Science & Technology Facilities Council

Technical Report

# The Exploitation of Parallel High Performance Systems in the FLAME Agent-Based Simulation Framework

C Greenough[1], LS Chin[1], DJ Worth[1], M Holcome[2], S Coakley[2]

June 2008

RAL-TR-2008-022

Library and Information Services
SFTC Rutherford Appleton Laboratory
Harwell Science and Innovation Campus
Didcot
OX11 0QX
UK
Tel: +44 (0)1235 445384
Fax: +44(0)1235 446403
Email: library@rl.ac.uk



**ISSN 1358-6254**

**RAL-TR-2008-022**

# The Exploitation of Parallel High Performance Systems in the FLAME Agent-Based Simulation Framework

C Greenough[1], LS Chin[1], DJ Worth[1], M Holcome[2], S Coakley[2]

June 2008

**Abstract**

Making use of high performance computers in agent-based simulation is a complex and difficult task. This paper looks 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] Department of Computational Science and Engineering
STFC Rutherford Appleton Laboratory

[2] Department of Computer Science, University of Sheffield

**Keywords:** Agent-based simulation, Parallelisation, Economic Modelling

---

Email: c.greenough@rl.ac.uk; m.holcombe@dcs.ush.ac.uk
Reports can be obtained from www.softeng.cse.clrc.ac.uk

Software Engineering Group
Computational Science & Engineering Department
STFC Rutherford Appleton Laboratory
Harwell Science and Innovation Campus
Didcot
Oxfordshire OX11 0QX

# 1 Introduction

The agent-based approach to the economy can be viewed as a vast complex adaptive system, i.e., as a system consisting of a very large number of independent agents, that interact in various ways and that change their state or actions as a result of the events in the process of interaction. The study of the economy by means of agent-based computational models is a relatively new field and dates back to the 90s when the increasing availability of cheap computing power it made possible to undertake the computationally expensive experiments modelling the interactions of large numbers of heterogeneous agents. See Testatsion [1] for a recent survey on agent-based computational economics (ACE).

Agent-based systems have also been a major interest to researchers in a number of application fields both within Computer Science and in other subjects. For computer scientists the interests are often in areas relating to intelligent agents and this is usually meant to mean agents that exhibit some aspects of perception, learning and so-on. Often the systems built are highly experimental and the focus is not so much on understanding the application domain, which could be almost anything, but on understanding intelligence, behaviour and related issues.

Often the systems are built in an ad hoc manner and are very difficult to maintain and extend by people who are not directly involved in their construction. Some general frameworks have been proposed for agent systems, for example agent modelling languages such as KQML, Finin *et al.* [2] have focused more on research into the nature of agents and agent-based systems rather than being a practical vehicle for building industrial strength agent systems for a specific application domain. A wide variety of frameworks and languages have been explored to provide useful support for many different types of research using agent systems. All have their strengths and weaknesses and much can be gained by exploring these. Many of these address the problems encountered in complex biological systems. However, only a few approaches have generated significant results that really deliver new insights in biology. For example, although there has been much work in the field of swarm intelligence its primary benefit has been in the discovery of new search algorithms for computer science. In bacterial biology the work of Gregory *et al.* [3] and Noble [4] has led to some new biological understandings.

# 2 The FLAME X-machine framework

There are many agent frameworks proposed for various application domains, but few share the flexibility or the power of the approach use within FLAME (FLexible Agent Modelling Environment) developed by Coakley [5]. FLAME develops the ideas of Kefalas *et al.* [13] which describes a formal basis for the development of an agent-based simulation framework using the concept of a communicating X-machine.

The framework has been used, successfully, by biologists to uncover new aspects of biology and a better understanding of the processes underlying some biological systems, Walker *et al.* [6, 7], Pogson *et al.* [8], Qwarnstrom *et al.* [9], Jackson *et al.* [10] and is now being extend for use in other application areas such economic modelling, EURACE [11]. The agent populations in these simulations range from a few hundreds of agents to tens of thousands. The agents themselves range in complexity from simple position detection and motion to the solution of algebraic and differential equations to provide data for agent decisions.

In order to develop an extensible framework which allows for the construction of a simulation environment that will enable the creation and operation of millions of autonomous agents it is necessary to take a fundamental look at the design concepts that will allow for highly efficient simulations and with the potential to exploit parallel computer systems.

