Project no.
035086
Project acronym
EURACE
Project title
An Agent-Based software platform for European economic policy design with heterogeneous interacting agents: new insights from a bottom up approach to economic modelling and simulation

Instrument: STREP

Thematic Priority: IST FET PROACTIVE INITIATIVE "SIMULATING EMERGENT PROPERTIES IN COMPLEX SYSTEMS

Deliverable reference number and title
D1.4: Porting of agent models to parallel computers
Due date of deliverable:
31/08/2008
Actual submission date:

Start date of project: September 1^{st} 2006 Duration: 36 months

Organisation name of lead contractor for this deliverable
STFC Rutherford Appleton Laboratory - STFC

Revision 1

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Contents

Abstract

Making use of high performance computers in agent-based simulation is a complex and difficult task. This report describes at the approach being used within the EURACE project to exploit parallel computing technology in large-scale agent-based simulations involving millions of agents. The underlying software is the FLAME framework and the paper will give an overview of features and use of FLAME and discuss the implementation of techniques that attempt to exploit large parallel computing systems. Some early simulation results will be presented together with a discussion of the requirements for efficient parallel simulations and the performance of the current implementation

1 Introduction

In this report we described the parallel implementation of the FLAME Framework [9]. We will cover some of the reason for parallelisation but also a detailed description of the approach being adopted. Some initial results from a set of benchmarks which hope to capture the computation and communications loads inherent in the EURACE application.

There are many agent-based modelling systems. A detailed survey of such programs, systems and frameworks in given by Mangina [4]. Many of these systems are based on Java as their implementation language. Although a good language for web-based and some communications applications it is not one often used in the area of high performance computing. Similarly there are relatively few agent systems that address the problem of scalable simulations.

Before considering the current parallelisation of FLAME it is worth considering the characteristics of a software system than make it worth taking the time and effort to parallelise. Some characteristics of when to parallelise:

- Code is practically incapable of running on one computer, memory requirements too great, run time too long
- Code will be reused frequently - parallelisation is a large investment
- Data structures are simple, calculations are local, easy to communicate and synchronize between processors

The converse should also be considered. When not to parallelise a code:

- Code will only be used once (or infrequently) - An efficient parallel code takes time to develop!
- Current performance is acceptable and execution time is short
- There will be frequent and significant code changes

It should also be remembered that some algorithms simply do not parallelise.

It is clear that developing a scalable agent-based framework will be difficult. As mentioned above there are few examples none of which attempt to utilise the power of high performance systems such the Cray XT4 or the IBM BlueGene or the multitude of Beowulf type systems being offered by vendors. Agent systems such as SAMAS [3],JADE [**?**], SIMJADE [**?**], MACE3J [**?**], JAMES [**?**] and SPADES [**?**] have been used to demonstrate scalable agent computing but these have been relatively small simulations.

The starting point of FLAME is different from these systems - high performance computing was thought to be essential and thus the implementation language is C.

2 General Parallel Implementation of FLAME

Because of its architecture FLAME does have some of the good characteristics but unfortunately it has a number of the bad. In the context of the EURACE project we believe that the good out-weigh the bad with the size and time to solution dominating the good. However in a fully connected and communicating agent population the communications may not be local but long range. However in many applications of FLAME, EURACE included, we have seen that there is sufficient locality to consider parallelisation given the general population sizes.

The use of simple read/write, single-type message boards allows the framework implementer to divide the agent population and their associated communications areas. This division could be based on any number of parameters or separators but the simplest to appreciate is position or

locality. If, as in EURACE, agents are people or companies for example, they will have locality defined either as location or by some group topology. It is also reasonable to assume that the dominant communications in both scenarios will be with neighbouring agents.

As explained above FLAME uses a collection of message boards to facilitate inter-agent communication. As the majority of large high performance computing systems currently use a distributed memory model a Single Program Multiple Data (SPMD) paradigm is considered most appropriate for the FLAME architecture. The parallelisation of FLAME utilises partitioned agent populations and distributed message boards linked through MPI communication. Figure 1 shows the difference between the serial and parallel implementation.

