Gemfony scientific

Scaling the Geneva library collection to large HPC clusters

Dr. Rüdiger Berlich, Dr. Sven Gabriel, Dr. Ariel Garcia
Gemfony scientific UG (haftungsbeschränkt)

With special thanks to
Jan Knedlik, Prof. Dr. Matthias Lutz, Dr. Kilian Schwarz
of GSI Darmstad

15.03.2016

Contact:
contact@gemfony.eu

Who we are

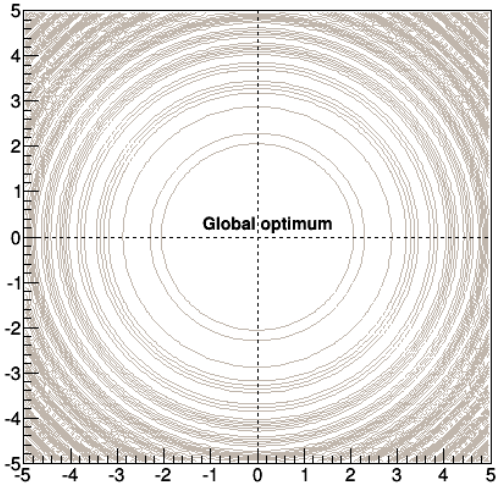
- Gemfony scientific
  - A spinoff from Steinbuch Centre for Computing at Karlsruhe Institute of Technology
- With particular experience in the fields of
  - Optimization of complex systems
  - Technical- and Science-Consulting
  - Implementation of IT-Solutions
  - Technical Marketing, PR and Training
- Long standing bacground in parametric optimiuation
  - Gemfony maintains the Geneva library collection of distributed optimization algorithms

