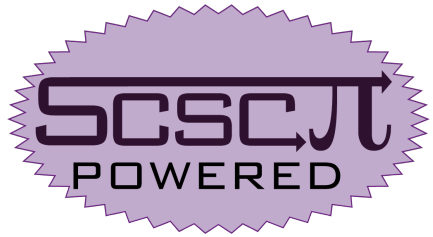
Combining computer algebra systems with the **S**ymbolic **C**omputation **S**oftware **C**omposability **P**rotocol

Alexander Konovalov

Centre for Interdisciplinary Research in Computational Algebra
University of St Andrews, Scotland

Fourth RISC/SCIEnce Training School in Symbolic Computation, Linz, June 29–July 10, 2009

*Supported by the EU FP6 project "**SCIEnce** – **S**ymbolic **C**omputation **I**nfrastructure for **E**urope"*



Talk outline

Infrastructure for *distributed* symbolic computation:

- Underlying protocol: **SCSCP**
- Encoding for mathematical objects: **OpenMath**
- OpenMath and SCSCP implementation in GAP (but expect similar in any SCSCP-compliant system)
- Parallel computations in GAP with SCSCP

Symbolic computation community needs

Research problems may require combinations of two or more instances of:

- computer algebra systems (e.g. GAP, KANT, Magma, Maple, Singular ...)
- constraint solvers (e.g. ECLiPSe, Minion ...)
- numerical tools (e.g. MatLab, Octave ...)
- statistical (e.g. R ...)
- etc.

Most common restrictions (from our GAP experience)

- interfaces do not support remote communication
- transmission of large or complex objects may be difficult
- Support of new system requires new I/O convertor. It relies upon the I/O format, may be subject to parsing errors and needs update if I/O format of the linked system changes
- not enough deeply (syntax, cd) and widely (other CAS) supported data encoding format (*OpenMath*)
- not interactive, just database access (Web services)
- not enough robust (*ParGAP*)
- less efficient for irregular parallel computing (*ParGAP*)
- shaped to deal with the particular problem (*dc*)
- may not work in some operating systems
- may be not easily customisable by the end-user

Composability goals

The SCIEnce project involves GAP, KANT, Maple, MuPAD:

- to deliver robust, usable and flexible extensions to our systems allowing any system to be easily used as a server or a client for both general and problem-specific services.
- to be open to connections to other clients and servers, both other powerful mathematical systems and special purpose programs

We want to offer an extendable framework for combining systems, which:

- allows both local and remote communication
- not relies on the input/output format of a particular system
- optimises average efforts needed from other systems to join

