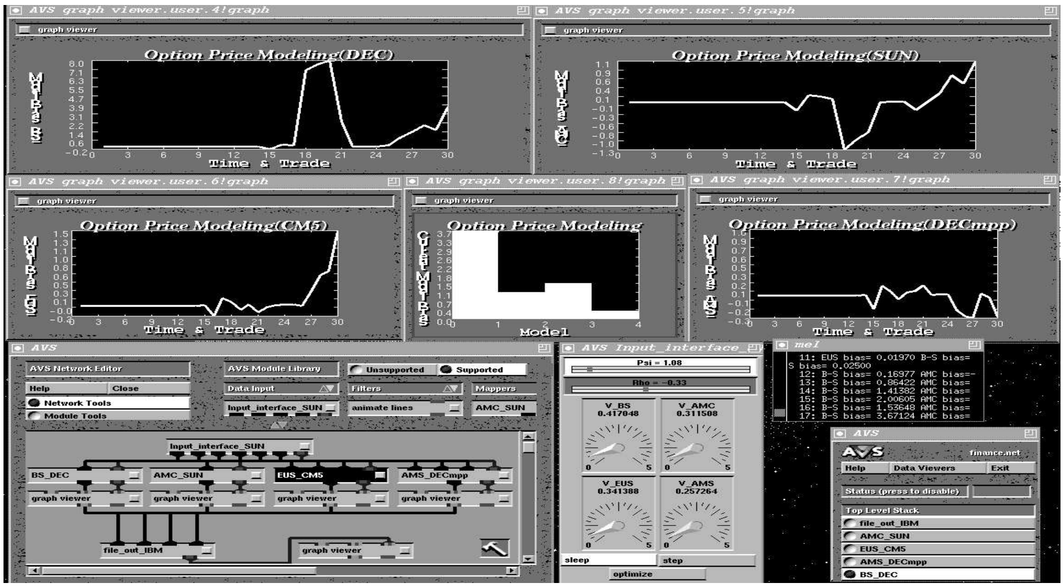

FIG. 2. *The Graphical User Interface for the option price modeling (1)*

the file server.

Figure 2 illustrates the GUI for managing user runtime input and output, and the system configuration. Runtime input includes user defined model parameters and system execution modes (continues, step or pause). Outputs include 2-dimensional displays of model and market prices calculated by the compute nodes. The system configuration includes choice of pricing models, network configurations and interface layouts.

Pricing models are extremely sensitive to model parameters for implied volatility, variance of stock volatility and correlation between stock price and its volatility. These parameters may be read from data files (historical estimates), calculated just prior to running the pricing models (by tuning on the optimization button from the GUI), or defined at run time (by expert user's parameter steering from the GUI).

Configuration 2 - NYNET + IBM SP2 + DEC Alpha Farm + Workstations

Figure 1.2 is the system configuration of the second prototype using the same visualization/computing framework while the two parallel pricing models are now ported to two clusters instead of the two MPPs in the previous configuration.

The two parallel pricing models (EUS model and AMS model) are implemented in PVM and run respectively on a 8-node IBM SP2, networked by an Ethernet at the time of evaluation, and a 8-node DEC Alpha cluster inter-connected by a FDDI-based GIGA-switch. They are coupled under the proposed AVS environment with the other two sequential simple models (BS model and AMC model) running on a SUN4 and a DEC5000 workstation, respectively. The nodal processor of SP2 is IBM RISC/6000 processor running at 62.5 MHz. The DEC Alpha farm consists of 8 Alpha model 4000 workstations which are supported by a high performance networking backbone of a dedicated, switched FDDI segments. The GIGA-switch provides full FDDI bandwidth and low latency switching to every workstation in the farm.

All models output are graphically displayed on the end-user's home machine (a SUN10)

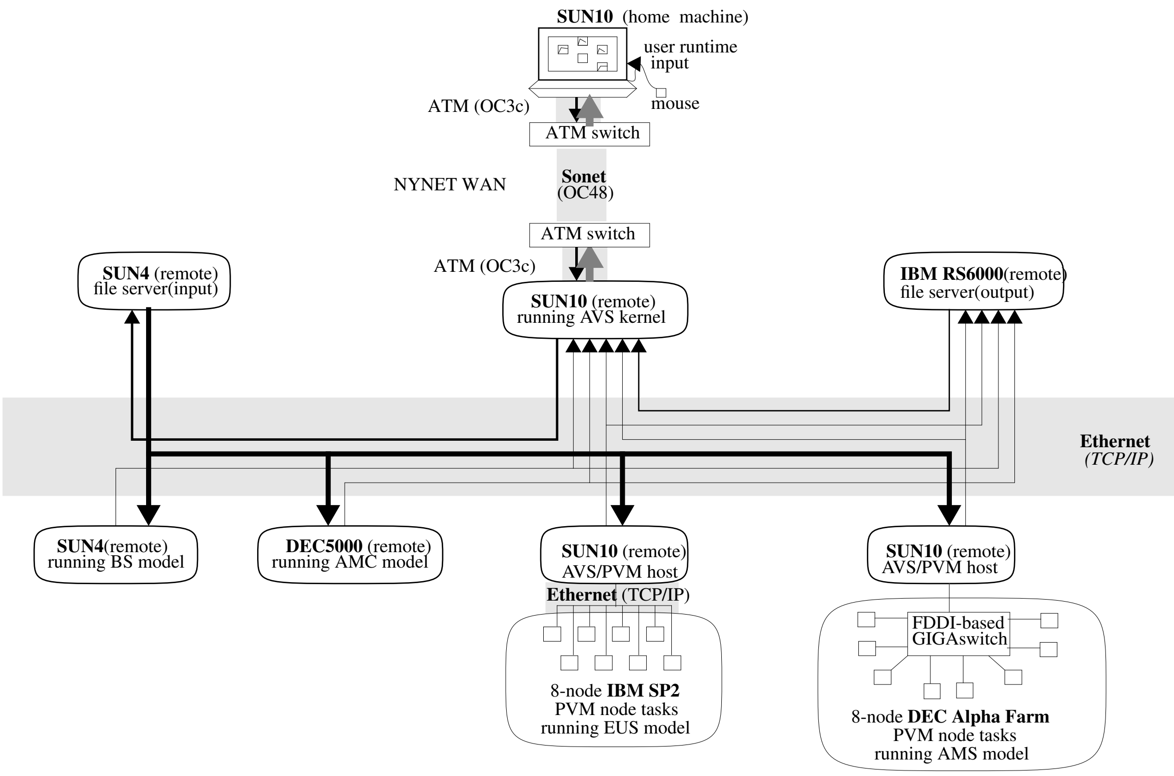

Figure 3.2: *System Configuration 2 for the Financial Modeling on IBM SP2 and DEC Alpha Farm over NYNET*

