OpenMP parallelization of agent-based models

Massimo Bernaschi, Filippo Castiglione
Istituto Applicazioni del Calcolo (IAC) "M. Picone"
Consiglio Nazionale delle Ricerche (CNR)
Viale del Policlinico, 137 – 00161 Roma, Italy
[massimo,filippo]@iac.cnr.it

Federico Massaioli
CASPUR
V. dei Tizii 6/b – 00185 Roma, Italy
federico.massaioli@caspur.it

June 20, 2003

1 Agent based models

Nowadays, high performance computing techniques are widely applied to applications like fluid-dynamics, materials science or meteorology. In these fields there are well established mathematical models and the procedure of the simulation is straightforward: (i) the equations of the model are discretized; (ii) the discrete version of the equations is employed in the computer simulation; (iii) the numerical stability of the results is checked. The computational features of the corresponding codes are also well known and much work has been devoted to the enhancement of their performances.

In other important fields the situation is not so good. In biology or finance the rules governing the micro-behaviour of entities such as cells, molecules or traders and brokerage agencies are mostly unknown. Only the behaviour at macro level is available. This can be studied either by means of specific experiments (for biology problems) or by applying statistical methods to the available data (this is true specially in finance where the ubiquitous use of databases allows to track any transaction worldwide).
In both cases there are complex systems composed of many heterogeneous elements which interact with each other. A possible approach for studying such systems is represented by agent-based simulations. An agent is an entity which has: (i) an internal data representation (memory or state); (ii) means for modifying its internal data representation (perception); (iii) means for modifying its environment (behavior).

Agent-based models can be considered as an extension and generalization of *cellular automata*. The description of complex systems like the Immune System or the Stock Market requires some extensions and changes of the original concept of cellular automata.

- different entities having a distinct micro-state space must be represented;
- the number of entities on each lattice site must be variable in space and in time;
- a diffusion process must be present by which entities move from a site to another (at this time we assume that the entities follow a classic *Brownian* motion);
- the interactions among entities may be local (*i.e.*, they happen on the same lattice site)

We represent *entities* (cells, molecules, traders on the market *etc.*) as a collection of information or *attributes*. The heterogeneous information corresponding to a single particle is represented by integer numbers, so that the *numerical stability* of the simulation algorithm is unconditionally guaranteed.

Since the entities follow a complete life-cycle during the simulation (*i.e.* birth, evolution and death) it is more appropriate to employ flexible *dynamic* lists of "records" that contain all the information about each entity.

In our models the complex behaviour of the entities is subject to precise *state-changes* upon interaction. Every single entity can be thought as a *Stochastic Finite State Machine* which processes information and changes its state according to the results of the interactions with other entities, or with external fields.

Transition probabilities can be fixed or can change in time. In our case they depend on the outcome of the interaction rules governing the entities. They can also depend on some global quantities or external fields.

Finally, the entities diffuse to adjacent sites but the bulk of the computation, that is the interaction, happens locally (*i.e.,* inside each lattice site).

2 Application I: Immunology

The Immune System (IS) response has been modelled by means of a generalized Cellular Automaton having the following features: (1) the automaton is defined on a triangular 2-d lattice with periodic boundary conditions (toroidal geometry); (2) the dynamics is probabilistic; (3) the evolution of each site depends just on the site itself (*internal* dynamics); (4) entities move from site to site (*diffusion* process); (5) each time step corresponds to ~ 8 hours of "real life".

The whole automaton corresponds to a single lymphnode. The primary lymphoid organs thymus and bone marrow are modeled apart: the thymus is implicitly represented by the positive and negative selection of immature thymocytes before they get into the lymphatic system, whereas the bone marrow

generates already mature B lymphocytes. Hence, on the lattice we find only immunocompetent lymphocytes.

The entities on the lattice are the major classes of cells of the lymphoid lineage and some of the myeloid lineage.

The model belongs to the class of the *bit-string models*. The bit strings represent the "binding site" of cells and molecules as for example lymphocyte receptors (T lymphocytes receptor TCR, B lymphocytes receptor BCR), Major Histocompatibility Complexes MHC, antigen peptides and epitopes, immunocomplexes, *etc.*. Currenty, we are able to deal with strings of more than 24 bits that allow the representation of more than 10^7 different binding sites.

The entities interact with a probability which is a function of the *Hamming distance* between the bit strings representing the entities' binding site. This probability is called *affinity potential*. For two strings s and s' such probability is max (*i.e.*, equal to 1) when all corresponding bits are complementary ($0 \leftrightarrow 1$), that is, when the Hamming distance between s and s' is equal to the bit string length. A good and widely used analogy is the matching between a lock and its key.

The Immune System is dynamic in nature with a population that, due to the combination of multiple mechanisms like evolution, learning and memory, may change significantly during the simulation. After the attributes for each cellular entity have been chosen, the information is organized in blocks of variables, *one block for each cell*. The blocks form lists, one for each class of cells: B list, Th list and so on. These are *forward* or *single linked* lists. The lists are initialized at startup time and are managed dynamically at run time. When a cell dies out it is removed from the list. Likewise, when a virgin cell comes in, a block of memory is allocated and the variables-attributes are filled with appropriate values. The new block becomes the head of the corresponding list.

Molecules like the pathogen-agents (antigens) do not have an internal state. Moreover, at any time, there are relatively few distinct antigens but their number can be very high (order of magnitude larger than cells). Basically, each type of molecule is represented by a global and unique record. Both molecules and any cellular entity that binds molecules have a pointer to the corresponding record.