Common protocol for communication

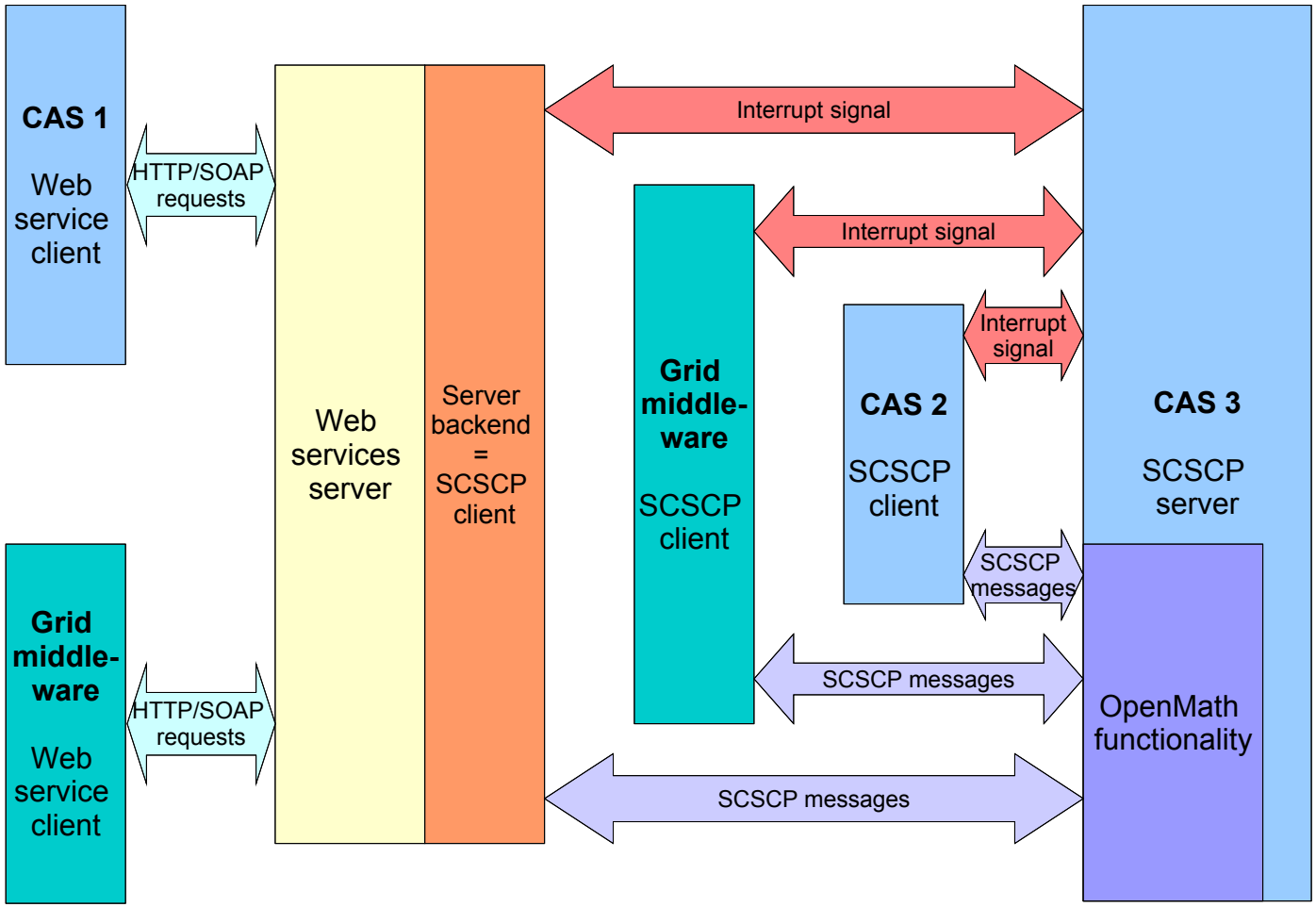
We designed the *Symbolic Computation Software Composibility Protocol* (**SCSCP**) by which a computer algebra system (CAS) may offer services for the following clients:

- Another instance of the same CAS (in a parallel computing context or on different architectures)
- Another CAS running on the same computer or remotely
- A Web server which passes on the same services as Web services using SOAP/HTTP protocols to another clients
- Grid middleware

Features of composability model

- SCSCP is in fact a lightweight sockets based RPC protocol
- it uses OpenMath standard for data encoding
- its implementation stays mainly within systems, rather than in wrappers
- it can be used for direct communication between systems
- SCSCP servers and clients can be “translated” into standard Web services servers and clients using Java SCSCP API
- Grid middleware (SymGrid-Par) also uses SCSCP to talk to CAS

Current vision of SCSCP usage



What is OpenMath?

- XML-based standard for representing mathematical objects with their semantics
- the current OpenMath Standard 2.0 is dated June 2004
- the worldwide OpenMath activities are coordinated within the OpenMath Society, based in Helsinki
- the idea is to allow their exchange between various programs, storing in databases, publishing on web, etc.
- two encodings: XML and binary format

What is OpenMath?

- basic objects: integers, floats, strings, byte arrays, variables, symbols
- symbols consist of a name and a reference to a definition in an external document called a *content dictionary* (**CD)**
- OpenMath objects can be combined recursively in a number of ways using standard OpenMath constructors: application, attribution, binding, error
- let us look at some examples in GAP
- support of various symbols may be different in various CASs, for each symbol it may be two-sided (both encoding and decoding) or only in one of these ways
- see `http://www.openmath.org` for further details
- if you need support of existing symbols or interested in creating new CD - please contact us!

SCSCP: OpenMath inside

- The feature of SCSCP is that protocol messages are also represented as OpenMath objects using OpenMath content dictionaries scscp1 and scscp2 developed for this purpose
- SCSCP specification defines semantical and technical descriptions and allowed sequences of OpenMath-encoded messages to and from CAS:
  - remote procedure call
  - returning result of successfully completed procedure
  - returning a signal about procedure termination

Flexible enough?