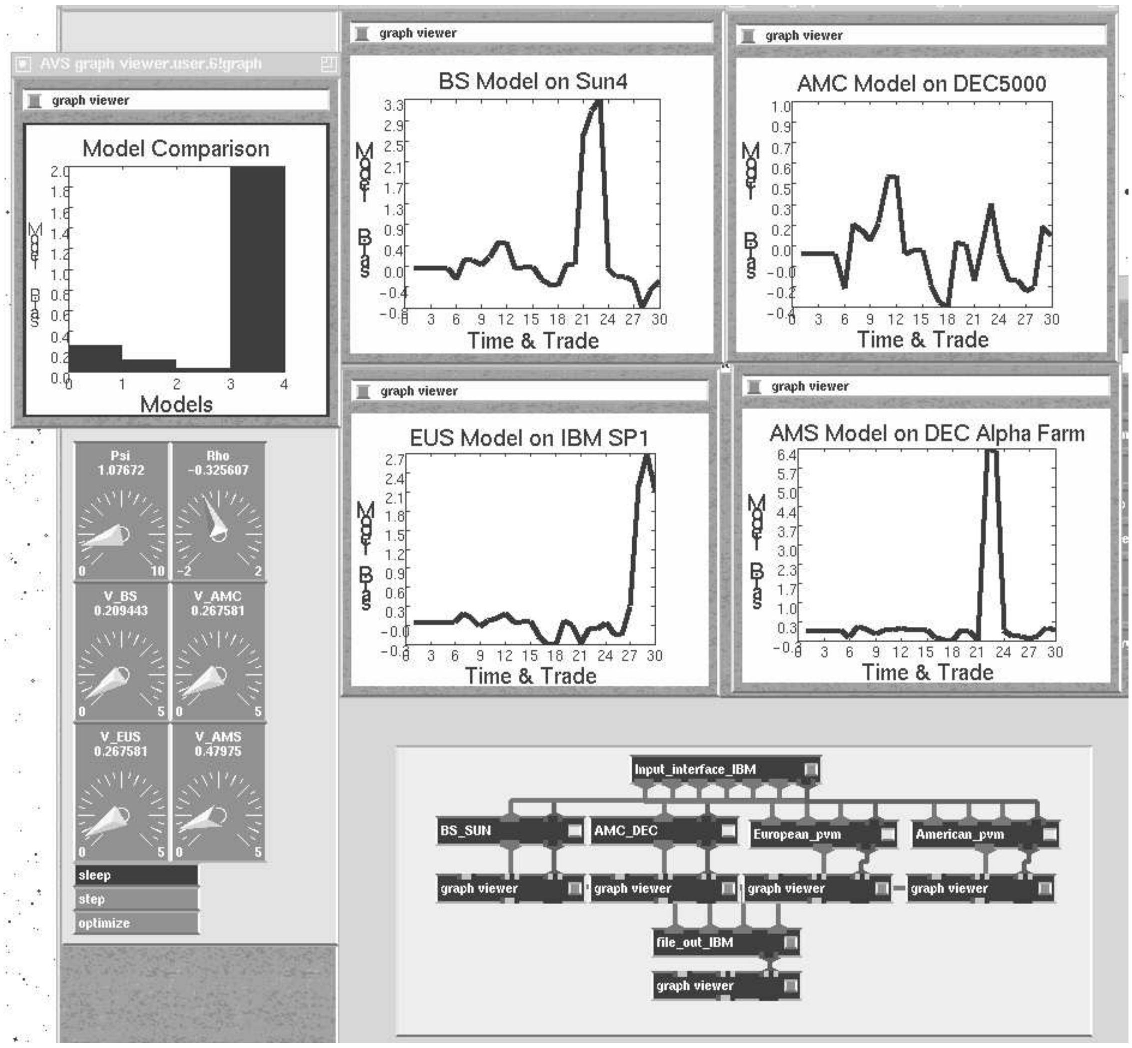

FIG. 3. *The Graphical User Interface for the option price modeling (2)*

in AVS graph viewers. Figure 3 is the screen dump of the GUI same as that in Figure 2 with a different layout, showing the simulation control panel(left), model output windows(top) and the flow network(bottom).

3.3 Performance Analysis

A generalized performance model of our networked system is shown in Figure 4. In one complete modeling cycle, starting with broadcast of new data from M_I, and ending with renderings on M_h, let $t_{\rm calc}$ be the summed calculation time on all machines and $t_{\rm comm}$ be the summed communication time in the system. Let T_d, T_s be the total time to complete such a cycle in the distributed system and in a single-machine system, respectively. Other symbols are defined as follows: $c_{\rm k}$ – communication time of required message passing from M_I to M_k; $d_{\rm k}$ – communication time of required message passing from M_k to M_h; $m_{\rm k}$ – calculation time of performing required computation on M_k; $r_{\rm k}$ – calculation time of performing required rendering by P_k on M_k; $t'_{\rm calc}$ – the total calculation time on a single-machine system; $m'_{\rm k}$ – calculation time of performing required computation on a single machine system.

In a single-machine system, $M_I = M_h = M_1 = M_2 = \cdots = M_n, t_{\rm comm} = 0$ and $T_s = t'_{\rm calc} = \sum_{k=1}^{n} (m'_k + r_k)$. In the distributed system based on TCP/IP protocol, we observe: sequential message passing from M_I to M_k, although the sending order can be scheduled; execution of computing process on M_k upon receiving new message from M_I; and sequential rendering of process P_k on a single-processor machine M_h, where each P_k is activated only after receiving a new message from M_k, and no other $P_j (j \neq k)$ is running on M_h.

Calculation and communication among different machines in the system are pipelined. We can overlap the following processes: (1) c_k with m_j, d_j, and r_j; (2) m_k with d_j and r_j; and (3) d_k with r_j. Figure 5 illustrates this overlap of processes in system of $(n = 2)$ machines. For each machine M_k, the total time to complete a cycle is $T_d = c_k + m_k + d_k + r_k + i_k$, where i_k is the idle time of M_k in the cycle, excluding the waiting time r_k for the rendering on M_h. Thus, $nT_d = t_{\rm comm} + t_{\rm calc} + t_{\rm idle}$, where $t_{\rm comm} = \sum_{k=1}^{n}(c_k + d_k)$, $t_{\rm calc} = \sum_{k=1}^{n}(m_k + r_k)$, and $t_{\rm idle} = \sum_{k=1}^{n} i_k$.

The speed-up S of a distributed system over a single-machine system is:

$$S = \frac{T_s}{T_d} = \frac{n}{\frac{t_{\rm calc}}{t'_{\rm calc}} + \frac{t_{\rm comm}}{t'_{\rm calc}} + \frac{t_{\rm idle}}{t'_{\rm calc}}},$$

Speedup depends on four components: (1) n, the number of distributed computing modules; (2) $t_{\rm calc}/t'_{\rm calc}$, the ratio of total calculation time on the distributed system to calculation time on a single-machine system; (3) $t_{\rm comm}/t'_{calc}$, the ratio of total communication time in the distributed system to calculation time on a single-machine system; and (4) $t_{\rm idle}/t'_{\rm calc}$, the ratio of total idle time in the distributed system to calculation time on a single-machine system. As Figure 5 illustrates, for a given application on a given system (fixed $c_{\rm k}$, $d_{\rm k}$,$m_{\rm k}$,$r_{\rm k}$) $t_{\rm idle}$ depends on the order of messages sent from M_I to M_k, making this a scheduling problem. In the best case, $t_{\rm idle} = \sum_{k=1}^{n-1} c_k(n-k) + \sum_{k=1}^{n} a_k r_k$, where $\{a_1, a_2, \ldots, a_n\}$ is a permutation of the set $\{0, 1, \ldots, n-1\}$.

