Journal of Computations & Modelling, vol.6, no.1, 2016, 107-133
ISSN: 1792-7625 (print), 1792-8850 (online)
Scienpress Ltd, 2016

High-performance Cloud Computing for Symbolic Computation Domain

Idris Al-Skloul Ibrahim[1], Hans-Wolfgang Loidl[2] and Phil Trinder[3]

Abstract

We investigate the potential benefits of high performance cloud computing (HPCC) for novel application domain namely symbolic computation, and specifically for computational algebra as provided by systems like Maple, Mathematica or GAP. HPCCs potentially offer the computational power of a large number of hosts, flexible configuration, and ease of access to a specialized, high-performance configuration. Computational algebra deals with the symbolic manipulation of mathematical problems, often and many algebraic computations are time consuming and therefore promising candidates for parallelism. However, the nature of these computations is fundamentally different to classic high-performance scientific computation: many are highly dynamic, use complex recursive data structures, exhibit high degrees of irregularity, generate large intermediate data structures and primarily use arbitrary precision scalars rather

[1] School of Mathematical and Computer Sciences Heriot-Watt University, Edinburgh, UK. E-mail: i.s.ibrahim@hw.ac.uk
[2] School of Mathematical and Computer Sciences Heriot-Watt University, Edinburgh, UK. E-mail: H.W.Loidl@hw.ac.uk
[3] School of Computer Science Glasgow University, Glasgow, UK. E-mail: Phil.Trinder@glasgow.ac.uk

Article Info: *Received* : September 23, 2015. *Revised* : December 2, 2015.
Published online : March 1, 2016.

than floating point values. We present an initial study on how to use existing HPCC frameworks for computational algebra. We port a C+MPI parallel implementation of a representative computational algebra application, the parallel determinisation of a non-deterministic finite state automaton, to an HPCC and evaluate the performance on several cloud infrastructures. The key issies for the HPCC implementation are the efficient management of massive intermediate data structures, up to 1.1TB, and fast access to the file system in order to store such big data structures.

Keywords: Computational algebra; CALCIUM; C+MPI; Cloud infrastructures; OpenNebula (key words)

1 Introduction

Cloud computing has the potential to simplify access to domain-specific software and to provide computational power far in excess of what is available locally to a user. The ease of access is realised by the concept of delivery of computing as a service rather than a product, which is core to the idea of cloud computing. This service allows the user (individuals and businesses) to use software and hardware that are located and managed (located remotely) by third parties. Accessing the cloud resources and information via cloud computing services is possible from anywhere while a network connection is available. The benefits of this type of service are reduced computing cost (e.g. cloud users do not have to invest in infrastructure, purchase hardware, or buy software licenses, etc.) and reduced complexity of owning and operating shared pool of resources (Computers, networks and data storage space) [1]–[4].

We investigate the potential benefits of high performance cloud computing (HPCC) for novel application domain, namely symbolic computation. The elasticity that comes with the Cloud offers the opportunity to adapt the configuration of a parallel programming platform on demand, in particular based on the size of the input data that needs to be processed. In order to deal with the computational complexity of symbolic computation problems, it is crucial that parallel programming is applied to deal with realistic inputs. Therefore, our focus is on combining established parallel programming technology, in particular using MPI for message passing on a distributed cluster of machines, and port one representative application from this area to modern Cloud infrastructures[1]–[4]. Using applications with significant processing time requirements, which require a reliable high bandwidth, and low latency networks and resources, that utilise supercomputers and computer clusters to address complex computational requirements (science, engineering, and business) are called High Performance Computing (HPC). Nowadays, Cloud computing centers (e.g. KTH PDC2 Cloud) provide such high specification hardware outside the established supercomputing centers, and make this hardware globally available. In particular, Cluster GPU (with thousands of cores), or Cluster Compute services on-demand to speed up and reduce the cost of such complex applications, and researches without large capital investments. Some high profile examples of clouds available for the public are Azure, Amazon Web Services, and Google App Engine [1, 5, 6].

The specific symbolic computation addressed in our work is the determinisation of a finite state automaton. It is representative for this domain because it requires massive compute resources (the sequential runtime for small inputs on a cluster is ca 40 minutes and reported parallel runtimes for large inputs exceed a full day [7], it generates large intermediate data structures (that are managed by a tailored library for file storage of symbolic data structures) and generates highly irregular parallelism (with large variations in the compute time between threads). It is therefore a challenging high-performance computing example, representative for computational mathematics applications.

Our work started with a parallel version of the application, developed for clusters, using C+MPI to expose massive parallelism and using a domain-specific library for fast access to files, that hold the intermediate data structures (Roomy [8], [9]). Our immediate goal was to port this application to cloud infrastructures, using the already existing parallelism. We explored several cloud infrastructures, starting with a private cloud and settling for KTH PDC2 OpenNebula cloud [10]–[12].

In this paper we reflect on the process of porting the determinisation application to several cloud infrastructures, in particular we assess the suitability of high-performance cloud computing for symbolic applications, and we summarise the performance results from running the application on one private and on one public cloud.

