SymGrid-Par: A Standard Skeleton-Based Framework for Computational Algebra Systems

Phil Trinder    The SCIEnce Team    The HPC-GAP Team

Dependable Systems Group
School of Mathematical and Computer Sciences
Heriot-Watt University, Edinburgh, UK

25/09/2010
Computational Algebra

- Computational Algebra (CA) systems manipulate symbolic structures like groups, rings, or sets of polynomials, to provide exact mathematical solutions.
- Examples: Maple, Mathematica, GAP, Kant, ...
- Unusual for parallel programming
Computational Algebra

- Computational Algebra (CA) systems manipulate symbolic structures like groups, rings, or sets of polynomials, to provide exact mathematical solutions.
- Examples: Maple, Mathematica, GAP, Kant, ...
- Unusual for parallel programming
  - Neither matrices, nor floating points
  - Relatively complex data structures, e.g. tuples, lists & sets
  - Highly dynamic parallelism: tasks created at runtime
  - Highly irregular parallelism: task sizes vary by 5 orders of magnitude
CA Community

- Seeking to **standardise** between multiple CA systems
- Seeking to **parallelise** large computations
- Seeking to **provide distributed access** to CA systems
- Willing to adopt high-level approaches like skeletons
SCIENCE Project

- Symbolic Computation Infrastructure for Europe (SCIENCE) is an EU FP6 project (£4.6M) 2007-2012 to address the challenges.

- **Partners:** St Andrews, T.U. Berlin, CNRS, T.U. Eindhoven, Heriot-Watt, Linz, Paderborn, Timisoara Universities, Maplesoft
Symbolic Computation Infrastructure for Europe (SCIEnce) is an EU FP6 project (£4.6M) 2007-2012 to address the challenges.

**Partners:** St Andrews, T.U. Berlin, CNRS, T.U. Eindhoven, Heriot-Watt, Linz, Paderborn, Timisoara Universities, Maplesoft

**Aim** to provide

- transparent access to complex, mathematical software, through Grid/Web Services
- interoperation between independent CA systems through OpenMath data format and the SCSCP protocol
- exploitation of modern parallel hardware with high-level orchestration of parallelism
SymGrid-Services and SymGrid-Par
SymGrid-Par Infrastructure

CA
client
socket

CAG
OpenMath
SCSCP

Coord. Server
socket

GCA

OpenMath
SCSCP

CA
server
socket

CA
server
socket

CA
server
socket
Components of the Architecture

- Either command-line client or a computer algebra system.
- High-level **Coordination Server** handling parallelism
  - makes essential use of parallel Haskell dynamic parallelism management to deal with irregularity
  - parallelism is mainly specified and coordinated here
  - automatic resource management on this level (load balancing etc)
- Any **SCSCP-based** (Symbolic Computation Software Composability Protocol) computer algebra server:
  - tested with GAP and a Haskell-side server
  - servers can themselves use parallelism, but
  - no direct communication between servers
Coordination Server

A Parallel Haskell component that
- implements a collection of (parallel) CA functions
- provides algorithmic skeletons for CA computation
- calls CAs to perform the heavy computation.
Classes of SymGrid-Par Skeletons

- Problem-Oriented
- Process-Oriented
- Computational Algebra specific
Problem-Oriented Skeletons

Standard, problem oriented higher-order functions:

\[
\begin{align*}
\text{parMap} & : (a \to b) \to [a] \to [b] \\
\text{parFold} & : (a \to b \to b) \to b \to [a] \to b \\
\text{parFold1} & : (a \to b \to b) \to [a] \to b \\
\text{parMapFold} & : (a \to b) \to (b \to c \to c) \to c \to [a] \to c \\
\text{parMapFold1} & : (a \to b) \to (b \to c \to c) \to [a] \to c \\
\text{parZipWith} & : (a \to b \to c) \to [a] \to [b] \to [c]
\end{align*}
\]
Problem-Oriented Examples I

zipWith (+) [1..10] [100,200..1000]

=>

[101,202,303,404,505,606,707,808,909,1010]
Problem-Oriented Examples 1

\[
\text{mapFold } f \ g \ n = \text{fold } g \ n \ . \ \text{map } f
\]
Problem-Oriented Examples II