Timings observed for the system configuration illustrated in Figure 1 are listed in Table 1. According to our analysis, expected $T_d = c_1 + m_1 + d_1 + r_1 + r_2 + r_3 + r_4 =$ 0.017 + 0.015 + 0.1 + 4*0.9 = 3.642 seconds (assuming the scheduled order is c_1, c_2, c_3, and c_4). This value will vary because the system runs under a resource-and-time-sharing environment.

To compare the performance of distributed and single-machine systems, we used a SUN4 single-machine system. Summarizing timings listed in Table 1, T_s = 0.015 + 0.085 + 4.05 + 4.25 + 4*0.9 = 12.0, and we calculate an expected speed-up of S = 12.0/3.642 = 3.3.

4 A Computational Electromagnetic Application

4.1 Introduction and Problem Description

Electromagnetic scattering is a widely encountered problem in electromagnetics[8], with important applications in industry such as microwave equipment, radar, antenna, aviation, and electromagnetic compatibility design. Figure 6 illustrates the EMS problem we are

price model/machine	m_k	c_k $(M_I \Rightarrow M_k)$	d_k $(M_k \Rightarrow M_h)$	r_k $(520 \times 550 window)$
BS/DEC5000(M_1)	0.015	0.017	0.1	0.9
AMC/SUN4(M_2)	0.085	0.017	0.1	0.9
EUS/8K DECmpp 12000(M_3)	0.075	0.017	0.1	0.9
EUS/SUN4(M_3)	4.05			
AMS/32-node CM5(M_4)	0.025	0.017	0.1	0.9
AMS/8K DECmpp 12000(M_4)	0.045			
AMS/SUN4(M_4)	4.25			

TABLE 1

Experimental timings for the option price modeling (in seconds)

modeling. Above an infinite conductor plane, there is an incident EM field in free space. Two slots of equal width on the conducting plane are interconnected to a microwave network behind the plane. The microwave network represents the load of waveguides, for example, a microwave receiver. The incident EM field penetrates the two slots which are filled with insulation materials such as air or oil. Connected by the microwave network, the EM fields in the two slots interact with each other, creating two equivalent magnetic current sources in the two slots. A new scattered EM field is then formed above the slots. We simulate this phenomena and calculate the strength of the scattered EM field under various physical circumstances. The presence of the two slots and the microwave load in this application requires simulation models with high performance computation and communication. Visualization is very important in helping scientists to understand this problem under various physical conditions.

In previous work, data parallel and message passing algorithms for this application were developed to run efficiently on massively parallel SIMD machines such as Connection Machine CM-2 and DECmpp-12000, and MIMD machines such as the Connection Machine CM-5 and iPSC/860. The data parallel algorithms run approximately about 400 times faster than sequential versions on a high-speed workstation[9]. Parallel models on high performance systems provides a unique opportunity to interactively visualize the EMS simulation in real-time. This problem requires response time of the simulation cycle that is not possible on conventional hardware.

Figure 6 also shows physical parameters of the electromagnetic scattering problem.

The moment method [8] is used as the numerical model for the EMS problem, which can be represented as:

$$\{\left[\ Y^a\ \right] + \left[\ Y^b\ \right]\}\vec{V} = \vec{I}$$

$$\left[\ H\ \right] = \mathcal{L}\{f\ (\vec{V}, \vec{M}, \left[\ H_0^2\ \right])\}$$

where,

$\left[\ Y^a\ \right]$: equivalent admittance matrix of the free space;

$\left[\ Y^b\ \right]$: equivalent admittance matrix of the microwave network;

$\vec{V}$: coefficient vector;

$\vec{I}$: the excitation vector;

$\vec{M}$: a vector of mode functions;

$\left[H_0^2 \right]$: matrix of Hankel functions;
f : a function;
$\mathcal{L}$: a linear operator on f;
$\left[H \right]$: final matrix of the simulated EMS field strength.

4.2 System Configuration and Integration

From the previous parallel algorithm design, we observed that:

1. Calculations of $\left[Y^a \right]$, $\left[Y^b \right]$, $\vec{I}$, $\vec{M}$, and $\left[H_0^2 \right]$ can be done independently;
2. Computation of $\left[Y^a \right]$, $\left[H_0^2 \right]$, and the linear solver for $\vec{V}$ have significant communication requirements and are computation intensive;
3. $\left[Y^b \right]$ is a sparse matrix and calculation of $\vec{M}$ requires little time. Calculation time for $\left[Y^a \right]$, $\left[Y^b \right]$ and $\vec{I}$ are relatively balanced.

Thus, we partition computations of this application into four loosely coupled computing modules (they are named as 'EM-1-SUN', 'EM-2-SUN', 'EM-3-CM5' and 'EM-all-CM5' in the AVS module network of Figure 8). Three modules can run simultaneously in the distributed computing environment (see Figure 7).

At the second level, i.e., data decomposition, because most computations of $\left[Y^a \right]$, $\vec{V}$(a linear solver), $\left[H_0^2 \right]$, and $\left[H \right]$ are matrix manipulations, data parallel algorithms are developed in Fortran90 and tailored to run on the Connection Machine 5, to take the advantages of CM5's balanced data network and control network. The CM Scientific Subroutine Library (CMSSL) is used in the data parallel implementation.

At a more general level, we can view the entire system a 'meta- computer' that makes use of both functional parallelism and pipelining. In this application, functional parallelism consists of graphical I/O (i.e., user interaction, 3D rendering) and decomposed simulation computations which are handled concurrently by different components of the meta-computer. Pipelining combines calculations and communications among different processors or groups of processor(e.g. the CM5) that are carried out simultaneously in consecutive stages of the simulation.

Figure 7 illustrates the system configuration and module components distributed over the network connecting three high-end workstations and a supercomputer Connection Machine 5. The network is a 10 MBit/s Ethernet-based local network. AVS software is used to provide sophisticated 3D data visualization and system control functionality required by the simulation.

The local machine in our system is a IBM RS/6000 with a 24-bit color GTO Graphics Adaptor. An AVS coroutine module (in C) on the local machine serves as a graphical input and system control interface to monitor and collect user runtime interaction with the simulation through keyboard, mouse and other I/O devices. The AVS kernel also runs on the local machine, coordinating data flows and control flows among AVS (remote) modules in the network.

The computation intensive modules of this application are distributed to a CM5, a MIMD supercomputer which is configured 32 processing nodes at NPAC. Each processing node(PN) of the CM5 consists of a SPARC processor for control and non-vector compu-

Module name	Calculation time	Communication time with 'Input-interface-IBM'	Communication time with 'EM-all-CM5'	Communication time with 'EM-3D-IBM'
EM-1-SUN	0.1(Sun)	0.02	0.045	n/a
EM-2-SUN	0.6(Sun)	0.02	0.5	n/a
EM-3-CM5	1.8(CM5) 1260(Sun)	0.02	0.0	n/a
EM-all-CM5	2.1(CM5) 120(Sun)	n/a	n/a	0.5
AVS Rendering Modules	3.5 (550x580 window)	n/a	n/a	n/a

TABLE 2

Timings of calculations and communications for the EMS (in second)

tation, four vector units for numerical computation and 32 MB of RAM. It also includes a Network Interface chip which gives the node access to the CM5 internal Data Network and Control Network. The two internal networks connect all the PNs with a control processor(CP) which runs a custom version of SunOS on a SPARC host. Two Sun SPARC workstations are used in our distributed visualization environment to run the computational modules with modest communication requirements.

