Computing the drainage network on huge grid terrains

Thiago L. Gomes
Salles V. G. Magalhães
Marcus V. A. Andrade
W. Randolph Franklin
Guilherme C. Pena

Universidade Federal de Viçosa (UFV) – Brazil
Rensselaer Polytechnic Institute (RPI) – USA
Introduction: GIS and Big Data
Drainage network computation
*EMFlow* algorithm description
Implementation details
Results
GIS: A Framework for Understanding and Managing Our Earth

Creating
Measuring
Organizing
Analyzing
Modeling

Applying
Planning
Managing
Acting

by Angela Lee – ESRI
An introduction to GIS

Geographic Knowledge

Holistic
Comprehensive
Systematic
Analytic
Visual
Recent advances in sensor technology have produced a huge volume of geographically distributed spatiotemporal data which have posed a massive challenge to GIS researchers.
On most computers, such huge volume of data do not fit in internal memory and need to be processed externally (mainly in disks);
On most computers, such huge volume of data do not fit in internal memory and need to be processed externally (mainly in disks);

But, in this case, the algorithms designed for internal processing do not run well since the time to access data on disk is much higher than the internal access;
• On most computers, such huge volume of data do not fit in internal memory and need to be processed externally (mainly in disks);

• But, in this case, the algorithms designed for internal processing do not run well since the time to access data on disk is much higher than the internal access;

• Thus, the communication between the fast internal memory and the slow external memory is often the performance bottleneck;
Thus, the algorithms must be designed focusing the optimization of I/O operations (data movements); not only the CPU processing;
Thus, the algorithms must be designed focusing the optimization of I/O operations (data movements); not only the CPU processing;

These algorithms are termed external memory (or I/O-efficient) algorithms;
Thus, the algorithms must be designed focusing the optimization of I/O operations (data movements); not only the CPU processing;

These algorithms are termed external memory (or I/O-efficient) algorithms;

To show how the algorithm performance can be affected by the external memory access, suppose you want to print a huge matrix $M$ with $n \times n$ cells stored in external memory;
Consider these two methods:

**Alg. 1**

```plaintext
for (i=1; i <= n; i++)
    for (j=1; j <= n; j++)
        cout << M[i,j];
```

**Alg. 2**

```plaintext
for (j=1; j <= n; j++)
    for (i=1; i <= n; i++)
        cout << M[i,j];
```
Consider these two methods:

**Alg. 1**

\[
\text{for } (i=1; i <= n; i++) \\
\text{for } (j=1; j <= n; j++) \\
\text{cout } \ll M[i,j]; 
\]

**Alg. 2**

\[
\text{for } (j=1; j <= n; j++) \\
\text{for } (i=1; i <= n; i++) \\
\text{cout } \ll M[i,j]; 
\]

- Based on CPU instructions, both algorithms are \( \Theta(n^2) \);
But, considering I/O operations, if the block size $B$ is smaller than the matrix row:

- Algorithm 1 executes $\Theta(n^2/B)$ I/O operations
- Algorithm 2 executes $\Theta(n^2)$ I/O operations
But, considering I/O operations, if the block size $B$ is smaller than the matrix row:

- Algorithm 1 executes $\Theta(n^2/B)$ I/O operations
- Algorithm 2 executes $\Theta(n^2)$ I/O operations

In a machine where the cache memory can store 10000 cells and the time to read a block is 10 milliseconds (9 for seek and 1 for read), the time to read (and print) a matrix with $50000^2$ cells is:

- Algorithm 1 $\approx 4$ minutes
- Algorithm 2 $\approx 10$ months
Hydrologic modelling
The whole process is extremely complex to be modelled in GIS
The whole process is extremely complex to be modelled in GIS.

