A Framework for Distributed Simulation for Asynchronous Handshake Circuits

E. Tsirogiannis, G. Theodoropoulos,
D. Chen, Q. Zhang

School of Computer Science
The University of Birmingham
Birmingham B15 2TT, UK

L. Janin, D. Edwards

Department of Computer Science
University of Manchester
Manchester M13 9PL, UK

Abstract

This paper presents PARBREEZE, *a distributed simulation kernel for asynchronous hardware behavioural simulation.* PARBREEZE *is based on the Logical Process paradigm and targets asynchronous handshake circuits generated by the Balsa asynchronous hardware synthesis environment. The paper describes the architecture of* PARBREEZE *and presents performance results for different partitioning strategies.*

1 Introduction

The increasing size and complexity of designs and quality assurance, reliability and relentless time-to-market pressures have rendered functional verification and performance evaluation the major bottleneck in hardware design, calling for ever faster and greater verification coverage. Two main approaches have been followed to address this issue, namely *testbench automation*[1] and *speeding up the simulation* per se, thus allowing an increase in the coverage through improved run times. Traditionally, it has been Switch and Logic level models whose demands in terms of time and memory made their simulation on conventional von Neumann machines extremely time consuming. However, the increasing complexity of architectural designs has also dramatically increased the requirements of higher level digital simulation (e.g. Register Transfer) and has long placed it in the highly computation intensive world, with computational requirements which far exceed the capabilities of conventional sequential von Neumann computer systems. As the complexity of the designs has increased, long execution times have made simulation a major and increasing bottleneck in the VLSI design process[2].

An approach to speed up the simulation is to exploit the inherent parallelism of digital systems and employ parallel and distributed simulation techniques whereby gates, functional blocks, etc. are modelled as "Logical Processes" (LPs) which may be executed on different processors [9]. Distributed simulation has emerged as a particularly promising and viable approach to alleviate the simulation bottleneck in VLSI design and over the past ten years has received considerable attention from researchers in mainstream Hardware Description Languages such as VHDL and Verilog [3, 15]. Distributed simulation techniques, albeit ad hoc, are progressively finding their place in innovative commercial hardware design environments (e.g. the VCK by Avery Design Systems[3], where SimCluster, an ad hoc distributed simulation engine, is used to leverage mainstream Verilog simulations).

Another important recent development in VLSI design has been a resurgence of interest in asynchronous design techniques, due to the significant potential benefits that the elimination of global synchronisation may offer to issues such as clock distribution, power consumption, performance and modularity [10].

A number of asynchronous processors have been developed including NSR and Fred at the University of Utah, STRiP at Stanford University, FAM and TITAC at Tokyo University and Institute of Technology respectively, Hades at the University of Hertfordshire, Sun's Counterflow pipeline processor, Sharp's Data-Driven Media Processor, CalTech's processors and Lutonium, the series of asynchronous implementations of the ARM RISC processor (AMULET1, AMULET2e, AMULET3i and SPA) developed by the AMULET group at the University of Manchester [1] and SAMIPS [21, 24, 25], a synthesisable asynchronous MIPS processor core developed at the University of Birmingham .

Synchronous VLSI modelling and simulation techniques are proving unsuitable for the asynchronous design style and therefore the last decade has witnessed an intense research activity aimed at developing notations and techniques appropriate for modelling and simulating asynchronous systems. I-Nets, Petri Nets, Signal Transition Graphs, CCS and in particular the concurrent process algebra Communicating Sequential Processes (CSP) are some of these tools and formalisms that have been employed in

[1]This approach has sought to automate the design environment, utilizing more efficient test and debugging tools, such as test plan specifications, code coverage tools, testbench generators and result checking techniques.

[2]In his keynote speech at the International HDL Conference (HDLCon) in March 2002, Raul Camposano, chief technology officer at Synopsys Inc., indicated that "complexity has caused simulation needs to grow a thousandfold since 1995, while simulator speed has only increased fiftyfold".