The most significant operation in the parallel implementation is providing the message information required by agents on one node of the processor array but stored on a remote node of the processor. The FLAME Message Board Library manages these data requests by using a set of predefined message filters to limit the message movement. This process could be considered a syncronisation of the local message boards within an iteration of the simulation. This syncronisation essentially ensures that local agents have the message information they need as the simulation progresses. The two main areas of algorithmic and technical development needed to

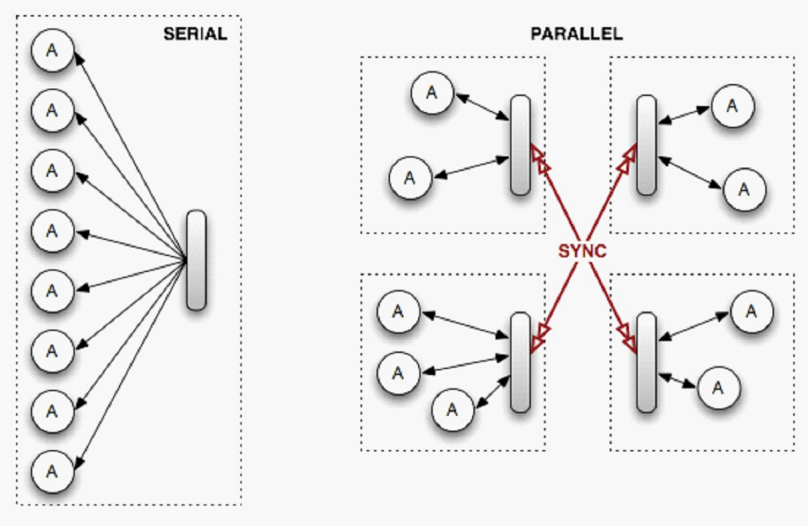

Figure 1: Serial and Parallel Message Boards

achieve an effective parallel implementation are: load balancing and communications strategy.

Initial load balancing to not too difficult: we have a population of agents - of various complexities to which we can assign relative weights - in the most general case the agents can be distributed over the available processors using the weights. This may well give an initial load balance but make no reference to the possible communications patterns of the agent population. As the simulation develops the numbers of agents in the population may change and adversely affect the load balance of the processors. It is a very interesting and difficult problem to gauge whether the additional work (computation and communication) involved in remedying a load imbalance is worth the gain. Given that the goal of any dynamic re-organising of the agents is to reduce the elapsed time of the overall simulation, determining whether a process of dynamically re-balancing the population will contribute to this is very problematic. It may well be that a slight load in-balance will have no significant effect of the wall clock time of the simulation. These problems are under investigation.

The communications patterns and volumes of the population will have a considerable impact of the performance and parallel efficiency of the simulation. In general agents are rather lightweight in the computational load. Where all agents can and do communicate with all others

the communications load within and across processors will be great. Fortunately communications within a processor are generally efficient. However across processors this communication can dominate the application. Within FLAME communications between agents is managed by the Message Board Library, which uses MPI to communicate between processors. The Message Board Library implementation attempts to minimise this communication overhead by overlapping the computational load of the agents with the communication.

Where the agents have some form of locality the initial distribution of agents makes use of this information in placing agents on processing nodes. During the simulation agents can be dynamically re-distributed to maintain computational load balance. However given the light-weight computational nature of many agent types the effect of dynamically re-distributing agents on the grounds of their communications load may well turn out to be more important than computational load.

Within the EURACE Project a parallel version of FLAME has been developed using these ideas and below are discussed some of the initial results in performing parallel simulations.

3 Detailed Desciption of Parallelisation

3.1 Overview of the Message Board Library

Within a FLAME simulation, every agent only interacts with its environment via the reading and writing of messages to a collection of Message Boards. This makes the Message Board component the ideal candidate for enabling parallelism. With a distributable Message Boards, agents can be farmed out across multiple processing nodes and simulated in parallel, while a coherent simulation can be maintained through the unified view of the distributed Boards.

