УДК 519.87

О РАСПАРАЛЛЕЛИВАНИИ АЛГОРИТМА ГЕНЕТИЧЕСКОГО ПРОГРАММИРОВАНИЯ ДЛЯ ЗАДАЧИ СИМВОЛЬНОЙ РЕГРЕССИИ

Становов В.В.,
научный руководитель д-р. техн. наук Семенкин Е. С.
Сибирский государственный аэрокосмический университет имени академика М.Ф. Решетнева

Алгоритм генетического программирования (ГП) является разновидностью интеллектуальных информационных технологий и может быть использован для решения широкого класса задач. В их числе задачи классификации, построения регрессии, анализа данных, а также определения структуры модели.

Широкому применению алгоритма генетического программирования, как и многих других эволюционных алгоритмов, препятствует их высокая вычислительная сложность и, как следствие, длительное время счета даже на современных ЭВМ. Ввиду повсеместного внедрения в последнее время компьютеров с несколькими процессорными ядрами, а также кластерных вычислений, актуальным вопросом видится возможность распараллеливания алгоритмов генетического программирования с целью ускорения счета. Однако стоит отметить, что написание параллельных приложений зачастую связано с множеством трудностей. Несмотря на то, что на сегодняшний день распространено множество программных средств и библиотек для автоматического распараллеливания, этот процесс остается нетривиальным и требует особого подхода в зависимости от структуры алгоритма.

Параллельным или многопоточным приложением называется такое приложение, которое использует несколько процессоров или ядер ЭВМ. Многопоточные программы отличаются тем, что в них необходимо синхронизировать работу различных потоков. Такая необходимость возникает вследствие того, что потоки работают с различной скоростью, даже выполняя один и тот же алгоритм. Такие эффекты возникают вследствие слабых колебаний тактовой частоты, работы операционной системы и некоторых иных факторов. То есть, если в некоторый момент времени существует необходимость собрать обработанные данные с нескольких потоков, то нужно ждать, пока все они завершат обработку. При этом существуют алгоритмы, которые невозможно распараллелить из-за того, что результат вычисления на каждом шаге зависит от результата на предыдущем шаге.

Эволюционные же алгоритмы, и в частности алгоритм ГП имеют дело с некоторым массивом индивидов, которые, как правило, обрабатываются независимо. По этой причине распараллеливание в данном случае видится достаточно простым.

Среди методов создания многопоточных приложений на языке C++ можно отметить функции *_beginthread*, *CreateThread*, набор библиотек *boost::thread*, а также стандарт *OpenMP* и многие другие. В данном случае для создания потоков использовалась библиотека *process.h* и функция *_beginthread*, так как, несмотря на некоторую сложность реализации, они позволяют производить тонкую настройку синхронизации работы потоков.

Замеры времени работы были проделаны на сервере с шестиядерным процессором Intel Xeon X5550 2.67ГГц. Тестирование ГП производилось по 100 запускам, при этом для задачи символьной регрессии число индивидов равнялось 200, поколений 200, максимальная начальная глубина 3. Алгоритм генетического программирования для решения задачи символьной регрессии был протестирован на двух функциях 1 и 2 переменных. Объем выборки – 251. При этом замерялось время работы каждого запуска алгоритма, а также полученное значение ошибки.

Тестирование производилось для одного, двух, трех и четырех потоков. На рисунке 2 представлены величины медиан времени работы алгоритма в зависимости от числа потоков, вычисленные по 100 прогонам алгоритма.

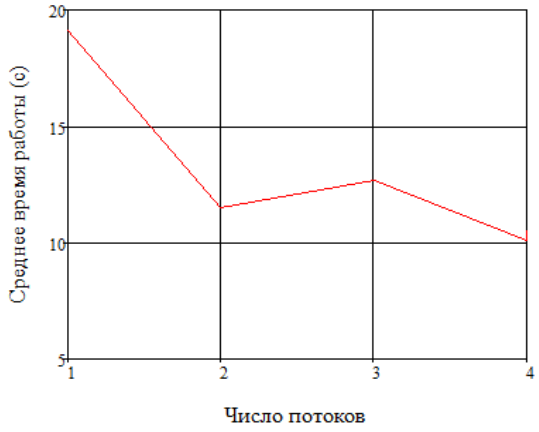

Рисунок 1. Зависимость времени работы от числа потоков, функция 1 переменной

Из графика можно увидеть, что существует значительное падение скорости работы при переходе от двух потоков к трем. Схожий график был получен для второй функции. График медиан приведен на рисунке ниже.

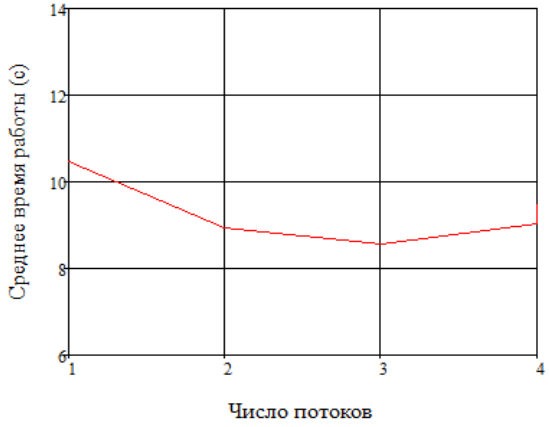

Рисунок 2. Зависимость времени работы от числа потоков, функция 2 переменных

В данном случае 4 потока показали эффективность на уровне двух. Из приведенных графиков можно сделать вывод, что какие-то процедуры в алгоритме реализованы неэффективно. При этом при переходе от одного к двум потокам время работы действительно значительно уменьшается. Внезапное же увеличение времени работы на 3 и 4 потоках может быть результатом неудачной реализации передачи данных между потоками.

Из полученных результатов можно сделать вывод, что суммарное время проведения операций передачи данных между потоками нивелирует то преимущество скорости, которое дает большее число потоков. При этом стоит отметить, что для разных тестовых функций подобное падение производительности, похоже, происходит при различном числе потоков. То есть из графиков, приведенных выше можно заключить, что, к примеру, тестируя на функции одной переменной отрицательный эффект можно наблюдать уже на трех потоках. Для функции двух переменных – на четырех потоках. Чем дольше вычисление функции пригодности, тем сильнее будет эффект от распараллеливания при увеличении числа потоков.