SymGrid-Par: a System for Parallel Symbolic Computation on Large-scale Distributed Systems

The SCIEnce Team

http://www.symbolic-computing.org/

September 22, 2012

Today's Challenges for Symbolic Computation

In the SCIEnce[1] project we addressed the following challenges:

- transparent access to complex, mathematical software, through **Grid or Cloud Services**;
- uniform data exchange between independent systems through **OpenMath data format and SCSCP protocol**;
- **exploitation of modern parallel hardware** through **high-level orchestration of parallelism**.

[1] http://www.symbolic-computing.org/

Today's Challenges for Symbolic Computation

In the SCIEnce[1] project we addressed the following challenges:

- transparent access to complex, mathematical software, through **Grid or Cloud Services**;
- uniform data exchange between independent systems through **OpenMath data format and SCSCP protocol**;
- **exploitation of modern parallel hardware** through **high-level orchestration of parallelism**.

Therefore, two main focus areas are

- system interoperability (SCSCP),
- parallel computation (SymGrid-Par).

[1] http://www.symbolic-computing.org/

Parallelism in Symbolic Applications

The characteristics of parallel symbolic applications are radically different from numeric applications:

- parallelism is highly irregular (computation sizes varying up to 10^4)
- parallelism is highly dynamic (new threads generated throughout with huge differences in thread residency)
- computations over complex data structures (rather than flat arrays)
- base operations are typically integer, not floating point, operations

Parallelism in Symbolic Applications

The characteristics of parallel symbolic applications are radically different from numeric applications:

- parallelism is highly irregular (computation sizes varying up to 10^4)
- parallelism is highly dynamic (new threads generated throughout with huge differences in thread residency)
- computations over complex data structures (rather than flat arrays)
- base operations are typically integer, not floating point, operations

$\Longrightarrow$ **dynamic, adaptive control of parallelism is needed**

Parallel Programming: Past and Future

Characteristics of parallel programming in the **future**:

- on off-the-shelf hardware such as **multi-core machines**
- architecture-independent
- high-level control of the code
- done by domain experts
- all areas of computing, especially symbolic computation

Parallel Programming: Past and Future

Characteristics of parallel programming in the **future**:

- on off-the-shelf hardware such as **multi-core machines**
- architecture-independent
- high-level control of the code
- done by domain experts
- all areas of computing, especially symbolic computation

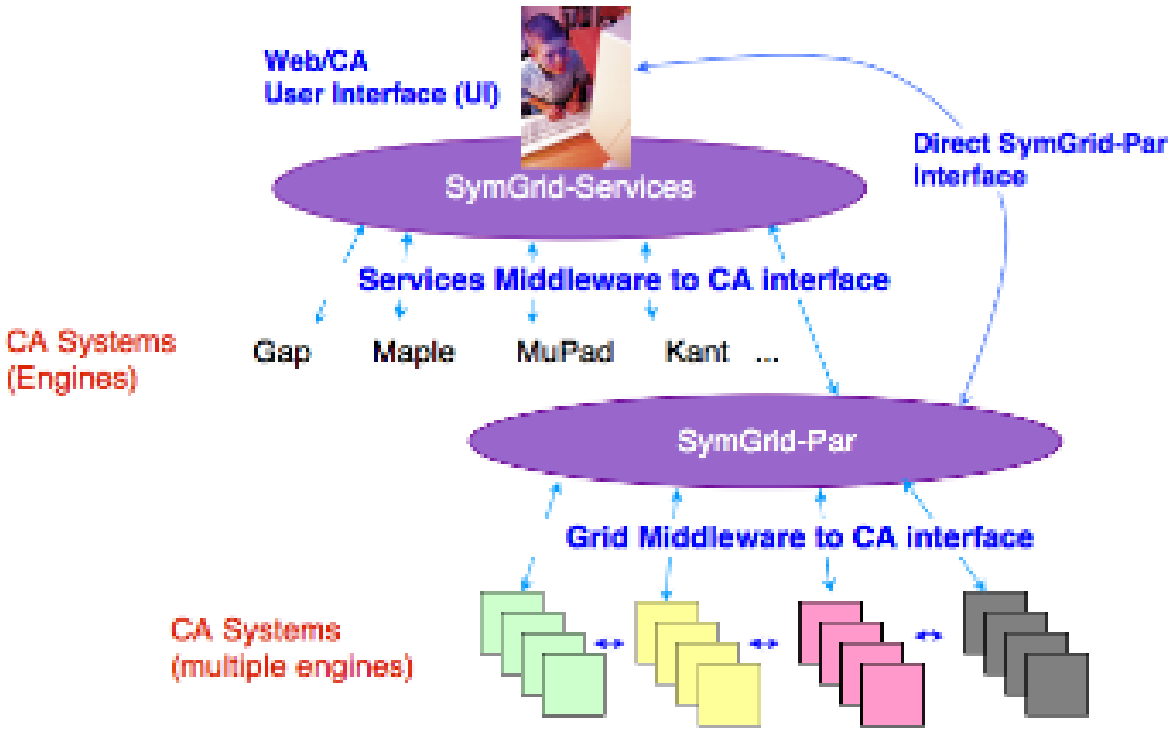
$\Longrightarrow$ **Desktop parallelism**

