Floyd Warshall x MinPlus

UFRJ - Algoritmos Paralelos 2015.2
https://github.com/edufgf/AllPairsShortestPathsPerformanceEvaluation

Comparação de performance entre algoritmos para o cálculo do menor caminho entre todos pares de vértices em um grafo.

Aluno: Eduardo Felipe Gama Ferreira

11/02/2016

Floyd Warshall

Floyd–Warshall algorithm

Class	All-pairs shortest path problem (for weighted graphs)
Data structure	Graph
Worst case performance	$O(\|V\|^3)$
Best case performance	$\Omega(\|V\|^3)$
Worst case space complexity	$\Theta(\|V\|^2)$

```
// Complexity O(N^3)
Matrix<int>& FloydWarshall(){
  setup_dist();

  for (int k = 0; k < V_SZ; k++){
    for (int i = 0; i < V_SZ; i++){
      for (int j = 0; j < V_SZ; j++){
        int new_dist = m_dist(i, k) + m_dist(k, j);
        if (m_dist(i, j) > new_dist){
          m_dist(i, j) = new_dist;
        }
      }
    }
  }

  return m_dist;
}
```

Min-Plus Matrix Multiplication

- Dado duas matrizes A e B, o produto distância C = A * B é definido como a matriz n x n onde C(i,j) = min(A(i, k) + B(k, j)) para todo k.

- Se modelarmos uma matriz D com as distâncias em um grafo, inicialmente apenas com as distâncias entre vizinhos, podemos calcular o all pairs shortest path realizando várias multiplicações na matriz D.

- D^k = matriz com as menores distâncias entre vértices usando até k arestas.

- Complexidade Tempo: O (N^3 log N)
  Complexidade Espaço: O (N^3)

Min-Plus Matrix Multiplication

```
// Complexity O(N^3)
void MatrixMultiplication(Matrix<int>& dest, const Matrix<int>& orig){
  for (int i = 0; i < V_SZ; i++){
    for (int j = 0; j < V_SZ; j++){
      for (int k = 0; k < V_SZ; k++){
        dest(i, j) = std::min(dest(i,j), orig(i, k) + orig(k, j));
      }
    }
  }
}

// Complexity O(N^3 * log N)
Matrix<int>& MinPlus(){
  setup_dist();

  int log_base2 = 0;
  int N = V_SZ;
  while (N >>= 1) log_base2++;

  for (int k = 0; k < log_base2; k++){
    Matrix<int> aux = m_dist;
    MatrixMultiplication(m_dist, aux);
  }

  return m_dist;
}
```

Floyd Warshall Parallel

- Podemos paralelizar o algoritmo de Floyd Warshall.
- O segundo for loop pode ser paralelizado.
- Note que:

m_dist(i, k) = min(m_dist(i, k),
m_dist(i, k) + m_dist(k, k))

m_dist(k, j) = min(m_dist(k, j),
m_dist(k, k) + m_dist(k, j))

```
// Complexity O(N^3)
Matrix<int>& FloydWarshallParallel(){
  setup_dist();

  for (int k = 0; k < V_SZ; k++){
    cilk_for (int i = 0; i < V_SZ; i++){
      for (int j = 0; j < V_SZ; j++){
        int new_dist = m_dist(i, k) + m_dist(k, j);
        if (m_dist(i, j) > new_dist){
          m_dist(i, j) = new_dist;
        }
      }
    }
  }

  return m_dist;
}
```

Min-Plus Matrix Multiplication Parallel

- Podemos paralelizar o algoritmo de Min-Plus Matrix Multiplication.
- A multiplicação de matrizes pode ser paralelizada no primeiro laço.
- Note que estamos lendo e escrevendo em matrizes diferentes (dist x orig), logo não há concorrência.

```
// Complexity O(N^3)
void MatrixMultiplicationParallel(Matrix<int>& dest, const
  cilk_for (int i = 0; i < V_SZ; i++){
    for (int j = 0; j < V_SZ; j++){
      for (int k = 0; k < V_SZ; k++){
        dest(i, j) = std::min(dest(i,j), orig(i, k) + orig
      }
    }
  }
}

// Complexity O(N^3 * log N)
Matrix<int>& MinPlusParallel(){
  setup_dist();

  int log_base2 = 0;
  int N = V_SZ;
  while (N >>= 1) log_base2++;

  for (int k = 0; k < log_base2; k++){
    Matrix<int> aux = m_dist;
    MatrixMultiplicationParallel(m_dist, aux);
  }
```