The solution is to use a simplified model.
A simplification generally used is to describe basically the overland runoff;
Hydrologic model and GIS

- Considering the precipitation falling on the land, it models the drainage network (the rivers), which then empty into the oceans;
Drainage network computation

DEM

<table>
<thead>
<tr>
<th>71</th>
<th>72</th>
<th>67</th>
</tr>
</thead>
<tbody>
<tr>
<td>68</td>
<td>62</td>
<td>65</td>
</tr>
<tr>
<td>63</td>
<td>61</td>
<td>58</td>
</tr>
</tbody>
</table>

3D Viewing
Drainage network computation

<table>
<thead>
<tr>
<th>71</th>
<th>72</th>
<th>67</th>
</tr>
</thead>
<tbody>
<tr>
<td>68</td>
<td>62</td>
<td>65</td>
</tr>
<tr>
<td>63</td>
<td>61</td>
<td>58</td>
</tr>
</tbody>
</table>

DEM

3D Viewing
Drainage network computation

<table>
<thead>
<tr>
<th>71</th>
<th>72</th>
<th>67</th>
</tr>
</thead>
<tbody>
<tr>
<td>68</td>
<td>62</td>
<td>65</td>
</tr>
<tr>
<td>63</td>
<td>61</td>
<td>58</td>
</tr>
</tbody>
</table>

DEM

3D Viewing

Flow direction
Drainage network computation

DEM

3D Viewing

Flow direction

Flow accumulation

<table>
<thead>
<tr>
<th>71</th>
<th>72</th>
<th>67</th>
</tr>
</thead>
<tbody>
<tr>
<td>68</td>
<td>62</td>
<td>65</td>
</tr>
<tr>
<td>63</td>
<td>61</td>
<td>58</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>71</th>
<th>72</th>
<th>67</th>
</tr>
</thead>
<tbody>
<tr>
<td>68</td>
<td>62</td>
<td>65</td>
</tr>
<tr>
<td>63</td>
<td>61</td>
<td>58</td>
</tr>
</tbody>
</table>

Flow direction

Flow accumulation

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>9</td>
</tr>
</tbody>
</table>
Threshold = 4
the drainage network is composed by all cells with “flow accum ≥ 4”
Drainage network computation

DEM

3D Viewing

Flow direction

Flow accumulation

www.engineering.usu.edu/dtarb

1st ACM BigSpatial Workshop, ACMGIS 2012, Redondo Beach, CA
In some cases, it is not possible to determine the flow direction in a cell:

Local minimum (depression)  Flat area
In some cases, it is not possible to determine the flow direction in a cell:

<table>
<thead>
<tr>
<th>71</th>
<th>72</th>
<th>67</th>
<th>71</th>
<th>72</th>
</tr>
</thead>
<tbody>
<tr>
<td>68</td>
<td>62</td>
<td>65</td>
<td>68</td>
<td>62</td>
</tr>
<tr>
<td>63</td>
<td>61</td>
<td><strong>58</strong></td>
<td>63</td>
<td>61</td>
</tr>
<tr>
<td>68</td>
<td>62</td>
<td>65</td>
<td>68</td>
<td>62</td>
</tr>
<tr>
<td>71</td>
<td>72</td>
<td>67</td>
<td>71</td>
<td>72</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>71</th>
<th>72</th>
<th>67</th>
<th>71</th>
<th>72</th>
</tr>
</thead>
<tbody>
<tr>
<td>67</td>
<td><strong>68</strong></td>
<td>68</td>
<td>68</td>
<td>62</td>
</tr>
<tr>
<td>63</td>
<td><strong>68</strong></td>
<td>68</td>
<td>68</td>
<td>61</td>
</tr>
<tr>
<td>68</td>
<td>62</td>
<td>65</td>
<td><strong>68</strong></td>
<td>62</td>
</tr>
<tr>
<td>71</td>
<td>72</td>
<td>67</td>
<td>71</td>
<td>72</td>
</tr>
</tbody>
</table>

- Local minimum (depression)
- Flat area

In general, these two cases are treated by a very time-consuming preprocessing step;
A depression is removed by filling it; that is, its elevation is raised to the elevation of its lowest neighbor;
A depression is removed by filling it; that is, its elevation is raised to the elevation of its lowest neighbor;

And, the flow direction in flat areas is oriented to the lowest neighbor cell;
A depression is removed by filling it; that is, its elevation is raised to the elevation of its lowest neighbor;

And, the flow direction in flat areas is oriented to the lowest neighbor cell;

In general, this preprocessing step takes about 50% of the total running time;
To avoid this time-consuming preprocessing step, we developed the *RWFlood* method which is very efficient when the whole terrain fits in internal memory;
To avoid this time-consuming preprocessing step, we developed the *RWFlood* method which is very efficient when the whole terrain fits in internal memory;

But, it does not scale well for huge terrains requiring external memory processing;
The **EMFlow** method