Domain-specific Skeletons

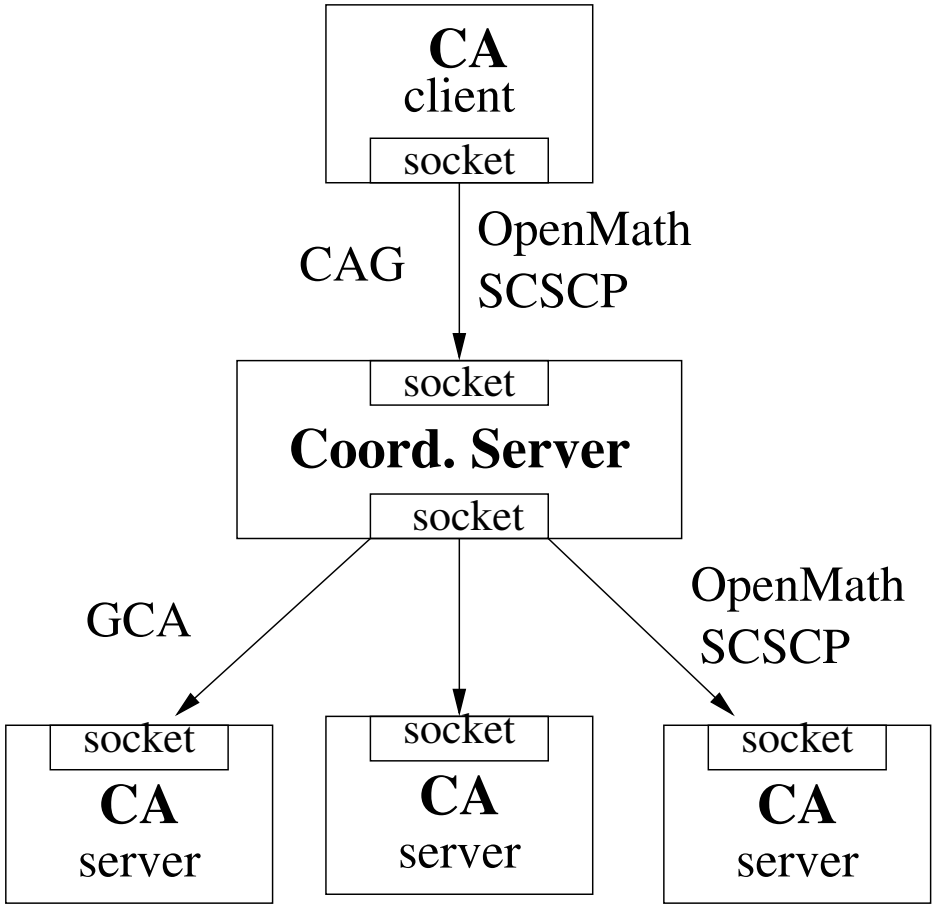
We raise the level of abstraction for parallel computation by using **domain-specific skeletons**:

- Skeletons encode commonly occuring patterns of parallel computation.
- Such patterns are increasingly used in main-stream parallel programming (Google MapReduce).
- We implemented a range of patterns specific to symbolic computation:
  - Orbit pattern
  - Multiple-homomorphic images pattern (using the modular method of problem solving)
  - Critical-pair Completion pattern (tested with Gröbner Bases)

SymGrid-Par Design



SymGrid-Par Infrastructure



Components of the Architecture

- Either command-line client or a client of a computer algebra system.
- High-level **Coordination Server** handling parallelism
  - uses parallel **Haskell** as high-level programming model
  - parallelism is mainly specified and coordinated here
  - automatic resource management on this level
- Any **SCSCP-based** computer algebra server:
  - tested with GAP and a Haskell-side server
  - servers can themselves use parallelism, but
  - no direct communication between servers

Key Technologies of SymGrid-Par

- Clear interfaces:
  - common data format: OpenMath
  - communication protocol: SCSCP
  - form of connection: sockets

Key Technologies of SymGrid-Par

- Clear interfaces:
  - common data format: OpenMath
  - communication protocol: SCSCP
  - form of connection: sockets
- A **Haskell-side Coordination Server** is
  - a server that implements a collection of (parallel) CA functions
  - a server that provides parameterisable skeletons of parallel computation
  - a client that calls CAs to perform the heavy computation.

Key Technologies of SymGrid-Par

- Clear interfaces:
  - common data format: OpenMath
  - communication protocol: SCSCP
  - form of connection: sockets
- A **Haskell-side Coordination Server** is
  - a server that implements a collection of (parallel) CA functions
  - a server that provides parameterisable skeletons of parallel computation
  - a client that calls CAs to perform the heavy computation.
- A minimal **interactive Haskell-side Client (cash)**
  - uses Haskell interpreter to provide direct access to SCSCP interface
  - useful for prototyping (sequential) code

High-level Parallelism

Modern parallel architectures are increasingly heterogeneous and hierarchical.
Low-level control (eg. C+MPI) becomes infeasible on such machines.
Modern parallel languages adopt an approach of high-level control of parallelism.

Modern Parallel Languages

- **X10**: Java-like with asynchronous, anonymous threads, no explicit communication, partitioned global address space (virtual shared memory), dependent types to express location constraints support for generic programming;
- **Fortress**: **"type-safe Fortran"** an object-oriented language with implicit parallelism (mostly through libraries), platform-independent, shared global address space (virtual shared memory); picks up concepts from Haskell, ML, Scala
- **Chapel**: **"provides a higher level of expression"** through anonymous threads, program abstractions to control parallelism, "separation between algorithmic expression and implementation", language constructs to control data locality supports object-oriented concepts and generic programming

Software Layers in SymGrid-Par

	SCSCP interface
Access Layer:	Grid
Service Layer:	Grid Service
Application Layer:	Skeletons
Coordination Layer:	parallel Haskell (Eden)
Communication Layer:	SCSCP
Data Layer:	OpenMath
Connection Layer:	Sockets

Parallel Orchestration

SymGrid-Par Infrastructure:

- Language: Eden 6.12.2
  - Stable implementation of a parallel Haskell dialect
  - Recent improvements in the management of parallelism
- Abstractions for parallelism: Algorithmic Skeletons
  - Capture common patterns of parallel computation
  - Building on ample experience in parallelising code
  - Extensibe for particular application domain of parallel symbolic computation
- Hardware:
  - Runs on any parallel hardware supporting PVM or MPI
  - Very good match for networks and clusters
  - Also good results on multi-cores

