

Introduction to Flexsim DS

Eamonn Lavery, Flexsim Corp.

www.flexsim.com

ABSTRACT

Flexsim DS is a simulation system for internet enabled distributed simulation. Traditionally flexsim is an Visual Object Oriented and Virtual Reality simulation platform for simulating dynamic systems. The main application of flexsim has been discrete event simulation with extensive object libraries for modeling physical and logistical systems. It features embedded RPL and C++ language integration with links to external databases. With the arrival of Flexsim DS, the power and flexibility of flexsim is now available for distributed simulation applications based on the tcpip standard. Features and applications are discussed and case studies are presented.

INTRODUCTION

Flexsim DS is a virtual reality simulation system which provides the ability to build, simulate and immerse the user in 3d worlds. It incorporates an advanced tcpip network layer to facilitate operation across the internet and intranet. It is a visual object oriented simulation tool which has several core abilities of virtual reality, discrete event modeling, a built in rapid prototyping language and full integration with c++. Flexsim DS incorporates the technologies of agent based modeling, high fidelity visualization and distributed simulation.

The key aspects of Flexsim DS are: high fidelity simulation applications; distributed simulation; and it's high fidelity graphics engine. High Fidelity Simulation Applications include:

- multi-user collaboration
- industrial workplace
- military
- architectural
- engineering

Why Distributed Simulation?

Distributed simulation leverages the qualities of conventional virtual reality simulation. It takes the technology to the next level and there are many compelling justifications for bringing the power of traditional simulation and distributed simulation together to form a new technology. the motivations include:

- super scale modeling
- collaborative design

- universal standards (e.g. HLA)
- separate computation and visuals

What is Distributed Simulation?

What is it that distinguishes distributed simulation from traditional simulation? One of the key aspects of distributed simulation is parallelism of logical processes (LPs). The benefit of this is to leverage the power of several computing stations to form one larger, more powerful computing resource. This has been done with tremendous success in the scientific computing world and the military and commercial gaming world already.

Technically there are broad strategies for implementing parallelized computation models: conservative, in which there is effectively a single synchronized model, where processes are dependent upon each other's results; optimistic, in which asynchronous models can operate with relatively loose coupling and utilize a state-rollback mechanism to resynchronize processes which have diverged.

High level considerations for distributing traditional simulation models include: DES models are queuing systems - that is they lend themselves to synchronization of computation. Though it is important to note that traditional simulation models are optimized to run on single stations and thus ignore or even exclude computational behavior that would lend itself to parallelization which exists in real world modeling situations. Plus, the traditional paradigm of simulation meaning 'process modeling' is itself evolving - consider the advance of agent based modeling into main stream simulation. For example

Physically detailed sub models can benefit from being part of a parallelized, or even sequentially computed distributed simulation. Whereas such models would be considered inappropriate for inclusion in a traditional simulation model because the detail was too high and even considered a different type of simulation altogether. Now, with distributed simulation different types of models can be included into a larger, more detailed model by distributing the simulation onto a computing cluster - effectively a simulation cluster. Consider the case where a simulation project would have to choose a simulation tool based on its level of detail capacity and exclude detail that may matter because of limitation of the different tools. The options typically are: (i) very simple high level such as supply chain at country or regional level (ii) intermediate level of detail such as single plant operation (iii) or high fidelity detail such as work cell modeling where ergonomics and spatial geometry can affect the model significantly, or even (iv) a high detail agent based model. The fact that one tool has to be chosen over another means compromise because no one tool fits the purpose by itself. This is primarily because the tools are designed on the assumption that the power of a single computing station is not powerful enough to run a model which can handle all the levels of detail of the subject matter by itself. Distributed simulation removes this assumption. A single distributed simulation could handle all of the aforementioned types of simulation in one model.

Flexsim DS design

The design philosophy of Flexsim DS focuses on leveraging technologies which are ubiquitous in a general computing sense but new to the field of general purpose simulation. Thus providing a new level of simulation capability over traditional simulation: multi user TCPIP architecture; focus on large scale visualization; distributed processing; and network model tools.

TECHNOLOGIES

Flexsim DS incorporates many state of the art technologies; an overview is given in the following sections.

- Distributed Simulation
- Collaborative Virtual Environments
- Virtual reality

- High Level Architecture
- Distributed computing
- Parallel computing

DISTRIBUTED SIMULATION

History

Computer simulation was developed hand-in-hand with the rapid growth of the computer, following its first large-scale deployment during the Manhattan Project in World War II to model the process of nuclear detonation. It was a simulation of 12 hard spheres using a Monte Carlo algorithm. Computer simulation is often used as an adjunct to, or substitution for, modeling systems for which simple closed form analytic solutions are not possible. There are many different types of computer simulation; the common feature they all share is the attempt to generate a sample of representative scenarios for a model in which a complete enumeration of all possible states of the model would be prohibitive or impossible. Computer models were initially used as a supplement for other arguments, but their use later became rather widespread.

Classification of simulation models

Computer models can be classified according to several criteria including:

- Stochastic or deterministic (and as a special case of deterministic, chaotic) - see External links below for examples of stochastic vs. deterministic simulations
- Steady-state or dynamic
- Continuous or discrete (and as an important special case of discrete, discrete event or DE models)
- Local or distributed.

Steady-state models use equations defining the relationships between elements of the modeled system and attempt to find a state in which the system is in equilibrium. Such models are often used in simulating physical systems, as a simpler modeling case before dynamic simulation is attempted.

Dynamic simulations model changes in a system in response to (usually changing) input signals.

Stochastic models use random number generators to model chance or random events; they are also called Monte Carlo simulations.

A discrete event simulation (DES) manages events in time. Most computer, logic-test and fault-tree simulations are of this type. In this type of simulation, the simulator maintains a queue of events sorted by the simulated time they should occur. The simulator reads the queue and triggers new events as each event is processed. It is not important to execute the simulation in real time. It's often more important to be able to access the data produced by the simulation, to discover logic defects in the design, or the sequence of events.

A continuous dynamic simulation performs numerical solution of differential-algebraic equations or differential equations (either partial or ordinary). Periodically, the simulation program solves all the equations, and uses the numbers to change the state and output of the simulation. Applications include flight simulators, construction and management simulation games, chemical process modeling, and simulations of electrical circuits. Originally, these kinds of simulations were actually implemented on analog computers, where the differential equations could be represented directly by various electrical components such as op-amps. By the late 1980s, however, most "analog" simulations were run on conventional digital computers that emulate the behavior of an analog computer.

