Математические структуры и моделирование
2012, вып. 26, с. 110–117

УДК 004.272.2:519.16

ГЕНЕТИЧЕСКИЕ АЛГОРИТМЫ С ЧАСТИЧНОЙ ПАРАЛЛЕЛИЗАЦИЕЙ В СИСТЕМАХ С ОБЩЕЙ ПАМЯТЬЮ НА ПРИМЕРЕ ЗАДАЧИ КОММИВОЯЖЁРА

К.В. Пугин, С.С. Ефимов

Данная статья является описанием опыта частичной параллелизации генетических алгоритмов для решения задачи коммивояжёра. Приведены примеры возможного кода, а также результаты тестов производительности различных вариантов алгоритма. Произведено сравнение вариантов реализации ГА как между собой, так и с эталонными вариантами.

Введение

Один из наиболее трудноразрешимых классов задач — NP-полные задачи. Одной из открытых математических проблем столетия является вопрос — можно ли построить полиномиальный алгоритм для их решения? Но пока ответа нет, и в данный момент для их приближенного решения используются всякие надстройки над полным перебором, одной из которых являются генетические алгоритмы. Генетические алгоритмы построены на основе подражания эволюции: каждый метод решения задачи кодируется набором хромосом (методы кодировки могут быть самыми различными), которые могут скрещиваться, мутировать, а также тестироваться на приспособленность при помощи функции решения задачи, которая называется фитнес-функцией.

Поскольку порядок выполнения операций в генетическом алгоритме строго детерминирован (отбор достойных $\rightarrow$ скрещивание $\rightarrow$ мутация $\rightarrow$ переход к следующему поколению), то не всегда их параллелизация осуществляется тривиально. Существует два подхода к распараллеливанию операций на генетических алгоритмах — полная параллелизация (у каждого потока имеется своя популяция, с которой он работает, и раз в несколько поколений они сообщаются), и частичная параллелизация (операции типа скрещивания, отбора и мутации выполняются параллельно, но слежение за поколениями происходит централизовано в главном потоке).

Цель данной работы — исследовать генетические алгоритмы с частичной параллелизацией на системах с общей памятью на примере условной и безусловной реализаций генетического алгоритма для решения задачи коммивояжёра.

Омский государственный университет им. Ф.М. Достоевского
E-mail: ria.freelander@gmail.com,s_efimov@mail.ru

Задачи — проанализировать доступную информацию по генетическим алгоритмам, возможности их параллелизации и применения, реализовать генетические и специализированные алгоритмы решения задачи коммивояжёра, исследовать их производительность в однопоточном и многопоточном вариантах.

1. Средства и возможности решения

Для работы использованы:

- Технические средства:
  - процессор Intel Core i5-2410m,
  - видеокарта NVIDIA GeForce 520m,
  - 4 GB RAM.
- Фреймворки и языки программирования:
  - C++ (Reference Genetic),
  - OpenMP,
  - CUDA 4.1 (Thrust 1.6),
  - Python (Testing frontends, non-performance parts).

В ходе работы было решено остановиться на частичной параллелизации с условным и безусловным скрещиванием, поскольку данные методы наиболее часто применяются в решении реальных задач при помощи генетических алгоритмов.

Реализовано два примера многопоточного решения задачи коммивояжёра при помощи генетического алгоритма на основе библиотеки Thrust. Один из них решает задачу коммивояжёра при помощи генетического алгоритма на основе маршрутов, выступающих там как хромосомы. Применено условное кольцевое скрещивание (постоянно проверяется условие отсутствия данного города в ребёнке, пункты маршрута выбираются с начального у первого родителя, с конечного — у второго), а в качестве мутации выступает полное перемешивание маршрута. Второй пример: применено классическое скрещивание, при этом каждая перестановка нумеруется в лексикографическом порядке. В качестве мутации выступает генерация случайного номера перестановки. Также эти примеры реализованы и в однопоточном варианте путём замены Thrust на STL.

Пример кода модуля алгоритма (условное скрещивание):

```
__host__ __device__ inline
  animal& animal::interbreed(animal& other)
{
  /* условие для выбора потомка */
  if(partial_calc(matrix,plist,len/2,len)<
     partial_calc(matrix,plist,len,len,len/2))
    for(size_t i=1,c1=len/2; ((i<len-1)&&(c1<len-1)); i++)
    {
      /* проверка на наличие поэлементно */
      if(!(in_place(plist,c1,other.plist[i])))
      {
        plist[c1]=other.plist[i];
        c1++;
      }
    }
  else
  {
    for(size_t i=1,c1=1; ((i<len-1)&&(c1<len/2)); i++)
    {
      /* также поэлементная проверка */
      if (!(in_place(plist,len,other.plist[i],c1)))
      {
        plist[c1]=other.plist[i];
        c1++;
      }
    }
  }
  /*
   *  скрещивание происходит с заменой одного
   *  из родителей для избегания выделения памяти
   *  (на CUDA операция занимает много времени)
   */
  return *this;
}
```

В качестве контрольного реализован алгоритм ближайшего соседа на языке Python, а также тестирующий модуль на том же языке, который подсчитывал время выполнения каждого алгоритма, а также результирующий маршрут и его длину. Каждому из алгоритмов на одной и той же итерации передавался один и тот же граф.

