N-Pancake Graph Diameter

Austen McClernon
University of Melbourne

ABSTRACT

We are given a stack of pancakes of different sizes and are only allowed to flip some number of pancakes from the top. Solving the maximum number of pancake flips one would need in order to sort any arbitrary stack is equivalent to finding the diameter of a pancake graph. The diameter of a pancake graph can be computed as a single source shortest path from one vertex to every other vertex in the graph. Finding the diameter of an n-pancake graph is a **very hard** problem and has only been solved up to $N = 19$. In this paper we propose a simple by design parallelization approach to solve for the diameter of an n-pancake graph. Our approach, implemented using OpenMPI achieves a linear speedup w.r.t processors used.

Author Keywords

Pancakes; A*; Breadth First Search; Prefix Reversals; MPI; Permutations; Cayley Graph

INTRODUCTION

Pancake sorting was first posed by Jacob E. Goodman, under the alias Harry Dweighter in a 1975 edition of Math Monthly [9].

> The chef in our place is sloppy, and when he prepares a stack of pancakes they come out all different sizes. Therefore, when I deliver them to a customer, on the way to the table I rearrange them (so that the smallest winds up on top, and so on, down to the largest at the bottom) by grabbing several from the top and flipping them over, repeating this (varying the number I flip) as many times as necessary. If there are n pancakes, what is the maximum number of flips (in terms of n) that I will ever have to use to rearrange them?

Pancake graphs are desirable for their symmetry, recursive structure and high number of vertices for the degree and diameter [10]. Pancake graphs have found applications in network topology for parallel processing systems and genomics [8] [12], with several studies exploring these applications. The exact diameter of an n-pancake graph is still unknown and is a NP-Hard problem [2], however notable advancements have been made in providing upper and lower bounds. Bill Gates the founder of Microsoft provided the first lower and upper bound in 1979 [5], this was Gates only published academic paper.

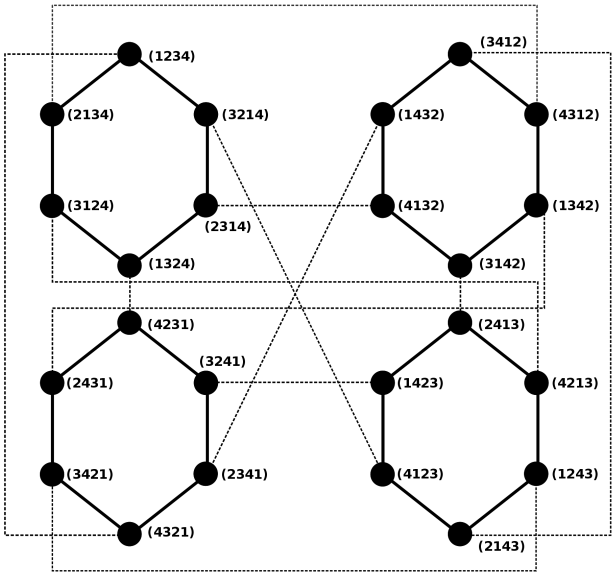

Figure 1. S_4 **Pancake Graph [10]**

FORMULATION

Let S_n be the set of all permutations of n symbols from 1 to n. Consider n distinct pancakes, where the the smallest pancake corresponds to 1, second largest to $n-1$ and largest to n. Let $\pi \in S_n$ represent a stack of pancakes of size n. Let the minimum distance, equivalent by the number of flips (prefix reversals) between some stack π and the sorted stack e be $f(\pi)$. Let S_n^m represent $\pi \in S_n$ where $f(\pi) = m$. Lastly, let $f(n) = max\{f(\pi) : \pi \in S_n\}$ representing the maximum number of flips necessary to sort any arbitrary stack $\pi \in S_n$.

Related Work

Early works focused largely on providing upper and lower bounds for $f(n)$ with Gates and Papadimitriou proving the earliest bounds of $\frac{17n}{16} \leq f(n) \leq (5n+5)$ in 1979 [5] . The upper bound has been improved since by Chitthuri et al [3] as $\frac{18n}{11} + O(1)$ and lower bound by Heydari et al [6] as $\frac{15n}{14}$.

