Distributed Parametric Optimization with the Geneva Library

Dr. Rüdiger Berlich
Karlsruhe Institute of Technology
What is the Geneva library?

- Geneva (read "Grid-enabled evolutionary algorithms") is a C++ library of optimization algorithms targeted at large scale problems
  - Designed with parallelism in mind
  - Allows multi-threaded or distributed execution
  - Is generic and applicable to virtually any (technical?) problem domain

- Geneva is an Open Source project (search for "Geneva" or "Gemfony" on Sourceforge)
  - Covered by the Affero GPL v3
  - Still in beta (0.5.3)
  - Suggestion to use the SVN version – new release is imminent
What do we want from you?

- **We want to build a community that**
  - Tells us its needs
  - Applies the code to new problem domains
  - Lets us know what does *not* work

- **We want to build an Open Source business on Geneva**
  - Upcoming spin-off from KIT
  - We want to make Geneva known also to an audience in different industry domains
  - Talk to us if you've got parametric optimization problems you would like us to work on
  - See [http://www.gemfony.com](http://www.gemfony.com)
Definition: A first attempt

- Lat. „Optimum“ = „The Best“
  - So: An optimum is the best of all possible solutions
  - Optimization then means finding the best solution

- Strategy 1:
  - Search for all possible solutions of the problem
  - Define a criterion that lets you distinguish good from best solutions
  - Find the best solution in the set
Example: Buying a car

- You want a car that
  - Is fast
  - Doesn't cost too much
  - Has low emissions
  - Is well-designed

- Scientific approach for finding a quality measure:
  - Combination of all criteria
  - E.g. square root of the squared parameter sum
  - If we combine just emission and price this way, than the best solution would be a car that costs nothing and has 0 emission. Likely not what was intended
Problems with this approach

- Suitable combinations of parameters to a single quality criterion can be difficult to find
- Equal solutions for different parameter sets are possible
- Some parameters are very difficult to quantify (e.g.: design)
- Quality $Q=f(x_1,x_2,...,x_{10})$
  - Test of 20 values per parameter means $20^{10}$ calculations of $Q$
  - At 1 millisecond per calculation this means 324 years of calculation (1 proc.)
  - 200 values per parameter == ~ $10^{12}$ years
Problems with this approach

- It is usually not possible (and not useful!) to take into account every possible choice
- We have to make a pre-selection
- The best of all solutions can usually not be reached
Definition: Second attempt

- **Second Attempt:**
  - The optimum is the best *achievable* result
  - In comparison: The *ideal* is the best possible result (which is usually unknown)

- **Strategy 2:**
  - Identify all necessary parameters
  - Define (and test!) a suitable numeric quality criterion $Q$
  - Search for a maximum or minimum of $Q$, that is as good as possible (under all boundary conditions)
Optimization Methods

- **Mathematical methods:**
  - Unsuitable for many real world problems, as no analytical function is available

- **Gradient descents**
  - Basic algorithm
    - Very simple!
    - „Go“ into the direction of the steepest descent of the function
  - Makes use of the surface geometry of the quality criterion
  - Works in any number of dimensions
  - Work well for many problem domains
Optimization Methods

- Problems of gradient descents:
  - Local optima
  - Algorithm can get stuck
Optimization Methods

- **Evolutionary Strategies**
  - Start with a known solution
  - Search predominantly in the neighborhood for better solution
  - Allow also values that are further away with a lower likelihood
Optimization Methods

- Why „Evolutionary“?
  - „Child individuals“ are created from „parents“ (the best known individuals so far)
  - Children are different from their parents
  - The best children become parents of a new generation

- Ingo Rechenberg, 1973
  - Optimization of technical systems according to the principles of biological evolution.
  - Was initially done without computers

- Genetic Algorithms
  - Use bits instead of floating point values
Optimization Methods

- Evolutionary Strategies:
  - Can „jump“ over a barrier. Thus very robust wrt. „noise“ (many local optima in the quality surface)
  - Uses properties of the quality surface, just like gradient descents
  - Quite easy to parallelise

![Graph of f(x) = (cos(x^2) + 2) * x^2](image1)

![Graph of f(x, y) = (cos(x^2 + y^2) + 2) * (x^2 + y^2)](image2)
Cluster Computing, Grids and Clouds

- Since ca. 1992:
  - New trend to do distributed computation (not the least due to Linux !!)
  - Large cluster systems

- Since ca. 1998:
  - Distributed computing on a global scale
  - Grids + Clouds
  - KIT is a Tier-1 centre in the LHC Computing Grid

- Computing resources for optimization become available on a large scale
Distributed Optimization

- Advantages are obvious
  - Better results in shorter time
  - Applicable to problem areas that could so far not be addressed due to the sheer amount of resources that were needed
  - Latency is not an issue for optimization problems with long-lasting quality calculation.
Generic Applicability

No assumptions about the underlying optimization problem

Currently working on protein folding examples

Fitness measure: energy
The Geneva Library: Design

- **Linux**
  - Widely used in Cluster, Grid, Cloud
  - Freely available and stable
  - Wide array of development tools

- **C++**
  - Efficient (cmp. Java)
  - Operators (num.problems)
  - Object oriented
  - g++ is reference for cross-platform C++ Compiler

- **Eclipse / CDT**
  - Standard
  - GEclipse ...
Geneva: Implementation

- **Optimization**
  - Evolutionary Strategies, Genetic Algorithms
  - Soon Swarm, Gradient Descent

- **Local parallelization**
  - Threads
  - Implementation of a “broker“

- **External Communication**
  - Network: Boost.Asio, Broker

- **Generation of random numbers**
  - Most „valuable“ good
  - Done in a multi-threaded random-number server
Geneva: Boost makes life easy

