FPGA Report

Implementation of FHAST on

Vitis HLS

Team members:

V.Karthikeya(IMT2021504)

Rohit Mogli(IMT2021503)

NVS Asrith(IMT2021508)

Introduction:

Traditional DNA sequencing tools like BOWTIE often have slow processing speeds due to computational bottlenecks. To solve this, FHAST, an FPGA-based replacement, was introduced. It speeds up the process by using parallelism and high memory bandwidth, achieving up to 70x faster performance compared to single-threaded BOWTIE and 12x faster than eight-threaded BOWTIE, while keeping the mapping accuracy nearly the same.

FHAST uses the FM-Index to quickly search for patterns in the reference genome. It processes reads using multiple hardware threads to reduce memory delays and improve speed. The system also uses precomputed memory addresses and external memory to handle large genomes, making FHAST much faster than traditional tools for DNA sequencing.

Some of the processes involved in performing FM Index are

Burrows-Wheeler Transform (BWT): BWT rearranges a sequence to group similar patterns, making it efficient for locating and matching sequences.

Suffix Array: Suffix Array is a sorted list of all possible suffixes of a sequence, which allows quick access to where specific patterns occur in the sequence.

C-Table: C-Table is a table that counts the occurrences of each character in the sequence up to a certain point helping to quickly narrow down the search range when looking up patterns.

I-Table: The I-Table stores the first occurrence of the each character in SBWT(first element of each string in suffix array). This would be helpful for the calculating the occurrence of the pattern.

Searching: When you search for a pattern, the FM-Index uses two markers (called "top" and "bottom") to narrow down where the pattern might be in the text. As it goes through the characters in the pattern, it keeps updating these markers to zoom in on the possible positions.

$$bottom_{new} = C - table[n, Bottom_{current} + I - table[n]$$

$$top_{new} = C - table[n, Top_{current} + I - table[n]$$

Example : GCTAATTAGGTACC$

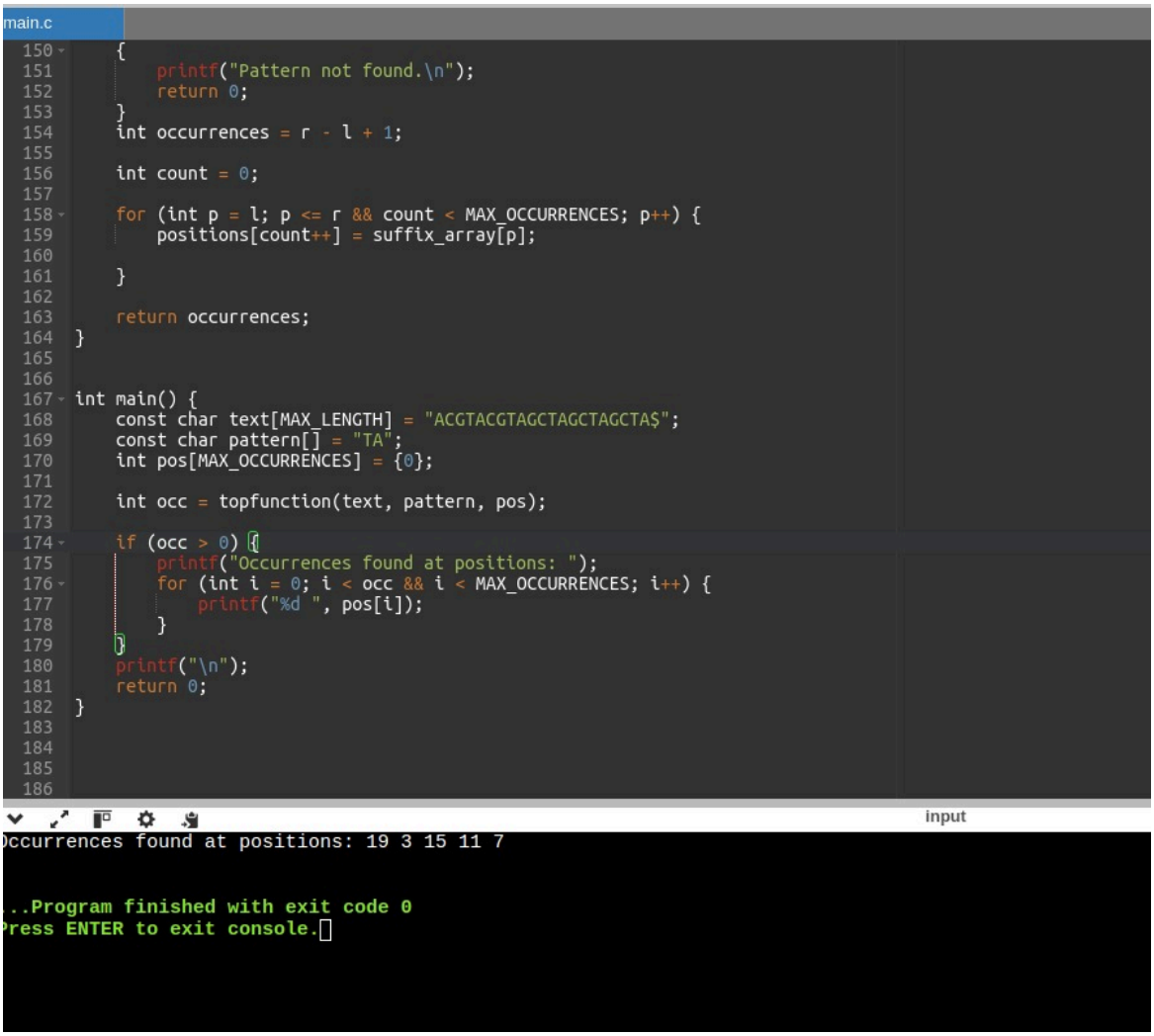
Original String: GCTAATTAGGTACC$		
Index	**Sorted Suffixes:**	**Suffix Array**
0	$	14
1	aattaggtacc$	3
2	acc$	11
3	aggtacc$	7
4	attaggtacc$	4
5	c$	13
6	cc$	12
7	ctaattaggtacc$	1
8	gctaattaggtacc$	0
9	ggtacc$	8
10	gtacc$	9
11	taattaggtacc$	2
12	tacc$	10
13	taggtacc$	6
14	ttaggtacc$	5