Heydari et al proposed a sequential algorithm which reduces the number computations greatly [6]. The authors, using the recursive properties of a pancake graph compute a S_n^m by computing $\overline{S_n} = \{S_{n-1}^m, S_{n-1}^{m-1}, S_{n-1}^{m-2}\}$ and appending n to the end of every stack. They show that for any π, appending n to the end does not increase the sorting distance as n is the largest

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

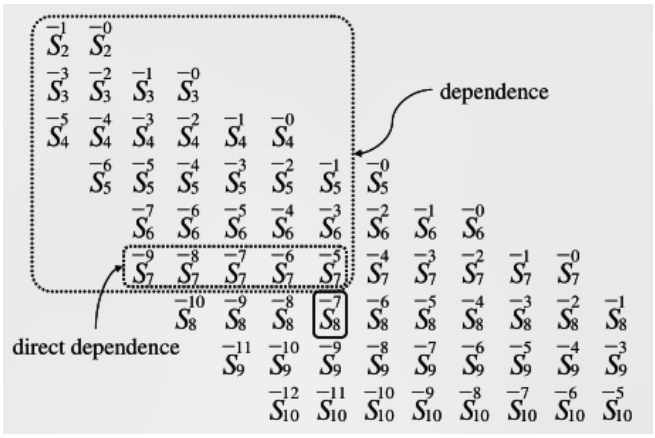

Figure 2. Dependency Relation between S_n^m[1]

pancake in the stack and in the last position. Following the append, the authors show that all elements in S_n^m can be generated by reversing each stack and then performing a prefix reversal on all possible positions. The resulting $\pi \in S_n$ is then filtered to S_n^m by removing any $\pi \in S_n$ where $f(\pi) \neq m$.

Currently no published paper details an algorithm for computing the diameter of a pancake graph in parallel. Earlier works make mention of parallelism, however we are unable to verify their exact approach. Kounoike et al make reference to a breadth first search approach in solving $f(14), f(15), f(16), f(17)$, using intermediate files for saving data [10] [1]. Cibulka computed the largest diameter currently of 19 [4]. Unfortunately the Cibulka does not detail what steps were taken to parallelize the sequential code provided alongside their paper apart from referencing use of the Metacentrum grid, a research computing grid [13].

PARALLEL ALGORITHM

In the following section we propose a layered approach to solve for $S_n^{f(n)}$. Notably our algorithm makes use of [7] for finding the distance between π and e. Figure 2 shows the dependency relation between stacks of size n, which are able to be sorted in m reversals. We adopt a similar approach for our algorithm and perform a layer by layer approach, where the next layer $\overline{S_{n+1}}$ is computed in parallel. As mentioned in the related works, we can calculate $\overline{S_{n+1}^m}$ by taking the union of $\overline{S_n^m}, \overline{S_n^{m-1}}, \overline{S_n^{m-2}}$ and then apply the following procedure for each $\pi_i \in \bigcup_{x=0}^{2} \overline{S_n^{m-x}}$.

- Append n to π_i
- Flip the entire resulting stack
- Partially flip the stack at every index
- Return all of the resulting π with required prefix reversals to sort $\geq lowerBound(n)$

In the naive algorithm, we BFS from the sorted stack of size n, reversing $n-1$ elements at each step and inserting them into the queue. This approach has a complexity of $O(N!)$ as the number of nodes in the graph grows as a factorial - introducing the recursive structure shown in 1. Our approach, whilst still having a sequential $O(N!)$ worst case run time, on average is much faster than this. We also explored a parallel meet in the middle BFS approach, however testing showed average case results were much closer to $O(N!)$ compared to the approach noted above.

Preliminaries

Let $\overline{S_n^p}$ be a subset of $\overline{S_n}$ such that $\overline{S_n} = \bigcup_{p=1}^{size} \overline{S_n^p}$. Where *size* is the number of processors. $p \in [1, size]$ is a processor identifier. Let $send(data, destination)$ and $receive(data, sender)$ be functions that allow for sending and receiving of data across multiple processes that do not share memory. Let $distance(\pi)$ return the distance between the sorted stack e of size n and π. Let $getLB(n)$ return the largest, lower bound necessary in order to create $\overline{S_n}$, where $\forall \pi_i \in \overline{S_n}, distance(\pi_i) \geq getLB(n)$ must hold.