Agent-based simulation A special type of discrete simulation which does not rely on a model with an underlying equation, but can nonetheless be represented formally, is agent-based simulation. In agent-based simulation, the individual entities (such as molecules, cells, trees or consumers) in the model are represented directly (rather than by their density or concentration) and possess an internal state and set of behaviors or rules which determine how the agent's state is updated

from one time-step to the next. Agent based simulation has been used effectively in ecology, where it is often called individual based modeling and has been used in situations for which individual variability in the agents cannot be neglected, such as population dynamics of salmon and trout (most purely mathematical models assume all trout behave identically). *Distributed models* run on a network of interconnected computers, possibly through the Internet. Simulations dispersed across multiple host computers like this are often referred to as "distributed simulations". There are several standards for distributed simulation, including Aggregate Level Simulation Protocol (ALSP), Distributed Interactive Simulation (DIS), the High Level Architecture (simulation) (HLA) and the Test and Training Enabling Architecture (TENA).

COLLABORATIVE VIRTUAL ENVIRONMENTS

Collaborative Virtual Environments, or CVEs, are used for collaboration and interaction of possibly many participants that may be spread over large distances. Typical examples are distributed simulations, 3D multiplayer games, collaborative engineering software, and others. The applications are usually based on the shared virtual environment. Because of the spreading of participants and the communication latency, some data consistency model has to be used to keep the data consistent.

The consistency model influences deeply the programming model of the application. One classification is presented below. Based on several criteria, like centralized/distributed architecture, type of replication, and performance and consistency properties. Four types of consistency models are described, covering the most frequently used CVE architectures:

Centralized Primaries

All primary replicas of each data item resides on the same computer called server.

Advantages: complete server control over the scene

Disadvantages: performance is limited by the server computer

Distributed Primaries

Primary replicas are distributed among the computers.

Advantages: high performance and scalability

Disadvantages: difficult programming model, weaker consistency

Data Ownership

Primaries are allowed to migrate among the computers. This approach is often called system with transferable data ownership.

Advantages: more flexibility compared to Distributed Primaries

Disadvantages: high amount of ownership requests may limit the system performance

Active Transactions

Active replication uses peer-to-peer approach while all replicas are equal. Usually, atomic broadcast is used to deliver updates to all of them, thus they are kept synchronized.

Advantages: complete scene synchronization (equal scene content on all computers)

Disadvantages: the performance is limited by the slowest computer in the system

VIRTUAL REALITY

Virtual reality (VR) is a technology which allows a user to interact with a computer-simulated environment, be it a real or imagined one. Most current virtual reality environments are primarily visual experiences, displayed either on a computer screen or through special or stereoscopic displays, but some simulations include additional sensory information, such as

sound through speakers or headphones. Some advanced, haptic systems now include tactile information, generally known as force feedback, in medical and gaming applications. Users can interact with a virtual environment or a virtual artifact (VA) either through the use of standard input devices such as a keyboard and mouse, or through multimodal devices such as a wired glove, the Polhemus boom arm, and omnidirectional treadmill. The simulated environment can be similar to the real world, for example, simulations for pilot or combat training, or it can differ significantly from reality, as in VR games. In practice, it is currently very difficult to create a high-fidelity virtual reality experience, due largely to technical limitations on processing power, image resolution and communication bandwidth. However, those limitations are expected to eventually be overcome as processor, imaging and data communication technologies become more powerful and cost-effective over time.

Virtual Reality is often used to describe a wide variety of applications, commonly associated with its immersive, highly visual, 3D environments. The development of CAD software, graphics hardware acceleration, head mounted displays, database gloves and miniaturization have helped popularize the notion. In the book *The Metaphysics of Virtual Reality*, Michael Heim identifies seven different concepts of Virtual Reality: simulation, interaction, artificiality, immersion, telepresence, full-body immersion, and network communication. The definition still has a certain futuristic romanticism attached. People often identify VR with Head Mounted Displays and Data Suits.

Application in Manufacturing

Virtual reality can serve to new product design, helping as an ancillary tool for engineering in manufacturing processes, new product prototype and simulation. Among other examples, we may also quote Electronic Design Automation, CAD, Finite Element Analysis, and Computer Aided Manufacturing. The use of Stereolithography and 3D printing shows how computer graphics modeling can be applied to create physical parts of real objects used in naval, aerospace and automotive industry.

HIGH LEVEL ARCHITECTURE

The High Level Architecture (HLA) is a general purpose architecture for distributed computer simulation systems. Using HLA, computer simulations can communicate to other computer simulations regardless of the computing platforms. Communication between simulations is managed by a Run-Time Infrastructure (simulation) (RTI).

Technical overview

The High Level Architecture (HLA) consists of the following components:

- Interface Specification. The interface specification document defines how HLA compliant simulators interact with the Run-Time Infrastructure (simulation) (RTI). The RTI provides a programming library and an application programming interface (API) compliant to the interface specification.
- Object Model Template (OMT). The OMT specifies what information is communicated between simulations and how it is documented.
- HLA Rules. Rules that simulations must obey to be compliant to the standard.

Common HLA terminology

- Federate - An HLA compliant simulation.
- Federation - Multiple simulations connected via the RTI using a common OMT.
- Object - A collection of related data sent between simulations.
- Attribute - Data field of an object.

- Interaction - Event sent between simulations.
- Parameter - Data field of an interaction.

Interface specification

The interface specification is object oriented. Many RTIs provide APIs in C++ and the Java programming languages. The interface specification is divided into service groups:

- Federation Management
- Declaration Management
- Object Management
- Ownership Management
- Time Management
- Data Distribution Management
- Support Services

Object model template

The object model template (OMT) provides a common framework for the communication between HLA simulations. OMT consists of the following documents:

Federation Object Model (FOM). The FOM describes the shared object, attributes and interactions for the whole federation.
Simulation Object Model (SOM). A SOM describes the shared object, attributes and interactions used for a single federate.

Base Object Model

The Base Object Model (BOM) is a new concept created by SISO to provide better reuse and composability for HLA simulations, and is highly relevant for HLA developers. More information can be found at Boms.info.

DISTRIBUTED COMPUTING

Distributed computing deals with hardware and software systems containing more than one processing element or storage element, concurrent processes, or multiple programs, running under a loosely or tightly controlled regime.

In distributed computing a program is split up into parts that run simultaneously on multiple computers communicating over a network. Distributed computing is a form of parallel computing, but parallel computing is most commonly used to describe program parts running simultaneously on multiple processors in the same computer. Both types of processing require dividing a program into parts that can run simultaneously, but distributed programs often must deal with heterogeneous environments, network links of varying latencies, and unpredictable failures in the network or the computers.

Organization

Organizing the interaction between the computers that execute distributed computations is of prime importance. In order to be able to use the widest possible variety of computers, the protocol or communication channel should not contain or use any information that may not be understood by certain machines. Special care must also be taken that messages are indeed delivered correctly and that invalid messages, which would otherwise bring down the system and perhaps the rest of the network, are rejected.

Another important factor is the ability to send software to another computer in a portable way so that it may execute and interact with the existing network. This may not always be practical when using differing hardware and resources, in which case other methods, such as cross-compiling or manually porting this software, must be used.