▶ parMap in GAP:

```gap
gap> Read("parallel.g");
gap> GAPparMap("Fibonacci",[100..105]);
==== Starting parallel execution on 6 processors ...
===== Setting up Gap around Haskell ....6
===== Setting down Gap around Haskell....
["354224848179261915075","573147844013817084101","927372692193078999176","1500520536206896083277","2427893228399975082453","3928413764606871165730"]
```
Problem-Oriented Examples II

- parMap in Maple:

```gap
gap> Read("parallel.g");
gap> mapleParMap("fibonacci",[100..103]);
==== Starting parallel execution on 4 processors ... 
====== Setting up Maple around Haskell....4
====== Setting down Maple around Haskell....4
["354224848179261915075","573147844013817084101","927372692193078999176","1500520536206896083277"]
```
Problem-Oriented Examples III

- A Chinese Remainder Algorithm operating on a list of values and a list of modules

\[
cra \: xs \: ms = \text{foldl} \: (\text{modSum} \: m') \: 0 \: (\text{parZipWith} \: \text{bin_cra} \: xs \: ms)
\]

where \( \text{bin_cra} \: x \: m = x * b * m0 \)

where \( b = \text{modInv} \: m \: m0 \)

\( m0 = m' \div m \)

\( m' = \text{product} \: ms \)
Process-Oriented Skeletons

- Create common process structures: farm, workpool, Google mapReduce
- A farm statically partitions the work.

```haskell
farm :: Int -> (a -> b) -> [a] -> [b]
    -- number of workers
    -- worker process
    -- input data
    -- output data
```
A workpool dynamically partitions the work.

\[
\text{workpool:: } \text{Int} \rightarrow \text{number of workers} \\
\text{Int} \rightarrow \text{prefetch} \\
(a \rightarrow a) \rightarrow \text{worker} \\
[a] \rightarrow \text{input data} \\
[a] \rightarrow \text{output data}
\]
Process-Oriented Skeletons III: mapReduce

- A variant of the parMapFold
- Core of Google/Hadoop Web engines
- SymGrid-Par provides scalability, but not reliability
Process-Oriented Skeletons III: mapReduce

Figure: Structure of the parallel Map-reduce pattern
Process-Oriented Skeletons III: mapReduce

parMapReduce :: Ord k2 =>
    Int -> -- No. of partitions
    (k2 -> Int) -> -- key partitioning
    (k1 -> v1 -> [(k2, v2)] -> -- mapF function
    (k2 -> [v2] -> Maybe v3 -> -- local reduceF function
    (k2 -> [v3] -> Maybe v4 -> -- global reduceF function
    [Map k1 v1] -> -- distributed input data
    [Map k2 v4] -- distributed output data
Computational Algebra Skeletons

- Capture CA-specific patterns of parallelism, currently:
  - Transitive Closure
  - Orbit
  - Critical-Pair-Completion
  - Multiple Homomorphic Images
Computational Algebra Skeletons

- Capture CA-specific patterns of parallelism, currently:
  - Transitive Closure
  - Orbit
  - Critical-Pair-Completion
  - Multiple Homomorphic Images
- Support irregular task sizes
- Combine data & task parallelism
- May have shared or distributed memory implementations
Transitive Closure

- Calculate the transitive closure of a relation from a set of initial elements.

\[
\text{transcl} :: (\text{Eq } a) \Rightarrow (a \rightarrow [a]) \rightarrow -- \text{ relation}
\]
\[
[a] \rightarrow -- \text{ seed elements}
\]
\[
[a] -- \text{ all reachable elements}
\]

- Shared-memory implementation
Transitive Closure: with Parallel Buffer

circular list: buffer

worker worker

relation
Orbit

- Recent CA pattern (2001)
- Similar to transitive closure, except that new elements are generated from existing elements by a set of generator functions.
Orbit Skeleton

\[
\text{orbit} :: (\text{Eq } a, \text{Ord } a) \Rightarrow \\
[ a \rightarrow a ] \rightarrow \text{ generator functions} \\
[ a ] \rightarrow \text{ initial set} \\
[ a ] \text{ result set}
\]
Orbit Implementation

- Currently only a shared-memory implementation [TFP’10]
- Design of a distributed-hash table implementation in progress
SymGrid-Par: Skeletons for Computational Algebra

Phil Trinder, SCIEnce Team, HPC-GAP Team

Context
SymGrid-Par Design
SymGrid-Par Skeletons
SymGrid-Par Performance
SymGrid-Par Dissemination
Discussion
Ongoing Work: SymGrid-Par2
Critical Pair Completion

- Starts with an arbitrary specification of an algebraic structure, and transforms it to an algebraic structure with nicer properties, typically common decision problems are far simpler.