Abstract Algorithm

Algorithm 1 Find pancake stacks where $f(\pi) = f(n)$

Input: $n, start$
Result: $S_n^{f(n)}$
$p \leftarrow getRank()$
$S_{start} \leftarrow initStacks(n, start)$
$len \leftarrow start$
while $len < n$ **do**
  if $p = root$ **then**
    foreach $p \in [1, size]$ **do**
      $send(\overline{S_{len}}, p)$
    end
  else
    $\overline{S_{len}} \leftarrow receive(\overline{S_{len}}, root)$
  end
  $\overline{S_{len+1}^p} \leftarrow ManageLayer(\overline{S_{len}})$
  $len \leftarrow len + 1$
  if $p = root$ **then**
    $\overline{S_{len}} \leftarrow \bigcup_{p=1}^{size} receive(\overline{S_n^p}, p)$
  else
    $send(\overline{S_{len}^p}, root)$
  end
end

Algorithm 2 ManageLayer returns a partial set of pancakes $\pi \in S_{n+1}$ where $distance(\pi) \geq lb$

Input: $\overline{S_n}$
Output: $\overline{S_{n+1}^p}$
Function `ManageLayer(`$\overline{S_n}$`):`

```
p ← getRank()
c ← 0
lb ← getLB(n)
ret ← {}
foreach π ∈ S_n do
    if c mod size = p then
        π_0 ← append(π, n)
        π_1 ← reverse(π_0, n)
        foreach i ∈ [0, n] do
            π_i ← reverse(π_1, i)
            if distance(pi_i) ≥ lb then
                ret ← ret ∪ π_i
            else
                continue
            end
        end
    else
        continue
    end
end
return ret;
```

End Function

Implementation

The implementation was conducted using solely OpenMPI via message passing. The approach followed the abstract algorithm proposed above, however multiple sends were condensed into a single *Bcast*. Likewise, receiving messages on the root node was condensed to use *Gatherv* in order to collate the data. Additional helper communication took place to signpost the size of incoming data on both worker and root nodes. Implicit blocking occurred in the program in two places: (1) In sending/receiving $\overline{S_n}$ and (2) creating $\overline{S_{n+1}}$ from $\bigcup_{p=0}^{size} \overline{S_{n+1}^p}$ on the root node. Further, lexographic representation of permutations was used in order to conserve space. The sorted stack was represented by 0, whilst the next smallest stack 1. This representation was stored into an 8 byte integer and operations involved a $O(1)$ conversion between the integer representation and array. This constant cost was necessary as the space complexity is $O(n!)$, where for larger calculations memory becomes a limiting factor.

Justification

OpenMP was not used although it would have benefits in conjunction with OpenMPI. OpenMP would be beneficial for larger problem sizes in allowing for parallelism with some degree of shared memory, as the problem becomes infeasible for $n > 16$ currently due to memory growing in the worst case by $O(n!)$. OpenMP however would incur overhead costs w.r.t synchronization of result stacks, requiring a barrier or atomic structure to avoid lost updates. OpenMPI alone appeared most appropriate for the problem sizes being tested in the next section.

EXPERIMENTS

Experiment Environment

All Experiments were conducted on Spartan [11], a high performance computing system. Each experiment was conducted on the Snowy Partition. Each experiment had access to identical memory, multiplied by the number of MPI tasks being tested. The following are the relevant specifications.

- **Processor:** Intel(R) Xeon(R) CPU E5-2698 v3 @ 2.30GHz
- **Compiler:** GCC 8.3.0, OpenMPI 3.1.4
- **Memory Per Processor:** 8GB
- **Network:** 25Gb networking with 1.5 µsec latency [11]

Steps to Reproduce

- Extract source code and submission scripts from the provided archive
- Submit the slurm file corresponding to the number of processors you wish to evaluate.
- Evaluate the results displayed in the timestamped .out file

Procedure

Testing was conducted in batches according to the number of MPI tasks (processors) required. Each batch calculated the diameter of a $n = 10, 11, 12, 13$ pancake graph and subsequently reported the set $S_n^{f(n)}$: all pancake stacks that require a minimum number of sorting reversals equal to the diameter. The time to find $S_n^{f(n)}$ was reported for each n using the provided C++ timing code, supplied for project 2. The run time reported does **not** include I/O operations. This process was repeated 10 times. The minimum time taken for each respective test was reported.

Justification

The testing procedure above was selected to draw out overhead such as synchronization conducted by the root node. The pancake graph range $\in [10..13]$ was chosen as the testing input as graphs of smaller n were not large enough to be appropriately parallelized with meaningful results. Pancake graphs > 13 were not included due to time and resource constraints. Memory was a notable constraint, with the problem size growing $O(n!)$ and requiring 8 bytes per permutation to store. $f(15)$ would require worst case $15! * 8$ bytes, or $10461gb$. Our solution did not require evaluating all of S_n, only $\overline{S_n}$; however memory constraints are still a consideration. Further, Spartan is a shared resource and each test needed to be evaluated at a minimum of 10 times to sample a sufficiently accurate value for run-time.