- May be limited to functionality/data types for which CDs exist
  - Avoid this by allowing transient CDs, which contain symbols specific to that service, obtainable from the server on request
- Encoding may be unreasonably bulky, or encoding costs may be too high for some applications
  - Perfectly OK for services which pass real data in some private format encoded in an OMSTRING, OMBYTES or OMFOREIGN element, if that suits the application.
  - Both transmission of actual mathematical objects and references to them are supported
  - Also working on new CDs for efficient representation of some common cases (eg matrices over finite fields)

scscp1 CD defines:

- three main kinds of messages: `procedure_call`, `procedure_completed`, `procedure_terminated`
- RPC identifier: `call_id`
- options that may be added to the `procedure_call` message: `option_runtime`, `option_debuglevel`, `option_min_memory`, `option_max_memory`, `option_return_object`, `option_return_cookie`, `option_return_nothing`
- information that may be supplied with the result: `info_runtime`, `info_memory`, `info_message`
- standard errors: `error_runtime`, `error_memory`, `error_system_specific`

scscp2 CD defines:

- procedures for remote objects: `store_session`, `store_persistent`, `retrieve`, `unbind`
- special procedures: `get_allowed_heads`, `is_allowed_head`, `get_transient_cd`, `get_signature`, `get_service_description`
- special symbols: `signature`, `service_description`, `symbol_set`, `symbol_set_all`, `no_such_transient_cd`

GAP implementation of the SCSCP

- GAP Package SCSCP by AK & Steve Linton:

  http://www.cs.st-andrews.ac.uk/~alexk/scscp.htm
- Allows GAP to work as an SCSCP server and client over the TCP/IP protocol
- Uses GAP packages IO, GAPDoc and OpenMath
- Client's version for GAP 4.4.12 works in Linux, Windows and Mac OS X
- The server works in GAP 4.4.12, and only needs functionality for the exception and error handling in GAP.dev added by Steve Linton (will appear in GAP 4.5) to be able to pass errors to the client
- Extends the OpenMath package:
  - new symbols from scscp1 and scscp2 OM CDs
  - support of OM attributes (OMATTR, OMATP)
  - support of OM references (OMR)

User-level functionality:

- The service provider installs procedures available as SCSCP services and starts the SCSCP server
- The client sends request to the server and gets back result
- The underlying technology is well-hidden: the end-user may know nothing about OpenMath and SCSCP !!!
- Store/Retrieve procedures allowing to work with remote objects
- InputOutputTCPStreams, compatible with other kinds of streams in GAP

Configuring SCSCP GAP server

1. Specify in `gap4r4/pkg/scscp/config.g`:
   - default InfoLevel, host name, port number etc.
2. Put all what you need in the configuration file (see `gap4r4/pkg/scscp/example/myserver.g`):
   - loading all necessary packages
   - other required GAP code or its reading from other file(s)
   - installation of SCSCP procedures using `InstallSCSCPprocedure("NameForClient", InternalName );`
   - starting the server with the command `RunSCSCPserver( <> );`
3. Start GAP with `gap myserver.g` or as a daemon (output may go on screen or be redirected to a file or to `/dev/null`)

Configuring SCSCP GAP server

- Why do we need to export GAP functions which will be available as SCSCP procedures, e.g.:
  `InstallSCSCPprocedure( "WS_IdGroup", IdGroup );`
- Because this allows to control which functions the client may call
- we may offer a specialised service with one or several specific functions available
- or some generic service e.g. to evaluate arbitrary OpenMath code or expression in CAS syntax etc.
- We are running an SCSCP server in St Andrews on `chrystal.mcs.st-andrews.ac.uk`, port 26133

Examples of simple calls

myserver.g

```
...
InstallSCSCPprocedure( "WS_Factorial", Factorial );
InstallSCSCPprocedure( "WS_Phi", Phi, "see ?Phi in GAP", 1, 1 );
...
```

To contact the server, the client need to know: the name of the remote procedure, the name of the server and the number of the port