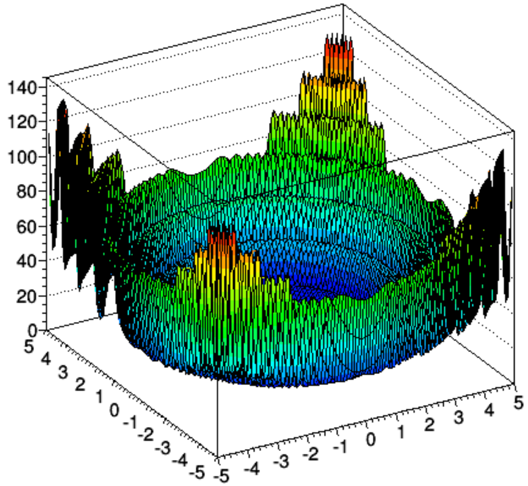
Dr. Rüdiger Berlich

Dr. Sven Gabriel

Dr. Ariel Garcia

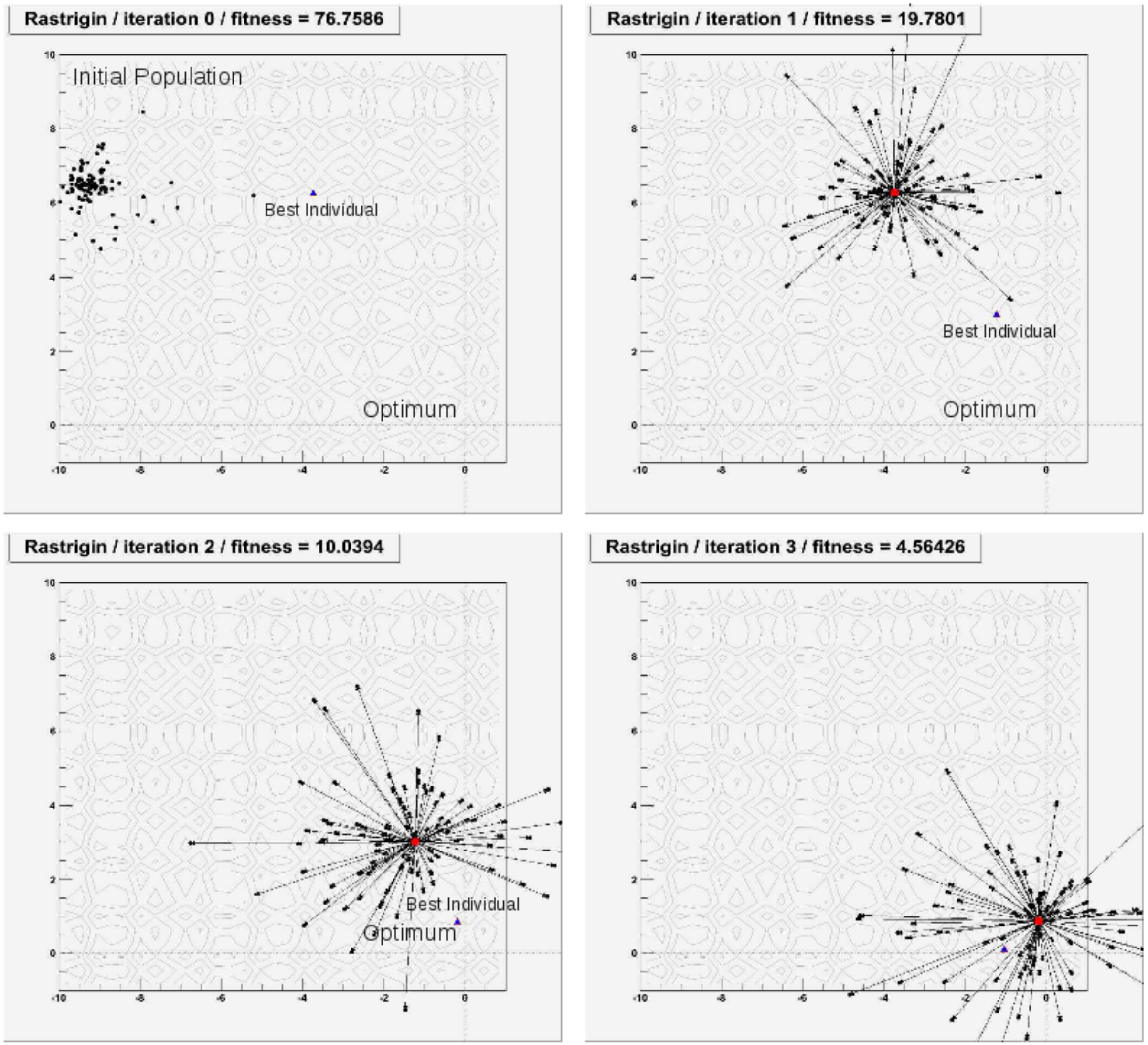
Parametric Optimization: Finding maxima or minina of $\vec{Q} = \vec{f}(x_1, x_2, \ldots, x_n)$





- Any mapping „f“ from input parameters to one or more numeric evaluations can be optimized
- For one evaluation criterion: optimization == finding suitable minima or maxima of solver „f“
  - Very similar to the search for extreme values of mathematical functions
  - „Solution space“ for multiple criteria
- However In the general case, „f“ will be a computer program
- Hence standard mathematical procedures cannot be applied easily
- Solvers may be computationally expensive
- As optimization algorithms will typically call the solver hundreds or thousands of times, such optimization problems will greatly benefit from parallelization

Parallelizability on the Example of Evolutionary Algorithms

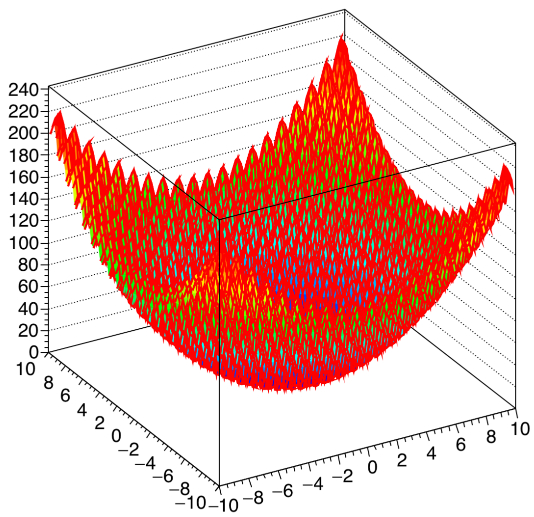

Rastrigin / iteration 0 / fitness = 76.7586

Rastrigin / iteration 1 / fitness = 19.7801

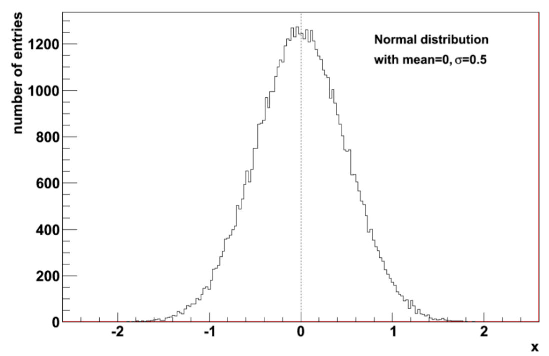
Rastrigin / iteration 2 / fitness = 10.0394

