Поиск кратчайшего пути в графе

обзор существующих решений

Марьина Анна, 371 группа, мат-мех, СПбГУ

16.05.2017

Мотивация

- Прирост скорости до 300x
- Широкое применение

Постановка задачи

- Исследовать параллельные алгоритмы поиска кратчайшего пути в графе
- Выбрать оптимальные для разных ситуаций

Особенности задачи

- Разные ситуации
- Не все алгоритмы эффективно распраралleливаются
- Эффективность зависит от представления графа

Фундаментальные результаты

- P. Martín, R. Torres and A. Gavilanes CUDA solutions for the SSSP problem (2009)
- Взят за основу алгоритм Дейкстры
- Сложность O(n^2)

Шаги алгоритма Дейкстры

- Initialization
- Edge relaxation
- Settlement
- Termination criteria

CUDA

- Устройство (device) — GPU. Выполняет роль «подчиненного» — делает только то, что ему говорит CPU.
- Хост (host) — CPU. Выполняет управляющую роль — запускает задачи на устройстве, выделяет память на устройстве, перемещает память на/с устройства.
- Ядро (kernel) — задача, запускаемая хостом на устройстве.

Варианты распараллеливания

- Распараллелить внутренние операции последовательного алгоритма
- Выполнять параллельно несколько алгоритмов Дейкстры через непересекающиеся подграфы

GPU implementation of Dijkstra's algorithm. CUDA kernels are delimited by <<<…>>>

```
<<<initialize>>> (U, F, δ);                //Initialization
while (Δ ≠ ∞) do
    <<<relax>>> (U, F, δ);                 //Edge relaxation
    Δ =<<<minimum>>> (U, δ);               //Settlement step_1
    <<<update>>> (U, F, δ, Δ);             //Settlement step_2
end while
```

Kernels

- relax kernel
- minimum kernel
- update kernel

Модификации

- An Economic Variant
- Martin et al. Successor Variant
- Martín et al. Predecessor Variant

Сравнение работы

- На матрицах смежности быстрее, чем на списках смежности
- Predecessor быстрее, чем Successor

All-pairs shortest-path

Подходы

- На основе Флойда-Уоршалла O (|V| 3),
- На основе Дейкстры O (|V| * |E| + |V| 2log |V|) для разряженных