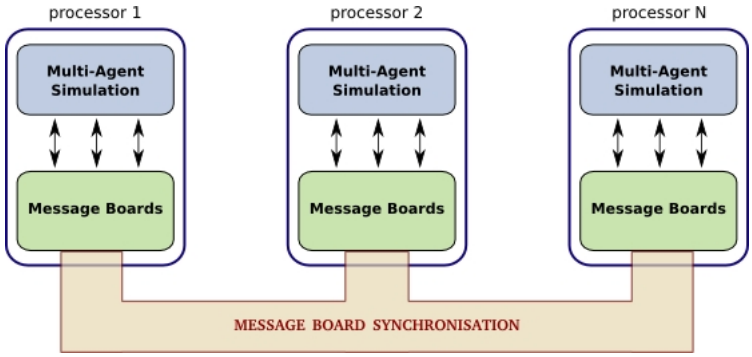

Figure 2: Parallelisation of FLAME using distributed Message Boards

In the recent code release, the Message Board was decoupled from the FLAME framework and implemented as a separate library. This will provides us with the flexibility to experiment with different parallelisation strategies while minimising the impact on current users of the FLAME framework.

The Message Board Library (*libmboard*) was designed as a static library that can be linked to the simulation binaries and accessed via the libmboard Application Program Interface (API).

libmboard uses MPI to communicate between processors, and POSIX threads (pthreads) to fork a separate thread for handling data management and inter-process communication. The use of threads enables the memory intensive operations of managing the Message Boards to be performed concurrently with the agent simulations.

Apart from potentially making better use of multi-core processors, delegating *libmboard* operations to a separate thread also allows us to minimise the overheads by overlapping the Board synchronisation time with useful computation.

3.2 The *libmboard* API

- Quick desc of the API. Point to User Doc for details.
- Opaque objects
- return codes

3.2.1 Library environment

- Init and finalise
- MPI env
- fork/join comm thread thread

3.2.2 Boards

- create/clear/delete
- AddMessage.. clone mem vs storing ptr

3.2.3 Iterators

- isolate users from internal data representation
- normal/filtered/sorted
- returning cloned mem vs ptr.
- randomisation
- rewind

3.2.4 Synchronisation

- Message Tagging vs Filtered Iterators.
- Why important? Impact on solution time?
- Details on how messages are packed and distributed

3.3 The Communication Thread

- motivation
- queues
- state diagram

3.4 Issues

- MPI thread support
- MPI sends/receives requires contiguous buffers. Expensive (space and time) packing of messages.
- Comm stages not fully non-blocking
- Total tagged messages may be more than actual message. Gets works with more proc. Scaling issues.

4 Detail on Dynamic Load Balancing

4.1 Dynamic Load Balancing Overview

The overall aim of parallelising the FLAME framework is to reduce the wall clock time for running a simulation. This relies on efficient parallelisation of communication and keeping the work load balanced between computing nodes. The message board library addresses the first of these and dynamic load balancing addresses the second.

To illustrate the problems in getting load balancing right the diagram in Figure 3 shows agents on two nodes and their communication patterns. The top portion shows an unbalanced number of agents but the frequent communication is internal to each node with only infrequent communication between the nodes. If agents are moved to try to balance the load then frequent communication between nodes is introduced (lower portion of figure) which could mean a large increase in communication time and hence wall clock time.

This example shows that measurement of communication between nodes must be part of the load balancing algorithm as well as elapsed time for various parts of the framework.

This section describes our initial thoughts on dynamic load balancing. The implementation has started with a timer which allows developers to insert timing into any section of FLAME from the framework itself to the user's model functions. It will be timing of different parts of the framework (which includes calls to the user's functions) that provides data for algorithms that seek to balance computational load.

4.2 The *timer* Package

Timers will be used to measure the elapsed CPU time for portions of the running code and this data will be used as input to the load balancing strategy used in the FLAME framework. The initial requirements against which the timer package was implemented are given below.