The following set of desirable features reflect not only the practical issues facing agent-based modelling but also the principal scientific needs of providing suitable information to enable other researchers to investigate the mechanics of the model, to adapt and to extend it and to trust that it is founded on a coherent, consistent and realistic set of principles.

- Each agent needs to be defined in a way that is as general and flexible as possible. A specification language is needed rather than using working source code or pseudo code or outline algorithms.
- The environment of the model also needs to be defined precisely. In particular the nature of the communication that exists between the agents and the external environmental that will influence the model.
- A standard framework for compiling the agent specifications onto a suitable platform for computation. This is likely to require mechanisms for utilising parallel and Grid computing in order to deal with realistic numbers of agent - millions rather than thousands.
- The framework would lead to an exchange mechanism whereby researchers would make their agent descriptions available to others to use within their simulations.
- The mathematical analysis of complex systems is extremely difficult and the use of formally defined agents could be a basis for the validation and verification of models in the longer term.

FLAME addresses many of these issues. It has an agent specification language, XMML (based on the XML standard), a set of tools to compile the specified agent-based systems into code using a set of standard templates and the potential to produce optimised code for efficient parallel processing. FLAME allows modellers to define their agent based systems and automatically generate efficient C code which can be compiled and executed on both serial and parallel systems. So the main elements of FLAME are: the XMML model definition, the functions files (contain C code) and the FLAME Xparser with its associated templates. Through

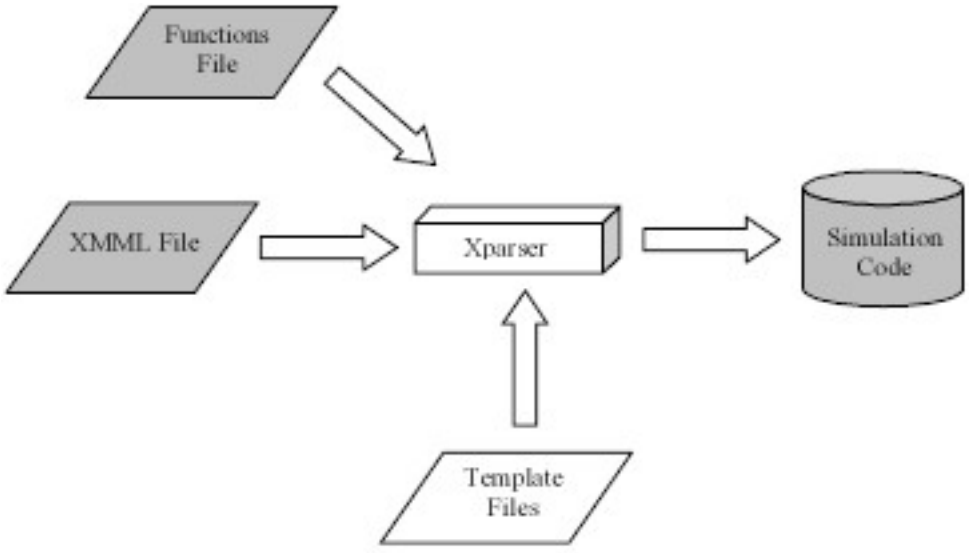


Figure 1: FLAME Components

an analysis of the XMML FLAME constructs an application specific code using the functions provided in the functions file. The ordering of agent state execution is determined through the generation of a function dependency graph. This graph drives the generation of the application's code.

At the software implementation level the essential architecture of FLAME is based on the premise that, during the simulation, agents pass through a number of defined state changes and communicate with other agents through a collection of message boards. Thus the Xparser generates a multiplicity of message boards to manage the community. This gives FLAME its

unique flexibility in the management of agents and their associated message boards. This also allows FLAME to exploit the potential benefits of utilising parallel computation.

The design and implementation of FLAME allow for detailed validation, systematic and formalised simulation and testing based on previous work by Holcombe [12], Kefalas *et al.* [13, 14], Eleftherakis *et al.* [15].

