Parametric optimization in parallel and distributed environments

Dr. Rüdiger Berlich
Mannheim CS Colloquium
Questions

Who in this room has heard before the term parametric optimization before this talk?
Questions

Who in this room has used parametric optimization to improve the results of his/her work?
Setting the scene

Geneva

(Grid-enabled evolutionary algorithms)

- Parallel optimization of problems from scientific and industrial domains
- Covering multi-core machines, clusters, Grids and Clouds
- Implemented in portable C++ (usage of ext. libraries limited to Boost)
- Version 0.82 will be released today (see http://launchpad.net/geneva)
- Open Source: Covered by the Affero GPL v3
Normally this presentation would have started
With an introduction to my home institution
– Karlsruhe Institute of Technology –

However:
Modelling the Mona Lisa

Subject of the optimization:
- Alpha-channel, coordinates and colors of 300 triangles
- Means that suitable values for 3000 variables must be found, with no known start-value
- Triangles should be super-imposed in such a way that they resemble the Mona Lisa
Protein Folding

Plots created with the Jmol molecular viewer
Engineering and Simulations

- Optimization of combustion engines
- Simultaneous calibration of large amounts of parameters
- Optimization of „const. parameters“ in simulations (weather, social, ...)
Elementary particle physics

Some examples:

- Partial wave analysis (see poster of Mathias Michel et.al.)
- Optimizing cuts (maximization of a peak's significance by varying cut parameters)
- Callibrating detector responses
  - Simultaneous optimization of very large numbers of parameters
Minimizing the error function of a feed forward neural network is a typical optimization problem.

Shown here:
- Two overlapping data distributions needed to be distinguished
- The output values of the trained network are printed on top of the data distribution
- It is visible that the network achieves an almost optimal separation
Optimization problems can be found in just about every field of engineering, natural sciences as well as business and economic sciences (and every other part of life)
Many can be described in terms of a set of parameters (e.g. floating point, integer, boolean) and an evaluation function that assigns a (usually numeric) quality to them.

\[(x_1, x_2, \ldots, x_n) \rightarrow f(x_1, x_2, \ldots, x_n)\]
So: This is very much like searching for maxima and minima of mathematical functions, right?
So why can't we just apply well-known mathematical algorithms?

Yes, indeed. There are many similarities between mathematical searches for maxima and minima and general purpose parametric optimization.

But some differences still apply.
Differences

- Analytic mathematical functions
  - Usually themselves expressed in terms of other functions (exp, sin, cos, ...)
  - At least subsets can be easily visually inspected
  - Well known methods for searching maxima and minima exist
  - Static once expressed as a formula

- General optimization problems
  - Usually expressed as a computer program or function
  - Impossible to apply analytic mathematical methods directly
  - Often discontinuous
  - Can depend on external boundary conditions
  - It can be difficult even to the expert to understand, what changes of parameters yield which change in quality
Some similarities

- There can be any number of local optima
- There can be many global optima (although more often there is just one)
- Some „traditional“ algorithms for searching minima/maxima of mathematical functions can be adapted to fit parametric optimization
Why brute force doesn't work

- Imagine an optimization problem with 100 parameters
  - Remember: There are many much larger problems
- Let us assume that the evaluation of a single parameter set takes 1 second on a single CPU core
- Now try out just two values per dimension / parameter
  - Means evaluation of 2 to the power of 100 parameter sets
- Equivalent to approx. 40000000000000000000000 years of calculation on a single core
- And noone tells you that the best solution is anywhere near those two parameters you tried
Defining the term „optimization“

- Realistic approach:
  - Optimization refers to the search for the *best achievable result* under a set of constraints
  - In comparison: „The ideal“ solution is the *best possible result*
    - Usually not practical: Imagine 3000 parameters, test 2 values each. Means computation of $2^{3000}$ parameter sets

- Strategy:
  - Identify all relevant parameters, including constraints
  - Assign a (computable) evaluation criterion to the parameters
    - Encapsulates experts knowledge
  - Search for maxima and minima of the criterion using one of many different optimization algorithms
    - Generic approach, applicable to many different problem domains
A simple solution

- Need to rely on other properties of the evaluation procedure that are more easily accessible
  - We can sample the surface
  - Thus we can make approximate statements about the shape of the surface in the near proximity
- Simple idea: „Walk down-hill“
- In mathematical terms: „Gradient descent“
- But: Need to make approximation

$$\frac{\partial f}{\partial x} \rightarrow \frac{f(x_2) - f(x_1)}{x_2 - x_1}$$

This will fail!
Easy and difficult local optima

Easy

Difficult

Final result / 20000

4000
Evolutionary strategies

- **Algorithm:**
  - Population of parents (best known solutions) and children
  - Cycle of duplication, mutation, selection
  - Mutation usually through addition of gaussian-distributed random numbers

- **Advantages:**
  - Tolerant wrt. local optima
  - Compute time scales with size of the population
  - Easy to parallelise