Discussion

The results shown in 3 and 4 illustrate the speedup w.r.t number of processors used. 1 summarises the results for each problem size and processor count.

Initial inspection of 3 and 4 demonstrates linear speedup, compared to the numbers of processors used. Referring to 1 the

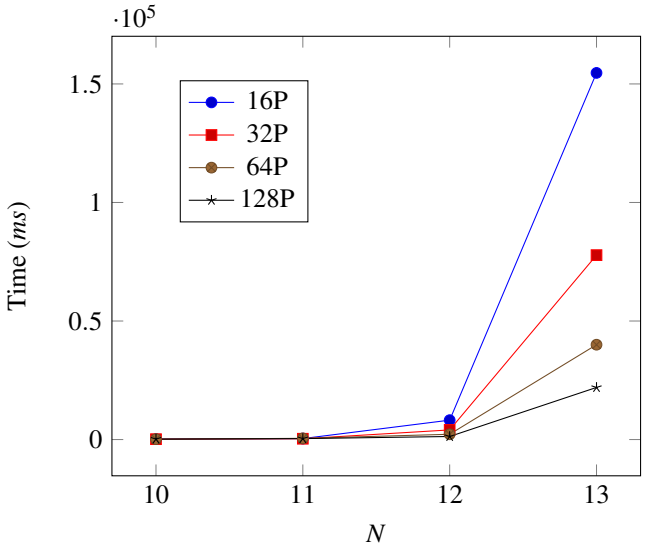

Figure 4. N Problem size vs run time (ms)

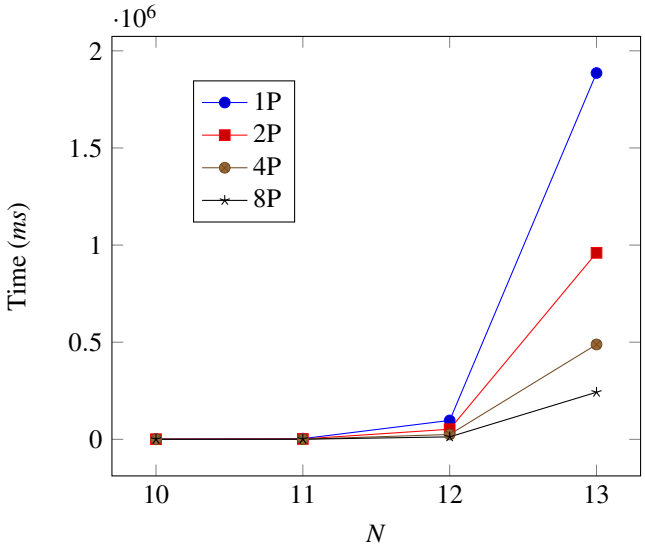

Figure 3. N Problem size vs run time (ms)

P	$N10$	$N11$	$N12$	$N13$
1	1.449	3.495	97.197	1885.744
2	0.734	1.845	51.896	960.077
4	0.379	0.859	25.055	488.362
8	0.181	0.438	12.646	242.423
16	0.243	0.418	8.185	154.651
32	0.295	0.345	4.065	77.789
64	0.284	0.513	2.227	40.004
128	0.282	0.395	1.29	21.976

Table 1. N-Pancake diameter runtime (s) w.r.t processors P

speedup is further evident, with 128 processors finding $S_{13}^{f(13)}$ 85 times faster than a single processor. Results for finding $S_{10}^{f(10)}, S_{11}^{f(11)}$ show that runtime increase is not linear w.r.t processors used for smaller problem sizes. As $N10$ and $N11$ are concluding in under half a second for $P \geq 8$, it is likely that the overhead of initial stack allocation, processor communication and OpenMPI initialization are providing a lower bound on run-time. The problem hardness is evident in 1 as $N13$ has 20x the run-time of $N12$ across all processor counts.

We expected approximately linear speedup for sufficiently large problem sizes w.r.t processors used. This is because finding the next layer can be broken evenly amongst each the processors available, with merging only necessary upon completion of each layer. The distance calculation between π and e is the most costly operation. Whilst the distance between any stack and the sorted stack is not uniformly random amongst the ordered $\overline{S_n}$, by each node processing a permutation $pi \in \overline{S_n}$, every *rank* steps as opposed to chunking work for cache locality benefits - we can ensure the work is sufficiently distributed.