Rastrigin / iteration 3 / fitness = 4.56426

Surface plot of the Rastrigin function



„Gauss-Mutation“

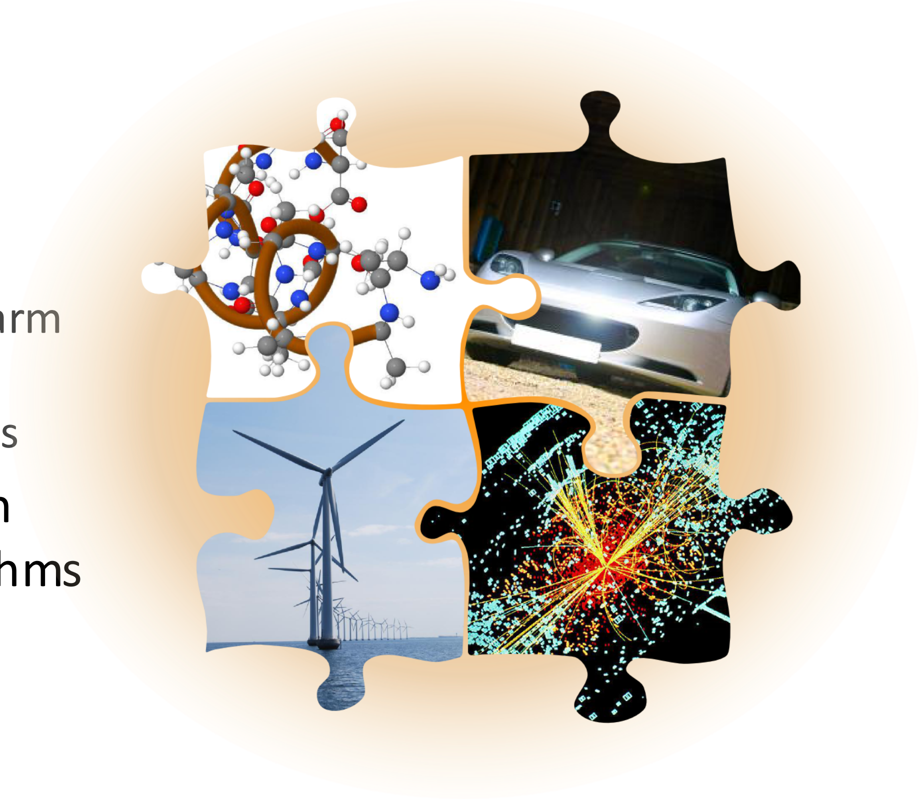


Why in „Massively Distributed Computing and Citizen Sciences“

- While this is probably not an exact fit:
  - Geneva is commercially supported Open Source (see http://www.launchpad.net/geneva)
  - Geneva particularly targets distributed and parallel execution
  - As optimization is a generic topic, application scenarios target just about every aspect of daily life
  - Free (simulation-)tools, along with cheap cloud resources, allow research to be performed by all scientifically interested parties

The Geneva Library Collection

- Generic solution for the search for **optimized solutions** of technical and scientific problems
- **„Metaheuristic“ Optimization**
  - Covering **Evolutionary Algorithms**, Swarm Algorithms, Simulated Annealing, Parameter Scans and Gradient Descents
- Data structures allow direct interaction between different optimization algorithms with **just one problem description**
- Written in portable C++
  - Uses the Boost library collection
  - Runs on different Unix variants (Linux, MacOS, ...) and Windows (experimental!)
- > 130.000 LOC (.hpp, .cpp, scripts, ...)