- Formally: Given a set of objects $T$ and a reduction relation defined as a set of rules $G \subseteq T \times T$, decide whether two elements $s, t \in T$ are in the reflexive, symmetric, transitive closure of $G$. 

Critical Pair Completion Implementation

\[
\text{parCritPairCompletion ::} \\
\quad (\text{NFData } a) \Rightarrow \\
\quad ([a] \rightarrow a \rightarrow a \rightarrow a) \rightarrow \quad \text{-- } f: \text{ operation on pair} \\
\quad ([a] \rightarrow [a] \rightarrow [a]) \rightarrow \quad \text{-- } f': \text{ normalization of the pool} \\
\quad ([a] \rightarrow a \rightarrow ([a], [(a,a)])) \rightarrow \quad \text{-- } g: \text{ construct new pool and pairs} \\
\quad ([a] \rightarrow a \rightarrow \text{Bool}) \rightarrow \quad \text{-- predicate, guarding addition to pool} \\
\quad [a] \rightarrow \quad \text{-- pool} \\
\quad [a] \rightarrow \quad \text{-- worklist} \\
\quad [a] \quad \text{-- result}
\]

- Application: Implementing Buchberger’s Gröbner Basis algorithm
Multiple Homomorphic Images

- Often a problem can be solved faster in a simpler domain because the data structures are smaller.

- The multiple homomorphic images approach:
  1. map the input data into several homomorphic images ($h$)
  2. compute the solution in each image ($f$)
  3. combine the results of all images to a result in the original domain (a fold of $g$).
Multiple Homomorphic Images Skeleton

\[
\text{multHomImg} :: \\
(p \rightarrow b \rightarrow b') \rightarrow -- \text{map input to homomorphic images} \\
(p \rightarrow b' \rightarrow c') \rightarrow -- \text{solve the problem in the hom. imgs.} \\
((p, c) \rightarrow (p, c')) \rightarrow (p, c) \rightarrow -- \text{combine the results} \\
(p, c) \rightarrow -- \text{neutral elem in the overall domain} \\
\text{Strategy} \ [c'] \rightarrow -- \text{strategy to generate parallelism} \\
b \rightarrow -- \text{input} \\
(p, c) -- \text{result}
\]

- Application: exact linear system solver
SymGrid-Par Performance

- Most performance results reported here use the **prototype** Coordination Server

- **Hardware:**
  - 28 node Beowulf cluster, 3GHz Intel Pentium 4, 512MB RAM
  - 8 core Dell PowerEdge 2950, 2.7GHz Intel Xeon, 16GB shared RAM
Sum the Euler totient function over a range of integers.

Moderately irregular as the totient function takes longer to calculate for larger integers

```haskell
sumTotient :: Int -> Int -> Int -> Int
sumTotient lower upper c =
sum (parMap (euler) (splitAtN c [lower..upper]))

euler :: Int -> Int
euler n = length (filter (relprime n) [1..n-1])

relprime :: Int -> Int -> Bool
relprime x y = hcf x y == 1
```
**sumEuler Example: GCA**

```haskell
sumTotient_GAP :: Int -> Int -> Int -> Int
sumTotient_GAP lower upper c =
    sum(parMap (euler_GAP) (splitAtN c [lower..upper]))

euler_GAP :: Int -> Int
euler_GAP n = gapObject2Int(gapEval "euler" [int2GAPObject n])
```
Direct/Recursive \texttt{sumEuler} Variants

- \texttt{sumEuler} would normally be written directly using iteration in GAP.
- A recursive GAP version is more directly comparable with the Haskell and GCA versions.

<table>
<thead>
<tr>
<th></th>
<th>GpH/GUM</th>
<th>GAP direct</th>
<th>GAP recursive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime</td>
<td>164s</td>
<td>500s</td>
<td>2298s</td>
</tr>
</tbody>
</table>

\textbf{Table}: Sequential \texttt{sumEuler} 32000
More Typical Example: **smallGroup**

- Searches for *groups* of a given order, with some property:
  - e.g. average order of their elements is an integer
- Two levels of irregularity
  - the number of groups of a given order varies by *5 orders of magnitude*!
  - the time to compute the conjugacy classes of each group.

