1. In the chapter *Analysis of Multithreaded Algorithms*, we studied the 2-way and 3-way construction of a tableau.
2. Describe in plain words, how to construct a tableau in a *k*-way fashion, for an arbitrary integer , using the same stencil (the one of the Pascal triangle construction) as in the lectures.

The Pascal stencil tableau can be constructed by dividing and conquering the elements of the Pascal Triangle. The structure of the initial case of the triangle is a half by table split diagonally through the middle and including the diagonal cells, from the locations to , where is the order of the table. Each of the elements is solved recursively until a base case is reached. The edge elements use the same structure as the base case while the non-edge elements use a square tableau structure.   
  
The following is a example of a Pascal Triangle with . With one recursive iteration.

../../../Downloads/Untitled%20Diagram.png

1. Determine the work and the span for an input square array of order .

Work:

Using the master theorem,

Therefore,

Span:

Using the master theorem,

Therefore,

1. Realize a Julia or CilkPlus a multithreaded implementation off that algorithm. Collect running times (both serial and parallel) for increasing values of (say consecutive powers of 2) and different values of (at least 2 and 3).

Program could be found under src/pascal

Make command: make

Run command: ./pascal n k, where n = k^x

|  |  |  |  |
| --- | --- | --- | --- |
| k | n |  | Time |
| 2 | 32 | Serial | 0m0.006s |
| Parallel | 0m0.013s |
| 256 | Serial | 0m0.009s |
| Parallel | 0m0.013s |
| 2048 | Serial | 0m0.050s |
| Parallel | 0m0.084s |
| 4096 | Serial | 0m0.174s |
| Parallel | 0m0.294s |
| 8192 | Serial | 0m0.681s |
| Parallel | 0m1.111s |
| 16384 | Serial | 0m2.646s |
| Parallel | 0m4.588s |
| 32768 | Serial | 0m11.206s |
| Parallel | 0m18.049s |
| 3 | 9 | Serial | 0m0.008s |
| Parallel | 0m0.012s |
| 81 | Serial | 0m0.007s |
| Parallel | 0m0.011s |
| 729 | Serial | 0m0.012s |
| Parallel | 0m0.020s |
| 6561 | Serial | 0m0.278s |
| Parallel | 0m0.611s |
| 8 | 64 | Serial | 0m0.007s |
| Parallel | 0m0.012s |
| 4096 | Serial | 0m0.118s |
| Parallel | 0m0.266s |
| 32768 | Serial | 0m7.468s |
| Parallel | 0m15.083s |
| 32 | 1024 | Serial | 0m0.050s |
| Parallel | 0m0.036s |
| 32768 | Serial | 0m7.606s |
| Parallel | 0m14.105s |