- Can have multiple timers running simultaneously
- Timers can be identified individually
- Functions to start/stop/reset a named timer
- Function to get elapsed time from a named timer
- Definition of a set of timers.
- Functions to get statistics from a set of timer
- Turn timing on/off during program execution

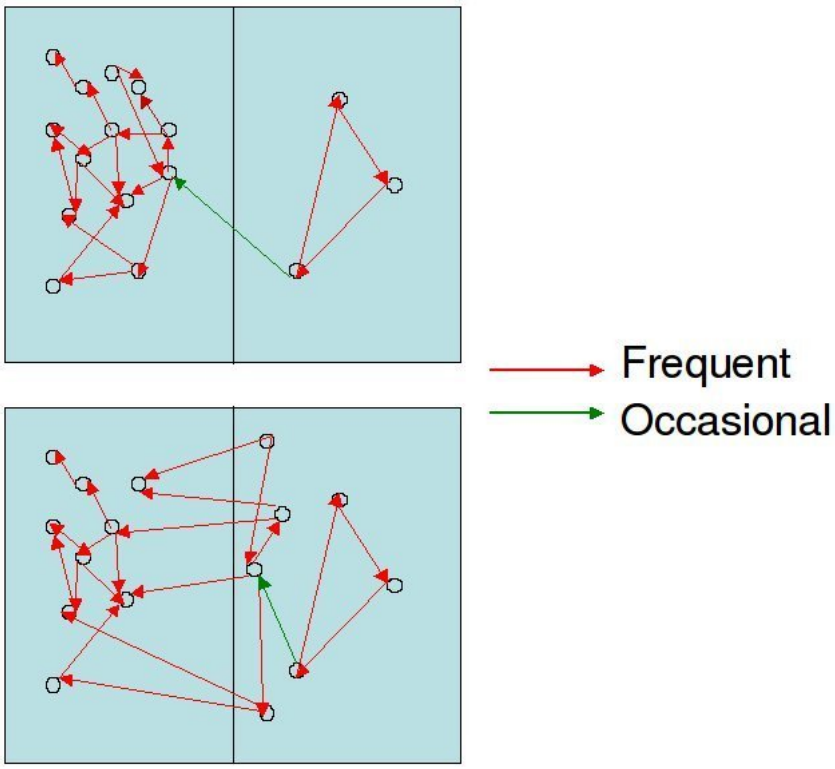

Figure 3: Illustrating some problems of load balancing

We have implemented all the functionality for individual timers and a set of unit tests and an example program using the timers. Code is stored under Subversion source code control in the FLAME project on the CCPForge site.

User documentation is supplied in [21].

5 Testing: Functional and Portability

5.1 Unit testing of the message board API

As mentioned above the message boards (are accessed by the FLAME framework via a Application Program Interface - the Message Board Library (the libmboard API). This provides the FLAME developer with a uniform interface to the functionality of the libmboard.

5.2 Testing serial and parallel implementations

It is important to ensure that application generated by the FLAME framework execute *correctly* in both their serial and parallel modes. Because of the stocastic nature of the agent-based approach to modelling it is unrealistic to expect complex simulations to following exactly the solution path although general trends should be similar. However for some simple applications we can expect to serial and parallel implementations to produce exactly the results throughout the simulation. Such example applications can be used to verify the correctness of both the serial and parallel implementations.

The *Circles Model* is one such application. The *Circles* agent is very simple. It has a position in two-dimensional space and a radius of influence. Each agent will react to its neighbours within its interaction radius repulsively. So given a sufficient simulation time the initial distribution of agents will tend to a field of uniformly spaced agents. Each agent has x, y, fx, fy and $radius$

in its memory and has three states: outputdata, inputdata and move. The agents communicate via a single message board, *location*, which holds the agent *id* and position. Given the simplicity of the agent it is possible to determine the final result of a number of ideal models.

A set of simple test models and problems have been developed based on the *Circles* agent. Each test has a *model.xmml* files and a set of initial data (*0.xml*).

Test 1 : Model: single *Circles* agent type; Initial population of no agents. Expected result:

Test 2 : Model: single *Circles* agent type; Initial population of one agent at (0,0).

6 Benchmark Problems

6.1 Approach to Benchmarking

6.2 The Circles Model

The Circles agent is very simple. It has a position in two-dimensional space and a radius of influence. Each agent will react to its neighbours within its interaction radius repulsively. So given a sufficient simulation time the initial distribution of agents will tend to a field of uniformly spaced agents.

The description of the agent is given as a example of XMML in the sections above. Each agent has x, y, fx, fy and *radius* in its memory and has three states: outputdata, inputdata and move. The agents communicate via a single message board, *location*, which holds the agent *id* and position.