- **Disadvantages**
  - Can be slower than gradient descent for smaller problems
  - Many configuration options (e.g. width of gaussian)
Evolutionary Algorithms: Minimizing the Rastrigin function

Rastrigin / iteration 0 / fitness = 76.7586

Done with Geneva; Plot created with the ROOT framework
Evolutionary Algorithms: Minimizing the Rastrigin function

Rastrigin / iteration 1 / fitness = 19.7801

Done with Geneva; Plot created with the ROOT framework
Evolutionary Algorithms: Minimizing the Rastrigin function

Rastrigin / iteration 2 / fitness = 10.0394

Done with Geneva; Plot created with the ROOT framework
Evolutionary Algorithms: Minimizing the Rastrigin function

Rastrigin / iteration 3 / fitness = 4.56426

Done with Geneva; Plot created with the ROOT framework
Other optimization algorithms

- Swarm algorithms
  - Members of „neighborhoods“ of candidate solutions are drawn in each iteration towards
    - The globally best solution
    - The best solution of the neighborhood
    - A random direction
  - Swarm algorithms have recently been added to Geneva (alongside gradient descents)

- Further interesting algorithms:
  - Deluge algorithms / Simulated Annealing
  - Line search, Simplex, ...

(Source: Wikipedia; Author Mila Zinkova; published under the Creative Commons license „Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Unported“)
Swarm Algorithms: Minimizing the Rastrigin function

Done with Geneva; Plot created with the ROOT framework
Swarm Algorithms: Minimizing the Rastrigin function

Rastrigin / iteration 1 / fitness = 1.34419e+06

Done with Geneva; Plot created with the ROOT framework
Swarm Algorithms:
Minimizing the Rastrigin function

Rastrigin / iteration 2 / fitness = 15951.3

Done with Geneva; Plot created with the ROOT framework

Picture: Wikipedia (public domain)
Swarm Algorithms: Minimizing the Rastrigin function

Rastrigin / iteration 3 / fitness = 15951.3

Done with Geneva; Plot created with the ROOT framework
Swarm Algorithms:
Minimizing the Rastrigin function

Rastrigin / iteration 4 / fitness = 4337.76

Done with Geneva; Plot created with the ROOT framework
The examples above were calculated with the Geneva library of optimization algorithms
Our assumption

- Geneva wants to provide users with an environment that lets them solve optimization problems of any size transparently, as easily on a single core-machine as in the Grid or Cloud.
- Geneva targets optimization problems, whose figure of merit requires long-lasting computations.
- We assume that many very large scale optimization problems so far have not been targeted as
  - Typical single- or multi-core machines do not offer sufficient computing power
  - The complexities of running optimizations in parallel and/or distributed environments lead to assumption that performing such computations is not feasible
Design criteria

- Focus on long-lasting, computationally expensive evaluation functions
  - Stability of core library rated higher than efficiency
  - Suitable for distributed environments
- Serial, multi-threaded and networked execution, transparent to users
  - Implications of networked and multi-threaded execution:
    - No global variables
    - User-defined data structures must be serializable
- Familiar interface
  - STL interface for data, individuals, populations, ...
- Fault tolerance of networked execution:
  - Algorithm must be able to repair itself in case of missing or late replies from clients
- Execution of clients in Grid and Cloud:
  - No push mode means: Server needs public IP, clients don't
- Easy, portable build environment:
  - CMake
- Quality assurance:
  - Unit-tests, based on Boost.Test library
  - Can be integrated into user code
Implementation

- **C++**
  - Efficient (cmp. Java)
  - Heavily uses Boost
- **So far largely Linux-based**
  - But: should be portable
- **Tested with Intel C++, var. g++**
- **Major components**
  - Repres. of parameter sets
  - Optimization framework
  - Parallelization and communication
  - Random number factory

With the upcoming version 0.85:

```cpp
int main(int argc, char **argv)
{
    GOptimizer go(argc, argv);

    // -------------------------------
    // Client mode
    if(go.clientRun()) return 0;

    // -------------------------------
    // Server mode

    // Create the first set of individuals.
    for(std::size_t p = 0 ; p<nParents; p++) {
        boost::shared_ptr<GParameterSet> functionIndividual_ptr
            = GFunctionIndividual<>::getFunctionIndividual();

        // Make the parameter collection known to this individual
        go.push_back(functionIndividual_ptr);
    }

    // Perform the actual optimization
    boost::shared_ptr<GParameterSet> bestFunctionIndividual_ptr
        = go.optimize();

    // Do something with the best individual
    // [...]  
    std::cout << "Done ...
return 0;
}
```
Boost