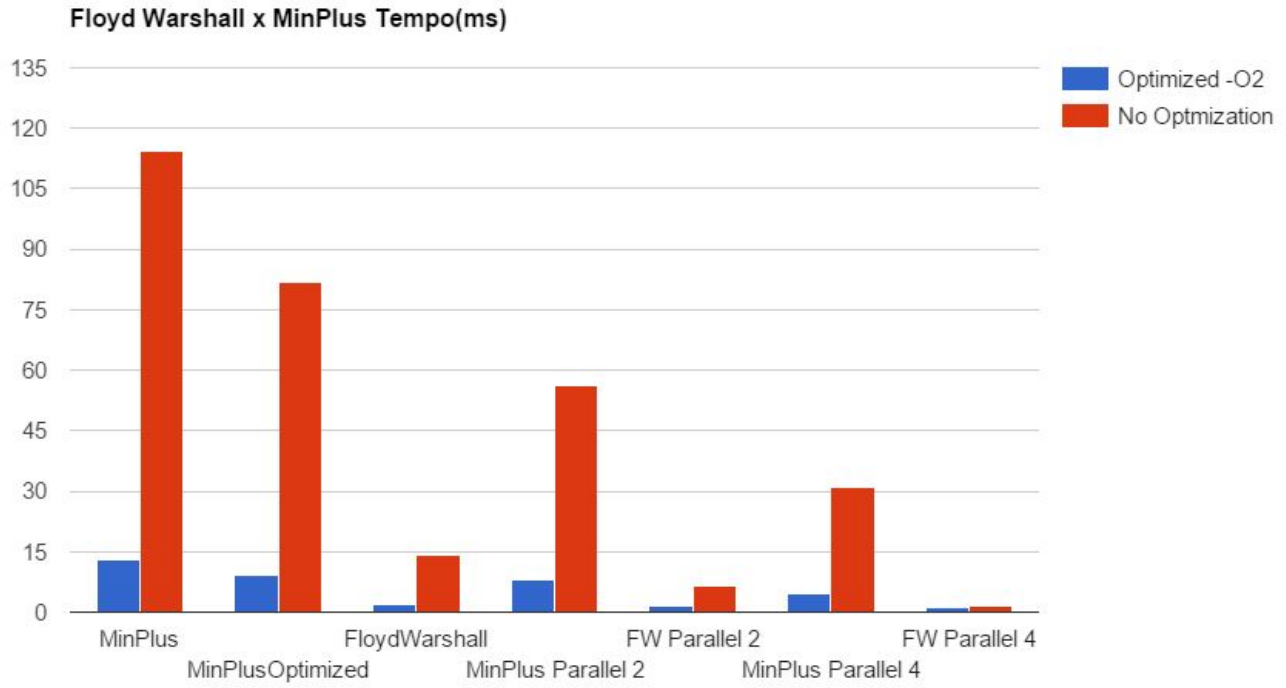
Matrix Multiplication Optimized

- O algoritmo de multiplicação de matrizes pode ser otimizado bastante.
- Acesso a memória cache otimizado pode trazer grandes ganhos.
- Técnicas de otimização:
  Blocking (memória cache)
  Loop unrolling (paralelismo das instruções no processador)
  SIMD (vetorização de instruções).
- Neste trabalho foi utilizado apenas blocking (32 bytes) com loop unrolling x5.
  Com o uso de matrizes de inteiros não foi possível utilizar instruções SIMD (processador utilizado possuia apenas instruções para float e doubles).

Testes

- Um grafo aleatório de 100 vértices foi gerado e usado para todos os testes.
- Um laço de 1000 iterações executou um mesmo algoritmo para calcular as distâncias mínimas e o tempo de execução do programa foi tomado.