Пример кода (Тестирующий модуль):

```
def main():
  with open("testslv.txt", "w") as f: # Пишется в файл
    writer=("Usual:\tGenetic:\tGenetic(C++):
\tGenetic(OpenMP):\tGenetic(CUDA)\n")
    f.write(writer)
    for i in xrange(23, 10000):
      fgraph = Generate(i); #Генерируется граф
      fgraph_array=Translate(fgraph)
      timed=time.time()
      #Вызывается функция ГА на CUDA с измерением времени
      cuda=GeneticCUDA(fgraph_array, fgraph)
      timecuda=time.time()-timed
      timed=time.time()
      #Вызывается функция ГА на OpenMP с измерением времени
      openmp=GeneticOpenMP(fgraph_array, fgraph)
      timeopenmp=time.time()-timed
      timed=time.time()
      #Вызывается функция ГА на C++ с измерением времени
      geneticc=GeneticC(fgraph_array, fgraph)
      timesingle=time.time()-timed
      timed=time.time()
      #Вызывается функция ближайшего соседа с измерением времени
      usual=SolveUsual(fgraph, 0)
      timeusual=time.time()-timed
      timed=time.time()
      #Вызывается функция ГА на Python с измерением времени
      genetic=SolveGenetic(fgraph, 0)
      timegenetic=time.time()-timed
      with open("testslv.txt", "a+") as f:
        # Все времена пишутся в файл
        writer=("{0}\t{1}\n").format(timeusual, timegenetic,
                                     timesingle,timeopenmp,timecuda)
        f.write(writer)
```

2. Полученные результаты

Пояснения к графику:

1. Наибольшее время выполнения алгоритма на CUDA объясняется простотой фитнес-функции и сетевым законом Амдала.

2. OpenMP позволяет получить в n раз больший выигрыш во времени выполнения (на 4 потоках время сокращается в 4 раза).

Пояснения к графику: Аналогично графику 1, только время выполнения уменьшилось в связи с уменьшением числа ветвлений в алгоритме.

Пояснения к графику:

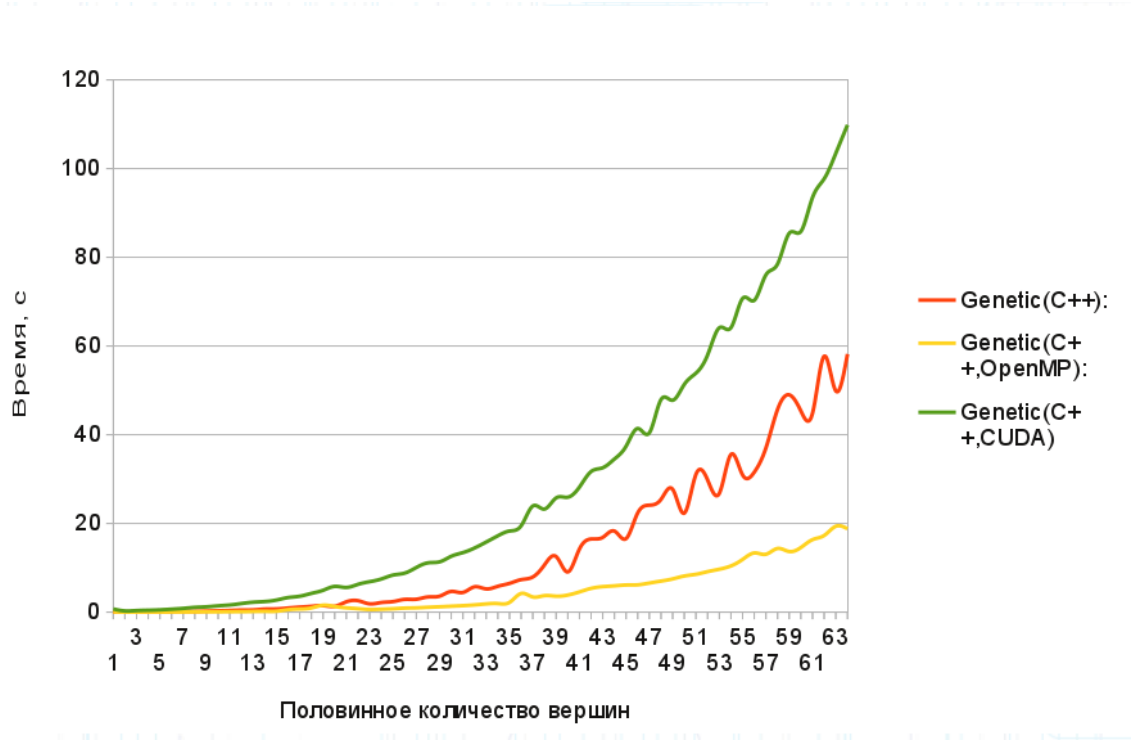

Рис. 1. График производительности условного генетического алгоритма в зависимости от вида параллелизации

1. Алгоритм ближайшего соседа проигрывает в точности генетическому алгоритму при применении на одних и тех же графах.

2. Согласно теореме, для любого количества городов большего трёх в задаче коммивояжёра можно подобрать такое расположение городов (значение расстояний между вершинами графа и указание начальной вершины), что алгоритм ближайшего соседа будет выдавать наихудшее решение [1].

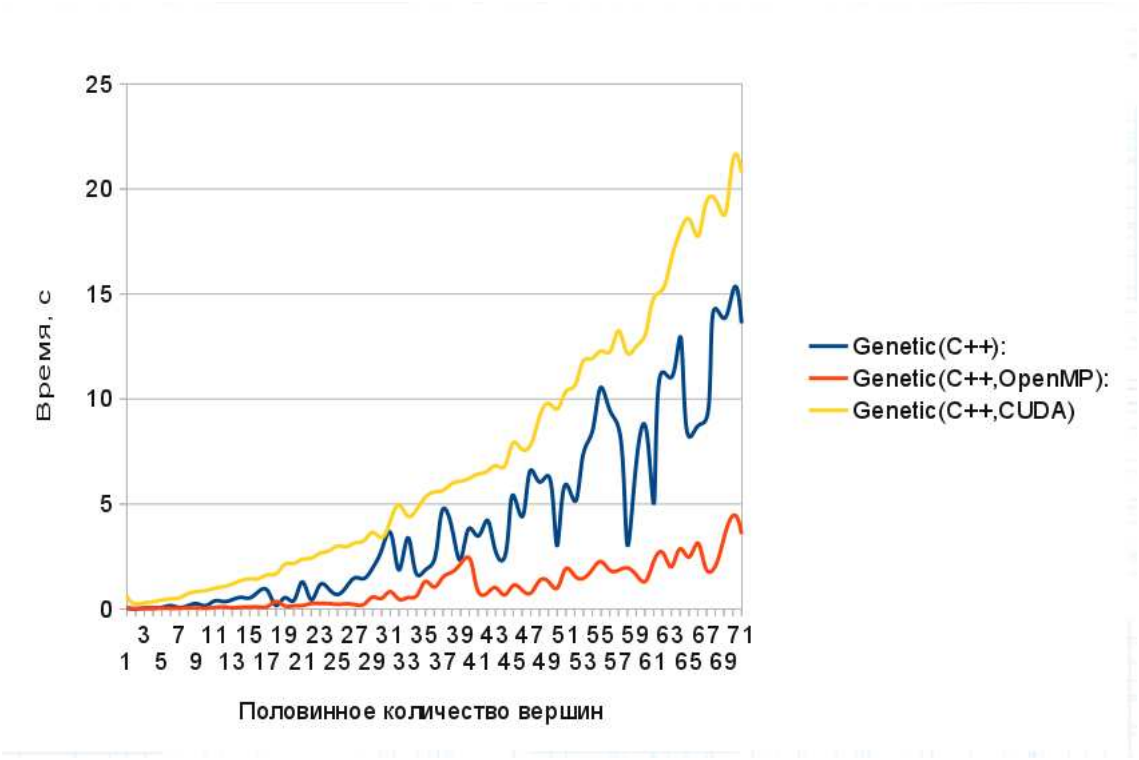

Рис. 2. График производительности безусловного генетического алгоритма в зависимости от вида параллелизации