- To avoid this time-consuming preprocessing step, we developed the *RWFlood* method which is very efficient when the whole terrain fits in internal memory;

- But, it does not scale well for huge terrains requiring external memory processing;

- Thus, the idea of this work (the *EMFlow* method) is to adapt the *RWFlood* for external processing;
The basic idea of **RWFlood** is:

- supposing a terrain being flooded by water coming from outside and getting into the terrain through its boundary;
The basic idea of RWFlood is:

- supposing a terrain being flooded by water coming from outside and getting into the terrain through its boundary;

- the course of the water getting into the terrain will be the same as the water coming from rain and flowing downhill (that is, the flow direction);
In other words, the idea is to suppose the terrain surrounded by water (as an island) and the flooding process is simulated raising the water level;
Initially, the water level is set to the elevation of the lowest cell in the terrain boundary;
Initially, the water level is set to the elevation of the lowest cell in the terrain boundary;

Then, two actions are executed iteratively:
- flooding a cell
- raising the water level
Flooding a cell $c$

- For all cells $d$ neighbors to $c$ do:
  - if the elevation of $d$ is smaller than the elevation of $c$ then $d$ is raised to the elevation of $c$;
  - the flow direction of $d$ is set to the cell $c$;
Raising the water level

- After flooding all cells with the same elevation as \( c \), the water level is raised to the elevation of the lowest cell higher than \( c \);
These cells are processed as previously and the level of the water is raised to the next level;
Now, the cell to be processed has some neighbor cells whose elevation is smaller that the water level (a depression);
The depression is filled;
Notice that the flooding process can creates islands;
Thus, the main idea of **RWFlood** is to store the cells in the boundary of flooded regions;
**RWFeed description**

- Thus, the main idea of *RWFeed* is to store the cells in the boundary of flooded regions;

- And, these cells are processed based on their elevation: from the lowest to the highest;
When a cell \( c \) in the boundary is processed, this boundary “moves toward” the lowest neighbor cell of \( c \);

Which means the terrain matrix is accessed non-sequentially since the cells that are “neighbors” in the two-dimensional matrix representation may not be close in the memory;

Thus, this process can be inefficient when the matrix is huge and is stored in external memory;
To reduce the number of disk accesses, we propose the *EMFlow* whose basic ideas are:

- subdivide the terrain in smaller pieces which can be processed separately;

- use a cache strategy to benefit from the spatial locality of reference present in the sequence of accesses;
The EMFlow algorithm

- **Terrain subdivision**: the flooding process can generate islands which can be processed separately;

- An island is a maximal connected component of non processed (flooded) cells;
The **EMFlow algorithm**

- **Spatial locality of reference**: a special library, named *TiledMatrix*, is used to subdivide the matrix in squared blocks (of cells);

- Some blocks are stored in internal memory and are managed as a cache using the *LRU* policy;
For performance improvements:

- **Islands identification**: uses a lower resolution matrix;

- **Scheduling the islands processing**: islands with a higher percentage of boundary cells stored in internal memory are processed first;

- **The islands boundary size**: the number of islands which can be processed simultaneously is limited by a threshold;
Experimental results

- **EMFlow** was implemented in C++ and compiled with g++ 4.5.2;

- It was compared against **TerraFlow** and **r.watershed.seg** (both included in **GRASS**) and the tests were executed using an Intel Core 2 Duo 2,8 GHz machine running Ubuntu Linux 11.04 64 bits with a 5400 RPM SATA HD;

- Also, it was used different internal memory sizes: 1GB and 2GB to evaluate the algorithms performance in different scenarios;
### Experimental results