# 3 X-Machine Agent Representation

The language of X-agents, XMML, provides a precise definition of the individual behaviour of agents by specifying their state transitions, internal memory and communication protocols. It also defines implicitly all the interactions between agents. There are many issues that need addressing in the definition of agent populations and their interactions. Clearly in multi agent systems agents have different characteristics and have varied life styles. For example, in biological systems cells divide, new agents appear and disappear and in economic systems Firms are formed and may subsequently be disolved. This is something that XMML attempts to deal with by defining a rich and flexible set of agent-based attributes.

The Extensible Mark-up Language (XML) is capable of describing many different types of data. It provides a standard way to share structured text. By having a defined way to structure an agent, agent-based models can be created, edited, shared, and merged between modellers. The current design is very straight forward and uses simple nested XML tags to describe the structure of an X-machine. Below we describe some of the basic elements of FLAMEs XMML based on Version 0.14.0 of the FLAME Xparser.

We use a very simple test model to illustrate the use of the XMML language. The Circles model is a collection of point agents, in two-dimensional space, that interact with neighbouring agents, within their radius of influence, by repulsion.

Before the X-machine is defined environment variables and functions can be defined. These are values and functions that can be used by transition functions of the X-machine. They typically include static values and functions that are called more then once by the X-machine. Environment functions can also be used to make the code of the transition functions more readable. The constituent parts of the XMML definition of a X-machine model are: its *environment*, the *xmachine* definitions and their associated *messages*.

```
<xmachine_agent_model>
      <name>Circles Test 1</name>
      <author>Chris Greenough</author>
      <date>15 May 2008</date>
      <environment>.....</environment>
      <xmachine>
            <memory>....</memory>
            <functions>....</functions>
      </xmachine>
      <messages>....</messages>
</xmachine_agent_model>
```

These statements define the basic XMML structure for the Circles Test 1 model. We will outline to four main elements of the model starting with environment. The environment block specifies essential global data for the model such as the names of function files and in future releases the definition of time in the model. In this example we only specify the file name which contains the C code functions associated with the model - `functions.c`.

```
<environment>
       <functions>
       <file>functions.c</file>
```

```
        </functions>
</environment>
```

The next step is to define the agents within the model: in this case one very simple agent called *Circle*. The agent has two main elements: its memory and its functions. The memory section defines all the elements of information to be carried by the agent. The functions section defines the states the agent can enter and how they are related. The ordering of the functions determines the execution sequence and the `depends` tags specifies any dependency on information from other agents provided through the message boards. In this case the *inputdata* function is dependent on location messages from other agents (this is defined later).

```
<xmachine>
        <name>Circle</name>
        <memory>
        <var><type>int</type><name>id</name></var>
        <var><type>double</type><name>x</name></var>
        <var><type>double</type><name>y</name></var>
        <var><type>double</type><name>fx</name></var>
        <var><type>double</type><name>fy</name></var>
        <var><type>double</type><name>radius</name></var>
        </memory>
        <functions>
        <function>
        <name>outputdata</name>
        </function>
        <function>
        <name>inputdata</name>
<depends><name>outputdata</name>
<type>location</type></depends>
        </function>
        <function>
        <name>move</name>
<depends><name>inputdata</name>
<type>internal</type></depends>
        </function>
        </functions>
</xmachine>
```

This example shows variables in the X-machine's memory that are of type integer or double and can be referenced by their name, for example *id*. Other variables can be added in the same way. We only show basic data types but there is scope to add other data-types such as arrays and structured data types. X-machine states are defined by the transition functions to other states. In this example there are three state transition functions: *outputdata*, *inputdata* and *move*. The actual operations performed in these will be defined in the C code associated with the function in functions.c. Within these functions FLAME provides methods to access the agent memory and to access the messages boards.

Lastly the messages that can be sent and received need to be well defined also. Each message is defined inside a message tag, is given a name, and any variables it needs to hold. The message defined below refers to the message used in the above X-machine function.

```
<messages>
        <message>
        <name>location</name>
        <var><type>int</type><name>id</name></var>
```

```
        <var><type>double</type><name>range</name></var>
        <var><type>double</type><name>x</name></var>
        <var><type>double</type><name>y</name></var>
        <var><type>double</type><name>z</name></var>
        </message>
</messages>
```

The format for the message variables is the same format as the variables defined in the X-machine's internal memory. Many types of messages can be defined with variables for different purposes.