The Circles problem is very simple but allows us an initial assessment of the performance of the parallelisation within FLAME. The simulation was started with a populations of 10^6 agents and experiments performed using from 4 to 100 processors. The averaged results are shown in Table **??** and Figure **??**.

6.3 The C@S Model

The C@S model was the first economic model to be implemented in FLAME by the EURACE Project. It is based on work detailed in Delli Gatti *et al.* [20] where an economy is populated by a finite number of *firms*, *workers/consumers* and *banks*. The acronym C@S stands for *Complex Adaptive Trivial System*.

This provides an initial economic model for testing FLAME. The EURACE version of C@S contains models for consumption goods, labour services and credit services. The population is a mix of agents: *Malls*, *Firms* and *People*. Each of these has different states and communicates with other agents in the population through 9 message types.

As the agents in the C@S Model have some positional/location data and the communication is localised, the initial distribution of agents to processors, as in the Circles Model, can be based on location. This helps reduce cross-processor communication.

The initial population contained: 20000 firms, 100000 people and 4000 malls (124000 agents in total).

7 Future Development and Optimisation

future

8 Conclusion

In this report we have described the parallel implementation of the FLAME framework and demonstrated its use in a number of EURACE related simulations including one of the complete EURACE model.

References

[1] T Takahashi, H Mizuta (2006) "Efficient Agent-Based Simulation Framework for Multi-Node Supercomputers", Simulation Conference, 2006. WSC 06. Proceedings of the Winter Volume , Issue , 3-6 Dec

[2] D Pawlaszczyk (2006) "Scalable Multi Agent Based Simulation - Considering Efficient Simulation of Transport Logistics Networks" 12th ASIM Conference - Simulation in Production and Logistics

[3] A Chaturvedi, J Chi *et al* (2004) SAMAS: Scalable Architecture for Multiresolution Agent-Based Simulation. In: M. Bubak et al. (eds.): ICCS 2004, LNCS 3038, Springer.

[4] Mangina (2002) "Review of software products for multi-agent systems", Agent-Link, http://www.AgentLink.org, July 2002

[5] Tesfatison (2006) "Agent-based computational economics: a constructive approach to economic theory" in Handbook for Computational Economics, Vol 2, Noth-Holland

[6] Finin et al (1994) "KQML as an Agent Communication Language", The Proceedings of the Third International Conference on Information and Knowledge Management

[7] Gregory et al (2001) "Computing Microbial Interactions and Communications in Real Life", 4th International Conference on Information Processing in Cells and Tissues

[8] Noble (2002) "Modeling the heart-from genes to cells to the whole organ", Science

[9] Coakley (2005) "Formal Software Architecture for Agent-Based Modelling in Biology", PhD Thesis, University of Sheffield

[10] Walker et al (2004) "Agent-based computational modeling of wounded epithelial cell monolayers", IEEE Transactions in NanoBioscience

[11] Walker et al (2004) "The Epitheliome: Agent-Based Modelling Of The Social Behaviour Of Cells", Biosystems

[12] Pogson et al (2006) "Formal Agent-Based Modelling of Intracellular Chemical Reactions", to appear in Biosystems

[13] Qwarnstrom et al (2006) "Predictive agent-based NFkB modelling - involvement of the actin cytoskeleton in pathway control", Submitted

[14] Jackson et al (2004) "Trial geometry gives polarity to ant foraging networks", Nature

[15] EURACE (2006) "Agent-based software platform for European economic policy design with heterogeneous interacting agents", EU IST Sixth Framework Programme.

[16] Holcombe (1998) "X-machines a basis for dynamic system specification", Software Engineering Journal

[17] Kefalas et al (2003) "Communicating X-machines: From Theory to Practice", Lecture Notes in Computer Science

[18] Kefalas et al (2003) "Simulation and verification of P systems through communicating X-machines", Biosystems

[19] Eleftherakis et al (2003) "An agile formal development methodology", Proceedings of the First South-East European Workshop on Formal Methods

[20] Delli Gatti et al (2006), "Emergent Macroeconomics An Agent-based Approach to Business Fluctuations", submitted.

[21] D Worth (2008), "Dynamic Load Balancing Library - Timer User API, Version 0.0.1", August 2008.