Поиск кратчайших путей в графе с использованием GPU

Обзор существующих решений

Поляков Александр
371 группа, Мат-Мех СПбГУ

Обозначения

- SSSP - нахождение кратчайших путей от выбранной стартовой вершины до всех остальных. Решают: алгоритм Форда-Беллмана; Дейкстры; поиск в ширину...

- APSP - нахождение кратчайших путей между всеми возможными парами вершин. Решают: алгоритм Флойда-Уоршелла, алгоритмы SSSP повторением для всех вершин...

Мотивация использования GPU

- Относительно высокая производительность

- Существующие средства значительно облегчающие разработку параллельных программ, например CUDA

Актуальность

- Сети дорог
- Картографические сервисы
- Компьютерные сети
- …

Анализ всего, что может быть представлено в виде графа

Фундаментальные результаты в области

- P. Harish and P. Narayanan, “Accelerating large graph algorithms on the gpu using cuda,” in High performance computing–HiPC 2007. Springer, 2007, pp. 197–208.

a) Ускорение в 20-60 раз на искусственных данных
b) Минимальное ускорение на реальных данных

Фундаментальные результаты в области

- L. Luo, M. Wong, and W.-m. Hwu, “An effective gpu implementation of breadth-first search,” in Proceedings of the 47th Design Automation Conference, ser. DAC ’10. New York, NY, USA: ACM, 2010, pp. 52–55.

  a) Модернизация предыдущей работы
  b) Ускорение в 2-6 раз на реальных данных

Фундаментальные результаты в области

- D. Merrill, M. Garland, and A. Grimshaw, “Scalable gpu graph traversal,” in Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, ser. PPoPP ’12. New York, NY, USA: ACM, 2012, pp. 117–128.

a) Основан на BFS, асимптотика $O(|V| + |E|)$
b) Достигнута скорость в 3.3 - 8.6 млрд обработанных ребер в секунду

Замечание

Выделенные основные различия алгоритмов:

- представление графа в памяти GPU
- уровни параллелизации

О них и пойдет речь

Представление графа в памяти

- Матрица смежности

  Требует $O(|V|^2)$ памяти

- Список смежности

  Доступ к элементу за $O(|E|)$ в худшем случае

Accelerating BFS Shortest Paths Calculations Using CUDA for Internet Topology Measurements

Eric Klukovich, Mehmet Hadi Gunes, Lee Barford, and Frederick C. Harris, Jr.

Тестовые данные

- Были собраны реальные данные, построена топология сети, содержащей 6.8 млн роутеров(вершин) и 12 млн связей (ребер)

- Данные были собраны с помощью платформы PlanetLab и метода построения топологии сети Интернет, использующего информацию о времени жизни пакетов данных в протоколе IP

Представление графа в памяти GPU

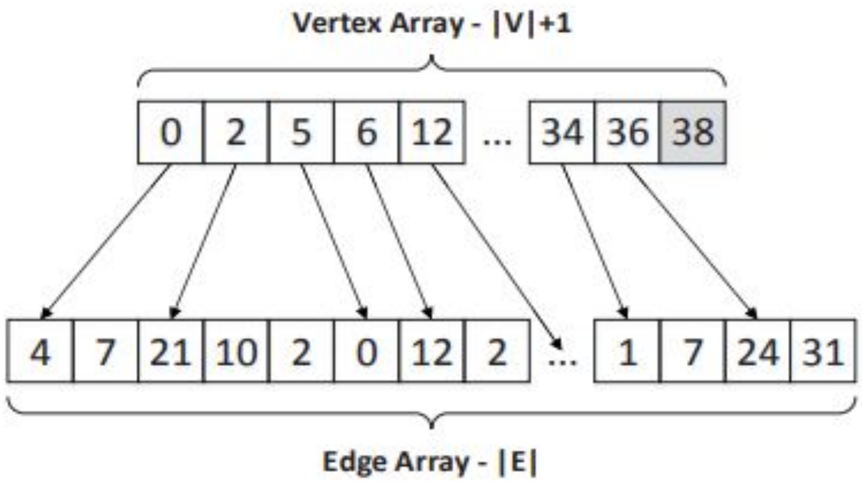
- Массив вершин
- Массив ребер

Пример:

0: 4, 7

1: 21, 10, 2

…



+: 1) легко узнать количество инцидентных вершине ребер

2) требует O(|V| + |E|) памяти

Алгоритм для одного GPU

Algorithm 1 SINGLE_GPU(*Graph (V,E)*, S_a, *numIngress*)

Create vertex array V_a from all the vertices and edge array E_a from all edges in *Graph(V,E)*
for $i = 0$ to $numIngress$ **do**
    Call PROCESS_GRAPH(V_a, E_a, $S_a[i]$)
end for

Algorithm 2 PROCESS_GRAPH(V_a, E_a, *Source S*)

Create cost array C_a, frontier array F_a, frontier update array FU_a, visited array X_a of size V
Initialize F_a, FU_a, X_a to false and C_a to -1
Initialize $F_a[S] \leftarrow true$, $X_a[S] \leftarrow true$, $C_a[S] \leftarrow 0$
$search \leftarrow true$
while $search$ **do**
    $search \leftarrow false$
    Send *search* to GPU
    Call BFS_KERNEL(V_a, E_a, F_a, FU_a, X_a, C_a)
    Call BFS_UPDATE_KERNEL(F_a, FU_a, X_a, *search*)
    Get *search* value from GPU
end while

Algorithm 3 BFS_KERNEL(V_a, E_a, F_a, FU_a, X_a, C_a)

$tid \leftarrow getThreadID$
if $tid < numVertices$ AND $F_a[tid]$ **then**
    $F_a[tid] \leftarrow false$
    for each edge *destID* in V_a **do**
        if NOT $X_a[destID]$ **then**
            $C_a[destID] \leftarrow C_a[tid] + 1$
            $FU_a[destID] \leftarrow true$
        end if
    end for
end if

Algorithm 4 BFS_UPDATE_KERNEL(F_a, FU_a, X_a, *search*)

$tid \leftarrow getThreadID$
if $tid < numVertices$ AND $FU_a[tid]$ **then**
    $F_a[tid] \leftarrow true$
    $X_a[tid] \leftarrow true$
    $search \leftarrow true$
    $FU_a[tid] \leftarrow false$
end if

Accelerating BFS shortest paths calculations using CUDA for Internet topology measurements

Алгоритм для N GPU

Algorithm 5 MULTIPLE_GPU(*Graph(V,E)*, S_a, *numIngress*)

Create vertex array V_a from all the vertices and edge array E_a from all edges in *Graph(V,E)*
Create thread array T, one thread for each device
$i \leftarrow 0$
while $i < numIngress$ **do**
    if $threadCount < numDevices$ **then**
        Launch PROCESS_GRAPH(V_a, E_a, $S_a[i]$) in a new thread
        $threadCount \leftarrow threadCount + 1$
        $i \leftarrow i + 1$
    else
        for $j = 0$ to $numDevices$ **do**
            Wait for thread to finish
        end for
        $threadCount \leftarrow 0$
    end if
end while

for $j = 0$ to $numDevices$ **do**
    Wait for the last threads to finish
end for

Accelerating BFS shortest paths calculations using CUDA for Internet topology measurements