All modules other than those on the local machine are implemented as AVS remote modules. Their input/output ports are defined by specific AVS libraries for receiving/sending data from/to other (remote) modules via socket connections. This configuration allows the interrupt driven user interface input mechanisms and rendering operations to be relegated to the graphical workstation, while the computation intensive components run on the CM5 coupled with the two workstations. This distributed simulation environment implemented in AVS provides a transparent mechanism for using distributed computing resources along with a sophisticated user interface component that permits a variety of interactive, application-specified inputs.

The graphical user interface shown in Figure 8 includes a main control panel, three individual input panels, and a 3D rendering window. The main control panel provides the user parameters input and simulation control at runtime. There are seven dial widgets representing simulation parameters used by all computing modules on the three remote machines, and a control button for starting a new simulation cycle (see lower left in Figure 8).

4.3 Performance Analysis

Our experiments show that under a typical working environment (only 0.5 MBits/s of the Ethernet's 10 MBits/s capacity are available), a complete simulation cycle takes about 8 seconds. This response time is quite satisfactory for this application. Table 2 lists timing data of major system components. For comparison, timings of sequential implementation on a SUN4 workstation of the two parallel modules are also given in the table.

5 A Computational Chemistry Application

5.1 Introduction and Problem Description

Computational chemistry has long been an important user of high-performance computing. Chemistry applications have historically used a significant fraction of available time on traditional supercomputers, and as efficient parallel implementations become increasingly available usage of massively parallel processors (MPPs) is likewise growing.

The field of computational chemistry includes a variety of methods ranging from $O(n^2)$ in cost (for molecules containing n atoms) to $O(n^4)$ and higher, and generally require memory, I/O, and other resources in accord with their cost. Realistic chemical simulations can involve substantial run times and produce significant amounts of data of interest to the chemist. Visualization of this data most often takes place at the end of the calculation, but because of the length of these calculations, there is an increasing interest in being able to visualize intermediate results as they become available for purposes of computational steering.

In this work, we have integrated a parallel version of the semi-empirical electronic structure package MOPAC [11] into an AVS environment which provides for control of MOPAC's inputs and visualization of intermediate and final results. Our focus is on one of the most common types of calculations using this method, *geometry optimization*, which provides information about energetically stable structures of molecules. These optimized structures are often used as inputs for further calculations. Geometry optimizations of large molecules can be slow to converge and could benefit greatly from the ability to examine intermediate results as the calculation progresses.

5.2 Parallelization of MOPAC

MOPAC is a general-purpose semi-empirical molecular orbital package for the study of chemical structures and reactions. Semi-empirical Hamiltonians are used in the Hartree-Fock [13] method to obtain molecular orbitals, the heat of formation and its derivative with respect to molecular geometry. Using these results MOPAC calculates the vibrational spectra, thermodynamic quantities, isotopic substitution effects and force constrains for a wide range of molecules, including radicals, ions, and polymers. Although MOPAC has been very successfully in many research projects, its application is limited to relatively small molecules, typically consisting of no more than 60 atoms. The limitation comes from the CPU demand that increases with the number of atoms n roughly as $O(n^3)$. A possible solution is to port the code to parallel or distributed computers which is in progress in our laboratory [14].

The structure of MOPAC is described in [11]. Because of the flexibility of the code, *nine* different paths are possible through the code. Central to most of these paths, including the optimization procedures on which we have focused our attention, is the driver routine COMPFG.

We have used three representative test cases to isolate the computation expensive components of the code: retinal (63 atoms), porphin (38 atoms), and tetrabenzene (62 atoms). As shown in Table 3, COMPFG and its descendants account for 99% of the total execution time. Within COMPFG, the primary routines are those involving the two $O(n^3)$ steps in the calculation: DENSIT calculates the electronic density matrix in what amounts to a simplified matrix multiplication and DIAG and HQRII are diagonalization routines. DIAG is, in fact a "fast pseudo-diagonalizer" [12] which takes into account the fact that in most cases, only block diagonalization is required in the Hartree-Fock method. The last

Test sets	retinal		porphin		tetrabenz	
	time(sec)	%	time(sec)	%	time(sec)	%
Total CPU time	4817.16	100.0	734.74	100.0	2905.28	100.0
COMPFG CPU time	4791.24	99.5	726.76	98.9	2884.62	99.3
DENSIT CPU time	795.24	16.5	106.49	14.5	469.61	16.2
DIAG CPU time	3090.48	64.2	481.56	65.5	2027.46	69.8
HQRII CPU time	400.97	8.3	12.86	1.8	56.00	1.9
DENSIT+DIAG+HQRII	4286.69	89.0	600.91	81.8	2553.07	87.9

TABLE 3
IBM SP2 timing of kernel subroutines (in second)

column shows that CPU time spent by the three kernel subroutines, DENSIT, DIAG, and HQRII, is more than 80% of total CPU time. Applying the following formula by assuming there are unlimited processors and no communications needed, we can get the theoretical maximum Speed-up of 9.1, 5.5, and 8.3 for our test sets.

$$Speedup = \frac{T_{Total}}{T_{non-parallel} + \frac{T_{parallel}}{N_{processor}}}$$

5.3 System Configuration and Integration

The system can be divided into four parts. Fig 9 shows the system configuration of our MOPAC visualization/computing system.

- **PCH (Parallel Computation Host)** a parallel computer which handles the computation of time-intensive modules (COMPFG in our experiment).
- **SCH (Sequential Computation Host)** a sequential computer which handles file I/O, data distribution, and other sequential modules. It also acts as the coordinator of all computation nodes.
- **AVSH (AVS server)** an AVS server which handles graphic output and controls the execution of modules.
- **XDH (X-windows Display Host)** a color workstation or X-terminal with at least 8-bits pseudo color to display the graphic output.

The PCH is a scalable meta-computer consists of any combination of a SUN4 workstations cluster, an IBM RS6000 cluster, an IBM SP2, and a DEC Alpha Farm. The inter-connection among those computation nodes is a heterogeneous network consists of an Ethernet, an IBM SP2 High Performance Switch, and a DEC Giga-switch. The MOPAC kernel module is implemented as PVM parallel modules and runs on PCH. It is initiated remotely by the SCH. Although the PCH consists of all the compute nodes physically connected, the actual configuration and number of nodes used can be dynamically determined and modified from the AVS GUI at run time to achieve system load balance and to fit the requirements of input data.