Goals and advantages

There are many different types of distributed computing systems and many challenges to overcome in successfully designing one. The main goal of a distributed computing system is to connect users and resources in a transparent, open, and scalable way. Ideally this arrangement is drastically more fault tolerant and more powerful than many combinations of stand-alone computer systems.

Openness

Openness is the property of distributed systems such that each subsystem is continually open to interaction with other systems (see references). Web services protocols are standards which enable distributed systems to be extended and scaled. In general, an open system that scales has an advantage over a perfectly closed and self-contained system. Openness cannot be archived unless the specification and documentation of the key software interface of the component of a system are made available to the software developer.

Consequently, open distributed systems are required to meet the following challenges:

Monotonicity

Once something is published in an open system, it cannot be taken back.

Pluralism

Different subsystems of an open distributed system include heterogeneous, overlapping and possibly conflicting information. There is no central arbiter of truth in open distributed systems.

Unbounded Nondeterminism

Asynchronously, different subsystems can come up and go down and communication links can come in and go out between subsystems of an open distributed system. Therefore the time that it will take to complete an operation cannot be bounded in advance.

Drawbacks and disadvantages

Technical issues

If not planned properly, a distributed system can decrease the overall reliability of computations if the unavailability of a node can cause disruption of the other nodes. Leslie Lamport famously quipped that: "A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable."

Troubleshooting and diagnosing problems in a distributed system can also become more difficult, because the analysis may require connecting to remote nodes or inspecting communication between nodes.

Many types of computation are not well suited for distributed environments, typically owing to the amount of network communication or synchronization that would be required between nodes. If bandwidth, latency, or communication requirements are too significant, then the benefits of distributed computing may be negated and the performance may be worse than a non-distributed environment.

Architecture

Various hardware and software architectures are used for distributed computing. At a lower level, it is necessary to interconnect multiple CPUs with some sort of network, regardless of whether that network is printed onto a circuit board or made up of loosely-coupled devices and cables. At a higher level, it is necessary to interconnect processes running on those CPUs with some sort of communication system.

Distributed programming typically falls into one of several basic architectures or categories: Client-server, 3-tier architecture, N-tier architecture, Distributed objects, loose coupling, or tight coupling.

- Client-server — Smart client code contacts the server for data, then formats and displays it to the user. Input at the client is committed back to the server when it represents a permanent change.
- 3-tier architecture — Three tier systems move the client intelligence to a middle tier so that stateless clients can be used. This simplifies application deployment. Most web applications are 3-Tier.
- N-tier architecture — N-Tier refers typically to web applications which further forward their requests to other enterprise services. This type of application is the one most responsible for the success of application servers.
- Tightly coupled (clustered) — refers typically to a cluster of machines that closely work together, running a shared process in parallel. The task is subdivided in parts that are made individually by each one and then put back together to make the final result.
- Peer-to-peer — an architecture where there is no special machine or machines that provide a service or manage the network resources. Instead all responsibilities are uniformly divided among all machines, known as peers. Peers can serve both as clients and servers.
- Space based — refers to an infrastructure that creates the illusion (virtualization) of one single address-space. Data are transparently replicated according to application needs. Decoupling in time, space and reference is achieved.

Another basic aspect of distributed computing architecture is the method of communicating and coordinating work among concurrent processes. Through various message passing protocols, processes may communicate directly with one another, typically in a master/slave relationship. Alternatively, a "database-centric" architecture can enable distributed computing to be done without any form of direct inter-process communication, by utilizing a shared database.

Multiprocessor systems

A multiprocessor system is simply a computer that has more than one CPU on its motherboard. If the operating system is built to take advantage of this, it can run different processes (or different threads belonging to the same process) on different CPUs.

Multicomputer systems

A multicomputer may be considered to be either a loosely coupled NUMA computer or a tightly coupled cluster. Multicomputers are commonly used when strong computer power is required in an environment with restricted physical space or electrical power.

Computer clusters

A cluster consists of multiple stand-alone machines acting in parallel across a local high speed network. Distributed computing differs from cluster computing in that computers in a distributed computing environment are typically not exclusively running "group" tasks, whereas clustered computers are usually much more tightly coupled. Distributed computing also often consists of machines which are widely separated geographically.

Grid computing

A grid uses the resources of many separate computers, loosely connected by a network (usually the Internet), to solve large-scale computation problems. Public grids may use idle time on many thousands of computers throughout the world. Such arrangements permit handling of data that would otherwise require the power of expensive supercomputers or would have been impossible to analyze.

PARALLEL COMPUTING

Traditionally, computer software has been written for serial computation. To solve a problem, an algorithm is constructed and implemented as a serial stream of instructions. These instructions are executed on a central processing unit on one computer. Only one instruction may execute at a time—after that instruction is finished, the next is executed.

Parallel computing, on the other hand, uses multiple processing elements simultaneously to solve a problem. This is accomplished by breaking the problem into independent parts so that each processing element can execute its part of the algorithm simultaneously with the others. The processing elements can be diverse and include resources such as a single computer with multiple processors, several networked computers, specialized hardware, or any combination of the above.

Fine-grained, coarse-grained, and embarrassing parallelism

Applications are often classified according to how often their subtasks need to synchronize or communicate with each other. An application exhibits fine-grained parallelism if its subtasks must communicate many times per second; it exhibits coarse-grained parallelism if they do not communicate many times per second, and it is embarrassingly parallel if they rarely or never have to communicate. Embarrassingly parallel applications are considered the easiest to parallelize.

Classes of parallel computers

Parallel computers can be roughly classified according to the level at which the hardware supports parallelism. This classification is broadly analogous to the distance between basic computing nodes. These are not mutually exclusive; for example, clusters of symmetric multiprocessors are relatively common.

Distributed computing

A distributed computer (also known as a distributed memory multiprocessor) is a distributed memory computer system in which the processing elements are connected by a network. Distributed computers are highly scalable.

Cluster computing

A Beowulf cluster

A cluster is a group of loosely coupled computers that work together closely, so that in some respects they can be regarded as a single computer. Clusters are composed of multiple standalone machines connected by a network. While machines in a cluster do not have to be symmetric, load balancing is more difficult if they are not. The most common type of cluster is the Beowulf cluster, which is a cluster implemented on multiple identical commercial off-the-shelf computers connected with a TCP/IP Ethernet local area network. Beowulf technology was originally developed by Thomas Sterling and Donald Becker. The vast majority of the TOP500 supercomputers are clusters.

Massive parallel processing

A massively parallel processor (MPP) is a single computer with many networked processors. MPPs have many of the same characteristics as clusters, but they are usually larger, typically having "far more" than 100 processors. In an MPP, "each

CPU contains its own memory and copy of the operating system and application. Each subsystem communicates with the others via a high-speed interconnect."