[3]`http://www.avery-design.com`

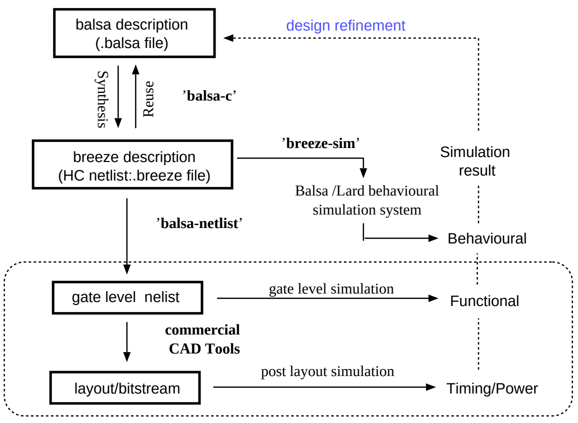

Figure 1. The Balsa System

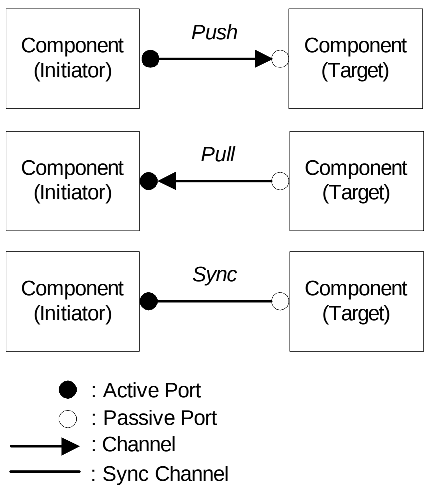

Figure 2. Push, Pull and Sync Channels

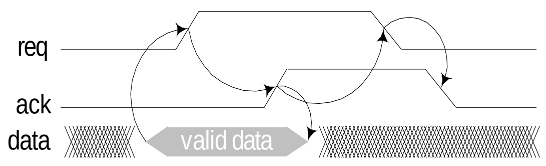

Figure 3. 4-phase Signaling

asynchronous logic design [17].

This paper presents PARBREEZE, a framework for the distributed simulation of asynchronous hardware. The framework targets the behavioural simulation of asynchronous hardware developed within Balsa, a CSP-based synthesis environment developed at the University of Manchester, UK [8]. The rest of the paper is organised as follows: section 2 provides an overview of the Balsa system and the associated handshake circuits; section 3 discusses the role of simulation in asynchronous hardware design; section 4 describes the architecture of PARBREEZE; section 5 discuss performance results; and section 6 summarises the paper and identifies areas for further work.

2 Balsa and Handshake Circuits

Balsa [6] is both an asynchronous hardware synthesis framework and a CSP-based language for describing asynchronous systems. It has been demonstrated by synthesising the DMA controller of Amulet3i, and SPA [16], an AMULET core for smartcard applications , and SAMIPS.

Figure 1 shows an overview of the Balsa system. Balsa uses CSP-based constructs to express Register Transfer Level design descriptions in terms of channel communications and fine grain concurrent and sequential process decomposition.

2.1 Handshake Circuits

Descriptions of designs (*.balsa* file) are translated (*Balsa-c*) into implementations in a syntax directed-fashion with language constructs being mapped into networks of parameterised instances of *handshake components* (*.breeze* file) each of which has a concrete gate level implementation [4]. Balsa handshake circuits are very similar to those introduced in the Philips Tangram system [7].

A number of tools are available to process the breeze handshake files [6]. *Balsa-netlist* automatically generates CAD native netlist files, which can then be fed into the commercial CAD tools that further synthesize the netlist to the fabricable layout.

Balsa has approximately fifty handshake components in total, most of them inherited from Tangram. Each component has a unique name, symbol, definition and several implementations (based on different technologies). In the handshake circuit, components communicate via point-to-point *channels* to exchange control information and optionally data. During a transaction, the *initiator* component requests the transfer of information (by issuing a *request* signal) and the *target* component responds (with an *acknowledgement*). Channels are connected to components via *ports* which may be either *active* (connected at initiator's side) or a *passive* (connected at target's side). Depending on the direction of the data flow, a channel is classified as a *push* channel (from the the initiator to the target) or a *pull* channel (from the target to the initiator). *sync* channels are used for synchronisation and do not carry any data. Figure 2 illustrates the channel types. Depending on the way that transitions on wires are interpreted, there are two protocols for implementing request and acknowledge signals, namely *2-phase* and *4-phase*. Figure 3 shows an example of a 4-phase signaling for push channels.

2.2 Levels of Simulation

Three levels of simulation are supported in Balsa, namely behavioural, gate-level and post layout simulation (figure 1). The latter two low simulation levels are carried out by the native simulators of the commercial CAD tools supported by Balsa. At the behavioural level, discrete-event simulation is used to simulate the network of handshake components. Two sequential simulators have been developed for this level, LARD [2] and *breeze-sim* [12].

3 Simulation in Asynchronous Hardware Design

Functional verification and performance evaluation is a more complex task in asynchronous systems than in their synchronous counterparts. In the latter, benchmark execution times are easy to interpret based on the number of clock cycles and the existence of a critical path. Delays in the critical path can determine the clock period while non-critical path delays have no effect on the performance of the system. In contrast, the temporal behaviour in asynchronous systems is more difficult to understand and interpret as delay inter-dependencies are more complex. Delays in one module may often be masked by occasional longer delays in another module, while the accumulation of delays through a chain reaction in a non-deterministic concurrent environment may have a chaotic effect on system performance. The need to evaluate the asynchronous architecture for different sub-system delays further complicates the process rendering simulation speed a crucial element [19].

For instance, the slow performance of LARD system often made it quicker to synthesise directly into a concrete realisation and then use the native CAD environment for simulation. As a result, functional simulation of the Amulet3i processor was severely constrained by the speed of the simulation . Consequently, the design was frozen prematurely in order to meet tape-out deadlines. A faster simulation environment would have allowed the design space to be explored more thoroughly [5].