The FLAME Xparser processes the XMML definition and using its templates converts the model description to C source code and an appropriate make file. The make file will compile and build the executable program. The parser can produce two versions: a serial version for running on standard workstations and a parallel version, that uses the message passing interface (MPI), for execution on parallel computing systems.

# 4 Parallelisation of the FLAME framework

Many agent simulation packages use the concept of a communications array which in the most general case will be an $nxnxm$ data space, where $n$ is the number of agents in the simulation population and $m$ the maximum number of messages. This is most often held as a single data space and addressed as required to store communication information. This, albeit a very large array, is an adequate implementation in a serial environment. However the single communications array is not a good structure when considering a parallel implementation of the simulator.

To introduce a mechanism by which the inherent parallelism in a simulation might be exploited FLAME abandons the communications array in favour of a collection of message boards. There is one message board for each message type in the system. Agents write and read information to these boards. By doing this the granularity of the information transfer mechanism is greatly increased: from one monolithic repository to a number of simpler and smaller repositories.

This break down of the single communications array allows the program to group agents and message boards which will have great benefits in parallel or distributed implementation of an agent framework. It can also have significant benefits in the serial implementation as well.

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.

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 2 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

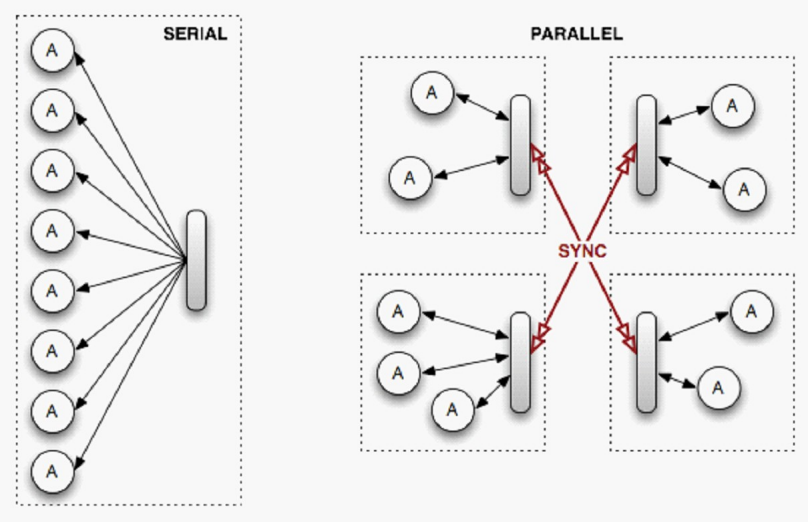


Figure 2: Serial and Parallel Message Boards

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 light-weight 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.

# 5 Some initial performance and efficiency results

Testing a parallel implementation of an agent-based simulation is quite difficult in general. However we have been able to test FLAME initially with small problems with known solutions: the Circles Agent is one of these test problems. The second phase of testing used more complex problems with multiple agent type: the C@S and the Labour Market models are these.

In the section below we present the initial results of running the FLAME generated applications on a variety of parallel systems. We will not dwell on the details of these system suffice to say they vary in processor and communications hardware and range from 128 to 2,500 processors in size.

## 5.1 The Circles Agent Test

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 1 and Figure 3.

The results indicate that this simulation benefits from using 30 to 50 processors after which the performance benefits flatten. It is interesting to note that this is essentially similar across the range of systems used. The variations between systems being attributed to memory, architecture and communications hardware differences.

| Processors | SCARF | HAPU | NW-GRID | HPCx |
|---|---|---|---|---|
| 4 | 2510.723 | 1710.531 | 2791.994 | - |
| 9 | 1021.234 | 277.672 | 890.378 | - |
| 16 | 321.057 | 106.169 | 388.098 | 403.122 |
| 25 | 189.211 | 67.646 | 196.751 | 161.757 |
| 36 | 67.413 | 52.912 | 121.571 | 78.468 |
| 49 | 37.080 | 19.682 | 58.068 | 42.503 |
| 64 | 23.199 | 16.482 | 42.881 | 25.122 |
| 81 | 14.826 | 12.971 | 38.033 | 15.849 |
| 100 | 6.653 | 6.641 | 27.603 | 10.322 |