Grid computing

Grid computing is the most distributed form of parallel computing. It makes use of computers communicating over the Internet to work on a given problem. Because of the low bandwidth and extremely high latency available on the Internet, grid computing typically deals only with embarrassingly parallel problems. Many grid computing applications have been created, of which SETI@home and Folding@Home are the best-known examples.

Most grid computing applications use middleware, software that sits between the operating system and the application to manage network resources and standardize the software interface. The most common grid computing middleware is the Berkeley Open Infrastructure for Network Computing (BOINC). Often, grid computing software makes use of "spare cycles", performing computations at times when a computer is idling.

Automatic parallelization

Automatic parallelization of a sequential program by a compiler is the "holy grail" of parallel computing. Despite decades of work by compiler researchers, automatic parallelization has had only limited success.

Mainstream parallel programming languages remain either explicitly parallel or (at best) partially implicit, in which a programmer gives the compiler directives for parallelization. A few fully implicit parallel programming languages exist—SISAL, Parallel Haskell, and (for FPGAs) Mitrion-C—but these are niche languages that are not widely used.

APPLICATIONS OF FLEXSIM DS

The applications of the Flexsim DS framework are many. The principle ones are listed below.

- Multi player login
- Multi designer login
- Scalable models via combination
- Multi processing
- Distributed visualization separating processing and visuals
- Virtual reality multi frustum displays
- Remote operation and execution via intranet and internet
- Classic agent models
- High fidelity agent models
- Interoperability

Multi player login

This is where many users are able to connect with a system and perform an interactive role in a common simulation affecting the simulation as it plays out over time

Multi designer login

This is similar to the multi player login in that many users can log in to a central world. The difference is that instead of playing a role within the simulation, the users affect the design of the model and can interactively experiment with and refine the model design.

Scalable models via combination

Scalable models can be achieved by linking models on individual stations together. In principle, the combined model can be any number of times the size of what a single station can represent by combining the models on each station into one large one, effectively performing the role of a single supercomputer.

Multi processing

By allocating non-overlapping computation tasks to component stations in a combined simulation, the computational load of the overall simulation can be parallelized.

Distributed visualization separating processing and visuals

Distributed visualization is the allocation of visualization computing tasks to the stations in a distributed simulation. The individual stations can handle the computation of visuals which are specific to them, such as part of the scene, a view of the scene from a different angle or location or a completely different informatic view of the simulation world.

Virtual reality multi frustum displays

In virtual reality multi frustum displays, a scene is visualized from different viewpoints simultaneously. Each view will be from a different angle in the users world. i.e. From straight ahead, from the left from the right, overhead etc. when combined together in a composite display, the views are displayed on juxtaposed physical displays according to the part of the overall scene the physical display represents. The term frustum refers to the geometric space which corresponds to the particular view, and requires separated computation from the other views.

Remote operation and execution via intranet and internet

This is running a model on a station other than the one on which the user is working. The model is visually represented on the user's station. The model run and parameters can also be controlled remotely from the users station but the model is run by a simulation engine on the remote computer.

Classic agent models

Classic agent based simulations can be run in Flexsim DS. These require model representation which contains a large number of objects interacting with each other. Flexsim DS is optimized to handle models with large numbers of interacting objects.

High fidelity agent models

A different type of agent model is one where agents' behavior is more sophisticated and designed to mimic complex behavior rules such as a human aircraft pilot. This could also be described as artificial intelligence. These agents may be put in the role of a real human taking the controls of a vehicle in a simulation.

Interoperability

Standardization of distributed simulation interfaces is important to ensure compatibility of distributed simulation models especially within large organizations. The United States' DOD HLA is one such interface. Flexsim DS is designed to interface with the HLA standard.

EXAMPLES OF DISTRIBUTED MODELS IN FLEXSIM DS

Multi User virtual environment

In this simulation, a central server coordinates a distributed simulation model which represents a virtual world into which users at local stations can join the simulation. The system coordinates the world across all the servers and client. Data integrity is maintained across all objects in the world via passing data messages.



Figure: Multi User virtual environment

Distributed computation (parallelized computation) Parallel processing is utilized significantly in this example. In the world there are vehicles and objects which possess physics based movement, and there is a physical landscape over which they travel. Collisions may occur between the objects and the landscape and other objects. The processing of calculations across the entire system is divided so that the calculation load is spread across the stations in the simulation. The division occurs as follows: each station represents a local simulation of a vehicle in the world and the local station performs the physics calculations needed for the movement of the vehicle. The simulation as a whole is coordinated by each of the stations sending it's vehicle position to the server. The server then receives all the vehicle positions from all the client stations, when it then performs the calculations necessary for determining collisions in the world as a whole. Upon determining the collisions and resultant states of all the objects in the environment, the server then sends the information back to the clients, at the same time informing the clients of the updated positions of all the objects in the world. Thereby, the distributed simulation spreads the calculation tasks across the computers in the network.

The vehicles at each station are controlled by humans. The control of the vehicles is by advanced user input devices such as flight controllers.

High fidelity agent environment (agent pilots) High fidelity agents play a role in this simulation. As well as the ability to

place a human player into the environment with manual real time control of the vehicle, an agent program can be used to control the vehicle and perform its given objectives in the environment. The agent does this by observing the environment information available to it and then adjusts its flight controls just as a human would. Then it reevaluates the consequences of its actions and adjust the controls again, and so on.



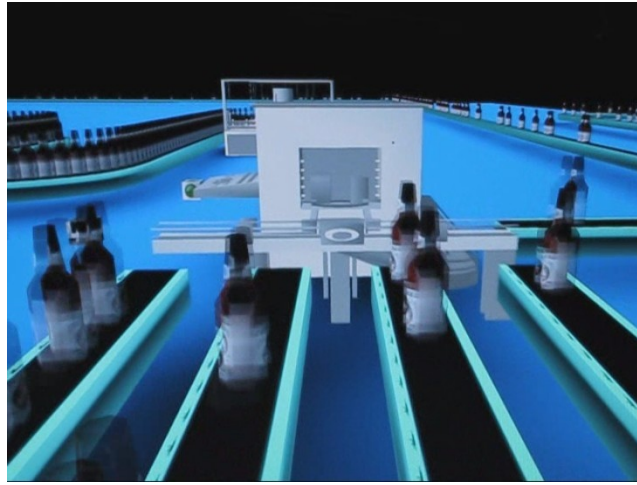
Figure: High Fidelity Agent

Distributed visualization (multi frustum displays; separation of visualization and computation; filtered visualization) Multi frustum displays can be used with this simulation. Any station can be used as a display controller, where the display presents the environment from a given viewpoint. A classic example of this is a wraparound vr theater display or a vr cave. The wraparound display may be typically a 270 degree field of view composed of three displays; a vr cave is six displays one for each face of a cubic space in which the human viewer stands thereby providing full immersion. In this example, such a wraparound display is achieved by using three client stations: one for left, center, right where the display angles set at are $-x;0;x$ degrees, where x may be typically 60 degrees to 90 degrees. A vr cave would use six stations. Any number of stations can be used and any flexible configuration is possible.