- Extremely portable C++ library collection
- Many components are reference implementations for the upcoming C++ library standard
- License (almost) free of Copyleft
- Many high-profile components
  - Boost::shared_ptr: Reference-counted Smart Pointer
  - Boost::Function: Generalised Callbacks
  - Boost::Bind: Parameter binding
  - Boost::Serialization: Object serialization
  - Boost::Asio: Networking, asynchr. IO
  - Boost::Thread: Portable Multithreading
  - Boost::Test: Portable Unit Testing
  - Boost::Lambda: Lambda expr. in C++
  - Boost::program_options: commandline and config. file parsing

See for yourself at http://www.boost.org
Implementation / Data representation (EA)

Parents

Children / candidate solutions

Population

Individual

(Sub-)Population

- Bit-Collection: [0, 1, 1, 0, 1, 0, 1]
- Double-Collection: [0.2, 1.1, 0.7, 8.0, 0.1, 3.3, 5.8]
- Integer-Collection: [0, 1, 3, 2, 5, -1, 10]
- Object-Collection: [Object, Object, Object, Object, Object, Object, Object]

Adaptor

- Smart-pointer

either/or

- Object: value 0.3
  l.bnd 0
  up.bnd 1
- GaussAdaptor
- GConstrainedDouble
Implementation:

Constrained values (e.g. GConstrainedDouble)

\[ f(x) = x - r^* (\text{max-min}) \]

\[ f(x) = x + (r-1)^* (\text{max-min}) + 2^* \text{max} \]

with \( r(x) = \text{floor}((x\text{-min})/(\text{max-min})) \)

---

Copyright Dr. Rüdiger Berlich and Forschungszentrum Karlsruhe Institute of Technology
Implementation: Broker

[Diagram showing a broker with multiple populations (Population 1, Population 2, Population n) and multiple clients (Client 1, Client 2, Client 3, Client m). The broker makes heavy use of Boost.Serialization.]
Using the Geneva library

- **Code example**
  - [http://www.launchpad.net/geneva](http://www.launchpad.net/geneva)
  - Try: Server and clients on laptop
  - Geneva is a toolkit – need to do some programming to perform optimization
  - Generally: need to specify evaluation function or run external evaluation executable

- **Running example**
  - See examples „GsimpleEA“ and „GSimpleSwarm“, part of the Geneva distribution
Performance

Nehalem system with 2 processors / 8 cores / hyperthreading
Performance: Amdahl's Law

- Roughly:
  - Speedup scales with the percentage of parallel execution time of the overall application runtime
  - Strong scalability constraints
    - Need very high percentage of parallel execution time to achieve significant speedup (as function of the number of parallel processing units)

\[
S = \frac{1}{(1 - P) + o(N) + \frac{P}{N}} \leq \frac{1}{1 - P}
\]

Author of picture: Bob Schwammerl
Performance: Scalability in a network

Multithreaded execution

Networked mode (local+cluster)

Average speedup

Evaluation time / individual [s]
Scalability: The 80-20 rule
Or: „The low hanging fruit“
Moving to a wide-area networking environment (Grid, Cloud)

- Geneva is Client/Server
  - Clients may have a private IP, work in pull mode. Server needs to be reachable, though
  - Server can repair itself in case of a lack of response
  - Late responses will still be considered in later iterations
  - Thus very suitable also for unreliable environments like Clouds

- Must take into account higher latency in WANs
  - Where 15-20 seconds of evaluation time will lead to close-to linear speedup in Cluster, deployment in a cloud environments makes sense for evaluation times beyond approx. 40 seconds (depending on the complexity of individuals – this example: 1000 parameters)
  - We observe „scheduling“ anomalies wrt. network performance similar to http://www.cs.rice.edu/~eugeneng/papers/INFOCOM10-ec2.pdf

- Data management in the cloud can be challenging
- Security is of course better in local clusters
- Otherwise no fundamental difference between cluster deployment and Amazon-style submission of Vms
- (EGEE-style) Grid deployment can be problematic due to very static environment
Upcoming Developments

- Currently implementing
  - GPGPU (based on OpenCL). Optimization algorithms are SIMD. Fits nicely
  - Simulated Annealing (can be expressed in terms of adaptors of individuals)
- Performance
  - Need to profile serialization (many tips from Boost community / Robert Ramey)
  - Reduce latencies
- Full Documentation with version 1.0 (to be released in a few weeks)
Summary

- Many low-hanging fruits for distributed optimization both in industry and science
- Deployment in Cluster/Grid/Cloud not only feasible, but highly useful
- Find further information about the Geneva library on http://www.gemfony.com
- Get the software from http://www.launchpad.net/geneva
- We are building a community. Please do contact us with your optimization problems, we are happy to help getting you started with Geneva
Thanks!

- I want to thank the audience and the organizers
- Steinbuch Centre for Computing as well as the department IMA of Karlsruhe Institute of Technology have supported my work – thanks a lot!
- Similarly, I want to thank the Helmholtz Society of German research centres for their kind help
- The Enabling Grids for E-SciencE project has given this work a scientific home for a long time – thanks!!
Question ? Questions!

ruediger.berlich@kit.edu
http://ruediger.berlich.com