The SCH runs an AVS remote module (*mopacavs*), which is essentially the host program of MOPAC kernel module. It takes the parameters from the AVS control panel. Via the PVM-based heterogeneous network, it handles runtime configuring of the PCH, distributes

	retinal		porphin		tetrabenz	
	Time	Speed-up	Time	Speed-up	Time	Speed-up
$P = 1$	31672	1.00	4869	1.00	19007	1.00
$P = 2$	19222	1.65	3395	1.43	11395	1.67
$P = 4$	16432	1.93	2753	1.77	9059	2.10
$P = 6$	15587	2.03	2539	1.92	8278	2.30

Table 4

The speed-up of the test set on Sun4 cluster

and initiates PVM node tasks on computation nodes. It also post-processes the results to translate them into AVS graphic calls. The SCH is a high performance workstation in our implementation.

The AVSH can be either a SUN4 or an IBM RS6000. It runs the AVS kernel and handles the communications among remote and local modules. The AVS flow network of MOPAC modules is shown in the right-lower of Fig 10. The input parameters and the configuration of PCH are set through the control panel of module *mopacavs*. Module *atomcntl* controls the display of atom structures. Users can move, zoom, or rotate the atom structures by clicking the bottoms on control panel to get the best view angle of atom structures. The connection between remote and local modules are handled by AVS channels.

5.4 Performance Analysis

The theoretical maximum Speed-up of 9.1, 5.5, and 8.3 for our test sets are based on the assumption of unlimited number of processor and no communications. In our present, straightforward parallelization, the first kernel subroutine, DENSIT, requires only one communication to gather results in the end of computation. However, DIAG and HQRII both require numerous small messages during computation. Due to the large amount of communication, we can expect the efficiency of CPU drops sharply when the number of processing node increases.

As shown in Table 4, overall parallel performance on a cluster of Sun4 workstations is not very close to the theoretical limiting values. This is primarily due to the high communication cost of the diagonalization. We are presently working on replacing the eigen-solvers with a more sophisticated parallel eigen-solver [10] , and expect performance to significantly improve. Moving to true parallel computers will also improve the situation compared to the network cluster environment.

6 Conclusion

The availability of high speed networks, combined with advances in computational power, will support a new set of applications designed for high performance distributed computing. In real world HPCC applications, computational power alone is not sufficient. We see a need for a more broader parallel software engineering environment such as the interactive visualization systems described in this paper, which combines and takes advantage of high performance computing and other well established computing areas such as data visualization, computer graphics, system integration and software engineering. We focus here on an execution model required to combine real-time modeling/simulation, parallel processing, visualization and interactive user control.

With its visual programming interface, modular program structure, dataflow based execution, interactive visualization functionality and its open system characteristics, a process-based integration CASE software like AVS provides an excellent framework to facilitate integration of various system components in a large, multi-disciplinary application, including integration of sophisticated interactive visualization, database management, heterogeneous networking, massively parallel processing and real-time decision making, as well as a useful tool for software development. We believe that with the adoption of HPCC technologies into industry applications, data visualization and software engineering will play an ever-growing important role in technology transfer and parallel software development.

References

[1] Advanced Visual Systems Inc. *AVS 4.0 Developer's Guide and User's Guide*, May 1992.

[2] G. C. Fox, et al, *InfoVision: Information, Video, Imagery and Simulation on Demand*, Technical Report, Syracuse Center for Computational Science SCCS-575, December, 1993.

[3] G. C. Fox, et al, *InfoMall: A Scalable Organization for the Development of HPCC Software and Systems*, Technical Report, Syracuse Center for Computational Science SCCS-531, October, 1993.

[4] K. Mills, M. Vinson, and G. Cheng, *A Large Scale Comparison of Option Pricing Models with Historical Market Data,* in the 4th Symposium on the Frontiers of Massively Parallel Computation, Oct. 19-21, 1992, McLean, Virginia.

[5] K. Mills, G. Cheng, M. Vinson, S. Ranka, and G. Fox, *Software Issues and Performance of a Parallel Model for Stock Option Pricing,* in the Fifth Australian Supercomputing Conference, Dec. 6-7, 1992, Melbourne, Australia.

[6] C. Upson, T. Faulhaber, Jr., D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz and A. van Dam, *The Application Visualization System: A Computational Environment for Scientific Visualization*, IEEE Computer Graphics and Applications, July, 1989.

[7] Silicon Graphics Inc. *Iris Explorer User's Guide*, 1992.

[8] R. F. Harrington, *Time-Harmonic Electromagnetic Fields*, McGraw-Hill Book Company, New York (1961).

[9] Y. Lu, A. G. Mohamed, G. Fox and R. F. Harrington, *Implementation of Electromagnetic Scattering from Conductors Containing Loaded Slots on the Connection Machine CM-2*, in Proc. of the 6th SIAM Conference on Parallel Processing for Scientific Computing, March 1993, Norfolk, VA.

[10] D.M. Elwood, G.I. Fann, and R.J. Littlefield. *PeIGS, Parallel Eigensystem Solver.* Pacific Northwest Laboratory, July 1995. available for anonymous FTP via ftp://ftp.pnl.gov/pub/peigs.

[11] Frank J. Seiler Research Laboratory, United States Air Force Academy, CO 80840. *MOPAC Manual*, DEC-3100 edition, December 1990.

[12] J.J.P. Stewart, P. Csaszar, and P. Pulay. Fast Semiempirical Calculations. *Journal of Comp. Chem.*, page pp. 227, 1982.

[13] Atilla Szabo and Neil S. Ostlund. *Modern Quantum Chemistry.* McGraw-Hill, New York, first edition, revised edition, 1989. ISBN 0-07-062739-8.

[14] Tomasz Haupt Tseng-Hui Lin and Geoffrey C. Fox. Parallelizing MOPAC on distributed computing systems within AV S framework. Technical Report Syracuse Center for Computational Science, SCCS-744, Northeast Parallel Architectures Center at Syracuse University, 1995.

[15] J. Cox, S. Ross, and M. Rubinstein, *Option Pricing: A Simplified Approach*, Journal of Financial Economics, Vol. 7, pp. 229-63, 1979.

[16] T. Finucane, *Binomial Approximations of American Call Option Prices with Stochastic Volatilities*, published in Journal of Finance, 1992.

[17] G. Cheng, K. Mills and G. Fox, *An Interactive Visualization Environment for Financial Modeling on Heterogeneous Computing Systems*, in Proc. of the 6th SIAM Conference on Parallel Processing for Scientific Computing, R. F. Sincovec, eds., SIAM, Norfolk, VA, March 1993.

[18] G. Cheng, Y. Lu, G.C. Fox, K. Mills and T. Haupt, *An Interactive Remote Visualization Environment for an Electromagnetic Scattering Simulation on a High Performance Computing System*, in Proc. of Supercomputing '93, Portland, OR, November,1993.

[19] G. Cheng, C. Faigle, G. C. Fox, W. Fumanski, B. Li, and K. Mills, *Exploring AVS for HPDC Software Integration: Case Studies Towards Parallel Support for GIS*, Proc. of the 2nd AVS Conference AVS'93, Lake Buena Vista, FL, May 1993.

[20] G. Cheng, G. Fox, K. Mills and Marek Podgorny, *Developing Interactive PVM-based Parallel Programs on Distributed Computing Systems within AVS Framework,* Proc. of the 3rd Annual International AVS Conference, JOIN THE REVOLUTION: AVS'94, Boston, MA, May 2-4, 1994.