Table 1: Execution Times for $10^6$ Circles

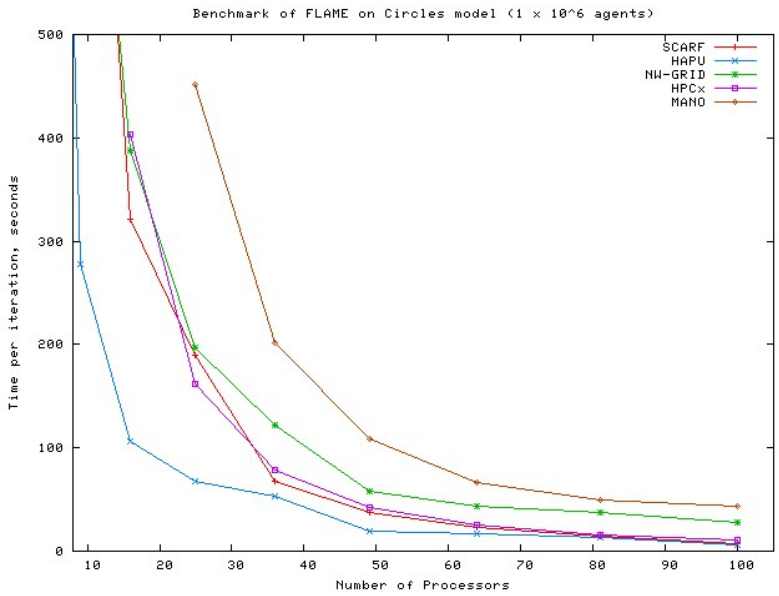


Figure 3: Graph of iteration times

## 5.2 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.* [16] 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).

The results show a potential reduction of the elapsed time of the simulation when using up to 30 processors.

| Processors | HAPU | NW-GRID | HPCx | MANO |
|---|---|---|---|---|
| 4 | 378.34 | 572.23 | 276.81 | 918.20 |
| 9 | 169.54 | 295.81 | 214.61 | 664.81 |
| 16 | 105.90 | 147.67 | 91.12 | 307.92 |
| 25 | 68.64 | 89.97 | 54.89 | 210.33 |
| 36 | 46.53 | 60.06 | 36.11 | 142.35 |
| 49 | 33.81 | 45.39 | 25.54 | 107.70 |
| 64 | 25.77 | - | 19.48 | 86.74 |
| 81 | 23.50 | 30.95 | 15.65 | 78.51 |
| 100 | 17.51 | - | 12.52 | 67.28 |

Table 2: Execution Times for C@S Model

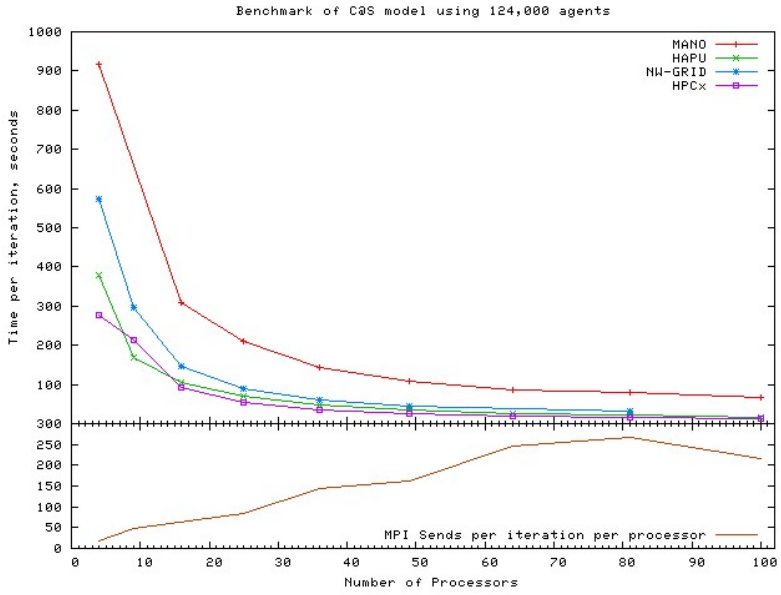


Figure 4: Graph of C@S Model execution times

### 5.3 The Labour Market Model

The Labour Market Model was the first attempt to develop a more realistic model within EURACE. The basic ideas were proposed by the University of Bielefeld and implemented jointly between Bielefeld and Sheffield. The model represents the matching of job seekers with the vacancies available in a selection of Firms.