The allocated memory grows in a linear way with the *number* of entities following the clonal growth of cells during the immune response. The growth is limited by an embedded mechanism which emulates the effect of having limited space for cell-proliferation in real bodies.

In the real immune system each cell has thousands of receptors on its surface. So, albeit very unlikely, it is possible that it binds more than one entity at the same time. In our simulator this mechanism is not represented: each cell has a single receptor, and a successful binding event removes the tied entities from the eliciting set to prevent further interactions during the same time step. Such mechanism may introduce an artificial bias in the simulation. Indeed, if the scanning order during the interaction phase were simply the cells' lists order, the cells close to the head of the list would get more chances to interact. To alleviate such problem the order of the cells' lists is shuffled at each iteration.

3 Application II: Financial Markets

The second application is an agent-based model of a financial market. It incorporates many features of other models proposed in this field, and it is aimed at being the first step towards the construction of a simulator that employs more realistic market rules, strategies, and information from the real-world.

In the simulator there are three kinds of agents who trade (with different strategies) a set of N assets (stocks, commodities, etc.): *fundamentalists*, *noisy* and *technical traders*. *Fundamentalists* consider a reference (or "fundamental") value to determine the "right" price of an asset. *Noisy* traders do not follow any reference value and do not look at charts. Their behaviour is mostly random. There is a very high number of noisy traders but most of them have a very limited capital at their disposal. The last group (*technical traders* also called *chartists*) represent those agents who take into account information about the evolution of prices (in our case a moving average over certain time horizons. The initial wealth (stocks and money) of each trader is selected according to a power-law distribution.

At each time step agents decide whether to trade or not. Active agents follow different decision paths depending on their trading strategy and may take different positions with respect to each stock in the market.

The size of the agents' lists depend on the *granularity* of the representation. If only major players in the market are represented (*i.e.*, brokers), there will be relatively few agents. In case each single "trader-on-line" (*i.e.*, people trading from their own computer) must be represented there will be, potentially, millions of agents at the same time. Regardless of their size, the lists are dynamic since existing agents disappear (when they run out of money) and new agents enter the market during the simulation. The management of these lists follows the same guidelines described in the case of the Immune System simulator.

The major difference from the computational viewpoint with the Immune System simulator is in the interaction among the entities, which is regulated by a *book-of-orders* defined by lists of buy-orders and sell-orders. Two agents (a *seller* and a *buyer*) interact if their order prices match. Matching orders are immediately satisfied (filed) whereas the rest wait for the arrival of a matching order for a time defined when the order enters the *book*.

Market rules require buy-orders to be sorted in descending order (from the highest to the lowest order price) whereas sell-orders must be sorted in reverse order (from the lowest to the highest order price) for each stock. This means that every time an order enters the book it can not be simply inserted at the beginning of the corresponding list but it must be placed in the "right" position.

Since most of the orders have a limited life-time (in the real word, up to a few days) it is necessary to check if an order has expired, which means an additional periodic scanning of the lists.

Note that in a real market there are hundreds of different stocks and hundreds of millions of transactions every day. A *quasi*-realistic simulation requires a significant amount of computing time for the management of the book-of-orders (up to 30% of the simulation time).

4 Parallelization

Applications like those presented in the previous sections look very suitable to parallel processing since they are computing intensive and easy to split among a set of processors. Each processor can deal with a subset of the agents exchanging information with other processors when required. However, there are non-trivial issues that we describe in the following subsections.

The Immune System simulator has been developed with message passing–based parallel processing in mind from the very beginning. MPI was adopted for its rich set of Collective Communication Primitives (CCPs). The lattice is decomposed among parallel processes, and all data are local to the processes, without replications. The problem is not "embarrassingly parallel" for two features of the simulation. First of all, there is a *diffusion* phase in which cells and molecules may migrate from a lattice site towards one of its six nearest neighbors. Second, as we described in section 2, a specific antigen is represented by a global and unique record. Since in some cases new antigens may appear in a lattice site during the simulation (either by mutation or by direct injection), it is necessary to synchronize the information among PEs to avoid the existence of duplicated records. The approach to the parallelization of the Stock Market simulator was very similar to the previous case. Each task of a parallel run is in charge of a subset of the agents.

In both applications, during the diffusion, agents migrate from a task to another and the communication is point-to-point. To evaluate global quantities required by all tasks CCPs are employed.

However, the presence of the book-of-orders changes significantly the situation in the stock market simulator. For each stock there is a single "book". The submission of a new order entails an interaction which can be either local or remote depending on the location of the agent. There are, at least, three messages for any successful non-local order: (i) a first point-to-point message for the submission of the order; (ii) a second point-to-point message to communicate the result of the order (iii) a third broadcast message to communicate the new price for that stock. Load balancing can be an issue as well.

It is apparent how a parallelization based on an explicit message passing mechanism in a distributed memory environment becomes unbearable. The clear trend towards large shared memory systems led us to consider a different approach to parallelization, using pthreads or OpenMP. The OpenMP Application Programming Interface (API) supports multi-platform shared-memory parallel programming in C/C++ and Fortran. However OpenMP seems more suitable to "classic" codes where there are arrays and loops on indexes whereas in the simulators we have described so far there are loops on dynamic lists managed using pointers.

During OMPlab, we intend to parallelize a simplified version of the two applications, trying different OpenMP idioms and proposed extension, and leveraging on the availability of tools and the presence of compiler and runtime experts to cope with the possible problems in work decomposition and synchronized access to global information.