Gurobi 并行计算的设置和操作 v202412

help@gurobi.cn

编注：Gurobi 版本 12 对于 Python 增加了异步优化函数，因此 Python 语言实现并发优化既可以采用和其他高级语言一样的异步优化函数的方式，也可以沿用之前介绍的借助 Python Multiprocessing 库的方式。

Gurobi 是目前被科研学术界和企业界广泛采用的数学规划求解器，不但内置了多种先进算法，也保持了对计算机前沿硬件技术的密切跟踪。随着计算机硬件配置升级，计算能力不断提升，利用最新计算机硬件系统进行并行计算，已经是提升算法整体效率的不可缺少的方法。并行计算不但可以发生在单台电脑中的多核多线程当中，也可以发生在多台计算机组成的集群或者网络中。针对不同的硬件配置，以及不同的算法参数设置，Gurobi 用户可以创建多种并行计算方法。

Gurobi 在 www.gurobi.com 官网上提供了在算法设计层面不同算法（单纯形法，内点法，分支定界法等）和并行计算的紧密关系和适用程度的说明，有兴趣的用户可以下载视频和资料观看。链接是 https://www.gurobi.com/resource/parallelism-linear-mixed-integer-programming/

在这篇文章中，我们将从设置和操作的层面，介绍 Gurobi 几个并行计算的应用场景，解释一些并行计算的概念和操作方法。为了说明方便，我们归纳一张表格，显示了一个模型或者多个模型在一台电脑上，或者多台电脑集群上进行并行计算的方式。我们以混合整数模型为例。

Gurobi 中并行计算设置和操作				
	一台机器内单发	一台机器内并发	多台机器间分布	多台机器间并发
一个模型	默认设置。多线程跑一个模型	设置 ConcurrentMIP 参数，多线程跑同一个模型的多个复制模	需要分布式许可，区分管理机（1 台）和工作机（多台），	需要分布式许可，区分管理机（1 台）和工作机（多

		型	在管理机上设置 DistributedMIPJOb 参数，启动模型优化任务，让多台工作机共同运行一个模型	台），在管理机上设置 ConcurrentJobs 参数，启动模型优化任务，让多台工作机的每台机器跑同一个模型的复制模型
多个模型	创建一个 env,派生出多个模型，串行进行，完成一个模型再跑下一个模型。	不允许多个模型简单地同时运行。对于 C, C++, .Net, Java 等语言，创建多个 env，每个 env 创建一个模型，然后每个模型调用 optimizeAsync() 函数启动多个模型异步优化，需要调用模型状态信息判断模型是否终止。对于 Python，采用 Python Multi-processing 机制，创建多个并行进程。每个进程创建不同的 Env，然后由 Env 派生模型。Gurobi 版本 12 对于 Python 增加支持 optimizeAsync() 函数	需要分布式许可。多个模型串行时，参考上面单一模型运行方法。多个模型需要并行时，一个模型需要配置一台管理机。工作机可以共享，但不推荐。	需要分布式许可。多个模型串行时，参考上面单一模型运行方法。多个模型需要并行时，一个模型需要配置一台管理机。工作机可以共享，但不推荐。

（一）一台机器内单发

这是目前最常见的使用方式。

（1） 一个模型：大部分情况下，Gurobi 用户创建环境对象 Env（Python 语言提供默认的环境对象，用户无需显性定义），然后由 Env 产生一个模型对象，用户对于这个模型对象进行各种变量、约束和目标的添加和修改，最终通过运行 optimize()函数启动单个模型的优化。当模型优化时，Gurobi 会自动根据模型结构、求解阶段和 Threads 等参数设置来决定使用一个或者多个线程。用户无需做额外过多设置，这个模型就已经在调用 Gurobi 内部的并行计算算法。

（2） 多个模型：一个 Env 对象可以产生多个模型对象，在 Gurobi 中不允

许多个模型简单的同时并行计算，会产生不可预见的错误。用户可以依次串行运行，一个模型运行结束之后再运行另外一个模型。

（二）一台机器内并发

（1） 一个模型：Gurobi 允许在一台电脑内通过设置 ConcurrentMIP 参数，运行同一个模型的多个复制模型。这样的好处是用户可以为不同的复制模型设置不同的优化参数。多个复制模型在不同参数设置下同时运行，胜者决定最终速度。例如一台机器的核数是 16 核，ConcurrentMIP = 4，那么就会同时有 4 个同样的模型运行，每个模型占用 4 个核。

（2） 多个模型：之前提到在 Gurobi 中不允许多个模型简单的同时并行计算。当多个不同模型同时运行时，如果开发语言是 C,Java,C++,.Net,Python 等高级语言，可以采用 Gurobi 的异步优化函数；如果开发语言是 Python，还可以利用 Python 的多并发进程模块。具体使用方式如下。