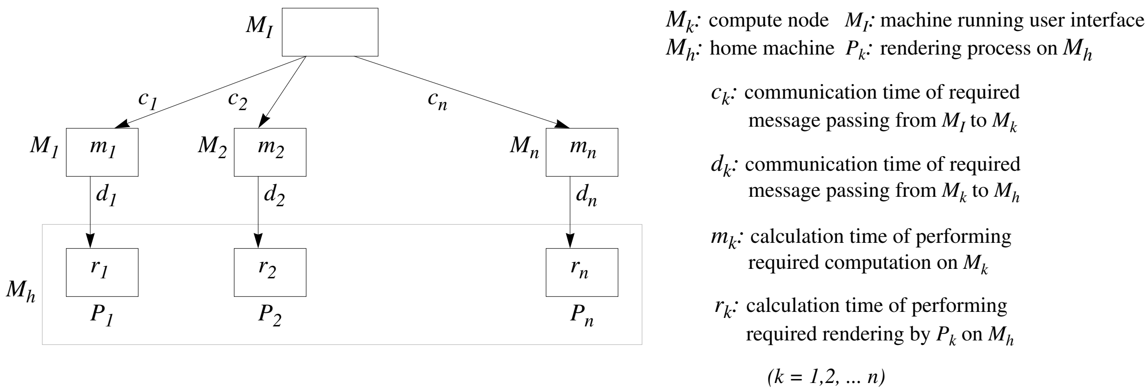

Figure 2.1: A generalized performance model

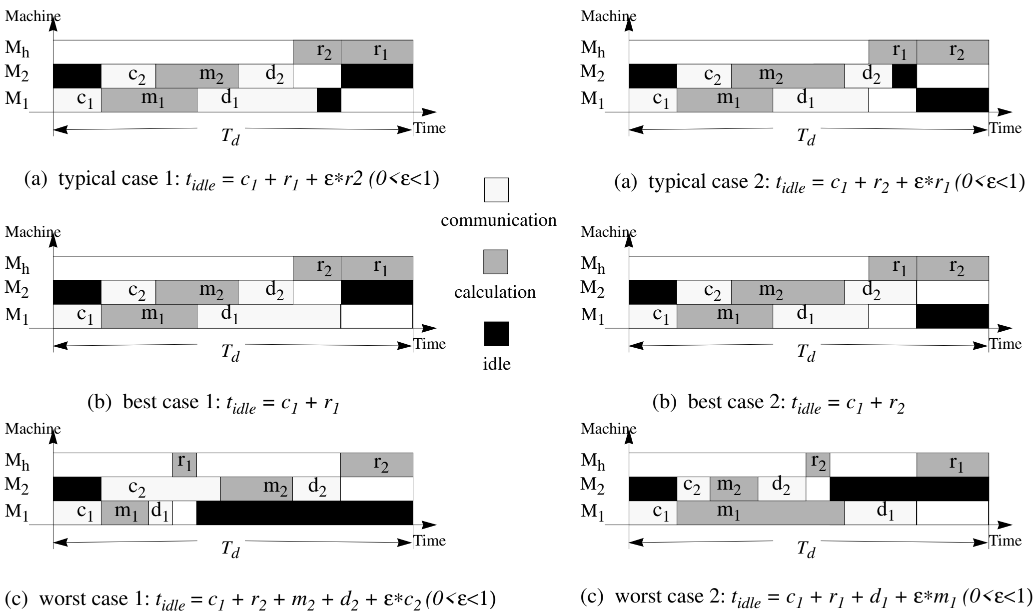

Figure 2.2: Different cases of the pipelined calculations and communications