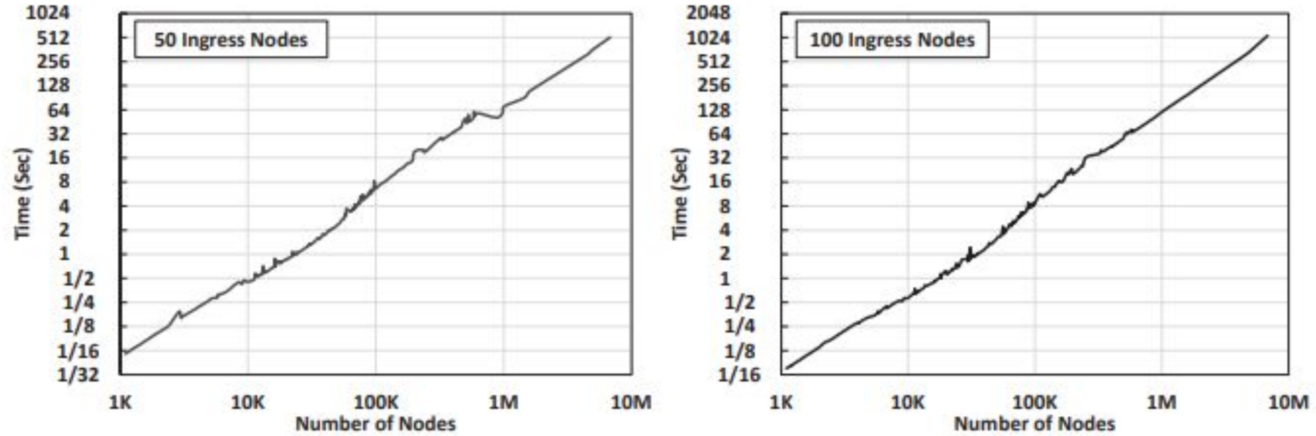


Fig. 4: Sequential execution timings for 50 ingress nodes and 100 ingress nodes

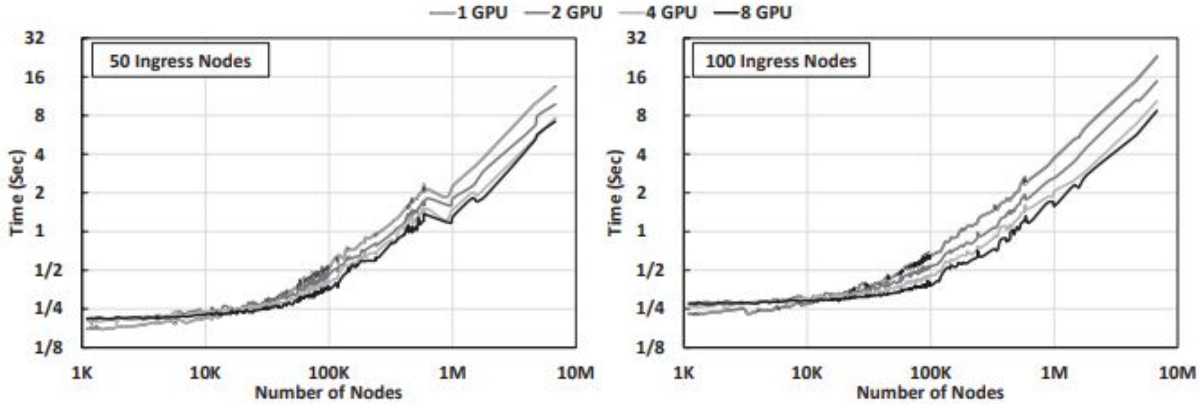


Fig. 5: GPU Execution timings for 50 ingress nodes and 100 ingress nodes

Accelerating BFS shortest paths calculations using CUDA for Internet topology measurements

Результат

Относительно непараллельных алгоритмов, решающих ту же задачу

- 1 GPU: ускорение в 38 раз
- 8 GPU: ускорение в 71 раз

Efficient Multi-GPU Computation of All-Pairs Shortest Paths

Hristo Djidjev, Sunil Thulasidasan, Guillaume Chapuis, Rumen Andonov, and Dominique Lavenier

Тестовые данные

- Были взяты данные Калифорнийской сети дорог, содержащей около 2 млн вершин и 5 млн ребер

```
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
```

Шаг 1:

На вход поступает взвешенный граф

Дробление графа на k равных частей, используя библиотеку Metis. Так как это является NP-полной задачей, реализация в библиотеке Metis основана на эвристиках и дает приближенное решение

```
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
```

Шаг 2:

Для каждой из компонент вызывается традиционный APSP алгоритм

Однако, результат работы данного шага не будет окончательным для каждой из компонент, так как расстояние между любыми двумя вершинами в одной компоненте может быть больше расстояние между ними в начальном графе

```
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
```

Шаг 3:

Граничная вершина - вершина, смежная с какой-либо вершиной из другой компоненты графа

Выделяется граничный граф BG, вершины которого - граничные вершины исходного графа, а ребра делятся на 2 типа:

1) ребра между граничными вершинами из разных компонент
2) виртуальные ребра - ребра между граничными вершинами из одной компоненты

Шаг 4:

Подсчитываются расстояния между парами вершин, в которых хотя бы одна из них не является граничной

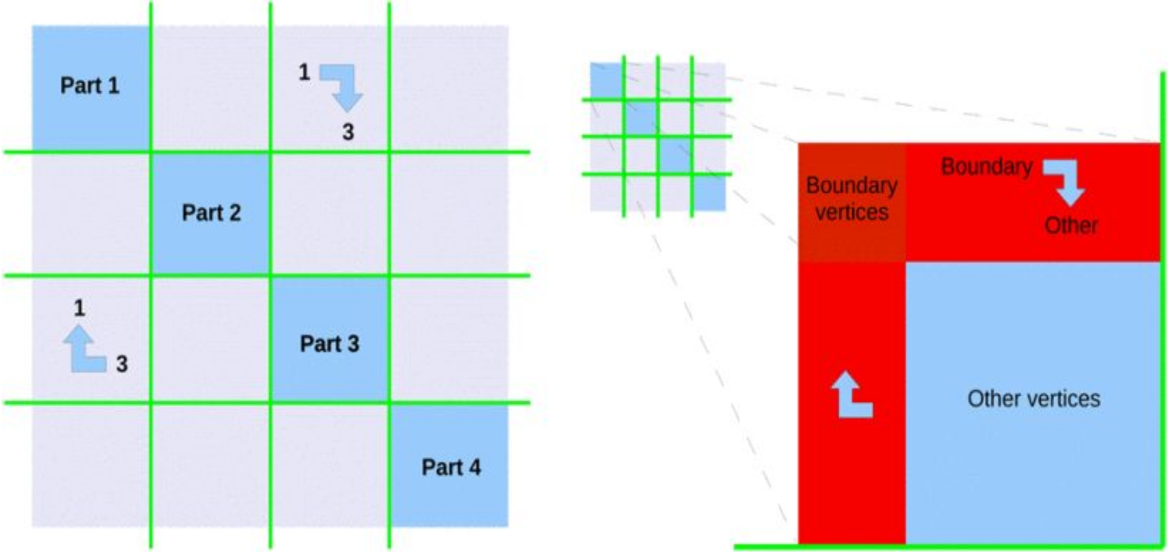
$$dist(v_i, v_j) = min(dist_2(v_i, b_i) + dist_3(b_i, b_j) +$$
$$+dist_2(b_j, v_j))$$

dist_i(a, b) - расстояние между вершинами **a** и **b**, полученное на **i** шагу