The remainder of this paper is organized as follows. In sections 2, 3, 4 and 5 we review the cloud models, high performance computing (HPC), computational algebra and finite state automata in sequence. Section 6 describes the determinisation application. In sections 7 and 8 the proposed work is outlined and described (CALCIUM and CALICIUM component & structure). Results and discussion are presented in section 9. Finally, in section 10, conclusions are drawn and future work is proposed.

2 Cloud Models

Clouds are different from one to another depending on their performance, resources and the underlying software infrastructure. Clouds as shown in Figure 1 could be a private (dedicated to a particular organization), public (managed by a service provider who hosts the cloud infrastructure) or hybrid (composition of two or more clouds).

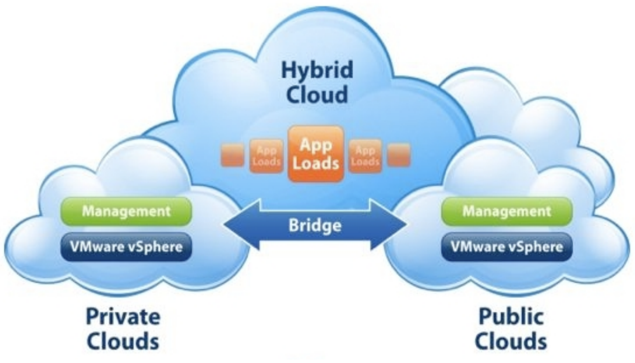

Figure 1: Cloud Models

Despite of the clouds infrastructure, most of them if not all provide the same service types as shown in Figure 2. In the Software as a Service (SaaS) modela third party service provider hosts and makes available a business application to clientusers on a subscription basis (GoogleApps, Webmail's, Salesforce.com etc.).

SaaS (Software-as-a-Service)
PaaS (Platform-as-a-Service)
IaaS (Infrastructure-as-a-Service)

Figure 2: Cloud Services

In the Platform-as-a-Service (PaaS) modelthe vendor (service provider) provides a platform and environment to allow clients (developers) to build applications and services over the internet. The third model is called Infrastructure as a Service (IaaS): On a client per-use basis the service provider owns the cloud resources (equipment) and is responsible for housing, running and maintaining it.

In IaaS, the client rents not only the computing power, but also the infrastructure, including power, cooling, and networking [5,1,13,14].

3 High Performance Computing

High Performance Computing (HPC) aims to deliver the computational power of massively parallel architectures to the programmer. Therefore, it needs to provide a model for parallel programming that can make use of this hardware. Traditionally the hardware is purchased by national institutions, managed by a specialized supercomputing centre and the parallel programming technology that is provided tends to be low level, for example Fortran with an MPI library for explicit message passing between parallel processes. More recently the trend is towards large clusters run by large multi-national institutions rather than dedicated supercomputers , and towards providing higher level language models that simplify the challenging task of parallelising an application. A high profile example of this approach is Google's MapReduce programming model that has been highly successful in parallelising a range of applications and is used in Google's warehouse compute centres. Open source implementations of this concept make it available to a wider class of users and can be applied to local clusters as well [3,2,15].

The next logical step beyond cluster and warehouse computing is to use the computational power of emerging cloud infrastructures to perform parallel programming. While this offers obvious benefits in terms of elasticity and the size of the available computing platform, it also raises new challenges, in particular how to map the existing parallel programming technology to clouds with dynamic allocation of resources. To outline the spectrum of these challenges, we start with a characterisation of High Performance Computing [16,15]:

A. *Compute-Intensive Problems:*

- Supercomputers and Computer clusters are used to solve advanced computation problems.

B. *Complex Data Structure:*

- HPC has the capacity to handle and analyse massive amounts of data at high speeds so that Tasks that can take months using normal computers can be done in days or even minutes.

C. *Pattern-based parallel programming:*

- An active technique to improve the performance of computationally intensive programs (eg. Google's MapReduce).
- Incurs costs of increasing the complexity of the programs, since new issues must be addressed for a concurrent application.
- Parallel programming environments provide a way for users to reap the benefits of concurrent programming without explicit parallelism in the code.

D. *Uses are diverse and examples include:*

- Facial reconstruction modelling, animated graphics, fluid dynamic calculations, nuclear energy research, petroleum exploration, car crash simulations, airflows over aircraft wings, data mining and storage.

4 Computational Algebra

In computer and mathematical science, the domain of symbolic computation deals with the processing of complex, structured data, rather than flat numerical data. Computational algebra deals with the symbolic manipulation of mathematical problems, often to simplify a class of problems into a form that can be more easily solved. It can therefore be seen as a powerful pre-processing step or a general approach to solve entire classes of problems. Algorithms in these systems encode mathematical knowledge that can be used to vastly reduce the complexity of a more concrete problem. Typically, modern computer algebra systems such as Maple or Mathematica [18]are used in engineering disciplines to simplify the mathematical model that has been developed for a concrete application. Because of this generality, computational algebra applications are often very time consuming and therefore promising candidates for parallel computation [17].