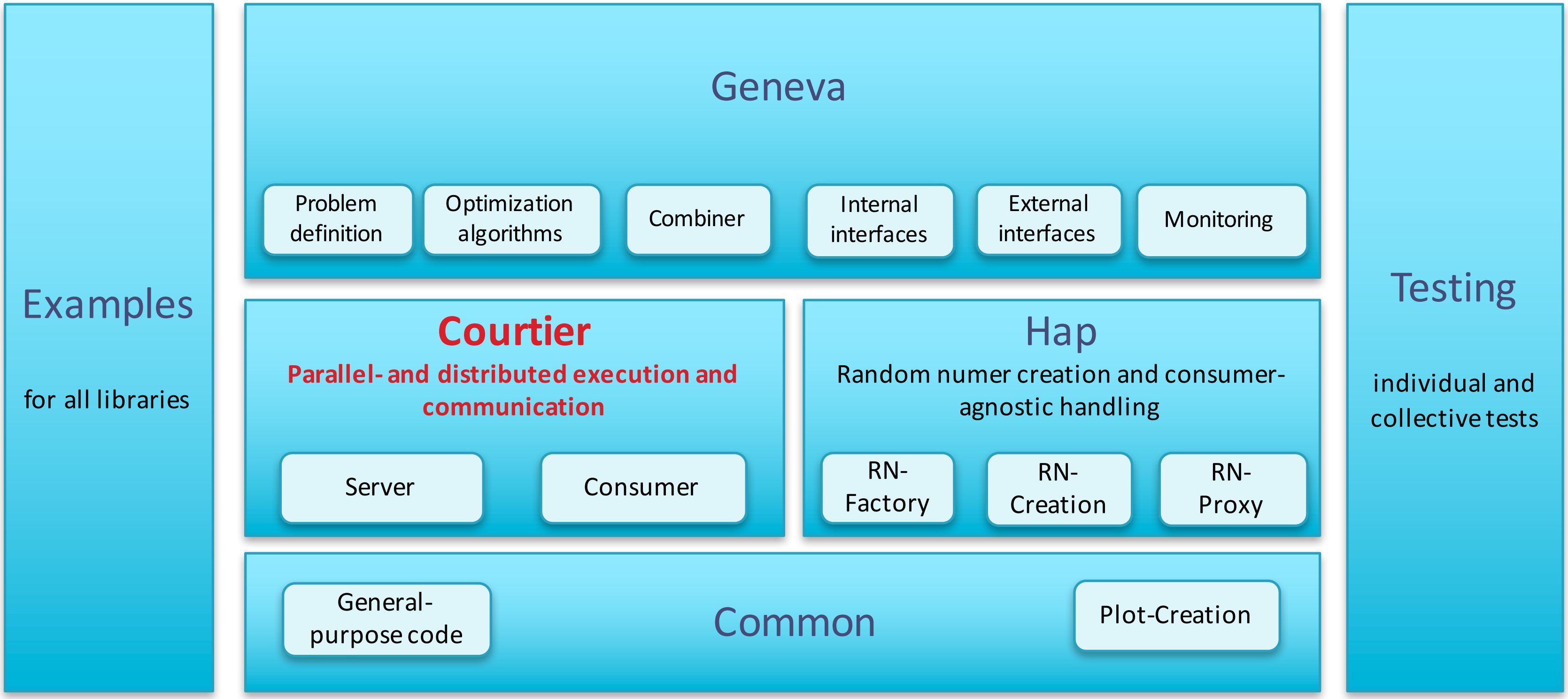
Sources:

- Car: Image courtesy of Simon Howden at FreeDigitalPhotos.net
- Wind turbines: http://www.flickr.com/photos/pebondestad/3533177131/sizes/l/in/photostream/ Creative Commons Attribution 2.0; By Pål Espen Bondestad
- Particle decay: https://en.wikipedia.org/wiki/File:CMS_Higgs-event.jpg Creative Commons Attribution Share-Alike 3.0; By CERN

Gemfony scientific

Library Components

Strong modularization allows for an efficient decoupling of evaluation and optimization!