Представление графа в памяти GPU

Граф хранится в виде матрицы смежности

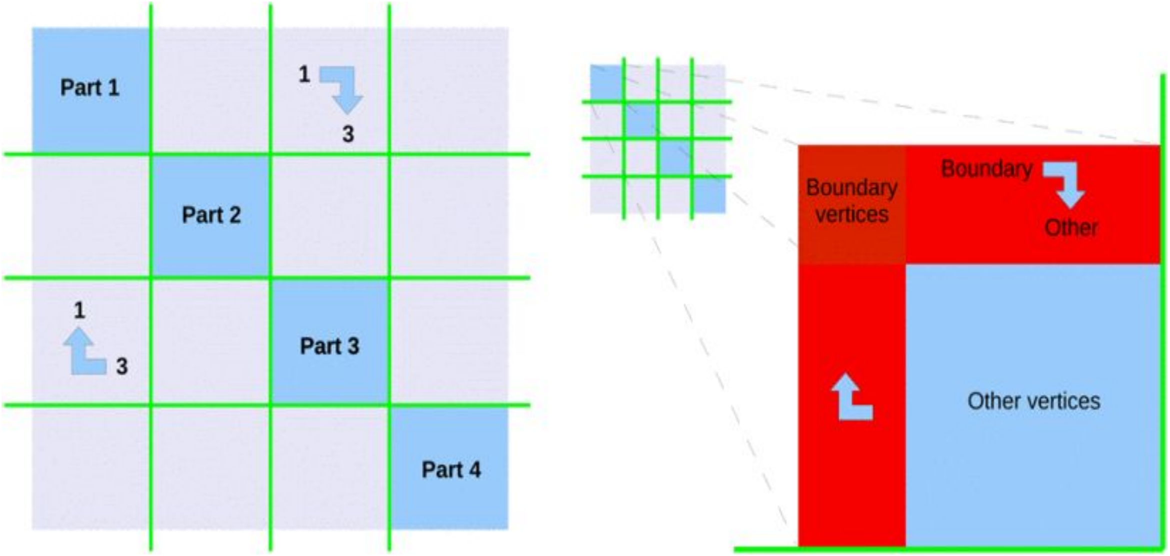
После деления графа на k частей, вершины перетасовываются так,



что вершины из одной компоненты нумеруются последовательно, начиная с граничных вершин

Представление графа в памяти GPU

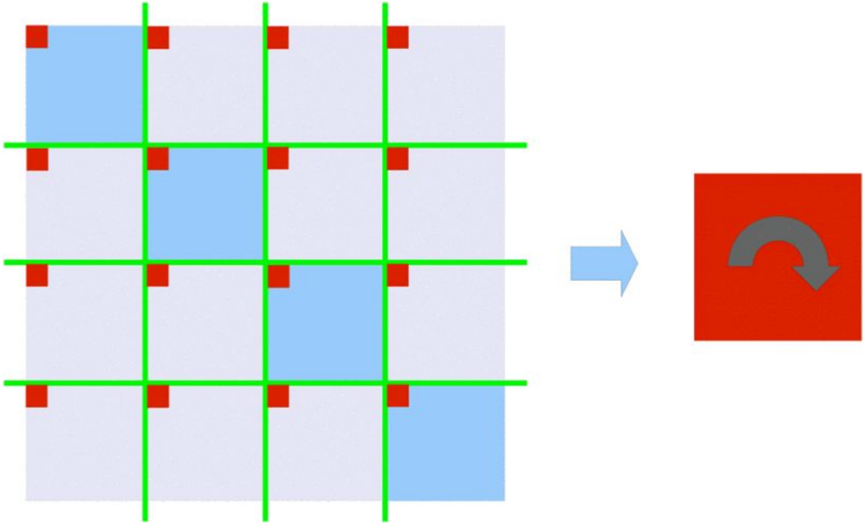
Диагональные подматрицы содержат информацию о подграфах каждой из компонент



Недиагональные подматрицы содержат информацию о кратчайших путях между компонентами