One instance of such model simplification is the algorithm for converting a NFA into a DFA [19]. NFAs model complex systems that exhibit non-deterministic behaviour, i.e. that behave differently on the same input at different points in time. General questions about NFAs are difficult to automate, because such questions need to consider all possible behaviours of the NFA. A DFA formulation that exhibits the same behaviour is therefore more desirable becauseoperations or queries on it can be implemented far more efficiently. There has been strong interest in this process of automata "determinisation" over the past 4 decades, building on a rich algorithmic theory in computational algebra and automata theory. Our goal in this work is to bring the knowledge currently encoded in computational algebra algorithms to cloud infrastructures, making it immediately available to a larger class of scientific users and to profit from the computational power available on cloud infrastructures. The nature of these computations is fundamentally different to classic high-performance scientific computation:

A. *Highly dynamic:*

- Generating massive parallelism at intermediate stages).

B. *Structures:*

- Use complex recursive structures rather than flat arrays.
- Generate large intermediate data structures (impacting the granularity of the parallelism).

C. Accuracy:

- Primarily use (arbitrary precision) scalar rather than floating point operations.

D. Speed:

- Computational algebra applications are often very time consuming.

Traditionally, experts in this area, mostly mathematicians, have been concerned with building theories that simplify the structures satisfying a particular set of axioms and proving properties over these structures. Algorithmic aspects often receive much less attention, and significant improvements in performance are possible even on sequential code. Parallelisation of such complex code would bring another boost in performance, but is rarely realised because it requires in depth knowledge of both the underlying mathematical domain and the intricacies of parallel programming. Combining both sequential and parallel optimisations can result in dramatic speedups, as we have recently shown in running a computer algebra application on the Archer super-computer [20]where we improved on the sequential performance by a factor of 350. The results in [20] also present the first ever modern HPC-scale parallelisation of a problem in computational group theory, yielding speedups of up to 548 on 992 cores. These results indicate the potential for performance improvements through HPC for symbolic computation applications, and in this paper we focus on cloud infrastructures as the underlying platform.

5 Finite State Automata

Finite State automata (FSA), also known as finite-state machines (FSM) are commonly used in the design and modelling of complex systems across a range of application areas, and can model a large number of problems in hardware and software engineering. They are also the basis for a lot of work in pattern matching (such as the pattern matching which is performed by Google in searching documents for keywords) and for research into computer applications for speech and language processing [19,21,22].

The behaviour of programs or systems can often be modelled as a set of states, with input triggering a state change and optionally producing an output. Not only Computer systems can be modelled, but also hardware devices across a range of domains (e.g. clocks, vending machines, washing machines, wish washers, microwaves), which perform a computational processing or predetermined sequence of actions depending on a sequence of events (inputs from the external world). These machines can be thought of as a reactive system (work by reacting to input signals). We can say that state machines are a method of modeling systems whose output depends on the entire history of their inputs, and not just on the most recent input [23,24, 25].

Finite State Automata (FSA) characteristics:

- A finite number of states
- A finite number of transitions in between, labelled by char. {E epsilon transitions}
- Used to decide if an input string is a member in some particular set of strings.

Finite state automata (FSA) can be nondeterministic finite automata (NFA), or deterministic finite automata (DFA).

A. NonDeterministic Finite Automata (NFA)

- A finite state machine (e.g. *M1*).
- A given input symbol (with a binary alphabet that determines if the input ends with a 1), the automaton may jump into several possible next states as shown in Figure 3.

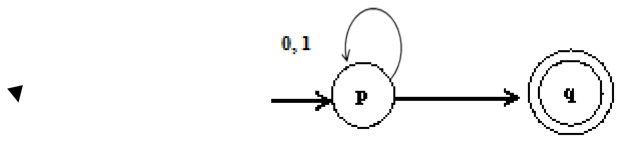

Figure 3: M1 a NonDeterministic Finite Automaton

•The transition relation Δ can be defined by the state transition specified in Table 1.

Table 1: m1 transition table

	0	**1**
P	{p}	{p, q}
Q	{ }	{ }

•Δ (p, 1) has more than one state therefore *M1* is a nondeterministic Finite Automata (NFA).

B. Deterministic Finite Automata (DFA)

- A finite state machine (e.g. M2).
- A given input symbol (with a binary alphabet, which requires that the input contains an even number of 0s), the next possible state is uniquely determined either S1, or S2 as shown in Figure 4.
-

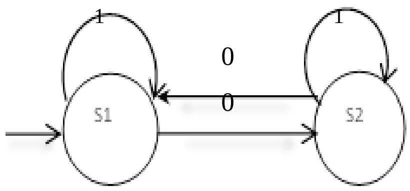

Figure 4: M2 a Deterministic Finite Automaton (DFA)

•Table 2. shows the state transition for M2.

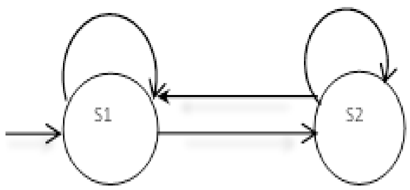

Table 2: m2 transition table

State	**Inputs**	
	0	**1**
S1	S2	S1
S2	S1	S2

- Δ (S1, S2, 0, 1) has only one state therefore *M2* is deterministic Finite Automaton (DFA).