![Graph showing the number of groups for different group orders](image-url)

- X-axis: Group Order
- Y-axis: Number of Groups

The graph illustrates the variability in the number of groups for different orders, highlighting the significant disparity seen across orders.
GCA smallGroup

smGrpSearch :: Int -> Int [(Int,Int)]
smGrpSearch lo hi = concat(map(ifmatch)(predSmGrp [lo..hi]))

predSmGrp :: ((Int,Int) ->(Int,Int,Bool)) -> Int ->[(Int,Int)]
predSmGrp (i,n) = (i,n,(gapObject2String (gapEval "IntAvgOrder"
    [int2GapObject n, int2GapObject i])) == ‘true’)

ifmatch :: ((Int,Int) -> (Int,Int,Bool)) -> Int -> [(Int, Int)]
ifmatch predSmGrp n= [(i,n) | (i,n,b) <-(masterSlaves predSmGrp
    [(i,n) | i<- [1 nrSmGrps n]],b]

nrSmGrps :: Int -> Int
nrSmGrps n=gapObject2Int(gapEval "NrSmallGroups" [int2GapObject n])
SymGrid-Par works with multiple CA systems: Maple, Kant, GAP, MuPAD etc

How does it compare with bespoke parallel CA systems?
## sumEuler [1..32000]

<table>
<thead>
<tr>
<th>PE</th>
<th>SymGrid-Par/GCA</th>
<th>ParGAP</th>
<th>GUM</th>
<th>Spd</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Recur Spd</td>
<td>Dir Spd</td>
<td>Recur Spd</td>
<td>Dir Spd</td>
</tr>
<tr>
<td>1</td>
<td>3006s 1</td>
<td>500s 1</td>
<td>3288 1</td>
<td>686s 1</td>
</tr>
<tr>
<td>2</td>
<td>1139s 2.6</td>
<td>320s 1.5</td>
<td>2117 1.5</td>
<td>481 1.4</td>
</tr>
<tr>
<td>4</td>
<td>542s 5.5</td>
<td>154s 3.2</td>
<td>1175 2.8</td>
<td>233 2.9</td>
</tr>
<tr>
<td>6</td>
<td>365s 8.2</td>
<td>107s 4.6</td>
<td>690 4.7</td>
<td>137 4.8</td>
</tr>
<tr>
<td>8</td>
<td>267s 11.2</td>
<td>84s 5.9</td>
<td>490 6.7</td>
<td>104 6.6</td>
</tr>
<tr>
<td>12</td>
<td>174s 17.2</td>
<td>59s 8.4</td>
<td>310 10.6</td>
<td>62 11.0</td>
</tr>
<tr>
<td>16</td>
<td>141s 21.3</td>
<td>51s 9.8</td>
<td>223 14.7</td>
<td>44 15.5</td>
</tr>
<tr>
<td>20</td>
<td>115s 26.1</td>
<td>45s 11.1</td>
<td>166 19.8</td>
<td>34 20.1</td>
</tr>
<tr>
<td>28</td>
<td>95s 31.6</td>
<td>40s 12.5</td>
<td>115s 28.5</td>
<td>25s 27.4</td>
</tr>
</tbody>
</table>
Recursive \textbf{sumEuler} Speedups
### Performance of generic SymGrid-Par is comparable with bespoke parallel CA systems