An effort to improve the performance of the sequential simulation for Balsa, has yielded impressive results, with the *breeze-sim* simulator achieving a speedup factor of more that 19000 compared to LARD[4] [12].

However, as asynchronous design techniques find their place in the mainstream VLSI industry and asynchronous designs become increasingly more complex, simulation speed will remain a crucial problem and, like in synchronous hardware design, distributed simulation will provide the only viable solution .

4 Distributed Simulation of Balsa: The PARBREEZE Kernel

The distributed simulation effort for Balsa has targeted the handshake circuit level. Simulation at lower levels (switch, gate) would require a complete (expensive) CAD suite and a (specific) technology file and its associated libraries to be installed, merely to investigate design alternatives at the architecture level. A fast distributed simulation environment tightly coupled to Balsa is more sensible from the designer's viewpoint, avoiding the need to negotiate complex general purpose commercial CAD frameworks.

A decentralised event-driven approach based on the Logical Process Paradigm has been adopted for the development of the Balsa distributed simulator (the PARBREEZE kernel), while MPI is used for interprocess communication. The breeze file is parsed and partitioned so that different subsets of handshake components are assigned to different Logical Processes (LPs). Finding an optimal partition for a given circuit graph is an NP-complete problem and various heuristics have been developed to address this problem [11, 20].

[4]For instance, a comparable LARD simulation completes in 12 days 22 hours 15 minutes whereas the Breeze simulator takes only 56.9 seconds.

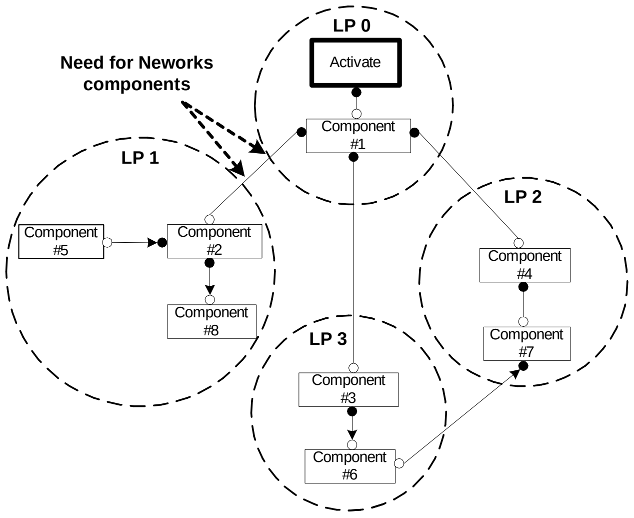

Figure 4. A Network of LPs in PARBREEZE

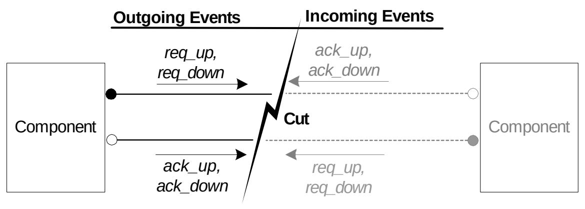

Figure 5. Cutting Handshake Channels

4.1 The Network Handshake Components

Figure 4 depicts an example configuration of the PARBREEZE kernel. Partitioning the handshake graph in different LPs generates a set of cut edges as depicted in figure 5. A edge cut defines a handshake channel which connects components in different LPs. To avoid modifying the implementation of the Balsa handshake channels, a new category of *Network Handshake Components* (NHC) has been defined to facilitate interprocess communication. NHCs are automatically inserted in an existing handshake circuit between the components over the cut. The NHCs implement the corresponding handshake channels via MPI messages while preserving their semantics.

Based on the channel types, six different NHCs have been defined, namely *BrzNetCompPassPush, BrzNetCompPassPull, BrzNetCompPassSync, BrzNetCompActPull, BrzNetCompActPush, BrzNetCompActSync* (figure 6).

Figure 7 shows the use of network components when sync, push and pull channels are cut. As an indicative ex-