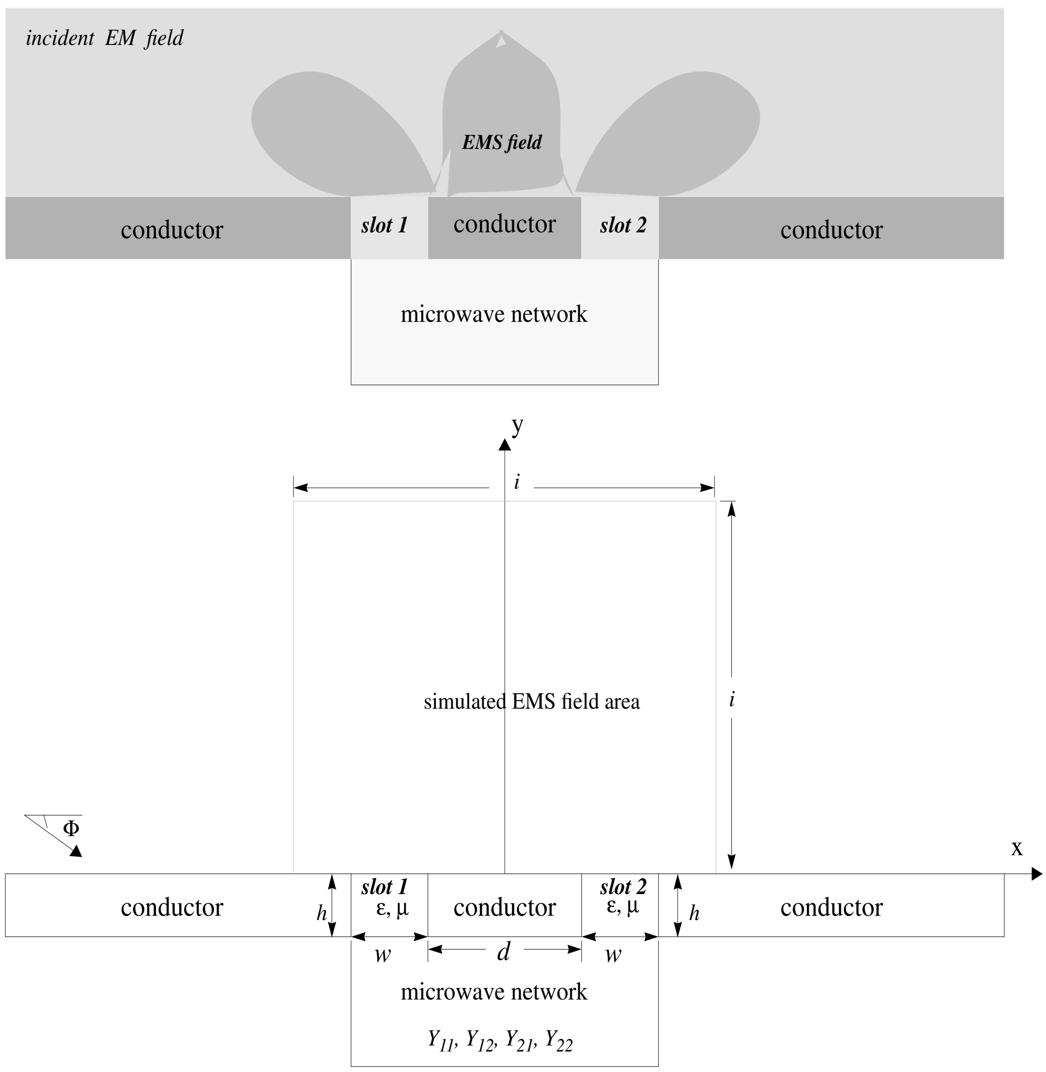

Fig. 6. *Physical parameters of the EM scattering problem*

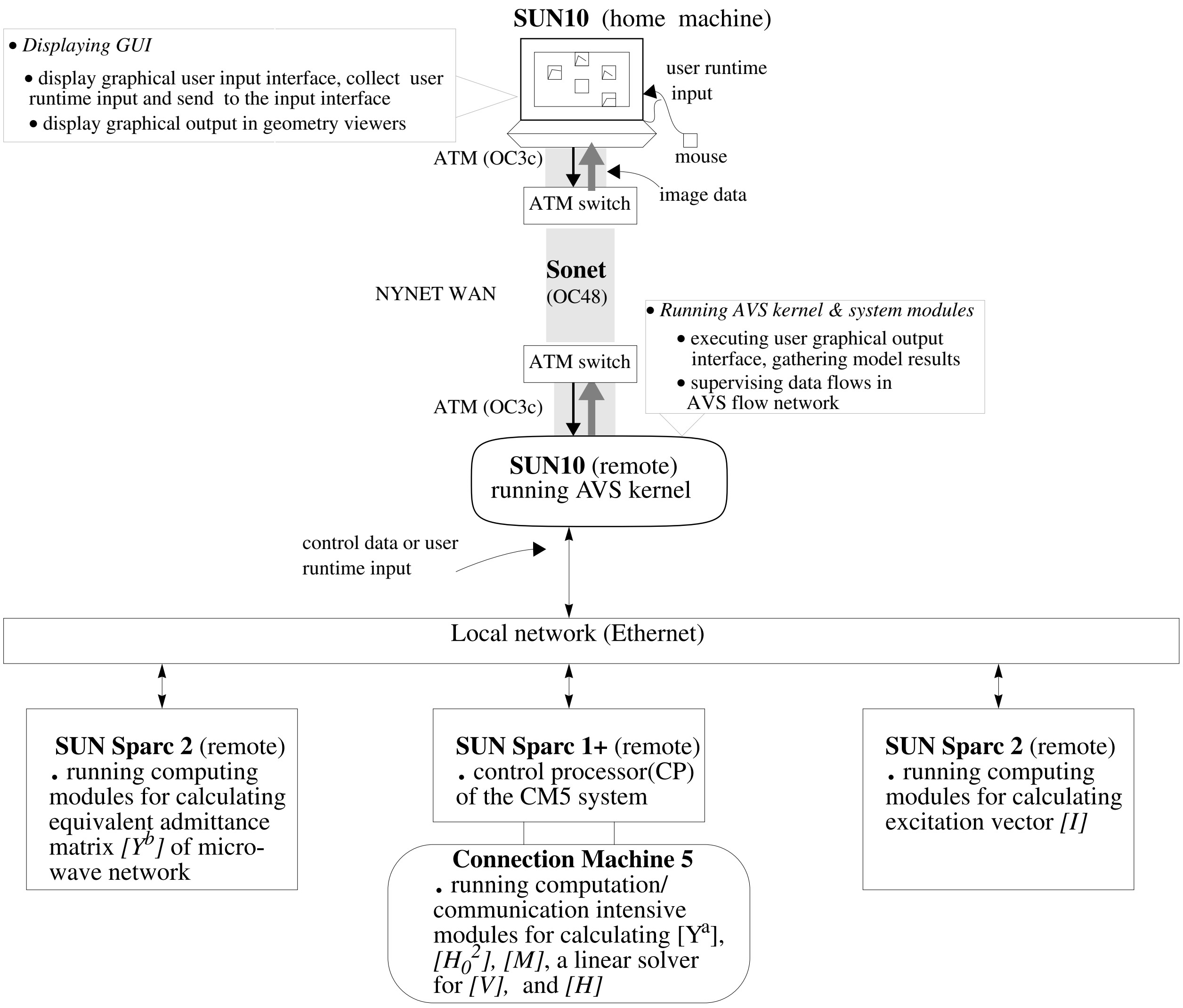

FIG. 7. *System configuration for the EMS application*

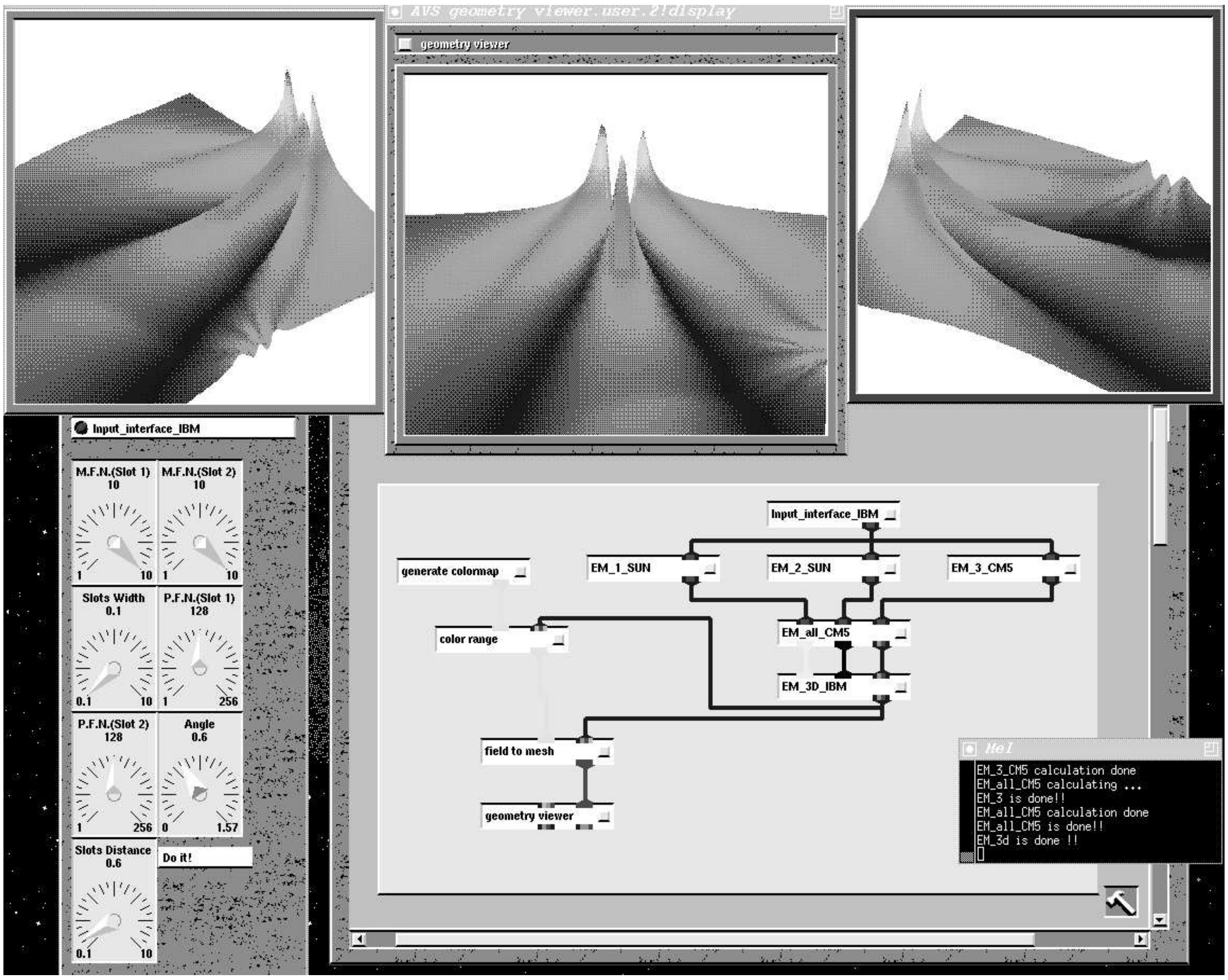

