Параллельный алгоритм

Добавим еще пару тестов с 16 процессорами – test31.xml и test32.xml

|  |  |  |  |
| --- | --- | --- | --- |
|  | test3.xml | test31.xml | test32.xml |
| Кол-во проц. | 16 | 16 | 16 |
| Кол-во прогр. | 160 | 128 | 150 |
| Суммарный предел | 1420 | 1450 | 1360 |
| Средний предел | 88.75 | 90.6 | 85 |
| Суммарная нагрузка | 1190 | 955 | 1155 |
| Средняя нагрузка | 74.4 | 60 | 72.2 |

Протестировав программу на данных файлах получим, что среднее время выполнения при мультипроцессировании больше, чем при вычислении на одном процессе. Но это на самом деле логично. Одним из условий окончания базового алгоритма было превышение счетчиком итераций фиксированного значения 5000. С таким условием тяжело оценить эффективность параллелизма с помощью времени исполнения программы, поскольку при решении задачи для 16 процессоров большинство генерируемых решений не проходят проверку, алгоритм в большинстве случаев просто прерывается после 5000 итераций, а в случае мультипроцессирования еще затрачивается время на создание процессов и очередей для передачи информации. Также при мультипроцессировании перебирается больше вариантов за единицу времени, значит больше вероятность нахождения решения, следовательно счетчик итераций сбрасывается чаще и программа работает дольше.

Нужно найти другой параметр характеризующий производительность программы. Разберем несколько вариантов.

1. Оценить вероятность решения

В данном случае будем использовать файлы test3.xml, test32.xml для них решения находятся довольно редко. Запустим программу для каждого файла по 50 раз, сначала базовым алгоритмом, а затем параллельным.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Базовый | 2 процесса | 4 процесса |
| test3.xml | 6 / 50 | 9 / 50 | 17/50 |
| test32.xml | 2 / 50 | / | / |

1. Убрать ограничение по количеству итераций и оценить время нахождения первого решения.
2. Оценить качество решения