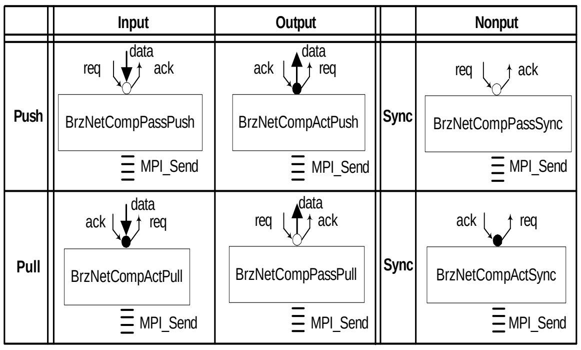

Figure 6. Network Handshake Components

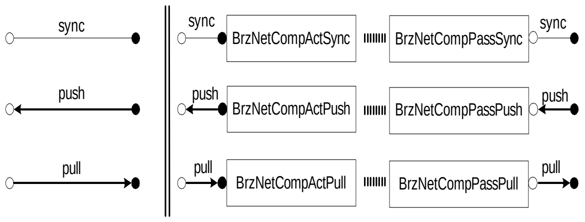

Figure 7. Use of Network Handshake Components

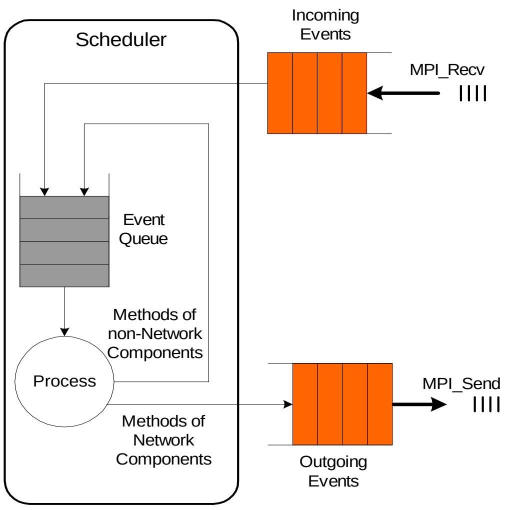

Figure 8. Event Driven Scheduler

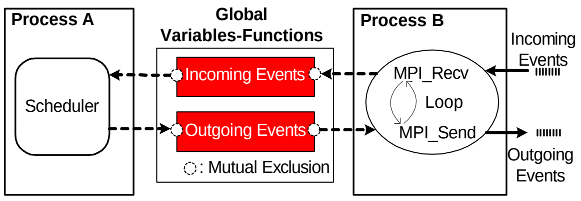

Figure 9. Mechanism for Incoming/Outgoing Events

ample, when a sync channel connecting an active port A to a passive port P is chosen by a partitioning algorithm to be an edge cut, then a pair of BrzNewCompActSync, BrzNetCompPassSync of NHC are introduced. After the cutting we obtain two new sync channels: the first one connects the passive port P with the active port of BrzNetCompActSync and the second connects the active port A with the passive port of BrzNetCompPassSync.

4.2 Event Driven Scheduler

Each LP in PARBREEZE is build around a typical event driven scheduler as depicted in figure 8. The scheduler extracts events from an chronologically ordered event queue and processes them invoking the model of the corresponding handshake component. Two message queues are also utilised to respectively send and receive MPI messages to remote LPs. If the processing of an input event results to a communication over a local channel, then the output event is placed in the event queue, otherwise the corresponding NHC is invoked and an MPI message is scheduled in the Outgoing Event queue.

Before the processing of the next event, the scheduler examines whether there are remote messages waiting in the Incoming Event queue, and inserts them all in the internal event queue. This may naturally result in causality errors, however, as we have shown in previous work [18], such errors can be ignored. The time stamps of the incoming events are all set to the the current value of the internal clock of the LP before they are inserted in the event queue.

The interaction with remote LPs is dealt with by a *listener* POSIX *thread* which runs in parallel with the main scheduler thread (figure 9). The listener receives MPI messages (using non-blocking *MPI_Iprobe* followed by the *MPI_Recv*) and inserts them in the Incoming Event queue. In the absence of incoming messages, the listener turns its attention to the outgoing queue sending all pending messages. Synchronisation between the two threads is achieved by means of semaphores, using the *pthread_mutex_lock* function. The proposed architecture is deadlock free.

5 Experiments and Results

A series of experiments have been performed in order to evaluate the performance of PARBREEZE, in relation to the sequential *breeze-sim* simulator. As a testbed, the SAMIPS asynchronous processor has been used. SAMIPS is built around a five-stage pipeline datapath (figure 10), namely Instruction Fetch (IF), Decode/Register File Read (ID), Execution or Address Calculation (EX), Memory Access (MEM) and Register Write-back (WB). The datapath includes an ALU, a Shifter, a Multiplier/Divider, an Address Adder, and a PC incrementer. SAMIPS has been modelled as a hierarchy of concurrent processes with approximately 900 lines of Balsa code , with the corresponding handshake circuit consisting of approximately 2300 handshake components and 3600 channels. SAMIPS executes MIPS machine language instructions produced by a MIPS cross-compiler and loaded during the initialisation phase as