Представление графа в памяти GPU

Граничный граф выделяется из диагональных подматриц



Уровни параллелизма

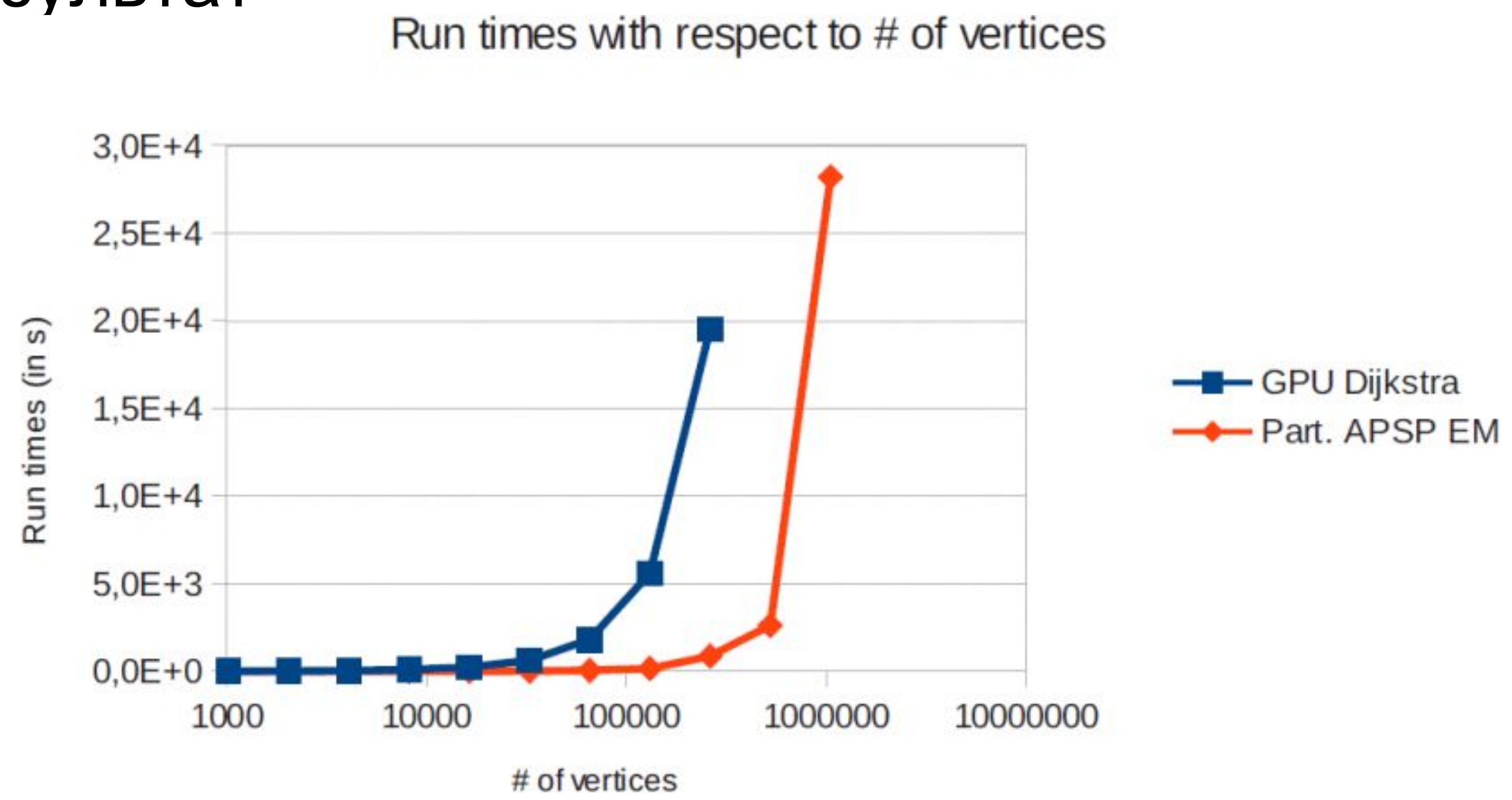
Шаги 2, 3, 4:

большое количество независимых действий дает возможность использовать крупное распараллеливание на уровне отдельных GPU

Алгоритм Флойда-Уоршелла:

распараллеливается для каждой компоненты на одном GPU

Результат



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

Список литературы

1. E. Klukovich, M. Hadi Gunes, L. Barford and F. C. Harris, "Accelerating BFS shortest paths calculations using CUDA for Internet topology measurements," *2016 International Conference on High Performance Computing & Simulation (HPCS)*, Innsbruck, 2016, pp. 66-73. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7568317&isnumber=7568299
2. H. Djidjev, S. Thulasidasan, G. Chapuis, R. Andonov and D. Lavenier, "Efficient Multi-GPU Computation of All-Pairs Shortest Paths," *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, Phoenix, AZ, 2014, pp. 360-369. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6877270&isnumber=6877223