Configuring SymGrid-Par

The configuration of SymGrid-Par follows the 3-level system architecture:

- several GAP servers performing the computation:

  `sgp_admin.sh launch ~/sgprc`

- a Coordination Server, using Eden for parallelism;

  `sgp_admin.sh start ~/sgprc`

- either a command-line or a GAP shell as client;

  `testClient ...`

The entries in the `~/sgprc` file are of the form

`<hostname> <port> <CAsystem>`

Example

SymGrid-Par provides a simple command-line client that can be used to start (parallel) computations, without having to install a full-fledged computer algebra system.

```
testClient 12321 SumEuler 8000 2000

starting up client, opening port 12321
Calling SumEuler with arguments [8000,2000]
Launching parallel sumEulerParSCSCP 8000 2000, coordinated by th
Result: 19455782
```

Note: the number of processors is only defined when starting the Coordination Server.

Use cases of writing parallel code

With **createProcess** parallel processes are created **explicitly**.
Coordination between the processes is **implicit**.

```
sumEuler :: Int -> Int -> IO Int
sumEuler n c = do
 let ranges = [[i*c+1, (i+1)*c] | i <- [0..(num c n)-1]]
 let xs' = map (createProcess (process (\ns ->
             unsafePerformIO (sumEulerRange ns)))) ranges
           `using` whnfspine
 let xs :: [Int]
     xs =  map deLift xs'
 return (sum xs)

sumEulerRange :: [Int] -> IO Int
sumEulerRange  = return .
  fromOM . (callSCSCP WS_SumEulerRange) . (map toOM)
```

Use cases of writing parallel code

The main worker function is a service, `WS_SumEulerRange`, provided by a GAP SCSCP server:

```
SumEulerRange:=function(n,m)
local result, x;
result:=Sum( [ n..m ], x -> euler(x) );
return result; end;
```

On the GAP server side the service is installed like this:

```
InstallSCSCPprocedure("WS_SumEulerRange", SumEulerRange,
                      "see sumEuler.g", 1, 2);
```

Using a parallel service from inside GAP

This parallel implementation of `sumEuler` is exported as an SCSCP service by the Coordination Server.
It can be called from inside a GAP client like this:

```
EvaluateBySCSCP("CS_SumEuler",
                [ 8000, 2000 ], "localhost", 12321);
```

Using a parallel skeleton

A very common skeleton is `parMapFold f g z zs`: it applies a function `f` to all elements of the list `zs` and then uses the binary operator `g` to combine the individual results to an overall result (`z` is the neutral element).

```
zs:=[87,88,89];
ParMapFold("WS_Phi", "WS_Plus", 0, zs);
```

Generating New Knowledge: Summatory Liouville

- Search for positive values of the Summatory Liouville function
- The *Liouville* function $\lambda(n) = (-1)^{r(n)}$, where $r(n)$ is the number of prime factors of n.
- The *summatory Liouville's function*, $L(x)$, is the sum of values of *Liouville*(n) for all n from $[1..x]$.