<table>
<thead>
<tr>
<th>PE</th>
<th>GCA</th>
<th>Spdup</th>
<th>Spdup (GAP)</th>
<th>ParGAP</th>
<th>Spdup</th>
<th>ParGAP/GCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>829s</td>
<td>1</td>
<td>1.1</td>
<td>1312s</td>
<td>1</td>
<td>1.5</td>
</tr>
<tr>
<td>2</td>
<td>416s</td>
<td>1.9</td>
<td>2.1</td>
<td>1421s</td>
<td>0.92</td>
<td>3.4</td>
</tr>
<tr>
<td>4</td>
<td>206s</td>
<td>4.0</td>
<td>4.4</td>
<td>549s</td>
<td>2.39</td>
<td>2.6</td>
</tr>
<tr>
<td>8</td>
<td>104s</td>
<td>7.9</td>
<td>8.7</td>
<td>185s</td>
<td>7.09</td>
<td>1.7</td>
</tr>
<tr>
<td>12</td>
<td>70s</td>
<td>11.8</td>
<td>13.0</td>
<td>105s</td>
<td>12.5</td>
<td>1.5</td>
</tr>
<tr>
<td>16</td>
<td>53s</td>
<td>15.6</td>
<td>17.2</td>
<td>79s</td>
<td>16.6</td>
<td>1.5</td>
</tr>
<tr>
<td>20</td>
<td>42s</td>
<td>19.7</td>
<td>21.7</td>
<td>64s</td>
<td>20.5</td>
<td>1.5</td>
</tr>
<tr>
<td>24</td>
<td>36s</td>
<td>23.0</td>
<td>25.3</td>
<td>53s</td>
<td>24.7</td>
<td>1.5</td>
</tr>
<tr>
<td>28</td>
<td>31s</td>
<td>26.7</td>
<td>29.4</td>
<td>45s</td>
<td>29.1</td>
<td>1.4</td>
</tr>
</tbody>
</table>
symGroup [1..400]

<table>
<thead>
<tr>
<th>PE</th>
<th>GCA</th>
<th>Spd</th>
<th>Spd GAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1,377s</td>
<td>1.2</td>
<td>1.2</td>
</tr>
<tr>
<td>2</td>
<td>698s</td>
<td>1.9</td>
<td>2.3</td>
</tr>
<tr>
<td>4</td>
<td>360s</td>
<td>3.8</td>
<td>4.6</td>
</tr>
<tr>
<td>6</td>
<td>243s</td>
<td>5.6</td>
<td>6.8</td>
</tr>
<tr>
<td>8</td>
<td>186s</td>
<td>7.4</td>
<td>8.9</td>
</tr>
<tr>
<td>12</td>
<td>132s</td>
<td>10.4</td>
<td>12.5</td>
</tr>
<tr>
<td>16</td>
<td>105s</td>
<td>13.1</td>
<td>15.7</td>
</tr>
<tr>
<td>20</td>
<td>89s</td>
<td>15.4</td>
<td>18.6</td>
</tr>
<tr>
<td>28</td>
<td>73s</td>
<td>18.8</td>
<td>22.7</td>
</tr>
</tbody>
</table>

- The skeletons deliver speedups even for problems with high levels, and multiple levels, of *irregularity.*
Generating New Knowledge: Summatory Liouville

▶ Search for positive values of the Summatory Liouville function

▶ The \textit{Liouville} function $\lambda(n) = (-1)^{r(n)}$, where $r(n)$ is the number of prime factors of $n$.

▶ The \textit{summatory Liouville’s function}, $L(x)$, is the sum of values of $\textit{Liouville}(n)$ for all $n$ from $[1..x]$. 

SymGrid-Par: Skeletons for Computational Algebra
Phil Trinder, SCIEnce Team, HPC-GAP Team

Context
SymGrid-Par Design
SymGrid-Par Skeletons
SymGrid-Par Performance
SymGrid-Par Dissemination
Discussion
Ongoing Work: SymGrid-Par2
Summatory Liouville

\[
L :: \text{Integer} \rightarrow \text{Integer} \rightarrow \text{Int} \rightarrow [\text{(Integer, Integer)}] \\
L \text{ lower upper c} = \text{sumL (myMakeList c lower upper)}
\]

\[
\text{sumL} :: [(\text{Integer, Integer})] \rightarrow [(\text{Integer, Integer})] \\
\text{sumL mylist} = \text{mySum ((masterSlaves liouville) mylist)}
\]

\[
\text{liouville} :: (\text{Integer, Integer}) \rightarrow \\
\quad [(\text{(Integer, Integer)}, \text{(Integer, Integer)})] \\
\text{liouville (lower,upper)} = \\
\quad \text{let} \\
\quad \quad l = \text{map gapObject2Integer (gapEvalN "gapLiouville" [integer2GapObject lower,integer2GapObject upper])} \\
\quad \quad \text{in} \\
\quad \quad ((\text{head l, last l}), (\text{lower,upper}))
\]
Summatory Liouville [1 .. 906150257] Speedups