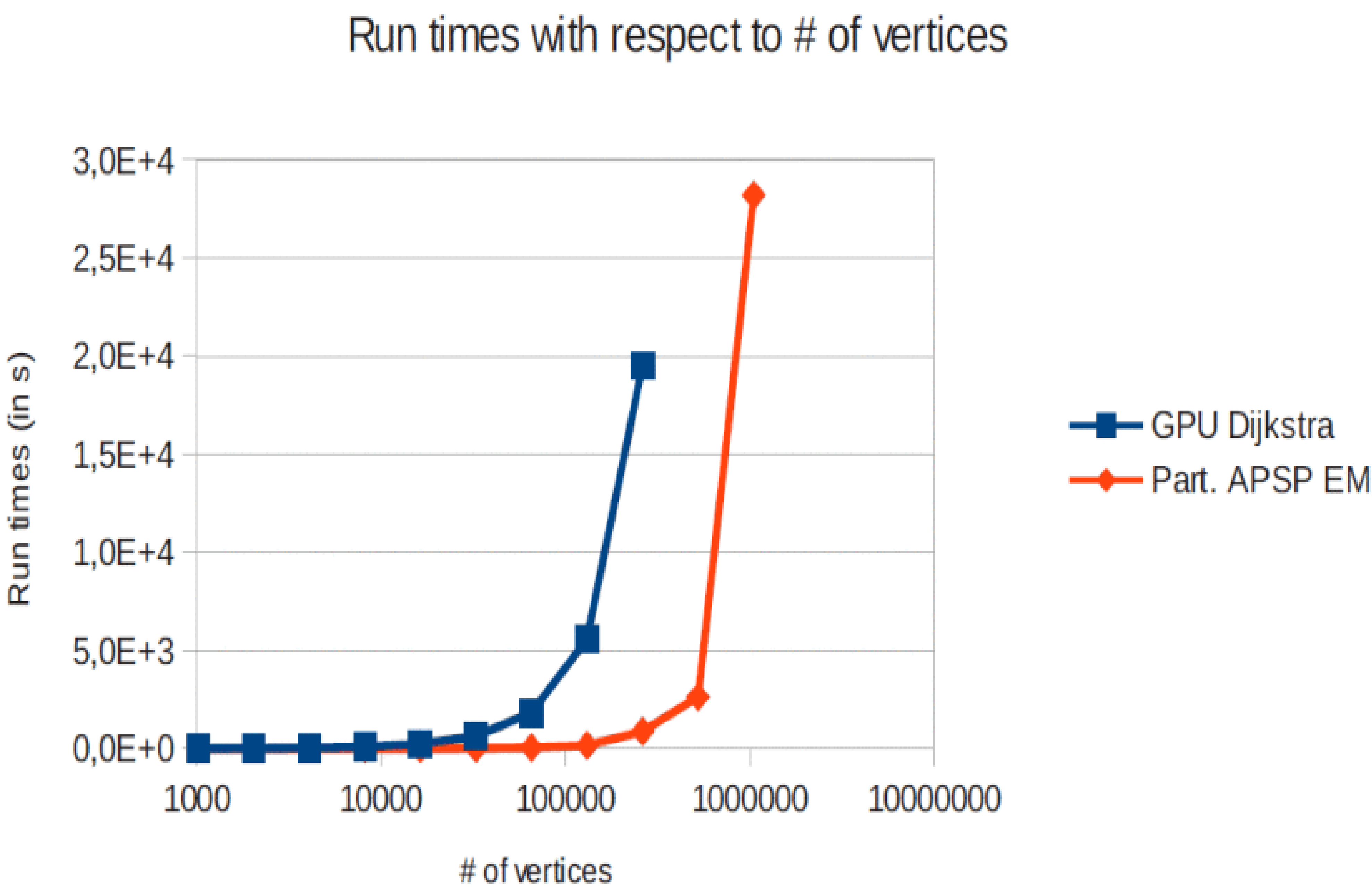
Floydwarshall algorithm

```
INPUT: A graph G(V,E), where V is a set of
    vertices
  and E a set of weighted edges between these
  vertices.
OUTPUT: The distance of the shortest path between
  any two pairs of vertices in G.

for each vertex v in V
    dist[v][v] = 0
end for
for each edge (u,v) in E
    dist[u][v] = w(u,v)  // the weight of the edge
     (u,v)
end for
for k from 1 to |V|
    for i from 1 to |V|
        for j from 1 to |V|
            dist[i][j] =
      min(dist[i][j], dist[i][k] + dist[k][j])
     end for
  end for
end for
return dist
```

Распараллеливание

```
INPUT: A graph G(V,E), where V is a set of
    vertices and E a set of weighted edges between
     these vertices.
OUTPUT: The distance of the shortest path between
    any two pairs of vertices in G.

function partitioned_APSP(G)
  // Step 1
  Partition G into k roughly equal components
    using Metis

  //Step 2
  for each Component C in G
      Floyd-Warshall(C) %compute_APSP(C)
  end for

  // Step 3
  Graph BG = extract_boundary_graph(G)
  compute_apsp(BG)
  for each Component C in G
    Floyd-Warshall(C) %compute_APSP(C)
  end for

  // Step 4
  for each Component C1 in G
    for each Component C2 in G
      compute_apsp_between_components(C1, C2)
    end for
  end for
end function
```

Сравнение


Run times with respect to # of vertices
3,0E+4
2,5E+4
2,0E+4
1,5E+4
1,0E+4
5,0E+3
0,0E+0
Run times (in s)
1000
10000
100000
1000000
10000000
# of vertices
GPU Dijkstra
Part. APSP EM