

A. Martin Wildberger
Electric Power Research Institute

Parallel and Distributed Simulation, the topic of this special issue, has many aspects to which artificial intelligence can be usefully applied. Several past columns have discussed agent-based modeling and simulation, in which agents are relatively autonomous and are designed to be capable of independent, intelligent, adaptive action. When a real-world situation can be modeled by agents, whether they are adaptive or unchangeable, the simulation based on that model is inherently parallel and distributed. The agents emulate as closely as possible the real world of humans and machines, operating simultaneously in different locations with limited information about each other. Agent-based models bypass the problems of dependency, latency and backtracking. These must be handled explicitly by programming when an inherently sequential model is converted to run in parallel on distributed processors. Unfortunately, it is often not convenient, and may not even be possible, to produce an agent-based model of the particular problem to be simulated. However, if the problem is very large, so that parallel computing is essential, then it is always worthwhile to expend some effort in an attempt to rethink the problem in terms of independent, parallel actions that can be conveniently distributed to multiple processors that are not necessarily collocated. Elsewhere in this issue, Edwin Naroska and Uwe Schwiiegelshohn describe some ways to exploit the natural parallelism within a model in terms of conservative parallel discrete-event simulation algorithms.

Computer simulation, since its very beginning, has been divided into two, sometimes antagonistic, segments: discrete event and continuous process. In part, this distinction had its roots in the differences between the analog and digital computers that were available to perform the early simulations, but there are many natural phenomena and man-made systems for which one or the other approach is particularly apt. As digital computers increased in power and availability, even the naturally continuous simulations came to be performed on them, and this required discretization. Today, the simulation of almost any continuous process requires the numerical solution of partial differential equations (PDE) by a finite step algorithm that operates on a generalized spatial and temporal grid. The generation of grids or meshes for this purpose is one of the greatest consumers of computer time.

Every simulation that attempts to solve a continuous system on a discrete grid is faced with a trade-off

between using a very fine mesh to ensure accurate results and using a larger mesh to reduce computing costs. Adaptive methods of mesh generation allow this trade-off to be made at every step. In order to guarantee that required accuracy is being achieved, mesh generation must be guided by computing estimates of discretization errors in the solution. One method, Adaptive Mesh Refinement (AMR), developed at Los Alamos National Laboratory in New Mexico, USA, was described in the June 1994 column. The system being simulated need not be particularly large in physical extent for adaptive grid generation to be useful. Simulation of large electric power grids and very small composite materials can both benefit from it. Automatic, adaptive iteration of mesh generation and solution error estimation in a parallel-computing environment is a particularly difficult challenge and an area in which machine learning has had only marginal success. Many techniques that work well for serial processing require too much communication overhead to remain synchronized on multiple parallel processors.

Multigrid methods attempt to bypass this problem by generating grids at many different scales arranged hierarchically. At the heart of multigrid is separate computation at each scale of the problem along with the appropriate handling of inter-scale interactions for the effective combining of micro-system and macro-system behavior analysis. Problems of this kind are characterized by being composed of a very large number, n , of individuals (for instance: particles), all of whose interactions, $O(n^2)$, are routinely calculated at each time step. One multigrid approach calculates the local, fine-grain behavior (say, inter-particle forces) directly, and smoothes those calculations by interpolating from the total forces calculated at a coarser scale. If that scale is twice the average distance between particles, the cost of calculation can be reduced to $O(n)$ per step. However, extremely small time steps are still required to avoid interactions during the time interval because of the very short distance between particles. If a hierarchy of grids are defined with, for instance, each one twice the size of the lower one, it is possible to move all the scales on a multigrid cycle, sweeping the field one or twice at each level from the coarsest to the finest, interpolating at each level down to the individual particles wherever necessary. This process can be shown to converge very quickly. The error after each complete cycle is about one-tenth the

error before. Additional gains can sometimes be achieved from hierarchical scaling by using different algorithms at the different scales. For instance, particles at the fine grain, but continuum mechanics at the coarse, or wave equations at the fine grain and "rays" at the coarse.