GAP session

```
gap> EvaluateBySCSCP( "WS_Factorial", [ 12 ], "localhost", 26133 );
rec( attributes := [ [ "call_id", "localhost:26133:12325:GxjuLOvp" ] ],
  object := 479001600 )

gap> EvaluateBySCSCP( "WS_Phi", [ 12 ], "localhost", 26133 );
rec( attributes := [ [ "call_id", "localhost:26133:12325:YtlnsRwb" ] ],
  object := 4 )
```

Group identification: three possible approaches

group $\longrightarrow$ group id

- client: GAP (slow machine? no small groups library?)
- server: fast and complete GAP installation

list of permutations $\longrightarrow$ group id

- client: CAS which "understands" permutations
- server: complete GAP installation

pcgs code (integer that encodes the group) $\longrightarrow$ group id

- client: GAP in Windows - ANUPQ package is not working :(
- server: GAP in UNIX environment - ANUPQ works :)

Three approaches (continued)

group → group id

```
InstallSCSCPprocedure( "WS_IdGroup", IdGroup );
```

list of permutations → group id

```
IdGroupByGenerators := function( permlist )
return IdGroup( Group( permlist ) );
end;

InstallSCSCPprocedure( "GroupIdentificationService", IdGroupByGenerators );
```

pcgs code (integer encoding the group) → group id

```
IdGroup512ByCode:=function( code )
local G, H;
G := PcGroupCode( code, 512 ); # recreate the group from code
H := PcGroupFpGroup( PqStandardPresentation( G ) );
return IdStandardPresented512Group( H );
end;

InstallSCSCPprocedure( "IdGroup512ByCode", IdGroup512ByCode );
```

How it works

Before, we need an auxiliary function ...

Client's counterpart for the 3rd example

```
IdGroup512:=function( G )
local code, result;
if Size( G ) <> 512 then
  Error( "G must be a group of order 512 !!!\n" );
fi;
code := CodePcGroup( G );
result := EvaluateBySCSCP( "IdGroup512ByCode",
                           [ code ],
                          "chrystal.mcs.st-and.ac.uk",
                           26133 );
return result.object;
end;
```

Now we can see the demo!

Example: remote objects

- Objects may "live" on the server while the client has only a reference (cookie) pointing to them
- There is no need to transmit objects repeatedly over the network
- The client may be a CAS which does not have objects of that type at all
- The client may retrieve object in its default OpenMath representation (be careful about its subobjects!)
- Also, the actual object may be deleted from the server

Example: stateful SCSCP service

Let $G = \langle \sigma_1, \sigma_2, \ldots, \sigma_n \rangle$ be a permutation group. For the orbit computation, we need a service that takes an integer k and returns a set of all distinct images of k under $\sigma_1, \sigma_2, \ldots, \sigma_n$

Code for the server

```
PointImages := function( G, n )
local g;
return Set( List( GeneratorsOfGroup(G), g -> n^g ) );
end;

InstallSCSCPprocedure( "PointImages", PointImages );
```

Now we can create the group G on the server as a remote object and next time pass only the reference to it!

Parallel computing in GAP

There are several possible approaches:

- ParGAP package (requires UNIX/Linux environment)
- dc (external coordinating software for specific research problem)
- Job submission systems like Condor, Grid middleware like Globus toolkit etc.
- SymGrid-Par (middleware developed in the JRA1 activity of the SCIEnce project)

Only the last one is SCSCP- compliant
But what can we do in GAP only, avoiding external binaries as much as possible?

Parallel tools in the SCSCP packages

- We present new parallel computing framework in the SCSCP package implemented purely in GAP, capable of:
  - creating multiple processes
  - synchronising processes
  - waiting for the first available result
- Easy to study and modify existing code for the end-user
- Big parallel Master-Worker skeleton – `ParListWithSCSCP`
- Can be easily modified to have parallel versions of other list arithmetic like ForAll, ForAny, First, Number, Filtered
- Also can be modified to extend the input list, e.g. as needed in orbit computation
- May orchestrate other SCSCP-compliant systems