Gemfony scientific

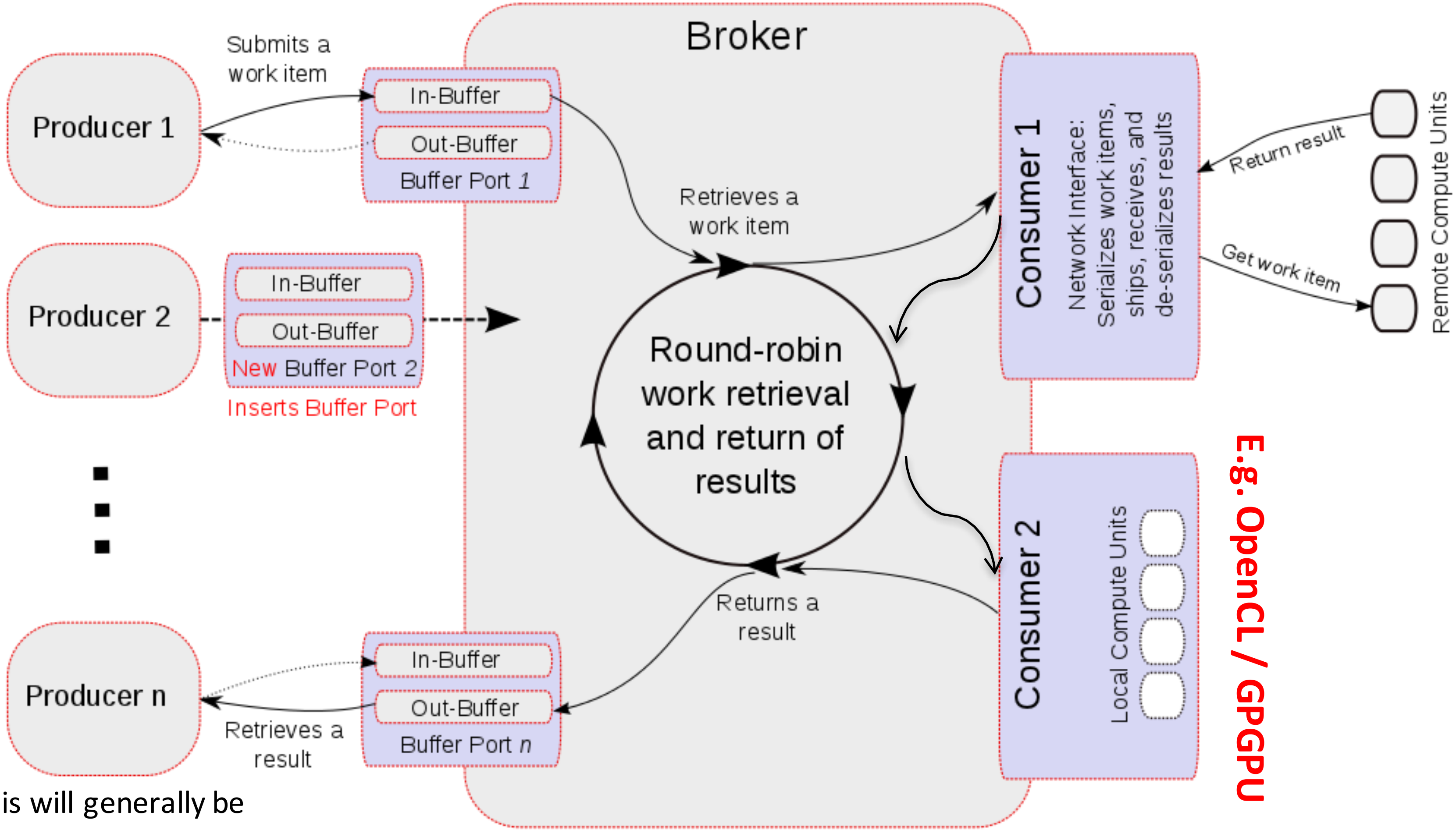
Source: Gemfony

Tuning Scalability

- „Performance“ is very problem-dependent
- In a nutshell, on the same hardware, performance improvements may be achieved in numerous ways, e.g.
  - Making the Geneva code more efficient
    - BUT: Focus on long-running evaluation-functions mandates focus on core-library stability rather than performance
    - Reducing run-time of the solver(s). But: task for the user
  - Making optimization algorithms converge faster
    - Minimization of Iterations needed to reach a given optimum
- But in particular: Parallelization of parallelizable parts
  - Reducing parallelization-overhead: „Amdahl“ may have a major impact on performance → Asynchronous transfer of candidate solutions
  - May need to cater for potentially thousands of clients, running for hours or days
- Reducing protocol overhead and improving stability is crucial