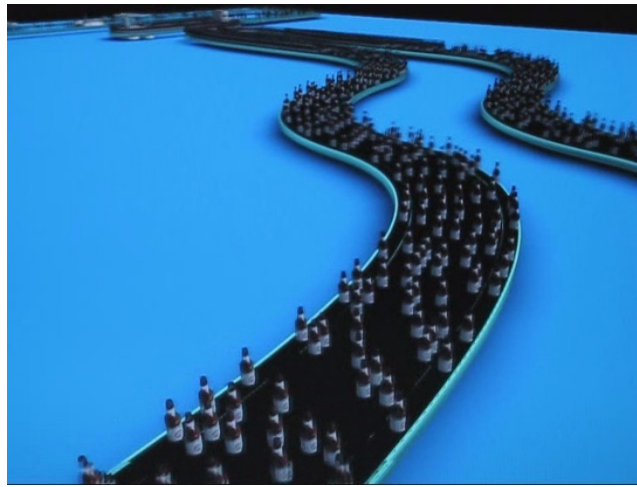
Distributed virtual environment (linked models)

In this example, there are several stations linked together. In the example three stations are used. Each client station logs into a server, which may share a station with a client and run on an available cpu. The model is of a manufacturing, distribution and warehousing operation.

Large scale agent environments The models are represented in a high fidelity virtual world which covers 36 square miles, contains over sixty miles of roads and contains tens of thousand of objects including vehicles, highway, intersections, manufacturing equipment, products, automated transportation machinery, trees, buildings and landscape topography. The objects and environment are modeled in high detail - right down to millimeter accuracy.



(a)



(b)

Figure: Virtual Reality Bottling Plant

The model is split as follows: (i) the topographical region contains a busy freeway system with main and local roads. A road shipping operation utilizing semi articulated trucks are employed to transport product from a factory to a warehouse. The factory and the warehouse are in this section modeled as black boxes which produce and consume goods respectively. (ii) the factory, a bottling plant to be exact, is itself a detailed model of the production operation. Thousands of items: bottles being prepared, filled, tested and packaged through detailed processing equipment are modeled in this part of the simulation. (iii) the warehouse receives and processes thousands of products for storage and dispatch through a large complex conveyor system.

The whole model is composed of the three sub models, each of which runs on a separate station. The sub models are combined at join points. The join points are the areas which production and consumption occurs across boundaries. There areas exist where: goods leave the factory and where goods enter the warehouse.



Figure: Virtual Reality Freeway

The linked model operates as one when running the simulation. The user interfaces of the sub models on the stations are synchronized, for example when the model run speed is changed, it changes on all the sub models, when it is reset on one of the stations, it is reset on all the stations. The local stations are visualized locally. i.e. The station which runs the factory part of the model is the one which displays the virtual reality view of the model.

This example also possesses the ability to have a multi frustum display. Another demonstration of distributed visualization in this example is that of filtered displays. One computer may have a view of the model with given view parameters and another station may have another view of the same model with different view parameters. This may be a high level view with visual orientation for viewing the system complexity all at once with high contrast coloration and low detail; and another view which shows detail at a local level with photo realistic high fidelity visuals and high detail of activity.

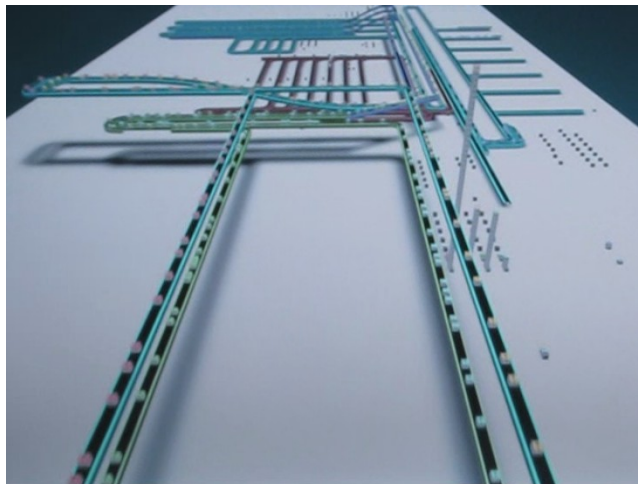


Figure: Virtual Reality Warehouse

Remote operation (separation of user interface and computation)

Flexsim has an open scalable network architecture, in the same philosophy as classical 3-tier or n-tier distributed computing and Internet architecture. This allows splitting of not just simulation sub models, collaborative VR and distributed simulation but also the splitting of user interface and computation centers. Thus this flexibility allows thin or thick clients, tight or loose coupled clients such that user interfaces can be used to distributed simulations in a wide range of ways. This includes remote model operation where the model may be run on a remote server. Parameterization and setup can be controlled remotely; and the visualization of the model and result from the model can be viewed concurrently on the client.

Immersive virtual reality (advanced user input devices)

Flexsim DS has ability to incorporate support for advanced user input devices such as flight controllers, 3d trackballs, 3d tracking devices other devices of that type. The high fidelity model representation within Flexsim DS has the power to then link 3d motion and position from devices to objects of any degree of complexity on the virtual world, be it an avatar, vehicle or anything required.

Interoperability (HLA)

HLA is a standard for simulation model interoperability used by the United States DOD. Flexsim DS has the ability to interface with this standard and operate as a federate in a HLA distributed simulation. However, Flexsim DS is not bounded by the capabilities of HLA. Flexsim DS has a fully abstracted tcpip interface for interoperability between stations on a tcpip network. It also has advanced support for structured data transfer which immensely simplifies linking simulations. The system is entirely flexible and can be used for applications far beyond the capability of HLA. Such as some of those mentioned, particularly remote user interfaces, multi frustum displays, and distributed visualization using standard pc's,

CONCLUSION

Overall, distributed virtual reality simulation provides a whole new world of possibilities which will cultivate applications even beyond that which we now envisage. Presently distributed simulation provides many benefits. It promotes team collaboration allowing teams to work together locally or remotely. Allows models to be deployed and run across the Internet; takes simulation to the next level in terms of power and model detail while requiring only standard off the shelf computing hardware; and finally, distributed simulation reflects today's technology. While every generation of software technology reflects the times in which it lives, the time has come for the inevitable arrival of distributed simulation.