Processes

- `EvaluateBySCSCP` uses a combination of `NewProcess` and `CompleteProcess`
- These works with new objects - *processes*
- Process is associated with corresponding *InputOutputTCPStream*
- You may start process, send a request and collect the result later, doing something else in the meantime
- There are functions for:
  - processes synchronization
  - waiting until the first available result and abandoning remaining processes
- We can terminate process locally (working on remote termination)
- See example of Karatsuba multiplication for polynomials in the SCSCP package manual

Example

```
gap> ParListWithSCSCP( List([2..6],n->SymmetricGroup(n)),"WS_IdGroup");
#I  master -> [ "localhost", 26133 ] : SymmetricGroup( [ 1 .. 2 ] )
#I  master -> [ "localhost", 26134 ] : SymmetricGroup( [ 1 .. 3 ] )
#I  [ "localhost", 26133 ] --> master : [ 2, 1 ]
#I  master -> [ "localhost", 26133 ] : SymmetricGroup( [ 1 .. 4 ] )
#I  [ "localhost", 26134 ] --> master : [ 6, 1 ]
#I  master -> [ "localhost", 26134 ] : SymmetricGroup( [ 1 .. 5 ] )
#I  [ "localhost", 26133 ] --> master : [ 24, 12 ]
#I  master -> [ "localhost", 26133 ] : SymmetricGroup( [ 1 .. 6 ] )
#I  [ "localhost", 26133 ] --> master : [ 720, 763 ]
#I  [ "localhost", 26134 ] --> master : [ 120, 34 ]
[ [ 2, 1 ], [ 6, 1 ], [ 24, 12 ], [ 120, 34 ], [ 720, 763 ] ]
```

SCSCP vs ParGAP

Master-slave in ParGAP	**Master-worker in SCSCP**
uses MPI	uses SCSCP
needs UNIX environment	client works in Windows as well
works only with GAP	worker: anything SCSCP-compliant
broken if one slave is lost	retries on another worker
no adding new slaves in the middle of computation	allows to add new workers from a previously declared range
no automatic marshaling of complex objects	automatic marshaling if object “supports” OpenMath

Other benefits of SCSCP

- Be nice to other users – you may shut down some services without breaking the whole computation
- If your jobs will run out of memory competing for it with other jobs (yours or other users, esp. in multi-core environment), the computation will be continued with a smaller number of servers without break (a compatibility feature in GAP 4.4.12 which happened to be a benefit!)
- Ability to dynamically add services and resist to failures is useful when you work in a guest environment and do not administer computers where SCSCP services are running

Speedup

Calculation of the Quillen invariant for the first 100 groups of order 512 from the GAP Small Groups Library with one master and two workers (average on 2 runs on MacBook Intel Core 2 Duo 2 GHZ):