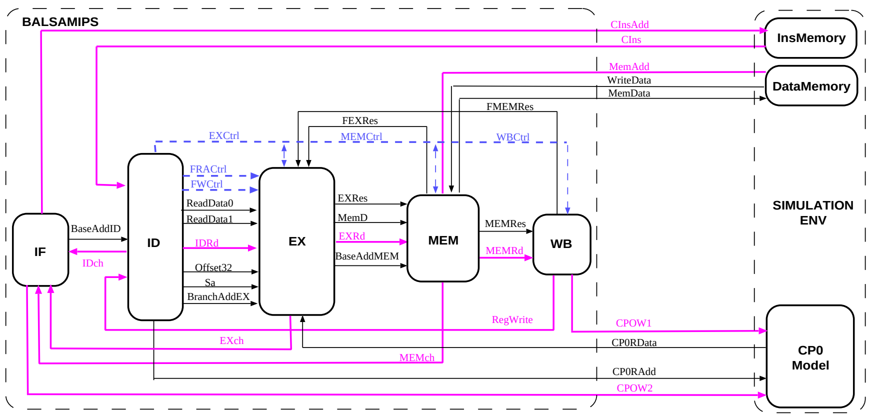

Figure 10. SAMIPS**: Top Level Process Graph**

32-bit quantities in hexadecimal format from an image file. The well established Dhrystone benchmark has been used for the experiments [23]. The execution platform is a cluster machine with dual-processor Intel Xeon 3GHz nodes and 2 GBytes of memory, interconnected via a Myrinet switch.

5.1 Partitioning Strategies

Two main partition strategies have been used, namely manual and using the well established graph partitioning system METIS [13]. The manual partitioning follows the pipeline stages of SAMIPS, from two up to five LPs.

METIS partitions a graph following a multilevel recursive bisection algorithm [14], where the vertices of the graph represent the set of the handshake components while the edges are the communication channels.

Four different partitioning strategies supported by METIS has been used (the first three require the execution of the simulation once, to collect the necessary information):

1. Weighted edges, where weights define the number of events on each channel. This strategy attempts to minimise communication costs.

2. Weighted vertices, where the weight of each vertex indicates *the number of events* processed by each handshake component. This strategy attempts to balance the load on each LP.

3. Weighted edges and vertices. Both load and communication are expected to be balanced.

4. No weights at all. The algorithm attempts to minimise the number of edgecuts and assign approximately the same number of vertices to each LP.

5.2 Execution Efficiency and Analysis

Figure 11 shows the performance achieved by PARBREEZE for the different partitioning algorithms using the MPICH-GM 1.2.6.14b for Myrinet.

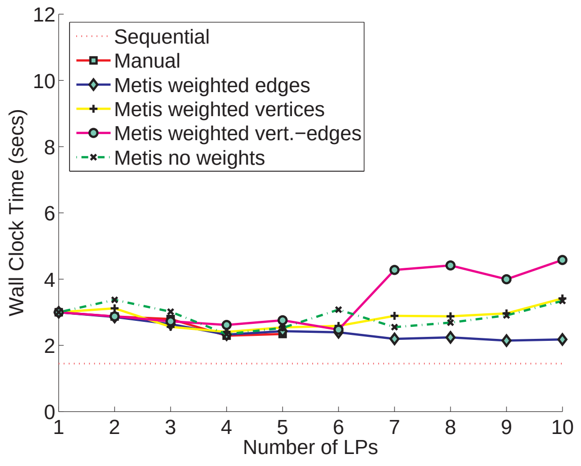

Figure 11. Performance of PARBREEZE **Using** MPICH-GM

Partitioning Method		# LPs	Max Speedup
Manual		4	1.31
METIS	No Weights	4	1.28
	Weighted Edges	9	1.40
	Weig. Vertices	4	1.24
	Weig. Edg.-Vert.	6	1.21

Table 1. Speedups for Different Partitions

The results show that the distribution of PARBREEZE can achieve a maximum speedup of 1.4 (26.8% reduction in execution time, see table 1) and that the choice of the partitioning strategy can have a significant impact on the execution efficiency of distributed simulator.

Understanding the factors that affect the performance of the simulator and the relationship amongst these factors is crucial for the selection of the appropriate partitioning algorithm.

As a first step in this endeavour, we have quantified the available parallelism in the model that can potentially be exploited. We define the *degree of parallelism* in the model