The „Courtier“ Library: Problem-Independent Parallelization

Source: Gemfony

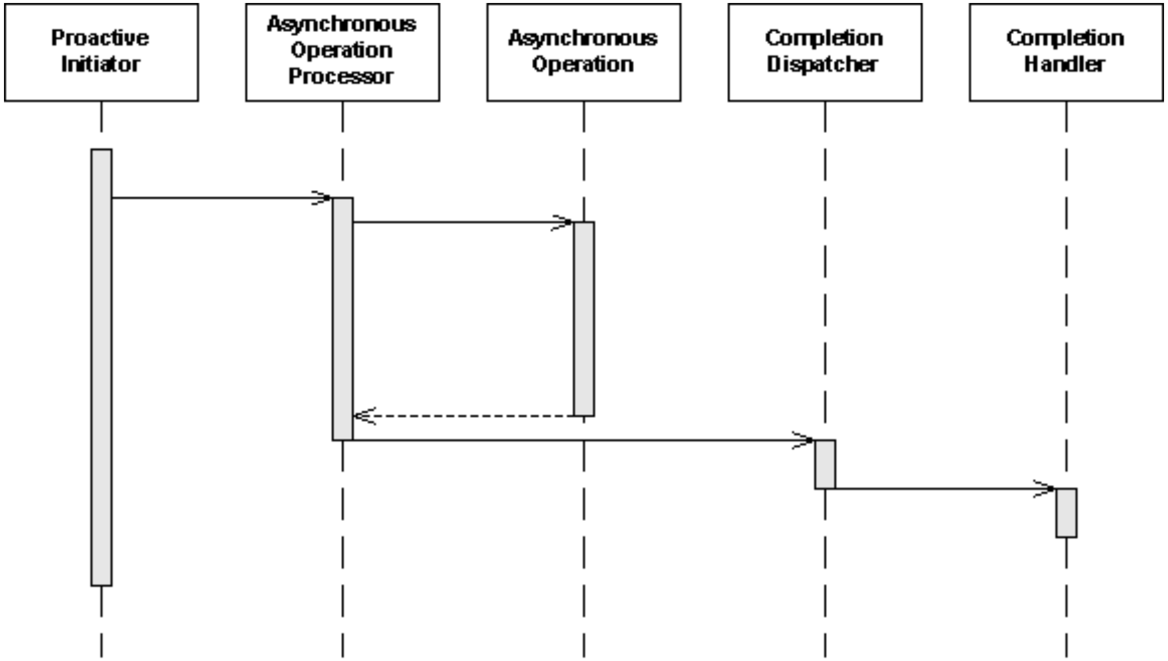


E.g. OpenCL / GPGPU

This will generally be
An optimization algorithm

Gemfony scientific

Proactor Pattern



- Boost.ASIO
  - Uses operating system parallelization
    - Thus very efficient
  - Will likely be part of the next standard C++17
  - Main usage: tcp networking in C++
    - Need to deal with TCP oddities on the lowest level

Picture source: Wikipedia „Proactor“, License CC-BY-SA 3.0, Author UlrichAAB

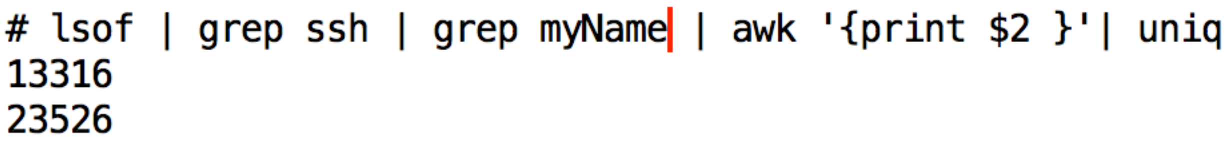
Dissecting the Network Mode (1)

- Current usage pattern in the release version (pull-mode!)
  - Client connects, retrieves work item, disconnects, does calculation, reconnects, transfers a result and retrieves the next work item
  - Many sockets are opened and closed
  - If calculation time and speed of clients is similar, connections may happen within a very small time window → long idle times for the server, followed by short periods of very high traffic
  - From the server-perspective, the problem may be quite similar to a high-load web-server
- Chosen, as, from the library perspective, there is no information about the length of an evaluation -- may take seconds or days
  - Server must cater for missing responses
  - Timeout must be calculated and possibly clients resubmitted, which may be problematic
- Scales to approx. 100 clients, but shows many problems beyond this

Dissecting the Network Mode (2) / How to tune a frequent-access Linux server ?