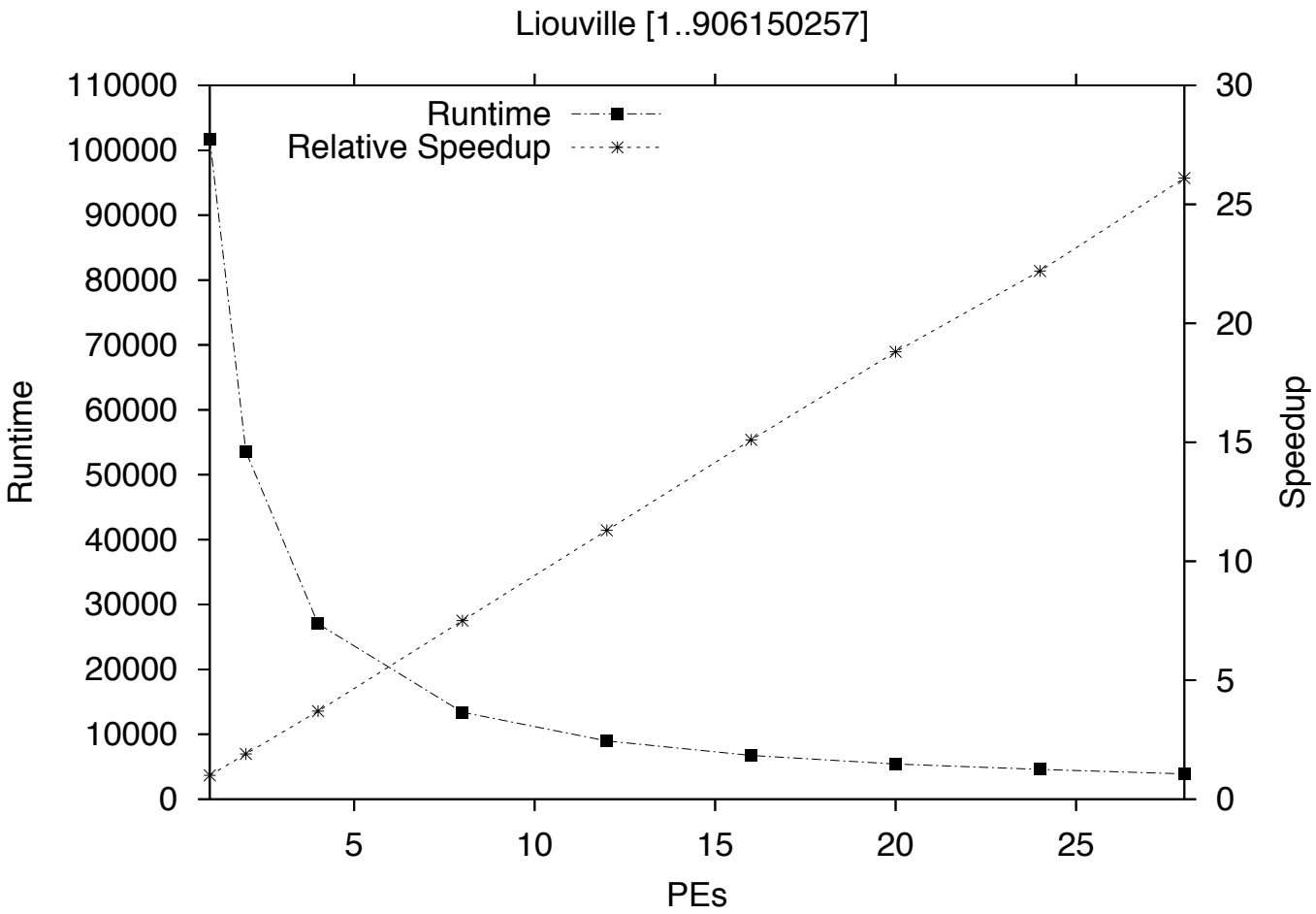
Summatory Liouville

```
L :: Integer -> Integer -> Int -> [(Integer,Integer)]
L lower upper c = sumL (myMakeList c lower upper)

sumL :: [(Integer,Integer)] -> [(Integer,Integer)]
sumL mylist = mySum ((masterSlaves liouville) mylist)

liouville :: (Integer,Integer) ->
             [((Integer, Integer),(Integer,Integer))]
liouville  (lower,upper) =
    let
    l  = map gapObject2Integer (gapEvalN "gapLiouville"
     [integer2GapObject lower,integer2GapObject upper])
    in
    ((head l, last l), (lower,upper))
```