The model population has four agent types: households, firms, markets and Eurostat. The *market* and *Eurostat* are agents that manage global data such as market data and wage costs. The Households and Firms make use of this data to inform their decisions.

So in this model there are a mix of agents communicating through 10 message types. An important point to note is that unlike the Circles and C@S agents there is no positional data in the model so there is a many-to-many communication.

The initial population contained: 10000 firms, 100000 households, 1000 markets and 1 Eurostat (110101 agents in total).

The results for this model show similar performance and the main performance gains are limited to around 20 to 30 processors despite the many-to-many communications. It the current implementation is results in large amounts of data replication. Although the trends are similar to the Circles and C@S models some curves show some erratic behaviour. These increases in

elapsed time may well be a result of the increased communications load.

| Processors | HAPU | NW-GRID | HPCx | MANO |
|---|---|---|---|---|
| 4 | 280.49 | 358.80 | - | 2420.93 |
| 9 | 134.32 | 203.71 | - | 360.86 |
| 16 | 67.23 | 107.75 | - | 251.16 |
| 25 | 59.28 | 119.75 | 104.70 | 192.34 |
| 36 | 46.92 | 107.97 | 86.411 | 53.77 |
| 49 | 40.19 | - | 85.10 | 124.04 |
| 64 | 38.17 | - | 83.81 | 96.48 |
| 81 | 36.56 | - | 91.30 | 85.02 |
| 100 | 36.21 | - | 83.97 | 116.90 |

Table 3: Execution Times for Labour Market Model

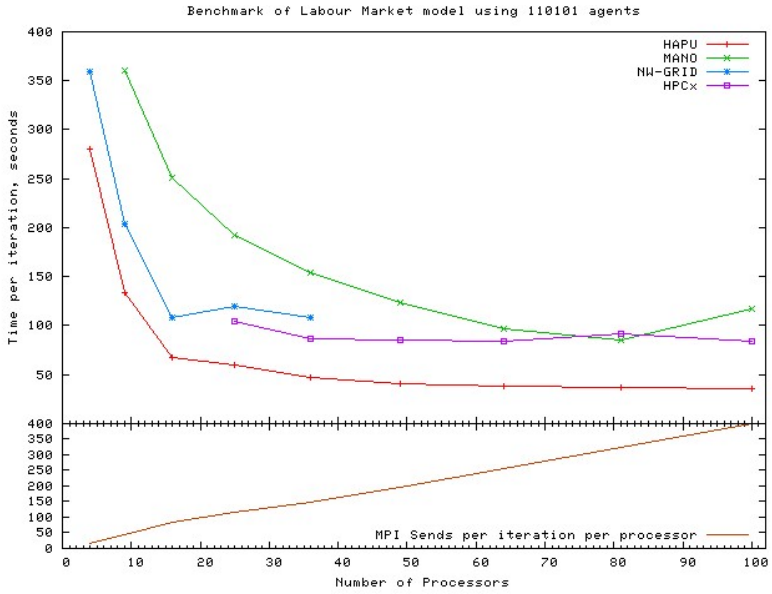


Figure 5: Graph of Labour Market Model iteraction times

## 6 Conclusions

The parallelisation of the underlying algorithms used in agent technologies is very difficult. The many-to-many communications network of the most general agent population is the primary bottleneck particularly if it is represented as an $n$-by-$n$ communications array. In the most general case all agents need to access this shared repository of information and mapping this requirement onto the architecture of most high performance computing systems is a challenge.

The FLAME framework uses the concept of message boards to replace the communications matrix and utilises the partitioning and distribution of these message boards to aid parallel implementation. This approach has been demonstrated to provide a sufficiently efficient parallel implementation in applications where the agent population has short range communication behaviour thus enabling agent neighbourhoods to be defined.

In applications where the communications network is less confined the message boards are treated as a shared resource and the shared data concepts in MPI are used to access the data.

Although this is not as efficient as having local populations with local communication networks the technique does allow the development of complex simulations of many tens of thousands of communicating agents.

## References


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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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