Fig. 8. *Graphic user interface for the EMS visualization system*

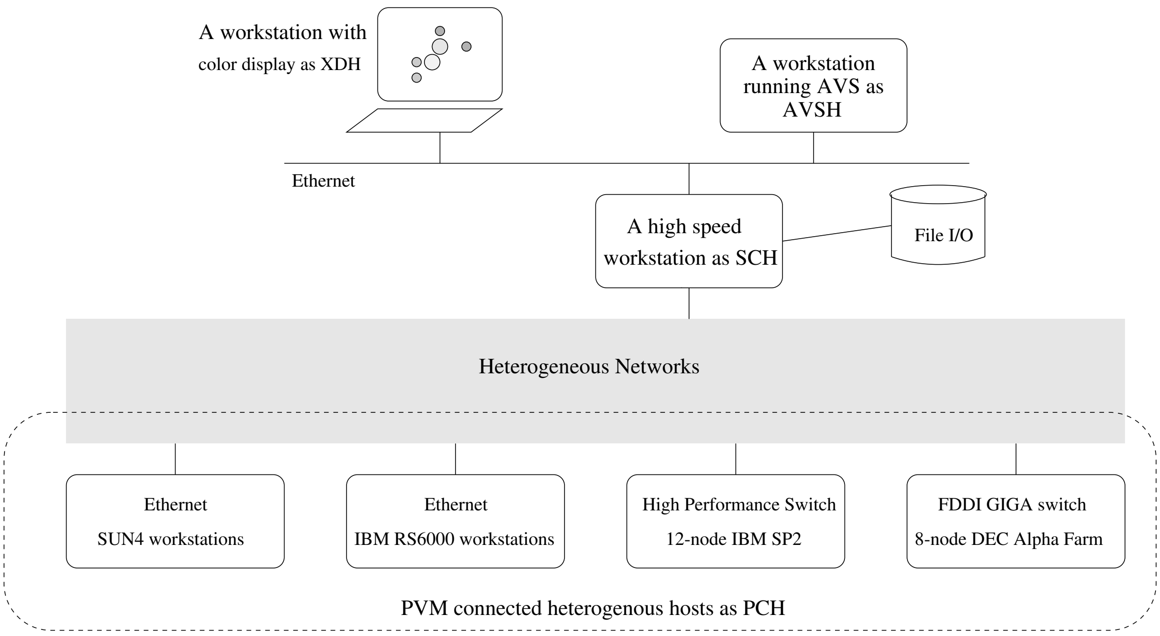

FIG. 9. *System configuration of MOPAC*

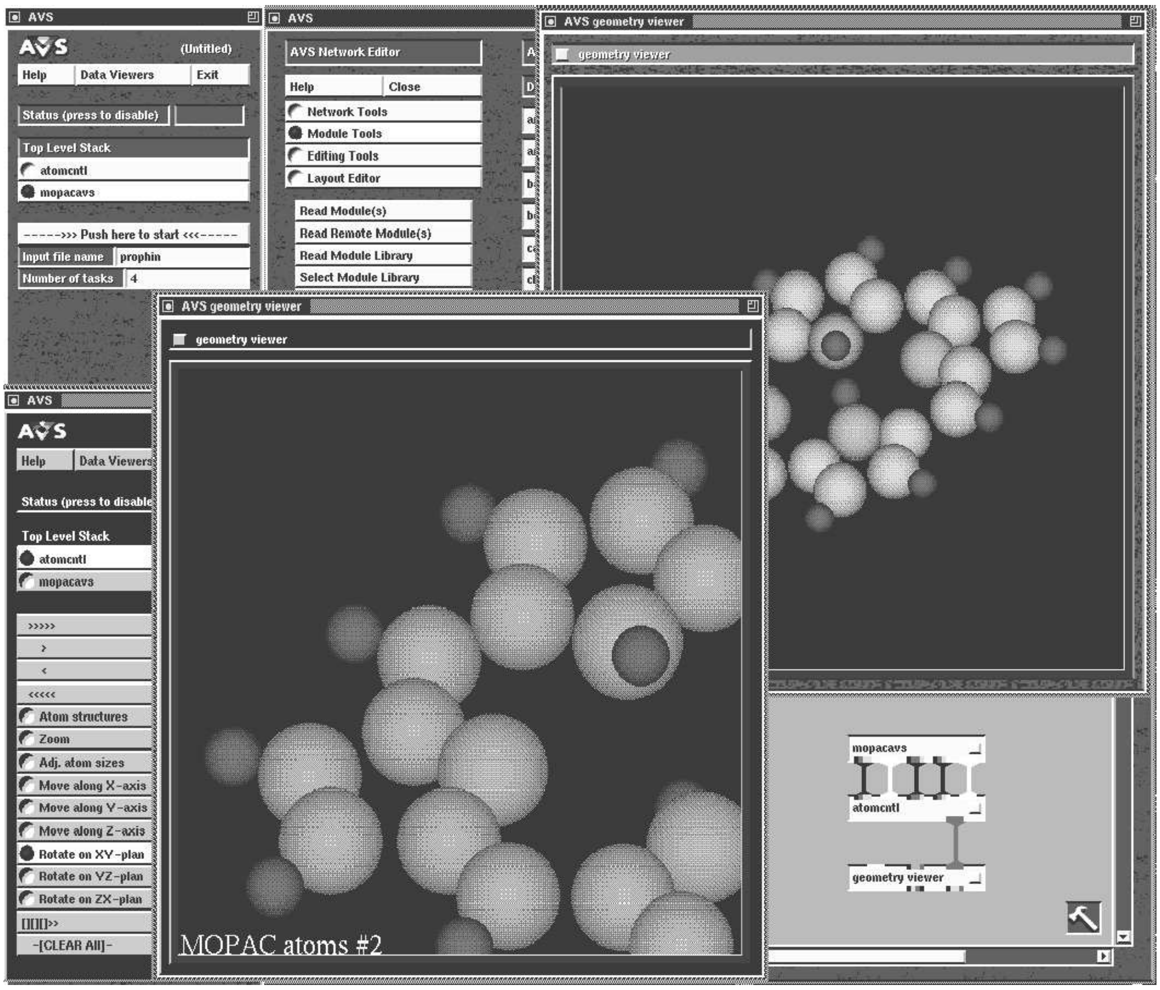

FIG. 10. *The MOPAC graphic interface*