6 Determinisation Application

The determinisation application is the demonstrator application, originally developed for clusters, that we have ported to the cloud infrastructure. This application is a parallel transformation of a non-deterministic finite state automaton (NFA) into a deterministic finite state automaton (DFA) [8]. The structure of the application is to first perform a standard sub-set construction followed by minimisation of the intermediate DFA [26]. One notable feature of the algorithm is that the intermediate DFA can be extremely large. We start from a parallel version of the application, developed for clusters, using C+MPI to expose massive parallelism. The application exhibits a parallel disk-based algorithm (based on a RAM-based parallel algorithm used on supercomputers in the late 1990s and early 2000s [27],[28]) to produce an intermediate DFA with almost two billion states and then continues by producing the corresponding unique minimal DFA with less than 800,000 states. The original computations were carried out on a 29-node computer cluster, each node's processor being a 4-core Intel Xeon CPU 5130 running at 2 GHz with 8 or 16GB RAM, at least 200GB free disk space and ran Red Hat Linux kernel version 2.6.9 [7].

The largest, previous computations of this kind were carried out in 1996 on a

512- processor CM-5 supercomputer to produce an intermediate DFA with 525,000 states. The corresponding minimal DFA states were unreported. In the original paper [7] the parallel disk-based computation is also compared with both a single-threaded and multi-threaded RAM-based implementation using a 16-core 128 GB large shared memory computer. One characteristic feature of the work in [7] is the usage of a scalable disk-based parallel algorithm, using the Roomy library [8,9], to relieve the critical bottleneck in many automata-based computations. This requires the construction of an intermediate non-minimal DFA whose, often very large, size has been the critical limitation on previous RAM-based computations. Thus, researchers may use a departmental cluster or a SAN (storage area network) to produce the desired minimal DFA off-line, and then embed that resulting small DFA in their production application. In the original paper [7] there is a motivating example, demonstrates the production of a two-billion state DFA that is then reduced to a minimal DFA with less than 800,000 states.

Roomy [8,9] is an open-source library for parallel disk- based computing, providing an API for operations with large data structures. The hash table, list and array are the three Roomy data structures that we have used. Operations to these data structures are batched and delayed until the user decides that there are enough operations for processing to be performed efficiently. In doing so, a latency penalty is paid only once for accessing a large chunk of data, and aggregate disk bandwidth is significantly increased. Data that needs to be sent to other compute nodes by Roomy is first buffered in local RAM, in buckets corresponding to each compute node. For a given piece of data, Roomy uses a hash function to determine which compute node should process that data and, hence, in which bucket to buffer that data. Once a given buffer is full, the data it contains is sent over the network to the corresponding compute node (or to the local disk, if the data is to be processed by the local node) [7].

7 Calcium

The primary scientific result of CALCIUM is a cloud-level, parallel transformation of non-deterministic finite state automata (NFA) into deterministic FSA [19,21,22]. The specific problem used as an instance of this problem is an enumeration of possible permutations generated by two (bounded) stacks connected in series. This is a generalization of a problem introduced by Knuth in The Art of Computer Programming (Volume 1). The parallel determinisation of FSA is of general relevance because FSA are widely used in computer science, e.g. in natural language processing, speech recognition, computer aided verification via model checking and modelling of complex systems, for example in UML. One specific example is that any UML 2 compiler must convert ND FSA into a deterministic FSA.

If an FSA can make several transitions based on the same input its behaviour is non-deterministic. A non-deterministic FSA is a convenient way of modelling inherently non-deterministic behaviour, but it is problematic from an implementation point of view, because all possible sequences of transitions have to be considered. Therefore, determinising a non-deterministic FSA can be seen as a pre-processing step, with benefits for all applications that use an FSA representation of a system.

8 Calcium Components & Structure

Our main development was done directly on OpenNebula, building on the existing C+MPI implementation of the determinisation application for clusters. We selected OpenNebula as a platform, to profit from the performance gains through para-virtualisation of the Linux-based application. As described in Table 3, our application was designed to be high performance, rather than high-throughput, like most cloud applications. By using C+MPI as an implementation platform it provides a scalable scenario of parallel execution. Thus, the majority of our work focused on scalability improvements for our application and corresponding measurements directly on OpenNebula.

Table 3: Infrastructure components

	Components			**Platform**		**Infrastructures**
	Roomy	OpenNebula	COMPSs	Private Cloud	KTH Cloud	Linux
Ver.	0.9.1	3.0	1.5	4 Nodes	PDC2	Ubuntu 12.04

Profiting from our experience with cloud infrastructures, we decided to first develop a technology exemplar that exhibits the characteristics of parallel symbolic computation within a small program (Parallel Totient Range, sumEuler). In early phases of the project we have used the same technology exemplar on our private cloud, and achieved near linear speedups on 4 nodes. On the KTH cloud we were given up to 3 VM instances and running our technology exemplar achieved a speedup of 2.9 on 3 VMs (Microsoft VitualBox4.1), indicating that the platform is a suitable basis for our high-performance, symbolic application. After validating the software stack, we ported our main determinisation application to an OpenNebula-based Cloud infrastructure. The initial port was done on our own 3-node private cloud, facilitating the development phase through unrestricted access to an infrastructure under our full control. The configuration of our private cloud is depicted below as shown in Figure 5.