- Processing time using 1GB of RAM

<table>
<thead>
<tr>
<th>Terrain Size</th>
<th>EMFlow</th>
<th>Region R2</th>
<th>r.wat.seg</th>
<th>EMFlow</th>
<th>Region R3</th>
<th>r.wat.seg</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000²</td>
<td>0,93</td>
<td>24,43</td>
<td>6,25</td>
<td>0,92</td>
<td>28,22</td>
<td>5,91</td>
</tr>
<tr>
<td>5000²</td>
<td>18,80</td>
<td>661,37</td>
<td>622,66</td>
<td>19,11</td>
<td>907,50</td>
<td>508,90</td>
</tr>
<tr>
<td>10000²</td>
<td>81,67</td>
<td>2329,71</td>
<td>25784,71</td>
<td>81,09</td>
<td>3358,42</td>
<td>55182,80</td>
</tr>
<tr>
<td>15000²</td>
<td>251,14</td>
<td>7588,33</td>
<td>∞</td>
<td>248,39</td>
<td>9046,13</td>
<td>∞</td>
</tr>
<tr>
<td>20000²</td>
<td>579,84</td>
<td>12937,30</td>
<td>∞</td>
<td>605,38</td>
<td>14404,76</td>
<td>∞</td>
</tr>
<tr>
<td>25000²</td>
<td>980,14</td>
<td>22220,89</td>
<td>∞</td>
<td>1065,78</td>
<td>24974,77</td>
<td>∞</td>
</tr>
<tr>
<td>30000²</td>
<td>1522,61</td>
<td>35408,11</td>
<td>∞</td>
<td>1890,35</td>
<td>41251,21</td>
<td>∞</td>
</tr>
<tr>
<td>40000²</td>
<td>3055,39</td>
<td>67076,04</td>
<td>∞</td>
<td>4117,65</td>
<td>78056,28</td>
<td>∞</td>
</tr>
<tr>
<td>50000²</td>
<td>7173,84</td>
<td>98221,64</td>
<td>∞</td>
<td>7618,78</td>
<td>110394,74</td>
<td>∞</td>
</tr>
</tbody>
</table>
## Experimental results

- **Processing time using 2GB of RAM**

<table>
<thead>
<tr>
<th>Terrain Size</th>
<th>EMFlow</th>
<th>TerraFlow</th>
<th>r.wat.seg</th>
<th>EMFlow</th>
<th>TerraFlow</th>
<th>r.wat.seg</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000²</td>
<td>0.74</td>
<td>19.32</td>
<td>6.03</td>
<td>0.98</td>
<td>19.44</td>
<td>5.79</td>
</tr>
<tr>
<td>5000²</td>
<td>20.02</td>
<td>400.84</td>
<td>630.60</td>
<td>19.98</td>
<td>442.97</td>
<td>513.88</td>
</tr>
<tr>
<td>10000²</td>
<td>87.66</td>
<td>2251.66</td>
<td>5290.46</td>
<td>86.94</td>
<td>2552.93</td>
<td>3911.23</td>
</tr>
<tr>
<td>15000²</td>
<td>209.02</td>
<td>5870.34</td>
<td>34252.23</td>
<td>202.36</td>
<td>6869.33</td>
<td>32518.89</td>
</tr>
<tr>
<td>20000²</td>
<td>437.58</td>
<td>13066.63</td>
<td>∞</td>
<td>415.37</td>
<td>13873.60</td>
<td>∞</td>
</tr>
<tr>
<td>25000²</td>
<td>776.98</td>
<td>19339.79</td>
<td>∞</td>
<td>764.86</td>
<td>22492.14</td>
<td>∞</td>
</tr>
<tr>
<td>30000²</td>
<td>1179.31</td>
<td>30364.31</td>
<td>∞</td>
<td>1196.58</td>
<td>33337.07</td>
<td>∞</td>
</tr>
<tr>
<td>40000²</td>
<td>2254.80</td>
<td>56421.36</td>
<td>∞</td>
<td>2162.17</td>
<td>59149.27</td>
<td>∞</td>
</tr>
<tr>
<td>50000²</td>
<td>4011.72</td>
<td>82673.22</td>
<td>∞</td>
<td>3470.99</td>
<td>86670.30</td>
<td>∞</td>
</tr>
</tbody>
</table>
Experimental analysis

Drainage network on Tapajos basin computed by EMFlow
Drainage network on Tapajos basin computed by TerraFlow
Drainage network on Tapajos basin computed by r.watershed
We developed a very fast and simple algorithm to compute the drainage network on huge terrains stored in external memory;

The algorithm doesn’t require a preprocessing step to remove depressions and flat areas;

It is linear in the number of cells in the terrain. That is, each terrain cell is read (and processed) only one time.
Conclusions

- **EMFlow code**, in C++, is available in:
  
  www.dpi.ufv.br/~marcus/EMFlowd

- **Contact:**
  
  marcus.ufv@gmail.com
  marcus@dpi.ufv.br
Questions