Multigrid techniques using adaptive numerical algorithms are not restricted to the solution of boundary value and fluid flow problems posed on spatial domains. The same methods can be applied to a broad spectrum of problems that have no "natural" association with any kind of physical grid. For instance, an industrial plant or an electronic system can be modeled hierarchically from coarser to finer grain as system, sub-system, device, component, circuit, etc. Methods ranging from Bond Graphs to Cellular Automata (CA) have abstracted from the concept of a multi-layered mesh in order to model problems where the grids are replaced by more general levels of organization. CA are spatially extended systems in which values, assigned to an array of sites, change synchronously in discrete steps over time by the application of relatively simple, local rules. CA were discussed in the February 1994 and January 1996 columns. The architecture of CA is ideal for a large class of parallel computations. However, it is extremely difficult to "program" a CA by selecting just the right combination of initial configuration and behavior rules. Lattice gases (LGA) constitute a distinct subclass of CA that can simulate complex fluid dynamics governed by non-linear equations such as the Navier-Stokes equation. There have been some significant recent developments in programming languages and parallel computer architectures that strongly support CA and especially LGA.

One way to help ease the problem of registration between solutions performed at different scales on different sized grids is to express the data in terms of a vector basis with local support. Wavelets (described in the August 1992 column) form one such set of basis functions, and the wavelet transform (WT) is an inner product that measures the similarity between the data being transformed and the particular set of bases (wavelets) representing different scale (frequency) components. Each wavelet can then be manipulated with a resolution matched to its scale. Since all the wavelets are derived directly as different scaled versions of the same "mother wavelet," registration among the different scales is always maintained. There are still many open questions as to the special properties of wavelet bases, the significance of these for specific applications, and whether a generic family of bases can be defined in a way that would include the radial basis functions, the Moebius transform and the Walsh transform, as

well as other bases that have similar characteristics.

Regardless of the particular algorithms used in a distributed simulation, individual routines must be deployed to the various processors in an optimal manner and must communicate with each other. Computing power that would be available otherwise can be lost to decomposition, repositioning, messaging, and the necessity of processors to wait for results generated by others. The most general use for AI in parallel and distributed simulations is to manage and minimize the delays and backtracking needed to maintain the logical and temporal dependencies among the separate processes. One way for the simulation designer to approach this would be to think how a human computer operator might manage the distributed system if simulated time were moving slowly enough for human intervention in the transmission of each message. If a simulation is intended to be run repeatedly with similar data, then a set of heuristics can often be derived that can substitute effectively for the human operator. These heuristics would attempt to strike an optimal balance between a "pessimistic" policy, with individual processors waiting until all messages that relate to their instructions have been received, and an "optimistic" policy, with assumption-based computing beginning as soon as partial data arrives, but with provisions for time-consuming back-out and revision should later messages conflict with key assumptions. So far, there has been little, if any, success in using machine learning and adaptation to improve time management over one or more runs of the simulation. Very small, and apparently random, fluctuations in processing or transmission time can have large effects on the outcome of the simulation. This seems to require either very fine granularity in the decision rules or else a very conservative and inefficient management policy. Adaptive granularity using rough sets may be useful, but there is little that can be learned in real-time from a stream of non-stationary random data.

To look further:

Articles in this issue address some of the ideas mentioned in this column. One describes synchronization methods for integrating a distributed simulation; another surveys existing languages and runtime libraries specifically meant for parallel discrete-event simulation.

The following books are excellent introductions to three of the topics mentioned in this column:

Briggs, William L. *A Multigrid Tutorial*, SIAM, Philadelphia, PA, 1987.

Strang, G., Nguyen, T. *Wavelets and Filter Banks*, Wellesley-Cambridge Press, 1996.

Toffoli, T., Margolus, N. *Cellular Automata Machines*, MIT Press, Cambridge, MA, 1987. ■