The initial port required some changes to the configuration of the Roomy library, to match the characteristics of a cloud versus a cluster and to improve performance of disk access. After these changes we managed to achieve speedups of up to 2.1 on a 4 node private cloud configuration, reducing execution time from 43 minutes 2 seconds to 21minutes. However, scalability of the results was limited due to the limited resources available on our private cloud, which was set up as development but not as a deployment platform. Therefore, our main goal in moving to the KTH cloud infrastructure was to run our parallel application on larger inputs and perform scalability measurements on a larger cloud configuration.

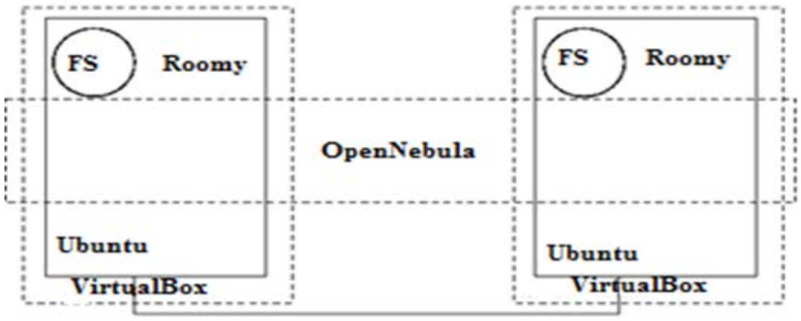

Figure 5: CALCIUM private cloud structure

However, when accessing the KTH cloud infrastructure we were given only a very restricted configuration of 1 GB main memory, 4 GB total disk space and 3 VM instances. As a result, we were only able to run the two smallest input sets on this infrastructure. Bearing these restrictions in mind, we achieved excellent results on the KTH cloud. For the input set A2, a representation of a TokenPassing network as described above, we achieved speedups of up to 6.6 on a 3 node configuration, reducing execution time from 39min48sec to 6min. This super-linear speedup is most likely due to more efficient handling of smaller files by the Roomy library when distributing the execution over several nodes. It can therefore be seen as a cache-effect, specialized to symbolic computation and the underlying library taking the role of the main memory.

9 Results And Discussion

For validation of the determinisation application on the KTH cloud, we ran the small input files (A1 and A2) through the application, and did a binary comparison of the output files with those produced on our private cloud, to ensure that the application produces the correct result.

Our primary input data is an NFA that is a representation of a TokenPassing network [29], elaborated on in[7]. TokenPassing networks are in important modelling tool for studying the behaviour of computer networks. In the context of symbolic computation, this application is of interest because it can be seen as an instance of a permutation group, describing which permutations to a sequence of values can be produced when passing the values through a series of stacks, allowing for limited re-ordering of the elements by each stack. The non-determinism in this application comes from the choice between passing an incoming element to the next phase or pushing it onto the local stack. The origin of studying these kinds of applications goes back to a problem introduced by Knuth in “The Art of Computer Programming” (Volume 1)[30], and has been studied in several forms in the areas of theoretical computer science and computational group theory. In particular, it has been used as a case study for applying automata-based algorithms in the GAP[31,32] system for computational group theory.

The A2 input data used in the measurements represents such an NFA with 3541 states and 8 transitions. The resulting, minimised deterministic finite state automata contains 1236 states. Although this output size is smaller than the input size, the non-minimised, intermediate deterministic finite state automaton, which is the result of a standard sub-set construction[26] in the first phase of the algorithm, is much larger, containing 646304 states in total, and the resource requirements are largely determined by this intermediate data structure. Such huge intermediate data structures are one typical characteristic of symbolic applications, and can also be observed in other algorithms such as Knuth-Bendix or Groebner Bases.

The physical setup of our experiments involves 3 VM instances both on our private cloud and on the public KTH cloud that was used as deployment platform in this project. We initially set up a small private cloud as main development platform for the code in order to retain complete control over the hardware and software architecture, after some initial experiences from using externally hosted clouds. This decision proved to be important to speed-up the development of the core application. The size of the deployment platform was prescribed by the hoster of the external cloud and is elaborated on below. Key components in our software architecture are the aforementioned Roomy library [8,9] for efficient disk-storage of large symbolic data structures, MPICH2 [33, 34] as a highly tuned implementation of the MPI standard for message passing on distributed memory architectures, and the GAP [31,32]system for computational group theory (mainly to validate the data produced by the parallel application). Our development built on the cluster-based implementation of the determinisation algorithm described in [7]. Our focus was on the port the application to cloud infrastructures and to test performance and scalability on these. In the bigger picture, we want to assess the suitability of HPC clouds for running parallel symbolic computations.