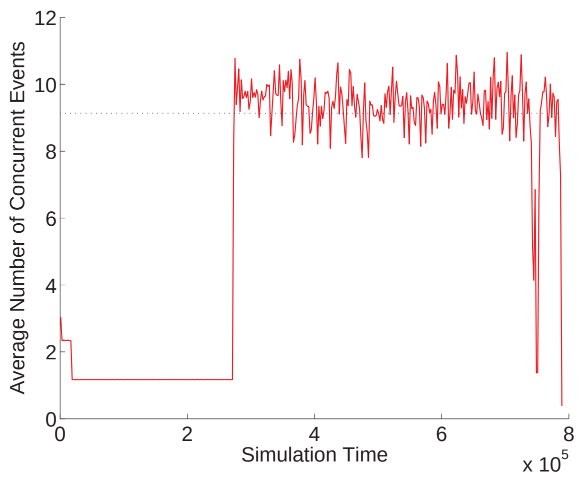

Figure 12. Degree of Parallelism of SAMIPS

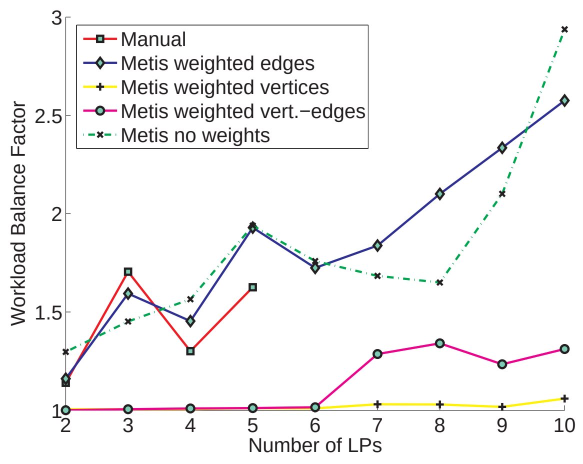

Figure 14. Workload Balance Factor

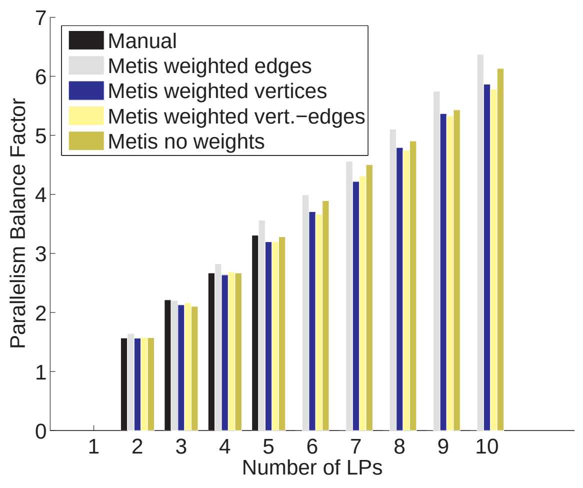

Figure 13. Parallelism Balance Factor

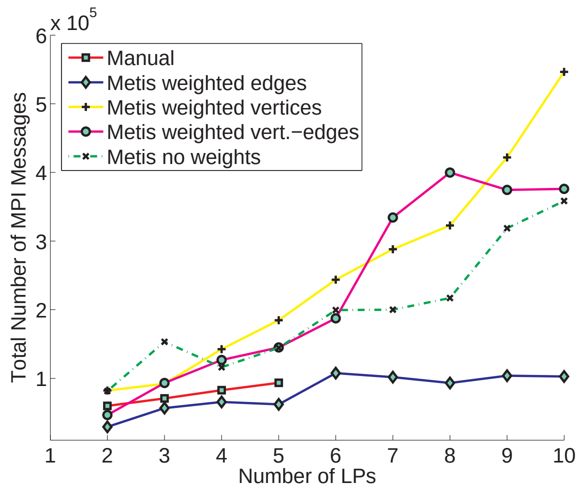

Figure 15. Total Communication Cost

as the average number of concurrent events in the system that can be executed at the same time step. The degree of parallelism of SAMIPS is presented in figure 12.

Figure 13 shows how well the partitioning strategies have exploited the available parallelism. The *parallelism balance factor* is defined as the (average) ratio of the maximum number of concurrent events in a particular time step assigned to a single LP (and therefore executed sequentially) over the number of events that the optimal, even distribution would have assigned to each LP.

Figure 14 illustrates the quality of the workload balancing in terms of the *workload balance factor*, namely the number of events executed by the busiest LP divided by the optimum number of events. The latter is defined as the total number of events in the whole graph divided by the number of LPs being used.

The results in figures 13 and 14 indicate that the partitioning strategies that achieve better exploitation of the parallelism and workload balance are those that include the weighted vertices as one of their parameters while the worse is the one that tries to minimise communication.

Figure 15 shows the total communication cost, in terms of the total number of MPI messages in the overall system. Clearly, as the number of LPs increases so does the total number of MPI messages that need to be exchanged between them. As expected, the partitioning that results is the lowest communication cost is the one that minimises the edgecuts in the graph.

Figure 16 shows the average idle times over all LPs for the different partitioning algorithms. The partitioning strategy which results in the lowest idle times is the one which minimises communication (which is also the one that achieves the maximum speedup in figure 1). This is explained by the fact that since there is a reduced number of messages in the system, the LPs spend less time waiting for remote messages.

From figure 11 it is clear that the reduction in communication yields the highest speedup even though the workload is not well balanced. This is mainly due to the fact that the simulation model is a communication bound system, rendering the communication overhead (figure 18) the