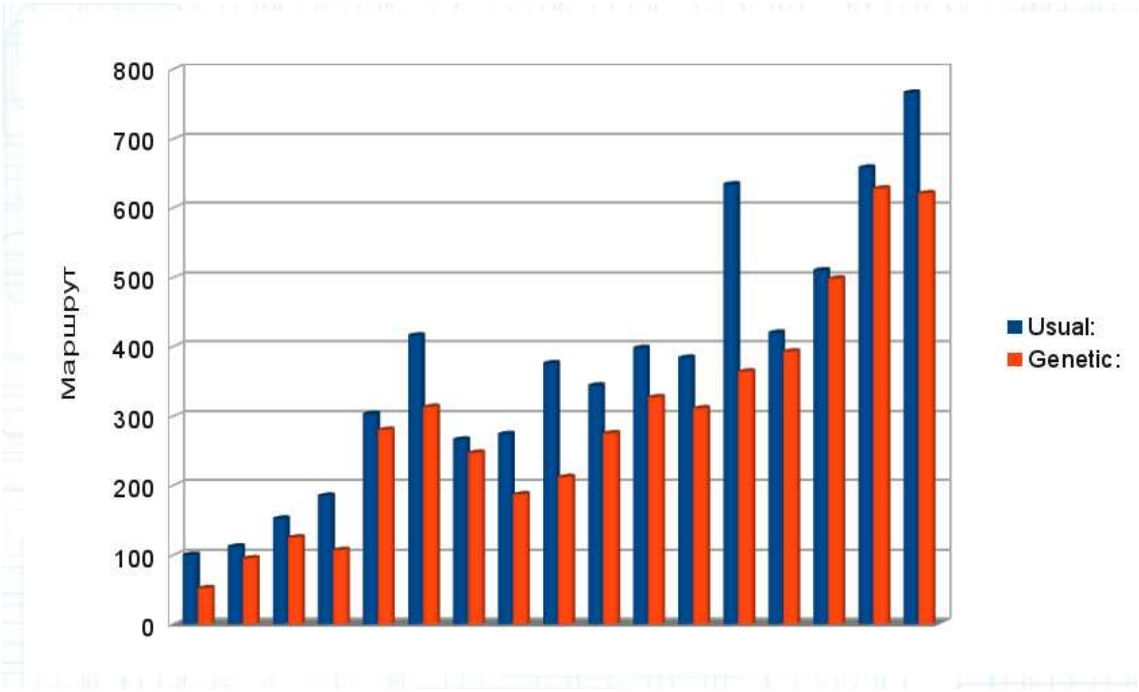

Рис. 3. График точности генетического алгоритма по сравнению с алгоритмом ближайшего соседа

Заключение и общие выводы

Выводы:

1. Применение генетического алгоритма для задачи коммивояжёра полностью оправдано.
2. Для систем параллелизации, использующих пересылки (в частности, для CUDA), критична вычислительная сложность параллельных частей.
3. Частичная параллелизация задачи коммивояжёра позволяет получить выигрыш производительности, сравнимый с количеством ядер.
4. Безусловный алгоритм быстрее для малых графов, но для большого числа вершин их номера не помещаются в восьмибайтную переменную, что влечёт неприменимость решения с нумерацией для большого числа вершин, так как при росте номера преимущества безусловного подхода практически теряются.

Перспективы расширения исследования:

1. Исследование полностью параллельных вариантов генетических алгоритмов с несколькими популяциями.
2. Более полное исследование влияния фитнес-функции и пересылок на производительность ГА.
3. Алгоритмическая оптимизация реализованных алгоритмов (изменение скрещивание под более ускоренный вариант либо под более точный вариант).
4. Программная оптимизация алгоритмов (написание более платформозависимого низкоуровневого кода).

Литература

1. Gutin G., Yeo A., Zverovich A. Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP. URL: `http://eprints.rhul.ac.uk/archive/00000356/` (дата обращения: 10.09.2012).
2. Debattisti S., Marlat N., Mussi L., Cagnoni S. Implementation of a Simple Genetic Algorithm within the CUDA Architecture. URL: `http://www.gpgpgpu.com/gecco2009/3.pdf` (дата обращения: 10.09.2012).
3. Van Luong Th., Melab N., Talbi E. GPU-based Parallel Hybrid Genetic Algorithms. URL: `http://www.gpgpgpu.com/gecco2010/0.pdf` (дата обращения: 10.09.2012).
4. Shah R., Narayanan P.J., Kothapalli K. URL: GPU-Accelerated Genetic Algorithms. URL: `http://cvit.iiit.ac.in/papers/Rajvi10GPU.pdf` (дата обращения: 10.09.2012).

5. Егоров К., Чураков М. Генетические алгоритмы. URL: http://yury.name/internet/03ia-seminar-note.doc (дата обращения: 10.09.2012).
6. Генетические алгоритмы. От теории к практике. URL: http://habrahabr.ru/post/138091/ (дата обращения: 10.09.2012).
7. Батищев Д.И., Неймарк Е.А., Старостин Н.В. Применение генетических алгоритмов к решению задач дискретной оптимизации. URL: http://www.unn.ru/pages/issues/aids/2007/15.pdf (дата обращения: 10.09.2012).