Resultados

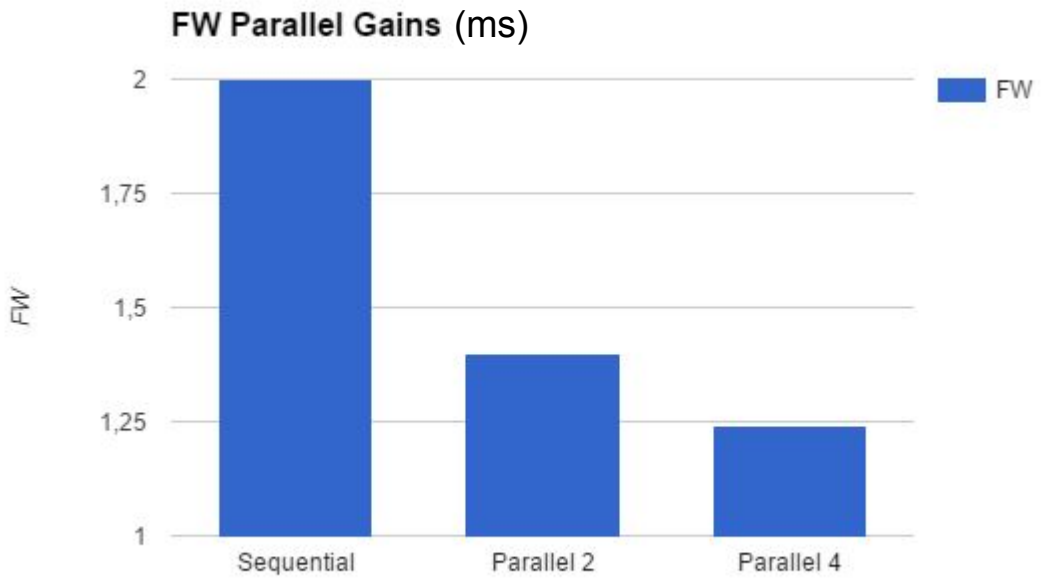

Floyd Warshall x MinPlus Tempo(ms)
135
120
105
90
75
60
45
30
15
0
Optimized -O2
No Optmization
MinPlus
MinPlusOptimized
FloydWarshall
MinPlus Parallel 2
FW Parallel 2
MinPlus Parallel 4
FW Parallel 4

Floyd Warshall Resultados

- SpeedUp

  2 Threads: 1,4

  4 Threads: 1,6

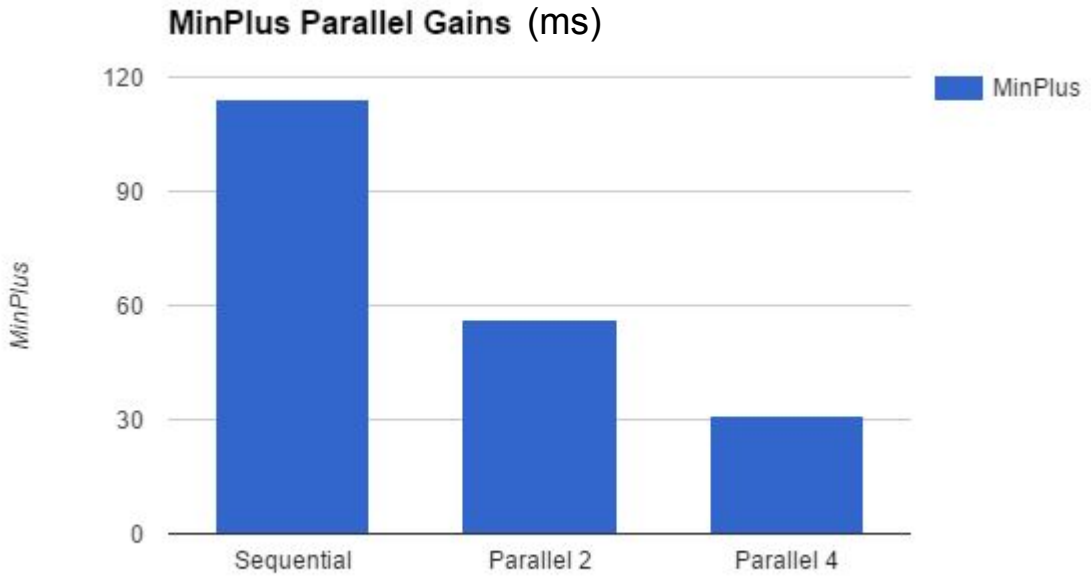


Min-Plus Multiplication Resultados

- SpeedUp

  2 Threads: 2,03

  4 Threads: 3,67



Resultados

- Otimização é importante, compilador ajuda bastante, utilize flags de otimização (-O2).

- Floyd Warshall executa em tempo menor que o algoritmo de MinPlus, mesmo com otimização na multiplicação de matrizes.

- O algoritmo de MinPlus apresenta ganhos maiores (speedup) com o uso de mais processadores do que o algoritmo de Floyd Warshall.
  Talvez o overhead da paralelização, com o tamanho do grafo utilizado neste caso (100 vértices), não foi tão vantajoso para o Floyd Warshall.

Obrigado

Código fonte, resultados e esta apresentação em:
https://github.com/edufgf/AllPairsShortestPathsPerformanceEvaluation