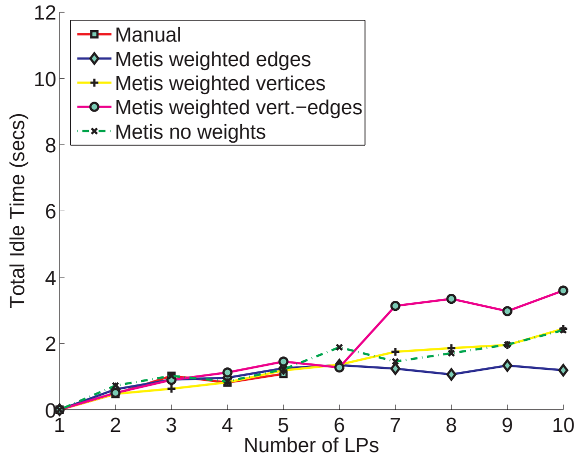

Figure 16. Average Idle Times Using MPICH-GM

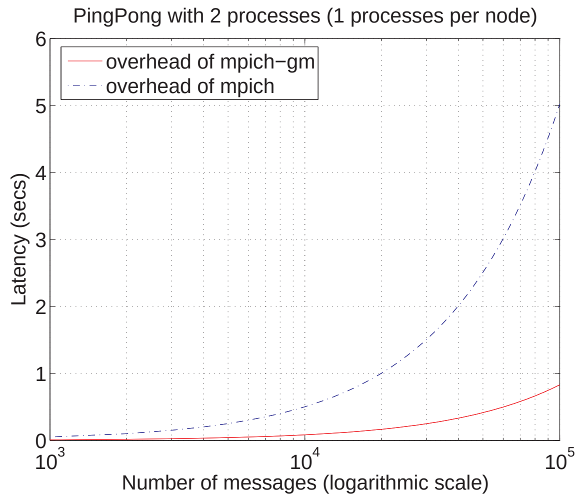

Figure 18. Communication Overhead

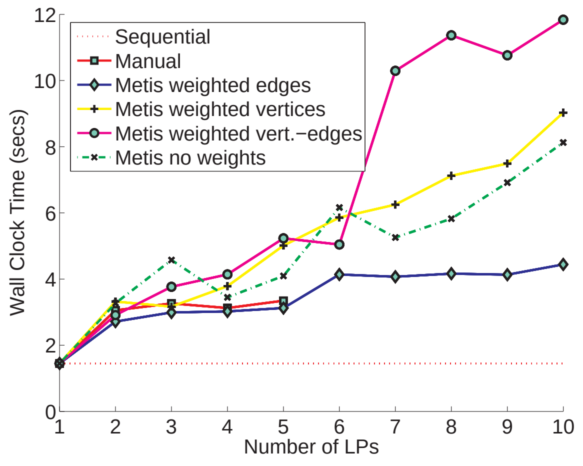

Figure 17. Performance of PARBREEZE Using MPICH

determining factor of the performance. Indeed, the cost of sending an MPI message corresponds to the processing time of approx. 18 events in the simulator. Figures 11 and 16 clearly show that the total execution times closely follow the patterns of idle times in the simulation.

5.3 Speedup and MPI Overhead

Despite the significant speedup achieved by the distribution of PARBREEZE, PARBREEZE has not managed to beat the performance of the sequential *breeze-sim* simulator. This is mainly due to the computation overhead that the MPICH-GM has introduced in the system, which has increased the execution time of the simulator on one node by 106% (figure 11). To investigate the cause of this computation overhead, a different version of MPI, MPICH has been tested. The results achieved are illustrated in figure 17. Clearly, as PARBREEZE is distributed onto more nodes, the execution times increase. This should be expected, as MPICH is for gigabit Ethernet and its communication efficiency is very poor compared to MPICH-GM[5] (figure 18). Hence the communication cost becomes the dominant factor of the simulation, as explained in the previous section. However, the computation overhead introduced by MPICH in PARBREEZE on one node is almost zero compared to the 106% introduced by MPICH-GM. This additional overhead of the MPICH-GM, can be attributed to the way it performs memory (*malloc*) and thread management. Indeed, a simple program which invoked *malloc()*, took 5.3 secs to execute on a single node when compiled with *gcc* and MPICH while within MPICH-GM it took 8.0 secs. Compiling the program using the *-pthread* paramemeter for POSIX threads, raised the execution time for MPICH-GM to 25 secs while it had no impact on that of MPICH (which remained at 5.3 secs).

6 Summary and Future work

Asynchronous Logic is progressively finding its place in the mainstream VLSI design, not least in the development of GALS (Globally Asynchronous Locally Synchronous) systems. As a result, there will be an increasing demand for appropriate efficient simulation techniques. This paper has presented a framework for the distributed simulation of asynchronous handshake circuits generated by the Balsa system. This work has shown that significant speedup can be achieved by the utilisation of distributed simulation. Our investigation has identified both the partitioning algorithm and the efficiency of the communication software (MPI) employed as significant factors for the performance of the simulator. Further work will investigate the cause of the computation overhead of MPICH-GM and will perform a more detailed analysis of the performance of the simulation, using additional benchmarks (such as the SPA processor) and partitioning algorithms (such as the JOSTLE system [22]). Synchronisation issues will also be investigated.

[5]These results have been obtained executing a ping-pong program with 2 MPI processes using blocking send and receive with message size of 40 bytes.

Acknowledgements

The work has been funded by EPSRC project GR/S11091[6]

References

[1] The Asynchronous Online Logic Home Page, The APT Group, School of Computer Science, University of Manchester, URL: `http://www.cs.manchester.ac.uk/apt/`.