如果开发语言是 C,Java,C++,.Net,Python 等高级语言，可以采用 Gurobi 的异步优化函数。当有多个模型时，需要为每个模型创建一个环境对象 Env，由该环境对象产生对应的模型，构造模型之后，调用 optimizeasync()启动异步优化。Gurobi 不用等优化结束，会将语句控制权直接跳到下个语句，用户可以启动第二、第三或者多个模型。用户可以不断查看模型当前优化状态，来判断模型优化是否结束。优化结束后，需要调用 sync()函数进行同步化，之后才能删除模型和环境对象。以下是一个 Java 示范案例。

```
/* Gurobi Example for Running Multiple Models in Parallel */
import gurobi.*;
public class GurobiParallel {
  public static void main(String[] args) {
    try {
      // Create three environments and start. One environment for one model
      GRBEnv env1 = new GRBEnv(true);
      env1.start();
```

```
GRBEnv env2 = new GRBEnv(true);
env2.start();
GRBEnv env3 = new GRBEnv(true);
env3.start();

// Create three models from mps files
GRBModel model1 = new GRBModel(env1, "misc07.mps");
GRBModel model2 = new GRBModel(env2, "glass4.mps");
GRBModel model3 = new GRBModel(env3, "p0033.mps");

// Set up parameters
model1.set(GRB.IntParam.Threads, 1);
model2.set(GRB.IntParam.Threads, 2);
model3.set(GRB.IntParam.Threads, 1);

// Start optimization
model1.optimizeasync();
model2.optimizeasync();
model3.optimizeasync();

// Check optimization status
while(true){
   int completed = 0;
 int status1 = model1.get(GRB.IntAttr.Status);
 if (status1 != GRB.Status.INPROGRESS) {
     System.out.println("Model 1 is completed!");
     System.out.println("The optimal objective is " +
             model1.get(GRB.DoubleAttr.ObjVal));
     completed ++;
 }

 int status2 = model2.get(GRB.IntAttr.Status);
 if (status2 != GRB.Status.INPROGRESS) {
     System.out.println("Model 2 is completed!");
     System.out.println("The optimal objective is " +
             model2.get(GRB.DoubleAttr.ObjVal));
     completed ++;
 }

 int status3 = model3.get(GRB.IntAttr.Status);
 if (status3 != GRB.Status.INPROGRESS) {
     System.out.println("Model 3 is completed!");
     System.out.println("The optimal objective is " +
             model3.get(GRB.DoubleAttr.ObjVal));
```

```
          completed ++;
      }

      if (completed == 3) break;

      try {
            Thread.sleep(500);
      } catch (InterruptedException e) {
            e.printStackTrace();
        }

      }

      model1.sync();
      model2.sync();
      model3.sync();

      model1.dispose();
      env1.dispose();

      model2.dispose();
      env2.dispose();

      model3.dispose();
      env3.dispose();

    } catch (GRBException e) {
      System.out.println("Error code: " + e.getErrorCode() + ". " +
                         e.getMessage());
    }
  }
}
```

如果开发语言是 Python，除了可以采用异步优化函数 optimizeAsync() 之外，还可以利用 Python 的多并发进程模块，为每个进程创建一个 Env 对象，然后由 Env 产生模型。多个模型在不同的进程内同时运行。以下是一个 Python 示范案例。

```
import multiprocessing as mp
import gurobipy as gp
```

```
def solve_model(input_data):
    with gp.Env() as env, gp.Model(env=env) as model:
        # define model
        model.optimize()
        # retrieve data from model

if __name__ == '__main__':
    with mp.Pool() as pool:
        pool.map(solve_model, [input_data1, input_data2, input_data3]
```

（三） 多台机器间分布（集群计算）

分布计算意味着多个计算资源共同运行同一个模型，而非一个模型的多个复制模型。对于基于分支定界的 Gurobi 混合整数模型而言，意味着多个计算资源作用于同一个搜索树的不同分支部分，相互协调。当模型的分支节点数量较大时，多台机器或者集群机可以有效地分担计算负载，加快搜索速度，提升求解模型的效率。

很多科研和企业配备有计算机集群，或者有数十台高性能计算机组成的计算网络，这些资源可以用来进行 Gurobi 分布式计算，增强复杂模型的计算能力。不论求解一个模型，还是多个模型，任何需要多台机器相互协调、分担负载、相互连通、同时运算的使用方式，都需要 Gurobi 的特殊分布式插件许可。

Gurobi 分布式计算需要配置一台管理机和多台工作机。管理机用于启动优化任务、配置工作机优化资源、协调和决定优化结果。而工作机则用于参与到分布式计算中。一般情况下，一台管理机启动一个优化任务。如果需要同时启动多个优化任务（多个并发模型），则需要配置多台管理机。

（1） 一个模型：在管理机上设置 DistributedMIPJOb 参数，启动模型优化任务，让多台工作机共同运行一个模型。这是典型的分布式计算方式。

（2） 多个模型：如果多个模型串行时，可以参考上面单一模型运行方法，

在管理机上依次串行启动模型。如果多个模型需要并行时，一个模型需要配置一台管理机。工作机可以共享，但不推荐。

（四） 多台机器间并发

除了让多台工作机运算同一个分支树的不同部分，Gurobi 分布式许可也允许每台工作机采用不同优化参数运行同一个模型的完整复制模型，哪台工作机速度快，哪台决定最终结果。

（1） 一个模型：在管理机上设置 ConcurrentJobs 参数，启动模型优化任务，让多台工作机的每台机器跑同一个模型的复制模型。

（2） 多个模型：如果多个模型串行时，可以参考上面单一模型运行方法，在管理机上依次串行启动模型。如果多个模型需要并行时，一个模型需要配置一台管理机。工作机可以共享，但不推荐。

总结：Gurobi 提供了多种灵活方式进行单发、并发和分布式计算。用户可以结合模型的特点，以及可调用的计算资源，进行配置和操作。如果使用过程中有任何问题，可以参考软件自带的使用手册和参考手册，或者发送邮件到 help@gurobi.cn 邮箱。