Building a Platform for Parallel Computational Algebra

Steve Linton
Oberwolfach 2011

reporting work of
Reimer Behrends, Alexander Konovalov, Paul Smith and others, supported by EPSRC
Every Year We Can Solve Bigger Problems

**1990's Practical Complexity Theory**

All EXPTIME problems can be solved in linear time

Wait a linear number of years while Moore's law makes computers exponentially faster

• Not any more :-((
• Individual cores have almost stopped getting faster
  • Just more numerous
• Modern gamers PC: 4 core CPU, 240 core GPU.

• Supercomputers: $10^5$ cores today, $10^6$ planned, $10^8$ being explored.
• Bigger problems in five and ten years, will need parallelism
What I Learnt at my Mother’s Knee

If you have two processors, use them to solve two problems at once.

- Good advice for most people, until recently
  - Less tenable in the world of 8-core laptops and 48-core desktops
- If you want to keep solving bigger problems for the next ten years you will have to engage with parallel programming
- We’re trying to make GAP a good platform for that
Some Prior Art

- ParGAPMPI -- Cooperman '99
  - explicit message passing for GAP on clusters
- DC -- Luebeck, Neunhoeffer 2010
  - very efficient specialised distributed orbit/condensation program
- Numerous projects in the wider computational algebra world
- SCSCP -- Freundt, Horn, Konovalov, L, 2009
Available now

- GAP package by Alexander Konovalov
- C(++) and Java libraries, interfaces to other systems
- one outcome of EU program SCIEnce

Simple robust client-server model

- set up GAP, various other systems or C or Java programs as servers
- call them from any of those systems as clients
- intended mainly for Web services but also useful for some parallel computation
- data encoding is OpenMath/XML -- not very compact
- suitable for course-grained computations
SCSCP Application

- Modular Isomorphism Problem for groups order 512
- are there two groups of order 512 with isomorphism group rings over F2?
- Konovalov, Eick ’11
- answer is no
- final checks 14000 CPU hours over 3 weeks on UK National Grid service organised with SCSCP
Goals of Present Project

• to support parallel computing in GAP for users and programmers
  • Threaded GAP sessions on multicore machine
  • Distributed computing on clusters (of multicores) and supercomputers
  • Easy ways for algorithm designers to describe and implement parallel algorithms
  • Allowing programmers to explore new parallel programming models
Closer-grained Parallel Programming is Hard

• A simple trap (out of myriads):

```plaintext
if not empty(queue) then
    take x from the queue and process it
fi
```

• In a parallel program another thread may have emptied the queue between the test and the action

• A less simple trap

```plaintext
a := < something>
b := true
if b then
    access a
```

• here the right hand thread may see b = true but a still uninitialized
Skeletions

- We don’t expect most parallel GAP users/programmers to work directly with threads, locks, message-passing, etc.

- Instead we will provide skeletons: high-level parallel functions which you flesh out with pieces of sequential code to do parallel computations.

- The skeleton takes care of distribution, communication, locking, scheduling

- Well designed skeletons should be stable interfaces across a range of underlying technology
# A Sample Skeleton

## Graph Traversal Skeleton

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Seed objects,</td>
<td>• Calls the visit function <em>once</em> on each object reachable from</td>
</tr>
<tr>
<td>• neighbours function,</td>
<td>the seeds by repeated application of neighbours</td>
</tr>
<tr>
<td>• equality test,</td>
<td>• maybe assigns each object a unique identity</td>
</tr>
<tr>
<td>• visit function,</td>
<td>• Calls the revisit function when a loop is discovered</td>
</tr>
<tr>
<td>• [revisit function]</td>
<td>• more?</td>
</tr>
<tr>
<td>• options</td>
<td></td>
</tr>
</tbody>
</table>
More About Skeletons

- A skeleton describes a *pattern* of parallel computation
  - orbit of a matrix group and determinisation of automata fit into the graph traversal skeleton
  - parallel List will also do a simple matrix multiplication
- Well known idea in CS community
  - Google’s MapReduce
- Some skeletons were already in ParGAP:
  - ParList, MasterSlave
Some Other Ideas for Skeletons

• **ParList**
  - Like GAP List, but no promises of order of execution

• **Isomorph Removal**
  - Input: a list of objects and a selection of invariants and binary isomorphism tests
  - Output: one object from each equivalence class

• **[Random] Search**
  - Input domain, success criterion, bound
  - Return object or fail

• **Task Farm**
  - Submit a task, which runs (perhaps after waiting for data to be ready) and may submit more tasks

• **Chain Reduction** Given an ordered list of objects, reduce each object against each earlier object in some way.
  - Naive Gaussian elimination
  - Schreier-Sims?

• **Pair Completion** Try to make new objects from all *pairs* of old ones. Interreduce?
  - Knuth-Bendix, Groebner

• **First to finish** Try a number of algorithms to solve a problem
  - finiteness and infiniteness testing
Questions for you:

• Can you imagine expressing your algorithms in terms of skeletons like these?

• If so, what are good skeletons and examples of applications of them?

• If not, what are the obstacles?

• What algorithms don’t fit?

• how to parallelise them?
# Under the Hood — Software Architecture

<table>
<thead>
<tr>
<th></th>
<th>Shared Memory</th>
<th>Distributed</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Low level</strong>&lt;br&gt;Primitives</td>
<td>Threads, Locks, Regions</td>
<td>Messages, UPC arrays</td>
</tr>
<tr>
<td><strong>Mid level</strong>&lt;br&gt;Common primitives</td>
<td>Channels, Single-assignment variables, Barriers</td>
<td></td>
</tr>
<tr>
<td><strong>Mid-High level</strong>&lt;br&gt;Data structures</td>
<td>Distributed/threadsafe arrays, hash tables, task queues</td>
<td></td>
</tr>
<tr>
<td><strong>High level</strong>&lt;br&gt;Skeletons</td>
<td>ParList, ParReduce, Graph Traversal, Chain Reduction, First to Finish, Task Farm,.....</td>
<td></td>
</tr>
<tr>
<td><strong>Thread-safe Library</strong></td>
<td>Full GAP library should be able to run in threaded environment — quite a bit of cleanup to do</td>
<td></td>
</tr>
<tr>
<td><strong>Parallelised Library</strong>&lt;br&gt;Selected Routines</td>
<td>Open to Suggestions</td>
<td></td>
</tr>
<tr>
<td><strong>Applications</strong>&lt;br&gt;Exemplars</td>
<td>Combinations in E₈, search problems, homology,.....</td>
<td></td>
</tr>
</tbody>
</table>
Hot off the Presses

First alpha-test release of HPCGAP

GAP 4.5 beta kernel extended with support for shared memory parallelism
Low and mid level support only
  no skeletons implemented yet,
  many parts of the library are still not thread-safe
Features:
  multiple GAP interpreter threads
  automatic object access control
  locks with deadlock prevention
  mid-level tools: channels, tasks, atomic lists and records, synchronization, thread-local variables...
  some documentation
  many bugs!
See Alexander Konovalov, Max or me for access.
ShowTime
Concluding Remarks

- Reasonably fine-grained parallel computing will be a practical necessity in the fairly near future if you want to make effective use of your computers.
- There is no silver bullet.
- But there are lots of interesting problems:
  - the whole corpus of algebraic algorithms to review.
  - do the theoretical frameworks out there give practically applicable insights?
    - NC? -- \((\log n)^c\) time on \(n^d\) processors.
    - what are the right general-purpose skeletons?
- We’re building software to let you explore these issues.
- Input and feedback are welcome.