Porting a substantial symbolic computation application to several state-of-the-art cloud infrastructures required us to deal with challenges specific to the domain. The determinisation application requires massive compute resources (the sequential runtime for small inputs is ca 40 min and reported parallel runtimes for large inputs exceed a full day), generates large intermediate data structures (that are managed by a tailored library for file storage of symbolic data structures) and generates highly irregular parallelism (with large variations in the in compute time between threads). Specifically, the original, cluster-based algorithm has a peak disk consumption of 1.1TB and a sequential runtime of 1 day 12 hours [7]. For these kinds of disk-intensive applications it is important to have efficient disk-access. Our preferred setup of OpenNebula enables us to make use of para-virtualisation, by running a Linux-based application on a Linux host system, thereby avoiding overhead that would come with an emulation-based, full virtualisation. This is likely to be a common issue for data-intensive applications in this domain, and the determinsation application is an extreme case for such an application (other examples in this class include a fast, parallel multiplication operation on large permutations[35] and a parallel library for the management of large binary decision diagrams [36]).

Irregularity of the parallelism is another major challenge for symbolic applications. The number of live threads may vary largely across the execution and so do the compute times of the generated threads, with variations up to five orders of magnitude. In a cloud context and through virtualisaton technology, the system has the additional flexibility of assigning VMs to physical machines. Therefore, VM-based load balancing mechanisms, that monitors the activity of a VM and, upon excessive load, migrate an entire VM to a different physical machine, are particularly promising for this kind of applications. It would devolve the challenging task of thread-based load-balancing inside the computer algebra engine to the higher VM level, employing generic techniques at that level.

The table below (Table 4) summarizes our main performance results, comparing the runtimes on our private cloud with those on the KTH cloud.

Table 4: Infrastructure components

Cloud	**Input**	**NFA size**	**DFA size**	**Runtime**		
				1 Node	***2 Node***	***3 Node***
Private	A2	3541	1236	43m2s	25m32s	21m
KTH	A2	3541	1236	39m48s	10m54s	6m

The runtimes reflect end-to-end execution times for the entire application and thus include time spent on system initialisation and data distribution. Using a single-program, multi-data (SPMD) parallel execution model, as prescribed by the MPI message-passing standard, the binaries are uploaded onto all nodes beforehand. This implies a limitation to the usage of this code on cloud infrastructure, since it cannot make use of joining nodes during the execution.

As mentioned above, our main test case was input A2, with a 3541-state, non-deterministic finite state automaton as input and a 1236-state deterministic finite state automaton as output, generating an intermediate automaton (before minimisation) with 646304 states. Running the determinisation on this input used 73 out of 1024 MB of main memory and 2.4 out of 4.0 GB of disk space available on our private cloud, used for the initial measurements.

We achieve good performance results on our private cloud already: a speedup of 2.1 on a 3 node configuration, and an almost linear speedup on 2 nodes. We then deployed this application on KTH cloud infrastructure, which is OpenNebula-based. In this setting, the resources are severely constrained with each application having an allowance of 1GB main memory, 4 GB total disk space and a maximum of 3 VM instances. Since our main goal was to provide a proof-of-concept study, we went ahead with this fairly limited configuration. Considering these tight resource constraints, we achieved excellent results on the KTH cloud. For the input set A2, a representation of a TokenPassing network as described above, we achieved speedups of up to 6.6 on a 3 node configuration, reducing execution time from 39 minutes 48 seconds to 6 minutes. Closer examination revealed that this super-linear speedup is due to the more efficient handling of smaller files by the Roomy library when distributing the execution over several nodes rather than using a very large file in the single node execution. It can therefore be seen as a “cache-effect” specialised to symbolic computation and the underlying disk-based library taking the role of the main memory.

While more realistic input sets would require substantially higher resources, the above results demonstrate that good speedups can be achieved for this challenging class of applications. The next data sets for this problem would require an estimated 2-8 GB of main memory, and should be executed on 16-64 VMs of a cloud infrastructure such as OpenNebula. At least 10 GB of disk space will be needed per node. For the larger input files this disk space will have to increase to about 150 GB per node, considering that reported intermediate sizes for this kind of input are in the range of 1.1TB.

One important outcome of this proof-of-concept study is the potential availability of highly specialised mathematical code through cloud infrastructures, based on a SaaS model. This is a drastic change from the model used so far, where licenses for entire computer algebra systems need to be purchased, and execution is restricted to a local cluster of machines to perform the computations. This set-up lacks this would have benefits for both the users of computer algebra systems and the developers of algorithms within these systems. The users would profit from the flexibility (or elasticity) in allocating additional hardware to a particularly large computation (depending on the input data set), which can be addressed using cloud infrastructures. The developers, typically mathematicians rather than computer scientists, don't have to deal with installation and setup issues of the system, which can be difficult for large, parallel systems, and can rather focus on the algorithmic changes required to enable parallel execution. In this sense, we believe that cloud infrastructures can provide a substantial benefit to the computer algebra community.