Follows http://www.lognormal.com/blog/2012/09/27/linux-tcpip-tuning/

- Simultaneously open files per process:
  - E.g. on MacOS: „ulimit –n“ returns 256. Even on a stock Ubuntu: only set to 1000
  - But every new socket requires a file handle ...
  - ... and even when a socket is closed, the handle is kept around, possibly for minutes, depending on the TCP/IP implementation
- Supply of “short-lived“ ports
  - By default 32768 – 61000 (range may be slightly increased)
  - Every new connection consumes a new port
    - E.g. two ssh connections to a server yields



```
# lsof | grep ssh | grep myName | awk '{print $2 }'| uniq
13316
23526
```

  - Unused ports are recycled ... after some time

Dissecting the Network Mode (3) / How to tune a frequent-access Linux server ?

- TIME_WAIT
  - Purpose: Packages returning after a connection is closed do not confuse TCP
  - May last long ... 2 minutes not uncommon
  - Affects open files, ephemeral ports
- iptables / Connection Tracking
  - iptables needs to allow two-way communication through the firewall
  - Needs to track connections
  - Keeps a list of connections, whose state is kept in a list
  - The list may not exceed a given limit
  - Will lead to silent failures if it does

Dissecting the Network Mode (4) / How to tune a frequent-access Linux server ?

- „nf_conntrack_tcp_timeout_established“
  - Timeout for established connections
  - Has very high default value of 432000 seconds
- Saturation of server by many simultaneous connections
  - De-Serialization in C++ (using Boost.Serialization) may be very costly
  - Must make sure (de-)serialization is disconnected from accepting new connections, or the server may not be responsive
- Queue-flooding in pull-mode
  - Where a timeout is reached, work items may need to be resubmitted.
  - Resubmission happens through a queue
  - Where timeout-values are not coupled (correctly) to the average compute time of clients, the queue will be flooded with work items
  - Need to make sure consumption rate is higher than submission rate

In summary, problems relate to …

- TCP/IP oddities
- Firewall deficiencies
- Internal architecture of Geneva

→ Not an easy problem

→ Need to reduce complexity!

→ Current design goal: Use a Websocket-type server architecture. May solve MANY of the above problems

Websockets (1)

- Client never disconnects
  - Reduces complexity on the TCP level
  - Reduces overhead for handshakes
  - May even allow a push-mode, without fear of queue-flooding
  - May inform clients about shutdowns (formerly they would have to terminate, when the server became unrechable)
- Must deal with TCP-timeouts
  - "Paylod-communication" may still only happen in very long intervals, as the client may just sit there, doing calculations
  - Will need a „heart-beat"

Websockets (2)

- Current implementation derived from „eidheim / Simple-Websocket-Server“ (see https://github.com/eidheim/Simple-WebSocket-Server)
  - MIT-licensed
  - Based on Boost.ASIO
  - Pure C++11
- Removed the pure „Websocket“ part, but kept part of the architecture

Websockets (3)

- Communication now happens on two levels:
  - Administrative (initial hand-shake, keep-alive in regular intervals)
  - Payload (exchange of work-items and results)
- Payload-protocol is implemented on top of the message-transfer
- Administrative and payload-messages enter the same queue → submission in the order they entered the queue
- Payload processing needs to happen in own thread-pool to keep client and server responsive

Summary

- High-throughput / high frequency TCP communication in C++ is CHALLENGING
- Current Design seems to be far more responsive
- Not yet part of a stable release
- Needs further tests
- Will be part of Geneva 2.0, which also represents a major redesign (C++14, simpler creation of new optimization algorithms, Broker-only architecture for parallelization, ...)

Thank you / Contact

Thanks
to the audience and the GSI team!

If you want to try Geneva:

http://launchpad.net/geneva

You may reach us at

contact@gemfony.eu

The project-team wants to thank Karlsruhe Institute of Technology, Steinbuch Centre for Computing as well as the Helmholtz-Association for supporting our work!

Masthead Gemfony

Address	Gemfony scientific UG (haftungsbeschränkt) Leopoldstr. 122 76344 Eggenstein-Leopoldshafen Germany
Telefone	+49(0)7247/934278-0
Fax	+49 (0)7247 934 2781
Email	contact@gemfony.eu
Registered at	Amtsgericht Mannheim (Germany)
Registration-Id	HRB 710566
Ust.-Id	DE274421406
Managing Director	Dr. Rüdiger Berlich