Proposta de projeto final

Matheus Pimenta
09/0125789

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Disciplina: Programação Paralela

7 de setembro de 2014

Para projeto final de Programação Paralela propõe-se que seja feita uma implementação de Δ-Stepping, um algoritmo paralelo para o problema de caminho mais curto com fonte única em grafos dirigidos [1, 2], utilizando C++1y e memória compartilhada. A Figura 1 ilustra o algoritmo.

1 Estratégias de paralelização

Na inicialização das estruturas de dados, é possível paralelizar a construção dos conjuntos *heavy* e *light*, uma vez que para cada vértice do grafo deve ser construído um conjunto de arestas cujos custos estão relacionados de alguma forma com o valor Δ.

Dentro do laço *while* mais interno, é possível paralelizar a construção do conjunto *Req*, de pares de vértices e custos, possivelmente atráves de subconjuntos menores para cada vértice de $B[i]$. Ainda dentro deste laço, outra potencial paralelização é a relaxação dos pares do conjunto *Req*, se cada relaxação for feita de maneira atômica.

Após o laço *while* mais interno, é possível paralelizar a criação do conjunto *Req* novamente através de subconjuntos menores, desta vez para cada vértice de S. Em seguida, há outra potencial paralelização sobre a relaxação dos pares de *Req*.

Referências

[1] Ulrich Meyer and Peter Sanders. Δ-Stepping: A parallel single source shortest path algorithm. In *Algorithms—ESA'98*, pages 393–404. Springer, 1998.

[2] Ulrich Meyer and Peter Sanders. Δ-Stepping: a parallelizable shortest path algorithm. *Journal of Algorithms*, 49(1):114–152, 2003.

```
foreach v ∈ V do                                          -- Initialize node data structures
    heavy(v) := {(v, w) ∈ E : c(v, w) > Δ}                -- Find heavy edges
    light (v) := {(v, w) ∈ E : c(v, w) ≤ Δ}               -- Find light edges
    tent (v) := ∞                                         -- Unreached
relax(s, 0); i := 0                                       -- Source node at distance 0
while ¬isEmpty(B) do                                      -- Some queued nodes left
    S := ∅                                                -- No nodes deleted for this bucket yet
    while B[i] ≠ ∅ do                                     -- New phase
        Req := {(w, tent(v) + c(v, w)) : v ∈ B[i] ∧ (v, w) ∈ light(v)}
        S := S ∪ B[i]; B[i] := ∅                          -- Remember deleted nodes
        foreach (v, x) ∈ Req do relax(v, x)               -- This may reinsert nodes
    od
    Req := {(w, tent(v) + c(v, w)) : v ∈ S ∧ (v, w) ∈ heavy(v)}
    foreach (v, x) ∈ Req do relax(v, x)                   -- Relax previously deferred edges
    i := i + 1                                            -- Next bucket

Procedure relax(v, x)                                     -- Shorter path to v?
    if x < tent(v) then                                   -- Yes: decrease-key respectively insert
        B[⌊tent(v)/Δ⌋] := B[⌊tent(v)/Δ⌋] \ {v}             -- Remove if present
        B[⌊x /Δ⌋] := B[⌊x /Δ⌋] ∪ {v}                       -- Insert into new bucket
        tent(v) := x
```

Figura 1: Algoritmo Δ-Stepping em alto nível [1].