Result almost immediately superceded by [ISSAC09]
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 \([1 \ldots 25 \times 10^6]\)]

<table>
<thead>
<tr>
<th>No PEs</th>
<th>Rtime</th>
<th>Spdup</th>
<th>CPU Utilis.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>526s</td>
<td>1</td>
<td>92.8%</td>
</tr>
<tr>
<td>2</td>
<td>264s</td>
<td>1.9</td>
<td>89.6%</td>
</tr>
<tr>
<td>3</td>
<td>178s</td>
<td>2.9</td>
<td>93.1%</td>
</tr>
<tr>
<td>4</td>
<td>132s</td>
<td>3.9</td>
<td>92.0%</td>
</tr>
<tr>
<td>5</td>
<td>106s</td>
<td>4.9</td>
<td>90.7%</td>
</tr>
<tr>
<td>6</td>
<td>89s</td>
<td>5.9</td>
<td>90.7%</td>
</tr>
<tr>
<td>7</td>
<td>76s</td>
<td>6.9</td>
<td>89.4%</td>
</tr>
<tr>
<td>8</td>
<td>68s</td>
<td>7.7</td>
<td>88.9%</td>
</tr>
</tbody>
</table>
8-core **smallGroup** [1...350]

<table>
<thead>
<tr>
<th>No PEs</th>
<th>Rtime</th>
<th>Spdup</th>
<th>CPU Utilis.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>480s</td>
<td>1</td>
<td>96.0%</td>
</tr>
<tr>
<td>2</td>
<td>246s</td>
<td>1.9</td>
<td>96.0%</td>
</tr>
<tr>
<td>3</td>
<td>165s</td>
<td>2.9</td>
<td>98.6%</td>
</tr>
<tr>
<td>4</td>
<td>125s</td>
<td>3.8</td>
<td>98.0%</td>
</tr>
<tr>
<td>5</td>
<td>104s</td>
<td>4.6</td>
<td>99.2%</td>
</tr>
<tr>
<td>6</td>
<td>91s</td>
<td>5.2</td>
<td>98.7%</td>
</tr>
<tr>
<td>7</td>
<td>82s</td>
<td>5.8</td>
<td>98.3%</td>
</tr>
<tr>
<td>8</td>
<td>76s</td>
<td>6.3</td>
<td>97.0%</td>
</tr>
</tbody>
</table>
Identifying Required Task Granularity

- Q: How coarse grained must the threads be to deliver a speedup?
- A: For some reasonable assumptions: approx. 70 Mcycles (approx. 30 ms on PowerEdge)
- Considered control & data parallel programs [DAMP09]
SymGrid-Par Dissemination

- Documentation
- Downloads
- Demos at CA Conferences, e.g. ISSAC’10
- Courses, e.g. Int. Summer School Symbolic Computation Linz 2008/09/10
Discussion

- SymGrid-Par aims to provide standardised parallelism for Computational Algebra systems
- Skeletons are a key part of the SymGrid-Par architecture
- There are problem-oriented, process-oriented, and domain-specific skeletons
- The skeletons are currently available from Maple, GAP, Kant & MuPAD
SymGrid-Par Performance Summary

- Performance of generic SymGrid-Par is comparable with bespoke parallel CA systems
- The SymGrid-Par skeletons deliver speedups even for problems high levels, 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
Ongoing Work

- Developing and disseminating SymGrid-Par
  - Implementing CA skeletons
  - Designing new CA skeletons
  - New applications of existing skeletons
  - Continuing dissemination
Ongoing Work

- Developing and disseminating SymGrid-Par
  - Implementing CA skeletons
  - Designing new CA skeletons
  - New applications of existing skeletons
  - Continuing dissemination
- New project: HPC-GAP
  - Aims, *inter alia* to scale GAP to large architectures
  - £1.8M, 2009 - 13
  - UK EPSRC funded
  - Aberdeen, St Andrews, Edinburgh, Heriot-Watt Universities
SymGrid-Par2 Design Goals

- Scale: designing for $10^6$ cores
  - Core failure every 5 minutes $\Rightarrow$ reliability
  - Topology awareness

- Rapidly evolving architectures $\Rightarrow$ performance portability is crucial

- Layered design: explicit parallelism at top layer