REFERENCES

- 1.1 Pečiva, J. 2007. Active Transactions in Collaborative Virtual Environments. PhD Thesis, Brno, Czech Republic, FIT VUT, ISBN 978-80-214-3549-0
- 1.2. MacIntyre, B. and Feiner, S. 1998. A distributed 3D graphics library, Proc. of ACM SIGGRAPH '98, Jul 1998, New York, NY, 361-370, <http://www.cc.gatech.edu/~blair/papers/siggraph98.pdf>, DOI=<http://doi.acm.org/10.1145/280814.280935>
- 1.3. Sung, U., Yang, J., and Wohn, K. 1999. Concurrency Control in CIAO. In Proceedings of the IEEE Virtual Reality (March 13 - 17, 1999). VR. IEEE Computer Society, Washington, DC, 22
- 2.1. Erik Davis, Techgnosis: myth, magic and mysticism in the information age, 1998
- 2.2. Garb, Yaakov (Winter 1987). "Virtual reality". Whole Earth Review (57): 118ff.

- 2.3. Jaron Lanier's Bio
- 2.4. Boyd, J. (2008). "NTT Becomes a Smell-o-Phone Company". IEEE Spectrum. 45 (1): 8.
- 2.5. Times Online
- 2.6. The Future of Virtual Reality with Mychilo Cline » Introduction to the Future of Virtual Reality
- 2.7. Example of anatomy instruction
- 2.8. Virtual surgery example
- 2.9. Rapid Marine Prototype -[Marine Prototyping, Yacht Prototyping, Marine Design, Boat Modeling, Design, and Engineering]- » Case Study
- 2.10. CEI : News
- 2.11. Werkzeug- und Formenbau - Motion Control Systems - Siemens
- 2.12. Dynatek delta | Silicone Medical Device Testing
- 2.13. Special Feature: Emerging Technologies | Medical Product Manufacturing News
- 2.14. "Nano": The new nemesis of cancer Hede S, Huilgol N, - J Can Res Ther
- 2.15. IngentaConnect Nanotechnology: Intelligent Design to Treat Complex Disease
- 2.16. Over the Horizon: Potential Impact of Emerging Trends in information and Communication Technology on Disability Policy and Practice
- 2.17. Future Medicine - Nanomedicine - 1(1):67 - Summary
- 2.18 Brooks Jr., F. P. (1999). "What's Real About Virtual Reality?", IEEE Computer Graphics And Applications, 19(6), 16
- 2.19 Burdea, G. and P. Coffet (2003). Virtual Reality Technology, Second Edition. Wiley-IEEE Press.
- 2.20 Hillis, Ken (1999). Digital Sensations: Space, Identity and Embodiment in Virtual Reality. University of Minnesota Press, Minneapolis, MN.
- 2.21 Kalawsky, R. S. (1993). The Science of Virtual Reality and Virtual Environments: A Technical, Scientific and Engineering Reference on Virtual Environments, Addison-Wesley, Wokingham, England ; Reading, Mass.
- 2.22 Kelly, K., A. Heilbrun and B. Stacks (1989). "Virtual Reality; an Interview with Jaron Lanier", Whole Earth Review, Fall 1989, no. 64, pp. 108(12)
- 2.23 Krueger, M. W. (1991). Artificial Reality II, Addison-Wesley, Reading, Massachusetts
- 2.24 Lanier, J., and F. Biocca (1992). "An Insider's View of the Future of Virtual Reality." Journal of Communication, 42(4), 150
- 2.25 Packer, Randall, and Ken Jordan (eds). 2002. Multimedia: From Wagner to Virtual Reality, Expanded Edition. W.W. Norton.
- 2.26 Packer, Randall, and Ken Jordan (eds). 2000. Virtual Art Museum - Multimedia: From Wagner to Virtual Reality
- 2.27 Rheingold, H. (1992). Virtual Reality, Simon & Schuster, New York, N.Y.
- 2.28 Robinett, W. (1994). "Interactivity and Individual Viewpoint in Shared Virtual Worlds: The Big Screen vs. Networked Personal Displays." Computer Graphics, 28(2), 127
- 2.30 Slater, M., Usoh, M.(1993). "The Influence of a Virtual Body on Presence in Immersive Virtual Environments" Virtual Reality International 93, Proceedings of the Third Annual Conference on Virtual Reality, London, April 1993, pages 34--42. Meckler, 1993
- 2.31 Stanney, K. M. ed. (2002). Handbook of Virtual Environments: Design, Implementation, and Applications. Lawrence Erlbaum Associates, Inc., Mahwah, New Jersey
- 2.32 Sutherland, I. E. (1965). "The Ultimate Display". Proceedings of IFIP 65, vol 2, pp. 506-508
- 2.33 Warwick, K., Gray, J. and Roberts, D. eds. (1993). Virtual Reality in Engineering, Peter Peregrinus.
- 2.34 Goslin, M, and Morie, J. F., (1996). "Virtopia" Emotional experiences in Virtual Environments", Leonardo, vol. 29, no. 2, pp. 95-100.
- 2.35 Robles-De-La-Torre G. The Importance of the Sense of Touch in Virtual and Real Environments. IEEE Multimedia 13(3), Special issue on Haptic User Interfaces for Multimedia Systems, pp. 24-30 (2006).
- 2.36 Hayward V, Astley OR, Cruz-Hernandez M, Grant D, Robles-De-La-Torre G. Haptic interfaces and devices. Sensor Review 24(1), pp. 16-29 (2004).