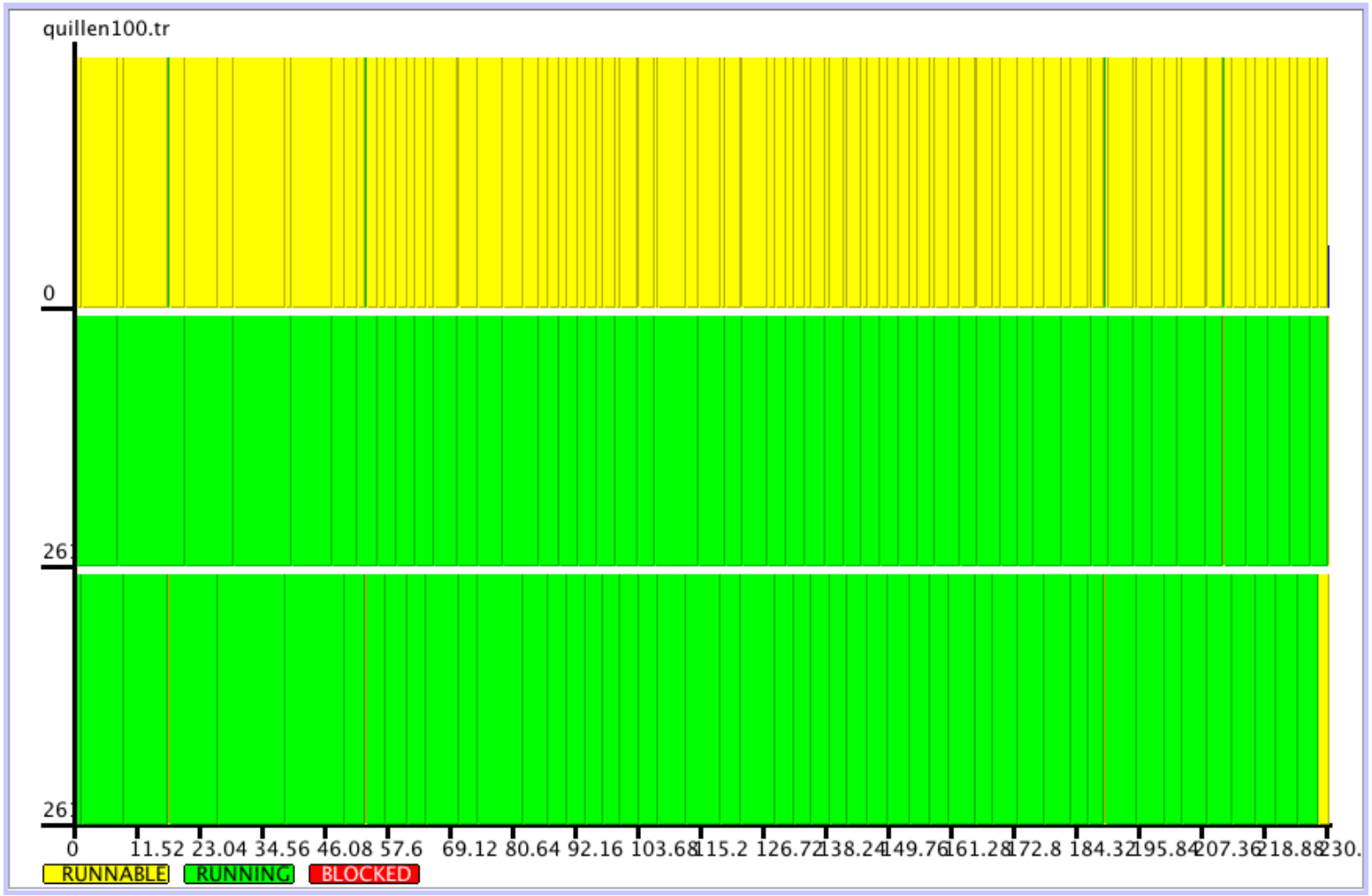
- GAP: 397 sec
- ParGAP: 251 sec (speedup 1.58)
- SCSCP: 205 sec (speedup 1.94)

EdenTV

- To analyse the performance of parallel SCSCP; framework, we make use of the EdenTV; program developed initially to visualize the performance of parallel programs written in functional programming language Eden, and now distributed under the GNU Public License and available from `http://www.mathematik.uni-marburg.de/~eden/?content=EdenTV`
- See Jost Berthold and Rita Loogen, **Visualizing Parallel Functional Program Runs - Case Studies with the Eden Trace Viewer**, in *Parallel Computing: Architectures, Algorithms and Applications. Proceedings of the International Conference ParCo 2007*, Advances in Parallel Computing 15(2007) for the description of EdenTV