CONCLUSION

We have presented a parallel algorithm for finding all stacks of pancakes, requiring the maximum number of flips to be sorted for a given length. The approach discussed showed efficacy in reducing running time required.The results show a speedup in linear proportion to the number of processors used, for sufficiently large problem size. The results also demonstrate a lower bound on processing time required for $N10$ and $N11$, where the serialized initialization and message passing overhead serve as a floor program run time. Fortunately, as the problem grows at a factorial rate - the numbers of processors required to reach this bottom also grows quickly as shown by calculating $N13$ with 128 processors still exhibiting linear speedup. Further, we provide the only complete description of a parallel approach to solve this problem.

Future Work

As previously mentioned, the largest N-Pancake graph for which the diameter is known is 19, which the author [4] notes would have taken 4000 days sequentially in 2011. In order to compute larger diameters, an algorithm must address memory constraints. Existing work has referenced a depth first approach [1] as opposed to layer approach discussed, however no work has examined the memory efficiency of different parallel algorithms to solve this problem. Future work should look to collate and evaluate existing parallel approaches for solving the N-Pancake graph diameter in hopes of selecting the best algorithm for finding $N = 20$.

REFERENCES

[1] Shogo Asai, Yuusuke Kounoike, Yuji Shinano, and Keiichi Kaneko. 2006. Computing the diameter of 17-pancake graph using a PC cluster. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. DOI: http://dx.doi.org/10.1007/11823285{_}117

[2] Laurent Bulteau, Guillaume Fertin, and Irena Rusu. 2015. Pancake Flipping is hard. In *Journal of Computer*

and System Sciences. DOI: http://dx.doi.org/10.1016/j.jcss.2015.02.003

[3] B. Chitturi, W. Fahle, Z. Meng, L. Morales, C. O. Shields, I. H. Sudborough, and W. Voit. 2009. An (18 / 11) n upper bound for sorting by prefix reversals. *Theoretical Computer Science* (2009). DOI: http://dx.doi.org/10.1016/j.tcs.2008.04.045

[4] Josef Cibulka. 2011. On average and highest number of flips in pancake sorting. *Theoretical Computer Science* (2011). DOI: http://dx.doi.org/10.1016/j.tcs.2010.11.028

[5] William H. Gates and Christos H. Papadimitriou. 1979. Bounds for sorting by prefix reversal. *Discrete Mathematics* (1979). DOI: http://dx.doi.org/10.1016/0012-365X(79)90068-2

[6] Mohammad H. Heydari and I. Hal Sudborough. 1993. On sorting by prefix reversals and the diameter of pancake networks. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. DOI: http://dx.doi.org/10.1007/3-540-56731-3{_}21

[7] Mohammad H. Heydari and I. Hal Sudborough. 1997. On the Diameter of the Pancake Network. *Journal of Algorithms* (1997). DOI: http://dx.doi.org/10.1006/jagm.1997.0874

[8] Morteza Keshtkaran, Roohollah Taghizadeh, and Koorush Ziarati. 2011. A novel technique for compressing pattern databases in the pancake sorting problems. In *Proceedings of the 4th Annual Symposium on Combinatorial Search, SoCS 2011*.

[9] D. J. Kleitman, Edvard Kramer, J. H. Conway, Stroughton Bell, and Harry Dweighter. 1975. Elementary Problems: E2564-E2569. *The American Mathematical Monthly* (1975). DOI: http://dx.doi.org/10.2307/2318260

[10] Yuusuke Kounoike, Keiichi Kaneko, and Yuji Shinano. 2005. Computing the diameters of 14- and 15-pancake graphs. In *Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks, I-SPAN*. DOI:http://dx.doi.org/10.1109/ISPAN.2005.31

[11] Bernard Meade, Lev Lafayette, Greg Sauter, and Daniel Tosello. 2017. Spartan HPC-Cloud Hybrid: Delivering Performance and Flexibility. *University of Melbourne* (2017).

[12] K. Qiu, H. Meijer, and S. G. Akl. 1991. Parallel routing and sorting on the pancake network. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. DOI: http://dx.doi.org/10.1007/3-540-54029-6{_}184

[13] Šimon Tóth and Miroslav Ruda. 2015. Distributed job scheduling in MetaCentrum. In *Journal of Physics: Conference Series*. DOI: http://dx.doi.org/10.1088/1742-6596/608/1/012025