- 2.37 Monkman. G.J. - An Electrorheological Tactile Display - Presence (Journal of Teleoperators and Virtual Environments) - Vol. 1, issue 2, pp. 219-228, MIT Press, July 1992.
- 2.38 Monkman. G.J. - 3D Tactile Image Display - Sensor Review - Vol 13, issue 2, pp. 27-31, MCB University Press, April 1993.
- 2.39 Klein. D, D. Rensink, H. Freimuth, G.J. Monkman, S. Egersdörfer, H. Böse, & M. Baumann - Modelling the Response of a Tactile Array using an Electrorheological Fluids - Journal of Physics D: Applied Physics, vol 37, no. 5, pp794-803, 2004
- 2.40 Klein. D, H. Freimuth, G.J. Monkman, S. Egersdörfer, A. Meier, H. Böse M. Baumann, H. Ermert & O.T. Bruhns - Electrorheological Tactile Elements. Mechatronics - Vol 15, No 7, pp883-897 - Pergamon, September 2005.
- 3.1 Adams, Ernest (July 9, 2004). "Postmodernism and the Three Types of Immersion" (HTML). Gamasutra. Retrieved on 2007-12-26.
- 3.2 Björk, Staffan; Jussi Holopainen (2004). Patterns In Game Design. Charles River Media, 423. ISBN 1584503548.
- 3.3 Immersive Ideals / Critical Distances : A Study of the Affinity Between Artistic Ideologies Based in Virtual Reality and Previous Immersive Idioms by Joseph Nechvatal 1999 Planetary Collegium
- 3.4 Oliver Grau, Virtual Art: From Illusion to Immersion, MIT-Press, Cambridge 2003
- 3.5 Oliver Grau (Ed.): Media Art Histories, MIT-Press, Cambridge 2007
- 3.6 Joseph Nechvatal, Immersive Excess in the Apse of Lascaux, Technoetic Arts 3, no3. 2005
- HLA.1. U.S. Defense Modeling and Simulation Office (2001). RTI 1.3-Next Generation Programmer's Guide Version 4. U.S. Department of Defense.
- 4.1. Strogatz, Steven (2007), "The End of Insight", in Brockman, John, What is your dangerous idea?, HarperCollins
- 4.2. "RESEARCHERS STAGE LARGEST MILITARY SIMULATION EVER" (news), Jet Propulsion Laboratory, Cal Tech, December 1997, webpage: JPL.
- 4.3. "Largest computational biology simulation mimics life's most essential nanomachine" (news), News Release, Nancy Ambrosiano, Los Alamos National Laboratory, Los Alamos, NM, October 2005, webpage: LANL-Fuse-story7428.
- 4.4. "Mission to build a simulated brain begins" (news), project of Institute at the École Polytechnique Fédérale de Lausanne (EPFL), Switzerland, NewScientist, June 2005, webpage: NewSci7470.
- 4.5. Baase, Sara. A Gift of Fire: Social, Legal, and Ethical Issues for Computing and the Internet. 3. Upper Saddle River: Prentice Hall, 2007. Pages 363-364. ISBN 0-13-600848-8.
- 4.6 R. Frigg and S. Hartmann, Models in Science. Entry in the Stanford Encyclopedia of Philosophy.
- 4.7 S. Hartmann, The World as a Process: Simulations in the Natural and Social Sciences, in: R. Hegselmann et al. (eds.), Modelling and Simulation in the Social Sciences from the Philosophy of Science Point of View, Theory and Decision Library. Dordrecht: Kluwer 1996, 77-100.
- 4.8 P. Humphreys, Extending Ourselves: Computational Science, Empiricism, and Scientific Method. Oxford: Oxford University Press, 2004.
- 5.1. Stewart Robinson (2004). Simulation - The practice of model development and use. Wiley.
- 5.2. Douglas W. Jones, ed. Implementations of Time, Proceedings of the 18th Winter Simulation Conference, 1986.
- 5.3. Douglas W. Jones, Empirical Comparison of Priority Queue and Event Set Implementations, Communications of the ACM, 29, April 1986, pages 300-311.
- 5.4. Kah Leong Tan and Li-Jin Thng, SNOOPY Calendar Queue, Proceedings of the 32nd Winter Simulation Conference, 2000
- 5.5 Michael Pidd (1998). Computer simulation in management science - fourth edition. Wiley.
- 5.6 Jerry Banks, John Carson, Barry Nelson and David Nicol (2005). Discrete-event system simulation - fourth edition. Pearson.
- 5.7 Averill M. Law and W. David Kelton (2000). Simulation modeling and analysis - third edition. McGraw-Hill.
- 5.8 Bernard P. Zeigler, Herbert Praehofer and Tag Gon Kim (2000). Theory of modeling and simulation: Integrating discrete event and continuous complex dynamic systems - second edition. Academic Press.
- 5.9 Roger W. McHaney (1991). Computer Simulation: A Practical Perspective. Academic Press.

- 5.10 William Delaney, Erminia Vaccari (1988). *Dynamic Models and Discrete Event Simulation*. Dekker INC.
- 6.1. Leslie Lamport. "Subject: distribution (Email message sent to a DEC SRC bulletin board at 12:23:29 PDT on 28 May 87)". Retrieved on 2007-04-28.
- 6.2. A database-centric virtual chemistry system, *J Chem Inf Model*. 2006 May-Jun;46(3):1034-9
- 6.3. CS236370 *Concurrent and Distributed Programming 2002*
- 6.4. David P. Anderson (2005-05-23). "A Million Years of Computing". Retrieved on 2006-08-11.
- 6.5. Attiya, Hagit and Welch, Jennifer (2004). *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. Wiley-Interscience. ISBN 0471453242.
- 6.6. Lynch, Nancy A (1997). *Distributed Algorithms*. Morgan Kaufmann. ISBN 1558603484.
- 6.7. Tel, Gerard (1994). *Introduction to Distributed Algorithms*. Cambridge University Press.
- 6.8. Davies, Antony (June 2004). "Computational Intermediation and the Evolution of Computation as a Commodity" ([dead link] – Scholar search). *Applied Economics*. doi:10.1080/0003684042000247334.
- 6.9. Kornfeld, William (January 1981). "The Scientific Community Metaphor". MIT AI (Memo 641).
- 6.10. Hewitt, Carl (August 1983). "Analyzing the Roles of Descriptions and Actions in Open Systems". *Proceedings of the National Conference on Artificial Intelligence*.
- 6.11. Hewitt, Carl (April 1985). "The Challenge of Open Systems". *Byte Magazine*.
- 6.12. Hewitt, Carl (1999-10-23–1999-10-27). "Towards Open Information Systems Semantics". *Proceedings of 10th International Workshop on Distributed Artificial Intelligence*.
- 6.13. Hewitt, Carl (January 1991). "Open Information Systems Semantics". *Journal of Artificial Intelligence* 47: 79. doi:10.1016/0004-3702(91)90051-K.
- 6.14. Nadiminti, Dias de Assunção, Buyya (September 2006). "Distributed Systems and Recent Innovations: Challenges and Benefits". *InfoNet Magazine*, Volume 16, Issue 3, Melbourne, Australia.
- 6.15. Bell, Michael (2008). "Service-Oriented Modeling: Service Analysis, Design, and Architecture". Wiley.
- 7.1. Almasi, G.S. and A. Gottlieb (1989). *Highly Parallel Computing*. Benjamin-Cummings publishers, Redwood City, CA.
- 7.2. Asanovic, Krste et al. (December 18, 2006). "The Landscape of Parallel Computing Research: A View from Berkeley" (PDF). University of California, Berkeley. Technical Report No. UCB/EECS-2006-183. "Old [conventional wisdom]: Increasing clock frequency is the primary method of improving processor performance. New [conventional wisdom]: Increasing parallelism is the primary method of improving processor performance ... Even representatives from Intel, a company generally associated with the 'higher clock-speed is better' position, warned that traditional approaches to maximizing performance through maximizing clock speed have been pushed to their limit."
- 7.3. Asanovic et al. Old [conventional wisdom]: Power is free, but transistors are expensive. New [conventional wisdom] is [that] power is expensive, but transistors are "free".
- 7.4. Patterson, David A. and John L. Hennessy (1998). *Computer Organization and Design, Second Edition*, Morgan Kaufmann Publishers, p. 715. ISBN 1558604286.
- 7.5. a b Barney, Blaise. "Introduction to Parallel Computing". Lawrence Livermore National Laboratory. Retrieved on 2007-11-09.
- 7.6. Hennessy, John L. and David A. Patterson (2002). *Computer Architecture: A Quantitative Approach*. 3rd edition, Morgan Kaufmann, p. 43. ISBN 1558607242.
- 7.7. Rabaey, J. M. (1996). *Digital Integrated Circuits*. Prentice Hall, p. 235. ISBN 0131786091.
- 7.8. Flynn, Laurie J. "Intel Halts Development of 2 New Microprocessors". *The New York Times*, May 8, 2004. Retrieved on April 22, 2008.
- 7.9. Moore, Gordon E. (1965). "Cramming more components onto integrated circuits" (PDF). *Electronics Magazine* 4. Retrieved on 2006-11-11.
- 7.10. Amdahl, G. (April 1967) "The validity of the single processor approach to achieving large-scale computing capabilities". In *Proceedings of AFIPS Spring Joint Computer Conference*, Atlantic City, N.J., AFIPS Press, pp. 483–85.
- 7.11. Brooks, Frederick P. Jr. *The Mythical Man-Month: Essays on Software Engineering*. Chapter 2 – The Mythical Man Month. ISBN 0201835959