- **Boost:**
  - Extremely portable
  - Contains implementation of upcoming C++ library standard
  - Boost::Function
    - Generalized function callbacks
  - Boost::Bind
    - Generalized parameter binding
  - Boost::Serialization
    - Serialization of C++ classes
  - Boost::Asio
    - Network programming in C++
  - Boost::Thread
    - Portable multi threading
  - Boost::lexical_cast
    - Gen. String conversion
  - Smart pointers
    - boost::shared_ptr<> avoids memory leaks without a garbage collector

---

```cpp
class test
public:
  test(){...}
};

main(){
  test *Test=new test();
  ostringstream ofs;
  boost::archive::xml_oarchive oa(ofs);
  oa << make_nvp("test",Test);
  cout ofs.str() << endl;
}
```
The Broker

GMemberBroker

STL::map

GBufferPort 1

GBuffer/Raw

GBuffer/Proc

GBufferPort 2

GBuffer/Raw

GBuffer/Proc

... ...

GBufferPort n

GBuffer/Raw

GBuffer/Proc

GAsioTCPConsumer

get()

GAsioClient 1

GAsioClient 2

GAsioClient 3

...

GAsioClient m

put(item)

put(item)

put(item)

ruediger@berlich.com / http://ruediger.berlich.com
The Broker

GMemberBroker

STL::map

GBufferPort 1
GBuffer/Raw
GBuffer/Proc

GBufferPort 2
GBuffer/Raw
GBuffer/Proc

GBufferPort n
GBuffer/Raw
GBuffer/Proc

GMemberBroker

GAsioClient 1

GAsioClient 2

GAsioClient 3

GAsioClient m

GTransferPopulation 1
get()

GTransferPopulation 2
get()

GTransferPopulation n
get()
GRANDOMFACTORY->setNProducerThreads(nProducerThreads);

// Set up a single parabola individual
boost::shared_ptr<GParabolaIndividual>
    parabolaIndividual(new GParabolaIndividual(parabolaDimension,
                                                 parabolaMin, parabolaMax, adaptionThreshold));

// Now we've got our first individual and can create a simple
// population with parallel execution.
GBoostThreadPopulation pop_par;
pop_par.setNThreads(10);

pop_par.push_back(parabolaIndividual);

// Specify some population settings
pop_par.setPopulationSize(populationSize,nParents);
pop_par.setMaxGeneration(maxGenerations);
// Calculation should be finished after maxMinutes minutes
pop_par.setMaxTime(boost::posix_time::minutes(maxMinutes));
// Emit information during every generation
pop_par.setReportGeneration(reportGeneration);
// The best parents have higher chances of survival
pop_par.setRecombinationMethod(rScheme);

// Do the actual optimization
pop_par.optimize();
class GParabolaIndividual
 :public GParameterSet
{
    /********************************************************************************
    friend class boost::serialization::access;

    template<class Archive>
    void serialize(Archive & ar, const unsigned int version) {
        using boost::serialization::make_nvp;

        ar & make_nvp("ParameterSet",
            boost::serialization::base_object<GParameterSet>(*this));
        // add all local variables here, if you want them to be serialized.
    }
    /********************************************************************************

    public:
    GparabolaIndividual() { /* nothing */ }

    GParabolaIndividual(std::size_t sz, double min, double max, boost::uint32_t as){
        boost::shared_ptr<GDoubleCollection> gdc(new GDoubleCollection(sz,min,max));
        boost::shared_ptr<GDoubleGaussAdaptor> gdga(new GDoubleGaussAdaptor(1.,0.001,0.000001,5));
        gdga->setAdaptionThreshold(as);
        gdc->addAdaptor(gdga);

        // Make the parameter collection known to this individual
        this->data.push_back(gdc);
    }

    GParabolaIndividual(const GParabolaIndividual& cp) :GParameterSet(cp) { /* nothing */ }
    // [...]

    GParabolaIndividual(GparabolaIndividual& cp) {
    /* nothing */
    }
const GParabolaIndividual& operator=(const GParabolaIndividual& cp) {
    GParabolaIndividual::load(&cp);
    return *this;
}

virtual GObject* clone() {
    return new GParabolaIndividual(*this);
}

virtual void load(const GObject* cp) {
    GparameterSet::load(cp);
}

protected:
virtual double fitnessCalculation() {
    double result = 0;
    std::vector<double>::const_iterator cit;
    boost::shared_ptr<GDualCollection> gdc_load = parameterbase_cast<GDualCollection>(0);
    for (cit = gdc_load->begin(); cit != gdc_load->end(); ++cit) result += std::pow(*cit, 2);
    return result;
};
Minimization with Geneva

[...]  
`h0xbfe7628c->Fill(35, 2992385.42928176);`
`h0xbfe7628c->Fill(36, 2982973.42688278);`
`h0xbfe7628c->Fill(37, 2975844.33568812);`
[...]  
`h0xbfe7628c->Fill(52, 2861077.65196095);`
`h0xbfe7628c->Fill(53, 2856462.27152501);`
[...]
Experiences made

- In the design of the application:
  - Virtually no dependency on underlying distributed environment. Design is thus applicable to both Grids and Clouds.
  - Latency not an issue.
  - But had to resort to largest common denominator of all target systems: Communication happens through sockets in the current implementation.
    - Not a large restriction.
    - Design is modular: Hence other communication models can be added easily.

- In the design of the business model:
  - Mostly a „traditional“ Open Source business model, but with the added complexity of having to communicate with vendors of distributed resources, in addition to buyer and community.
I want to thank the audience and the organizers of this event!

Helmholtz society, the German federal ministry of education and research BMBF, the departments MAP/SI and SCC have supported my work – thanks!!

If you find our approach interesting: Talk to us!