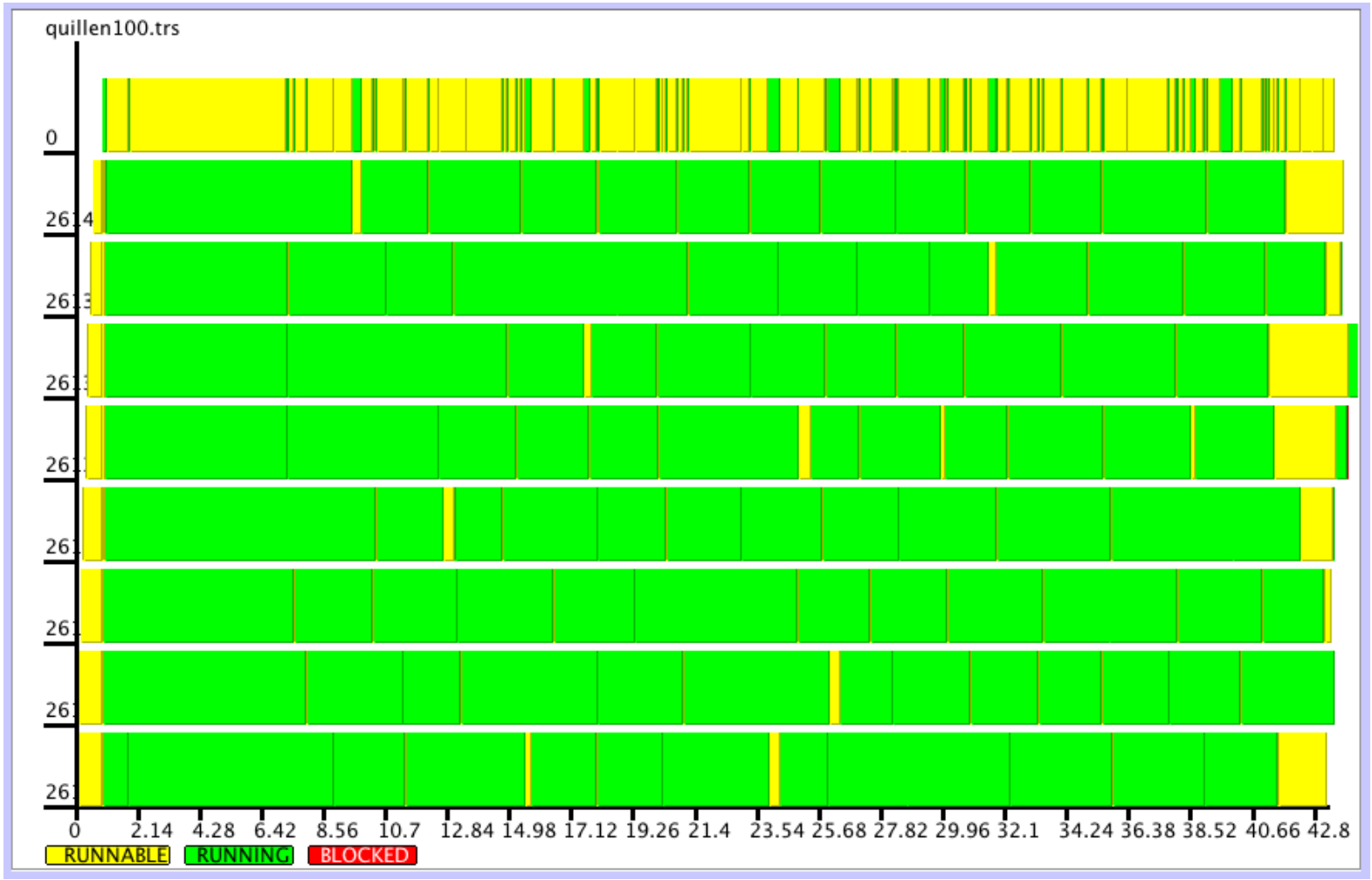
Quillen 100 : master + 2 workers

MacBook Intel Core 2 Duo 2 GHZ



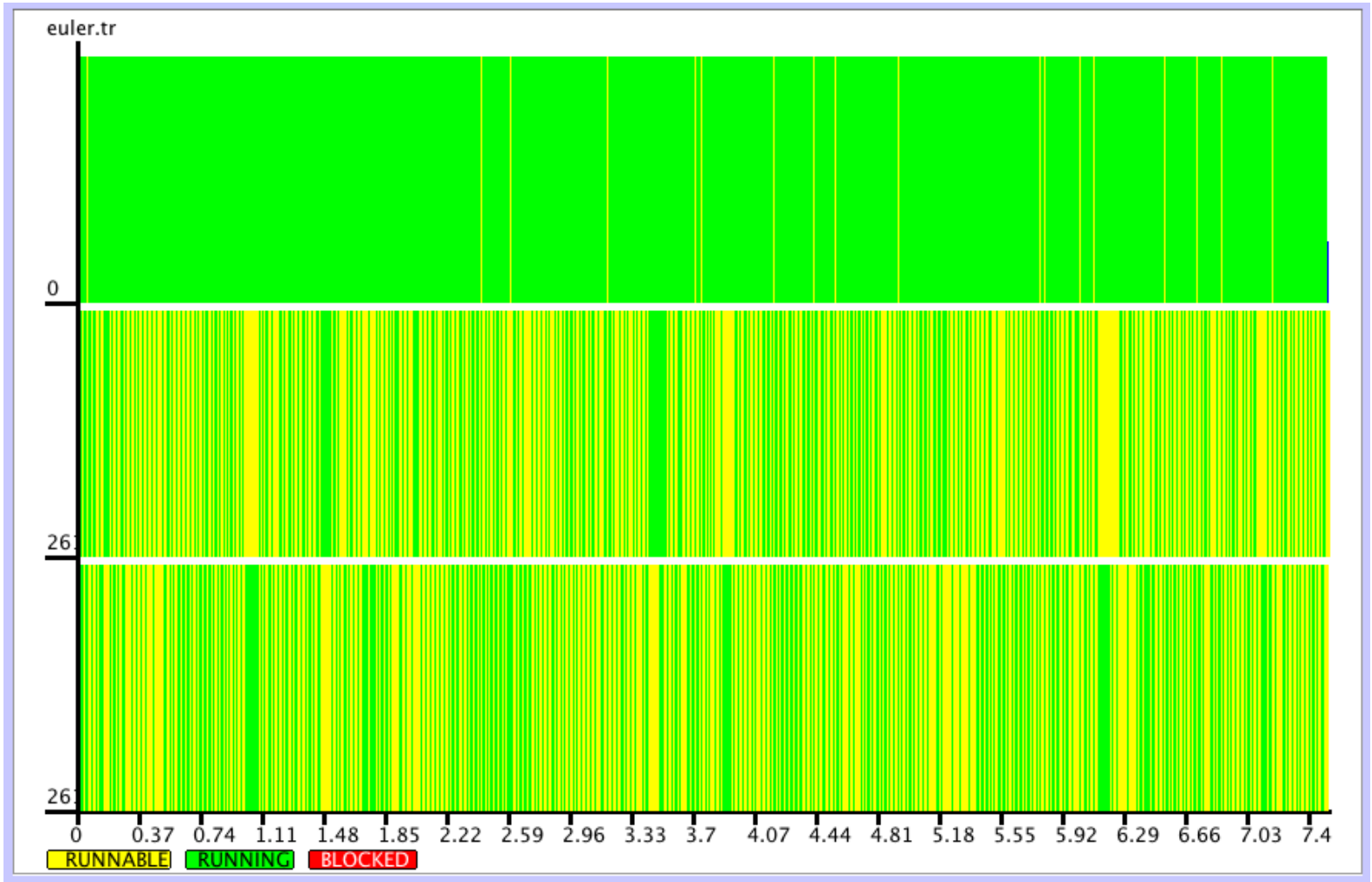
Quillen 100 : master and 8 workers

8-core Dell Poweredge 2950:
two quad-core Intel Xeon 5355 @ 2.66 Ghz;



Euler 1000 : master and 2 workers

MacBook Intel Core 2 Duo 2 GHZ



Conclusions

We would be very interested:

- in connecting other systems or stand-alone programs
- in test problems and use cases:
  - for cooperating systems
  - for parallel computation
  - for CA-based Web services

References

SCIEnce — Symbolic Computation Infrastructure for Europe, project homepage
http://www.symbolic-computation.org/

S. Freundt, P. Horn, A. Konovalov, S. Linton, D. Roozemond.
Symbolic Computation Software Composability Protocol (SCSCP), Version 1.3, 2009
http://www.symbolic-computation.org/scscp

A. Konovalov, S. Linton.
GAP package *SCSCP — Symbolic Computation Software Composability Protocol*
http://www.cs.st-andrews.ac.uk/~alexk/scscp.htm

M. Costantini, A. Konovalov and A. Solomon. GAP package *OpenMath*.
http://www.cs.st-andrews.ac.uk/~alexk/openmath.htm.

S. Freundt, P. Horn, A. Konovalov, S. Lesseni, S. Linton, D. Roozemond.
OpenMath Content Dictionary **scscp1**
http://www.win.tue.nl/SCIEnce/cds/scscp1.html

S. Freundt, P. Horn, A. Konovalov, S. Lesseni, S. Linton, D. Roozemond.
OpenMath Content Dictionary **scscp2**
http://www.win.tue.nl/SCIEnce/cds/scscp2.html