Advances in parallelization of cosmic rays simulations in CORSIKA 8

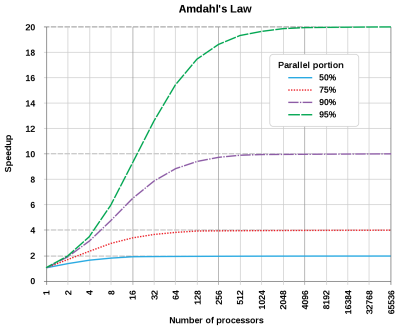
A. Augusto Alves Jr, P. Sampathkumar, R. Ulrich

Presented at DPG Spring Meeting - Heidelberg
March 15, 2022

Introduction: Amdahl's law

Predicts the expected speedup from parallelism:

> Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities
>
> Amdahl, Gene M.
>
> AFIPS Conference Proceedings (30): 483-485 (1967)
> doi:10.1145/1465482.1465560



It is expressed as

$$S(n) = \frac{1}{(1-p) + \frac{p}{n}}$$

where: $S(n)$ is the speedup in function of the number of cores/threads. n is number of cores/threads and p is the fraction of code that is parallelizable.

Calculation model

- Perform simulation in rounds, parallelizing the proceessing of the particles on each generation of the shower.
- Output is managed using side effects.
- Input data, RNG, geometry, filters etc are services available to modules, and accessible from the processing threads in read-only mode.
- The simulation manager thread can operates aside a IO manager thread, a monitoring thread etc. The worker threads are commissioned and released by the simulation manager thread.

In summary, it is necessary an efficient, flexible and self-balancing facility to manage the task distribution and execution, in a transparent and scallable way.

Gyges

`Gyges` is a lightweight C++20 header-only library to manage thread pooling.

- With `Gyges`, thread creation and destruction costs can be paid just once in the program lifetime.
- Threads from the pool pick-up tasks as they became available. If there is no task, the threads go sleeping.
- Tasks can be submitted from multiple threads.
- The submitter gets a `std::future` for monitoring the task in-place.
- Task assignment and running can be interrupted at any time.
- A `gyges::gang` can be created with any number of threads.

Status: Released. Already usable and available here:

`https://gitlab.iap.kit.edu/AAAlvesJr/Gyges`

`gyges::gang` interface

```
1  class gang
2  {
3      gang(unsigned int const thread_count=std::thread::hardware_concurrency(), bool release = true ) ;
4      gang( gang const & other ) = delete;
5      gang( gang && other )      = delete;
6
7      template<typename FunctionType>
8      inline std::future<void> submit_task(FunctionType f) requires Dispatchable<FunctionType>;
9      inline void stop(void);
10     inline std::size_t size(void);
11 };
12
13 template<typename Iterator, typename Predicate>
14 void for_each(Iterator begin, Iterator end, Predicate const& functor,  gang& pool);
15
16 template<typename Iterator, typename Predicate>
17 void for_each(Iterator begin, Iterator end, Predicate const&  functor);
```

The `concept Dispatchable` specifies signature `void operator()(std::stop_token token) noexcept`.

`gyges::concurrent_queue` interface

With this data structure, push and pop operations can be performed concurrently.

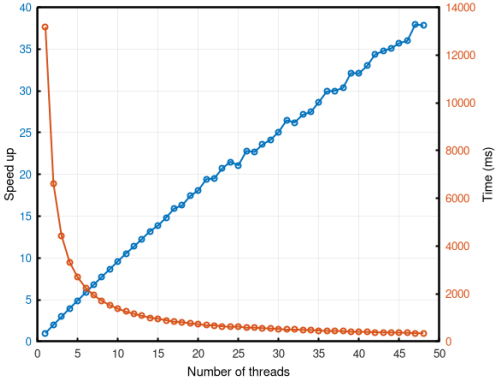
```
1  template<typename T>
2  class concurrent_queue
3  {
4      concurrent_queue();
5
6      concurrent_queue(const concurrent_queue& other)=delete;
7      concurrent_queue& operator=(const concurrent_queue& other)=delete;
8
9      std::shared_ptr<T> try_pop();
10     bool               try_pop(T& value);
11
12     std::shared_ptr<T> wait_and_pop(std::stop_token stoken);
13     void               wait_and_pop(std::stop_token stoken, T& value);
14
15     void push(T value);
16     bool empty();
17 };
```

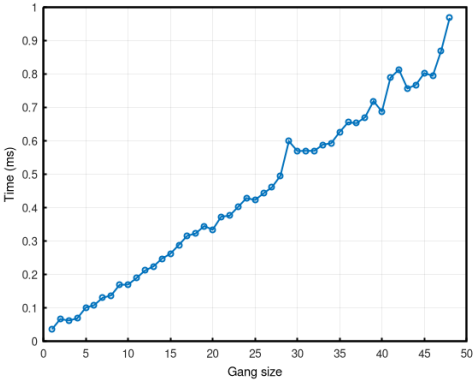
Performance and scalling

- The cost for creation and destruction of `gyges::gang` objects with diffent sizes has been measured using a `AMD Ryzen Threadripper 3960X 24-core/48-thread` processor running at 2.2 GHz with frequency boost enabled.
- Using same processor scalling behavior has been measured for the task below, performed 2048 times and distributed over different numbers of threads.

```
1  //number of random numbers to accumulate per task
2  unsigned nsamples = 1000000;
3  //seed
4  size_t seed = 0x1234abcd;
5  //functor
6  auto functor = [ seed, nsamples ]( double& x)  {
7
8          std::mt19937_64 generator( seed );
9          std::uniform_real_distribution<double> distribution(0.0, 1.0);
10
11         for(unsigned i=0; i < nsamples; ++i) x += distribution(generator);
12 };
```

Performance and scalling





Conclusions and prospects

- Low-level software infrastructure for parallelizing CORSIKA 8 is ready.
- Gyges is very small and concise package and has the required performance, scalability and self-balancing features.
- Gyges also provide `gyges::concurrent_queue`, which can substitute the current particle stack.
- Integration into CORSIKA 8 requires modules and other routines to be reimplemented in a thread-safe fashion.