10 Conclusion and Future Work

In this project we have undertaken a proof of concept study for high performance computational algebra on cloud infrastructures. We have been able to demonstrate the feasibility of the approach by producing a cloud-enabled version of a parallel automata determinisation application. In particular, we make use of the large number of processors and the distributed disk-space available on these infrastructures to deal with the high computational demand and the large intermediate data structures in this application. We have measured the performance on one private and one public cloud. Our results show, that we can achieve excellent speedups even on severely resource-constrained infrastructures. In the best case we achieved a super-linear speedup of 6.6 on a 3 node cloud, due to the more efficient handling of smaller files in the distributed configuration by the underlying Roomy library.

A significant portion of the development time was expended on basic infrastructure access, first to private cloud, later to the KTH cloud. Using a private cloud initially gave us full control over the infrastructure, which proved important to develop and test the behaviour of the application initially. We made sure to have a self-contained software infrastructure around our application, so that we only had to make minor adjustments to different cloud infrastructures once the initial port was done. When deploying the application on the external cloud we had to deal with administrative issues of getting access to the infrastructure itself. We conclude that good, responsive support by the cloud service provider is crucial in order to make it an attractive platform for high performance computing, which has stronger demands on tuning and configuring the software infrastructure. One concrete limitation in our case was the constrained main memory provided for each node. This is a general concern, across providers, who often offer “small”, “medium” and “big” configurations on the cloud; but in most cases the ratio of processors to main memory and to disk space is constant. Thus, our applications, that are often very memory intensive, cannot exploit the full number of processors, without exhausting the available main memory. This lack of elasticity in the per-node cloud configurations means, that often the nodes are under-populated and use only a few of the available processors.

Overall, however, our experiences with cloud infrastructures have been positive. We conclude that they provide significant benefits for both the users and the developers of computer algebra algorithms. The user get immediate access to parallel implementations running on clouds, using a SaaS model, and don't have to install specialised software on a local cluster with most likely more limited processing power. For the developers this setup avoids having to deal with low-level configuration issues, and allows them to focus on algorithmic improvements, often based on more abstract, mathematical results. We therefore believe that high performance cloud computing has a very strong, so far unused, potential to speedup existing applications, and to apply them to problems that have so far been out of reach.

One obvious line of future work is to extend the measurements to larger inputs running on larger cloud configurations. For practical reasons this should be done on OpenNebula infrastructures. Another interesting direction of future work would be to rephrase this application in terms of an Orbit pattern. The Orbit pattern is a computational pattern, repeatedly occurring in symbolic computation that we have identified and studied in earlier work [37], producing a parallel implementation of this pattern. Making the determinisation application an instance of this pattern would profit from the tuning that we have already performed on this pattern.

Acknowledgements. We would like to thank the VenusC consortium and the EU funding agency that funded the CALCIUM project as a pilot project in the Research Infrastructure project VenusC (Virtual Multi-Disciplinary Environments Using Cloud Infrastructures) (RI-261565). This works was also partially funded by EPSRC under grant EP/G055181/1 “HPC-GAP: High Performance Computational Algebra and Discrete Mathematics”. We also would like to thank KTH for giving us access to their cloud infrastructure as part of this project, and Microsoft Research for giving us initial access to MS Azure in the early stages of the project. We are indebted to Vlad Slavici, Daniel Kunkle, Gene Cooperman, and Stephen

Linton for providing the source code for the cluster-based determinisation algorithm, and for useful discussions on how to port this code and to develop it further. We would also like to thank the GAP community, in particular Alexander Konovalov, for providing expertise on the GAP systems and the GAP algorithms.

References

[1] Q. Zhang, L. Cheng, and R. Boutaba, Cloud computing: State-of-the-art and research challenges, *J. Internet Serv. Appl.*, **1**, (2010), 7–18.

[2] V. Mauch, M. Kunze, and M. Hillenbrand, High performance cloud computing, *Futur. Gener. Comput. Syst.*, **29**, (2013), 1408–1416.

[3] J. Ekanayake, X. Qiu, T. Gunarathne, S. Beason, and G. Fox, High Performance Parallel Computing with Cloud and Cloud Technologies, *Technology*, **1**, (2010), 1–39.

[4] Y. Peng and F. Wang, Cloud computing model based on MPI and OpenMP, in ICCET 2010 - 2010 International Conference on Computer Engineering and Technology, *Proceedings*, **7**, (2010).

[5] N. Antonopoulos and L. Gillam, Cloud Computing: Principles, Systems and Applications, **54**, (2010), 379.

[6] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, A survey of mobile cloud computing: Architecture, applications, and approaches, *Wirel. Commun. Mob. Comput.*, **13**, (2013), 1587–1611.

[7] V. Slavici, Finding the Minimal DFA of Very Large Finite State Automata with an Application to Token Passing Networks, pp. 1–14.

[8] H. Ave and D. Kunkle, *Roomy : A System for Space Limited Computations,* pp. 22–25, 2010.

[9] *Roomy: A C/C++ Library for Parallel Disk-based Computation.* [Online]. Available: http://roomy.sourceforge.net/. [Accessed: 17-Jul-2014].

[10] D. Miloji, I. M. Llorente, and R. S. Montero, OpenNebula: A cloud management tool, *IEEE Internet Computing*, **15**, (2011), 11–14.

