Libcmaes automatically scales the number of threads according to the running platform capabilities (e.g. computer cores). There are two types of mulithreaded operations:

1. matrix / matrix and matrix / vector operations, through Eigen3
2. the objective function can be evaluated in parallel for several offsprings. This is usually highly beneficial when the number of offsprings reach above 100 individual or so when the objective function is not costly. When evaluation is costly, it is almost always beneficial. In order to activate the parallel evaluation:

```
int dim = 10; // problem dimensions.
double sigma = 0.1; // initial step-size, i.e. estimated initial parameter error.
std::vector<double> x0(dim,10.0); // initialize x0 as 10.0 in all 10 dimensions
CMAParameters<> cmaparams(dim,&x0.front(),sigma); // &x0.front() allows to turn the vector into a double*
cmaparams.set_mt_feval(true); // activates the parallel evaluation
```

VERY IMPORTANT: 2. requires that the objective function is designed so that it is thread-safe. Correct behavior cannot be certified otherwise, unless you either:

- tell the library to deactivate the parallel evaluation of the objective function:

```
cmaparams.set_mt_feval(false); // deactivates the multithreaded evaluation of objective function
```

- wrap the relevant section of the objective function code with protective mutexes;
- wrap the whole objective function with protective mutexes, here is an example:

```
#include <mutex>

FitFunc ff = [&](const double *x, const int N)
{
  std::lock_guard<std::mutex> lck(fmtx);
  double fval = (*fitnessfunction)(x);
  return fval;
};
```

where fitnessfunction() is the true objective function and ff is a wrapper around it, with a local protective mutex.