Summatory Liouville [1 ..906150257] Speedups



Performance Portability

- SymGrid-Par designed for distributed memory architectures
- Many CA problems have large thread granularity and hence perform well on multicores [DAMP09]

8-core Summatory Liouville $[1 \ldots 25 \times 10^6)]$

No PEs	Rtime	Spdup	CPU Utilis.
1	526s	1	92.8%
2	264s	1.9	89.6%
3	178s	2.9	93.1%
4	132s	3.9	92.0%
5	106s	4.9	90.7%
6	89s	5.9	90.7%
7	76s	6.9	89.4%
8	68s	7.7	88.9%

8-core smallGroup [1...350]

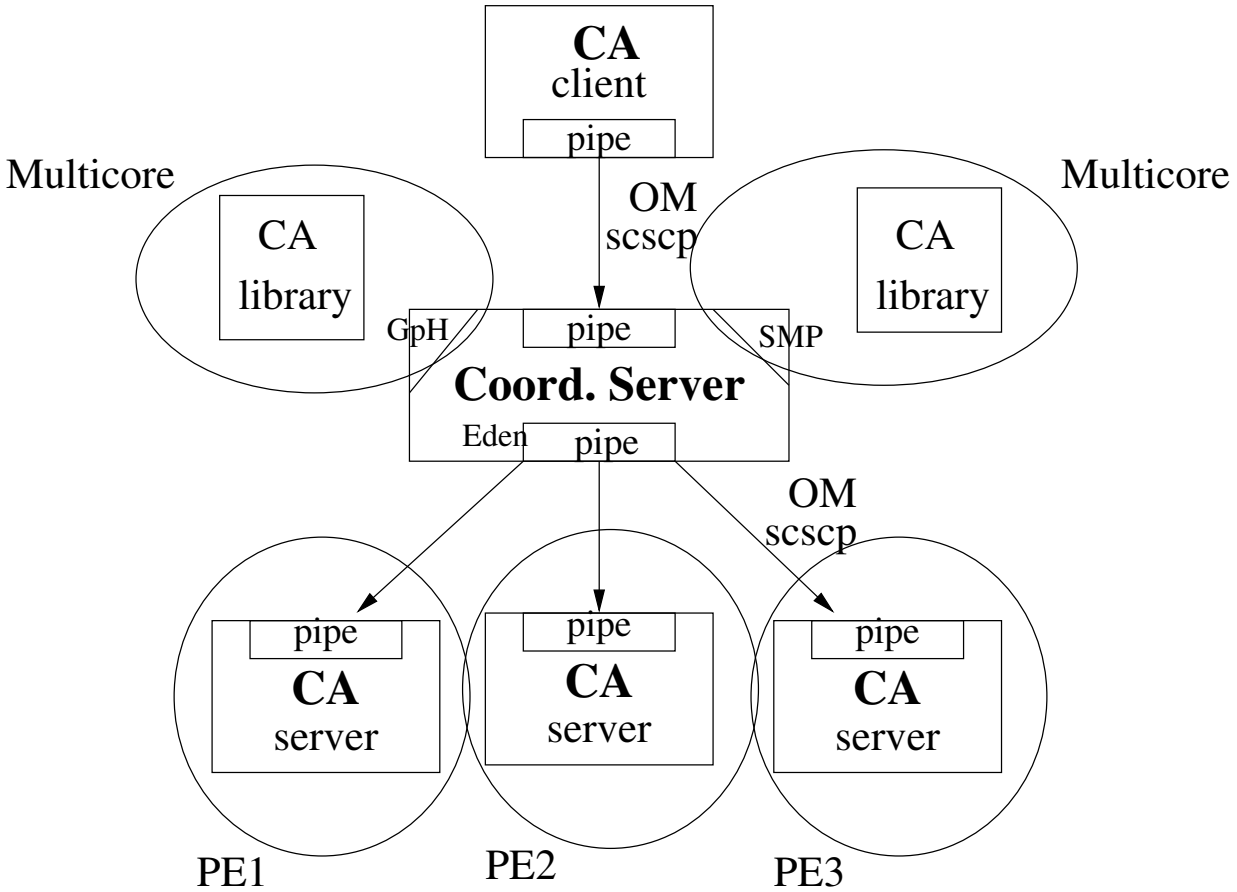
No PEs	Rtime	Spdup	CPU Utilis.
1	480s	1	96.0%
2	246s	1.9	96.0%
3	165s	2.9	98.6%
4	125s	3.8	98.0%
5	104s	4.6	99.2%
6	91s	5.2	98,7%
7	82s	5.8	98.3%
8	76s	6.3	97.0%