[11] X. Wen, G. Gu, Q. Li, Y. Gao, and X. Zhang, Comparison of open-source cloud management platforms: OpenStack and OpenNebula, in *Proceedings - 2012 9th International Conference on Fuzzy Systems and Knowledge Discovery,* FSKD 2012, (2012), 2457–2461.

[12] P. Sempolinski and D. Thain, A comparison and critique of Eucalyptus, OpenNebula and Nimbus, in *Proceedings - 2nd IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2010,* (2010), 417–426.

[13] L. Qian, Z. Luo, Y. Du, and L. Guo, Cloud computing: An overview, in *Lecture Notes in Computer Science* (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), **5931** LNCS, (2009), 626–631.

[14] W.T. Tsai, X. Sun, and J. Balasooriya, Service-Oriented Cloud Computing Architecture, *Seventh Int. Conf. Inf. Technol. New Gener.*, (2010), 684–689.

[15] V. Eijkhout, E. Chow, and R. van de Geijn, Introduction to High-Performance Scientific Computing, *math-cs.gordon.edu*, **5**, (1997), 73–74.

[16] C. Vecchiola, S. Pandey, and R. Buyya, High-performance cloud computing: A view of scientific applications, in *I-SPAN 2009 - The 10th International Symposium on Pervasive Systems, Algorithms, and Networks,* (2009), 4–16.

[17] R. Liska, L. Drska, J. Limpouch, M. Sinor, M. Wester, and F. Winkler, *COMPUTER ALGEBRA, Algorithms, Systems and Applications,* 1999.

[18] I. Shingareva and C. Lizrraga-Celaya, *Maple and mathematica*: A problem solving approach for mathematics, 2007, pp. 1–234.

[19] Mark V. Lawson, *Finite Automata*, 1st ed. Chapman and Hall/CRC, p. 326, 2003.

[20] P. T. Patrick Maier, Daria Livesey, Hans-Wolfgang Loidl, High-Performance Computer Algebra --- A Parallel Hecke Algebra Case Study, in EuroPar'14: *Parallel Processing*.

[21] M. Mukund, Finite-state automata on infinite inputs, TCS, **96**, (1996), 2.

[22] P. E. Black, Finite state machine, *Dict. Algorithms Data Struct.*, **2007**, (2008), 1–7.

[23] R. Alur and D. L. Dill, Automata for modeling real-time systems, in *Proceedings of the seventeenth international colloquium on Automata, languages and programming*, (1990), 322–335.

[24] C. L. Lucchesi and T. Kowaltowski, Applications of finite automata representing large vocabularies, *Software\emdash Pr. Exp.*, **23**, (1993), 15–30.

[25] M. O. Rabin and D. Scott, Finite Automata and Their Decision Problems, *IBM J. Res. Dev.*, **3**, (1959), 114–125.

[26] D. Kozen, *Automata and Computability*, Springer Verlag, 1997.

[27] J. F. J´. and K. W. Ryu., An efficient parallel algorithm for the single function coarsest partition problem, *Proc. fifth Annu. ACM Symp. Parallel algorithms Archit.*, (1993), 230–239.

[28] B. Ravikumar and X. Xiong. parallel algorithm for minimization of finite automata. Parallel, A parallel algorithm for minimization of finite automata, *Parallel Process. Symp. Int.*, **187**, (1996).

[29] D.T. M.D. Atkinson, M. J. Livesey, Permutations generated by token passing in graphs, in *Theor. Comput. Sci.*, **178**, 103–118.

[30] D. E. Knuth, *The Art Of Computer Programming*, 1st ed., Addison-Wesley Professional, p. 1872, 2005.

[31] EPSRC, *The HPC-GAP Project*, Heriot-Watt University. [Online]. Available: http://gow.epsrc.ac.uk/NGBOViewGrant.aspx?GrantRef=EP/G05553X/1. [Accessed: 18-Jul-2014].

[32] *GAP System for Computational Discrete Algebra.*, [Online].

Available: http://www.gap-system.org/. [Accessed: 17-Jul-2014].

[33] *MPICH2 home page.* [Online].
Available: http://phase.hpcc.jp/mirrors/mpi/mpich2/.[Accessed: 18-Jul-2014].

[34] *MPICH | High-Performance Portable MPI.* [Online]. Available: http://www.mpich.org/. [Accessed: 18-Jul-2014].

[35] D. K. and G. C. Vlad Slavici, Xin Dong, Fast Multiplication of Large Permutations for Disk, Flash Memory and RAM, in the *International Symposium on Symbolic and Algebraic Computation* (ISSAC '10), (2010), 355–362.

[36] G. C. Daniel Kunkle, Vlad Slavici, Parallel Disk-Based Computation for Large, Monolithic Binary Decision Diagrams, in the *International Workshop on Parallel Symbolic Computation* (PASCO '10), (2010), 63–72.

[37] Hans-Wolfgang Loidl. Vladimir Janjic, Christopher Brown, Max Neunhoffer, Kevin Hammond, Steve Linton, Space Exploration using Parallel Orbits: a Study in Parallel Symbolic Computing, in ParCo2013: *Parallel Computing: Accelerating Computational Science and Engineering* (CSE), (2013), 225 – 232.