- 7.12. Reevaluating Amdahl's Law (1988). *Communications of the ACM* 31(5), pp. 532–33.
- 7.13. Bernstein, A. J. (October 1966). "Program Analysis for Parallel Processing," *IEEE Trans. on Electronic Computers*. EC-15, pp. 757–62.
- 7.14. Roosta, Seyed H. (2000). "Parallel processing and parallel algorithms: theory and computation". Springer, p. 114. ISBN 0387987169.
- 7.15. Lamport, Leslie (September 1979). "How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs", *IEEE Transactions on Computers*, C-28,9, pp. 690–91.
- 7.16. Patterson and Hennessy, p. 748.
- 7.17. Culler, David E.; Jaswinder Pal Singh and Anoop Gupta (1999). *Parallel Computer Architecture - A Hardware/Software Approach*. Morgan Kaufmann Publishers, p. 15. ISBN 1558603433.
- 7.18. Culler et al. p. 15.
- 7.19. Patt, Yale (April 2004). "The Microprocessor Ten Years From Now: What Are The Challenges, How Do We Meet Them? (wmv). Distinguished Lecturer talk at Carnegie Mellon University. Retrieved on November 7, 2007.
- 7.20. a b Culler et al. p. 124.
- 7.21. a b Culler et al. p. 125.
- 7.22. a b Patterson and Hennessy, p. 713.
- 7.23. a b Hennessy and Patterson, p. 549.
- 7.24. Patterson and Hennessy, p. 714.
- 7.25. What is clustering? *Webopedia computer dictionary*. Retrieved on November 7, 2007.
- 7.26. Beowulf definition. *PC Magazine*. Retrieved on November 7, 2007.
- 7.27. Architecture share for 06/2007. *TOP500 Supercomputing Sites*. Clusters make up 74.60% of the machines on the list. Retrieved on November 7, 2007.
- 7.28. Hennessy and Patterson, p. 537.
- 7.29. MPP Definition. *PC Magazine*. Retrieved on November 7, 2007.
- 7.30. Kirkpatrick, Scott (January 31, 2003). "Computer Science: Rough Times Ahead". *Science*, Vol. 299. No. 5607, pp. 668 - 669. DOI: 10.1126/science.1081623
- 7.31. a b c D'Amour, Michael R., CEO DRC Computer Corporation. "Standard Reconfigurable Computing". Invited speaker at the University of Delaware, February 28, 2007.
- 7.32. Boggan, Sha'Kia and Daniel M. Pressel (August 2007). *GPUs: An Emerging Platform for General-Purpose Computation (PDF)*. ARL-SR-154, U.S. Army Research Lab. Retrieved on November 7, 2007.
- 7.33. Maslennikov, Oleg (2002). "Systematic Generation of Executing Programs for Processor Elements in Parallel ASIC or FPGA-Based Systems and Their Transformation into VHDL-Descriptions of Processor Element Control Units". *Lecture Notes in Computer Science*, 2328/2002: p. 272.
- 7.34. Shimokawa, Y.; Y. Fuwa and N. Aramaki (18–21 November 1991). A parallel ASIC VLSI neurocomputer for a large number of neurons and billion connections per second speed. *IEEE International Joint Conference on Neural Networks*. 3: pp. 2162–67.
- 7.35. Acken, K.P.; M.J. Irwin, R.M. Owens (July 1998). "A Parallel ASIC Architecture for Efficient Fractal Image Coding". *The Journal of VLSI Signal Processing*, 19(2):97–113(17)
- 7.36. Kahng, Andrew B. (June 21, 2004) "Scoping the Problem of DFM in the Semiconductor Industry." University of California, San Diego. "Future design for manufacturing (DFM) technology must reduce design [non-recoverable expenditure] cost and directly address manufacturing [non-recoverable expenditures] – the cost of a mask set and probe card – which is well over \$1 million at the 90 nm technology node and creates a significant damper on semiconductor-based innovation."
- 7.37. a b Patterson and Hennessy, p. 751.
- 7.38. Shen, John Paul and Mikko H. Lipasti (2005). *Modern Processor Design: Fundamentals of Superscalar Processors*. McGraw-Hill Professional. p. 561. ISBN 0070570647. "However, the holy grail of such research - automated parallelization of serial programs - has yet to materialize. While automated parallelization of certain classes of algorithms has been

demonstrated, such success has largely been limited to scientific and numeric applications with predictable flow control (e.g., nested loop structures with statically determined iteration counts) and statically analyzable memory access patterns. (e.g., walks over large multidimensional arrays of float-point data)."

7.39. Asanovic, Krste, et al. (December 18, 2006). The Landscape of Parallel Computing Research: A View from Berkeley (PDF). University of California, Berkeley. Technical Report No. UCB/EECS-2006-183. See table on pages 17-19.

7.40. Menabrea, L. F. (1842). Sketch of the Analytic Engine Invented by Charles Babbage. Bibliothèque Universelle de Genève. Retrieved on November 7, 2007.

7.41. a b c Patterson and Hennessy, p. 753.

7.42. da Cruz, Frank (2003). "Columbia University Computing History: The IBM 704". Columbia University. Retrieved on 2008-01-08.

7.43. a b c d e Wilson, Gregory V (1994). "The History of the Development of Parallel Computing". Virginia Tech/Norfolk State University, Interactive Learning with a Digital Library in Computer Science. Retrieved on 2008-01-08.

7.44. Anthes, Gry (2001-11-19). "The Power of Parallelism". Computerworld. Retrieved on 2008-01-08.

7.45. Patterson and Hennessy, p. 749.

7.46. Patterson and Hennessy, pp. 749–50: "Although successful in pushing several technologies useful in later projects, the ILLIAC IV failed as a computer. Costs escalated from the \$8 million estimated in 1966 to \$31 million by 1972, despite the construction of only a quarter of the planned machine ... It was perhaps the most infamous of supercomputers. The project started in 1965 and ran its first real application in 1976."