SymGrid-Par Prototype Performance Summary

For compute-bound problems:

- Performance of generic SymGrid-Par is comparable with bespoke parallel CAs
- The SymGrid-Par skeletons deliver speedups even for problems exhibiting a high degree, and multiple levels, of irregularity
- We have generated new CA results using the SymGrid-Par skeletons
- The SymGrid-Par skeletons provide performance portability across clusters and multicores

Complete SymGrid-Par Infrastructure



Summary

SymGrid-Par provides clear interfaces (OpenMath, SCSCP), a high-level parallel programming model (parallel Haskell), and a (flat) client-server architecture for parallel symbolic computation.

- The current release of SymGrid-Par (0.3.2) comes with an automated install script and some example programs
- A set of algorithmic skeletons is provided
- We provide a range of domain-specific skeletons (eg. Orbit computation)
- Other example applications exist for the older bespoke interface

Download:
`http://www.cs.st-andrews.ac.uk/~hwloidl/SCIEnce/SymGrid-Par/v0.3`
with `INSTALL` notes and `USE` examples.

Summary

Novel features of SymGrid-Par:

- parallel orchestration of SCSCP-based servers;
- **high-level parallel coordination** with domain-specific skeletons to allow easy parallelisation for non-specialists in the area of parallel programming;
- direct embedding of the interface into a familiar computer algebra shell;
- potentially co-ordinate several SCSCP-based systems in parallel;

Reflecting on SCIEnce results

- Freedom of choice w.r.t. CA system is crucial: don't force users into using a particular system, rather provide interfaces (SCSCP, domain-specific skeletons) for additional functionality
- Interoperability of CA systems opens new opportunities: use the best algorithmics across all systems
- Profit from mathematical expertise encoded in CA software (cash interface from Haskell side)
- Focus on high-level abstractions for parallelism and domain-specific skeletons to ease parallel computing
- **High-performance symbolic computation is mostly unexplored territory!**

Symbolic Computation on Clouds

The CALCIUM project explored symbolic computation on High-Performance Cloud architectures:

- **Goal**: Assess the suitability of Clouds for symbolic computation
- **Application**: Determinisation of a non-det. finite state automaton
- Instance of the Orbit skeleton
- Based on the existing cluster implementation by Cooperman et al.
- Ported to an OpenNebula based Cloud

[1] http://www.macs.hw.ac.uk/~hwloidl/Projects/CALCIUM.html

CALCIUM Results

- Good speedups on a private Cloud and small configurations
- Gaining access to larger Clouds is administratively difficult
- Current Cloud middleware is geared for high-throughput rather than high-performance computing
- High-performance is challenging due to the dynamic and irregular nature of the parallelism
- For symbolic computation, **ease-of-access** should be the main selling point:
  "Mathematical Software as a Service (SaaS)"

The Future of Parallel Symbolic Computation

- Hierarchical **clusters of multi-cores** will become the dominant high-performance computing platform
- Fault-tolerance will be crucial to use these for long running computations
- High-level abstractions will be needed to cope with these architectures
- Modern functional languages can be used as either computation or coordination languages
- Can we identify a parallel symbolic challenge application?

[1] www.macs.hw.ac.uk/sicsawiki/index.php/MultiCoreChallenge