[2] LARD Documentation Home Page URL: `http://www.cs.manchester.ac.uk/apt/projects/tools/lard/`.

[3] M. L. Bailey, J. V. Briner, and R. D. Chamberlain. Parallel Logic Simulation of VLSI Systems. *ACM Computing Surveys*, 26(3):255–294, September 1994.

[4] A. Bardsley. *Implementing Balsa Handshake Circuits*. PhD thesis, Deptartment of Computer Science, University of Manchester, 2000.

[5] A. Bardsley and D. Edwards. Synthesising an Asynchronous DMA Controller with Balsa. *Journal of Systems Architecture*, pages 1310–1319, December 2000.

[6] A. Bardsley and D. A. Edwards. Balsa - An Asynchronous Hardware Synthesis Language. *The Computer Journal*, 45(1):12–18, January 2002.

[7] C. H. V. Berkel, J. Kessels, M. Roncken, R. Saeijs, and F. Schalij. The VLSI-Programming Language Tangram and its Translation into Handshake Circuits. *Proceedings of EDAC*, pages 384–389, 1991.

[8] D. Edwards, A. Bardsley, L. Janin, and W. Toms. Balsa: A Tutorial Guide, Department of Computer Science, The University of Manchester, 2005.

[9] R. Fujimoto. *Parallel and Distributed Simulation Systems*. John Wiley & Sons, 2000. ISBN 0471183830.

[10] S. B. Furber. *Computing without Clocks: Micropipelining the ARM Processor*. Springer Verlag, January 1995. ISBN 3540199012.

[11] B. Hendrickson and T. G. Kolda. Graph partitioning models for parallel computing. *Parallel Computing*, 26(12):1519–1534, 2000.

[12] L. Janin. *Simulation and Visualisation for Debugging Large Scale Asynchronous Handshake Circuits*. PhD thesis, Deptartment of Computer Science, University of Manchester, 2004.

[13] G. Karypis. METIS Library ver 4.0.1, November 1998, URL: `http://www-users.cs.umn.edu/~karypis/metis/metis/index.html`.

[14] G. Karypis and V. Kumar. A Fast and Highly Quality Multilevel Scheme for Partitioning Graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[15] G. D. Peterson and J. C. Willis. High performance hardware description language simulation: Modelling issues and recommended practices. *Transctions of the Society for Computer Simulation International*, 6(1):1–10, January 1999.

[16] L. A. Plana, P. A. Riocreux, W. J. Bainbridge, A. Bardsley, S. Temple, J. D. Garside, and Z. C. Yu. Spa - a secure amulet core for smartcard applications. *Microprocessors and Microsystems*, 27:431–446, October 2003.

[17] J. Sparsø and S. Furber. *Principles of Asynchronous Circuit Design - A Systems Perspective*. Kluwer Academic Publishers, Hardcover ISBN 0-7923-7613-7, 2001.

[18] G. Theodoropoulos. Distributed Simulation of Asynchronous Hardware: The Program Driven Synchronisation Protocol. *Journal of Parallel and Distributed Computing, Special Issue on "Parallel and Distributed Discrete Event Simulation–An Emerging Technology", Academic Press*, 62(4):622–655, April 2002.

[19] G. Theodoropoulos. Modelling an Asynchronous Microprocessor. *Transactions of the Society for Computer Simulation International*, 79(1):377–409, July 2003.

[20] G. Theodoropoulos and D. Edwards. Towards a Framework for the Distributed Simulation of Asynchronous Hardware. *5th United Kingdom Simulation Society Conference, UKSim 2001, Emmanuel College, Cambridge, United Kingdom*, March 28-30, 2001.

[21] G. Theodoropoulos and Q. Zhang. A Distributed Colouring Scheme for Dealing with Control Hazards in Asynchronous Microprocessors. *The 7th International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN), Hong Kong, ISSN: 1087-4089*, pages 266–271, May 10-12, 2004.

[22] C. Walshaw. JOSTLE-graph partitioning software URL: `http://staffweb.cms.gre.ac.uk/~c.walshaw/jostle/`.

[23] R. P. Weicker. Dhrystone: A Synthetic Systems Programming Benchmark. *Communications of the ACM*, 27(10):1013–1030, 1984.

[24] Q. Zhang and G. Theodoropoulos. Towards an Asynchronous MIPS Processor. *The Eighth Asia-Pacific Computer Systems Architecture Conference (ACSAC'2003), Aizu-Wakamatsu City, Japan, ISBN 3-540-20122-X*, 2823:137–150, September 23-26, 2003.

[25] Q. Zhang and G. Theodoropoulos. Modelling SAMIPS: A Synthesisable Asynchronous MIPS Processor. *37th Annual Simulation Symposium (IEEE/ACM/SCS), Part of the Advanced Simulation Technologies Conference (ASTC 2004), Hyatt Regency Crystal City Arlington, VA, ISSN: 1080-241X*, pages 205–212, April 18 - 22, 2004.

[6] `http://www.cs.bham.ac.uk/research/projects/parlard`