C-table

Index	BWT(Q)	A	C	G	T
0	C	0	0	0	0
1	T	0	1	0	0
2	T	0	1	0	1
3	T	0	1	0	2
4	A	0	1	0	3
5	C	1	1	0	3
6	A	1	2	0	3
7	G	2	2	0	3
8	$	2	2	1	3
9	A	2	2	1	3
10	G	3	2	1	3
11	C	3	2	2	3
12	G	3	3	2	3
13	T	3	3	3	3
14	A	3	3	3	4
15	Total	4	3	3	4

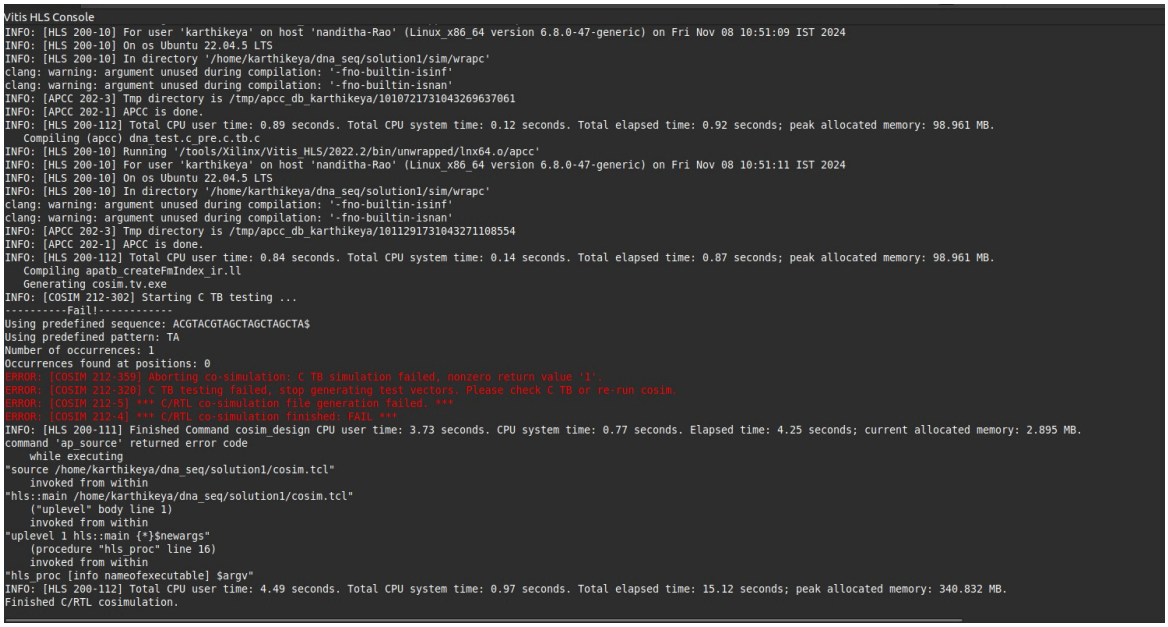
Methodology:

First we have written a c code implements fm-index and finds all the occurrences of a required read(string) in the reference sequence.



Next we converted it into HLS format and ran it in vitis_hls 2023.2. The board we used for this purpose initially was zybo (xc7z010clg400-1) but we had violations in memory as our task required around 59,000 which were not present in zybo so we resorted to using ZCU104.

During the process of C/RTL cosimulation we got this error given below



The test bench was working properly for c-simulation and c-synthesis but for c/rtl cosimulation it was able to synthesise the suffix_array so we tried implementation of each module in the code seperately to see if they are synthesizeable or not and have updated parts which were not syntesizeable like (while loops , pointers , struct). Even then we weren't able to make the rtl co-simulation work It went on running for hours. We had discussed with our Mentor and tried debugging it but we weren't able to do it.

in directory 'D:/Seafile/diplwmatikh/hls_files/HLS_dev_nonbonded_v2/solution1_Zynq_ZC706/sim/wrapc'
clang: warning: argument unused during compilation: '-fno-builtin-isinf'
clang: warning: argument unused during compilation: '-fno-builtin-isnan'
@I [APCC-3] Tmp directory is apcc_db
@I [APCC-1] APCC is done.
Generating cosim.tv.exe
@I [SIM-302] Starting C TB testing ...
Is there any way to know if the simulation is still running or it has failed?
I'm using Vivado HLS 2015.4.
Thanks in advance

HLS

赞 答案 共享 1 个回答 · 685 次查看

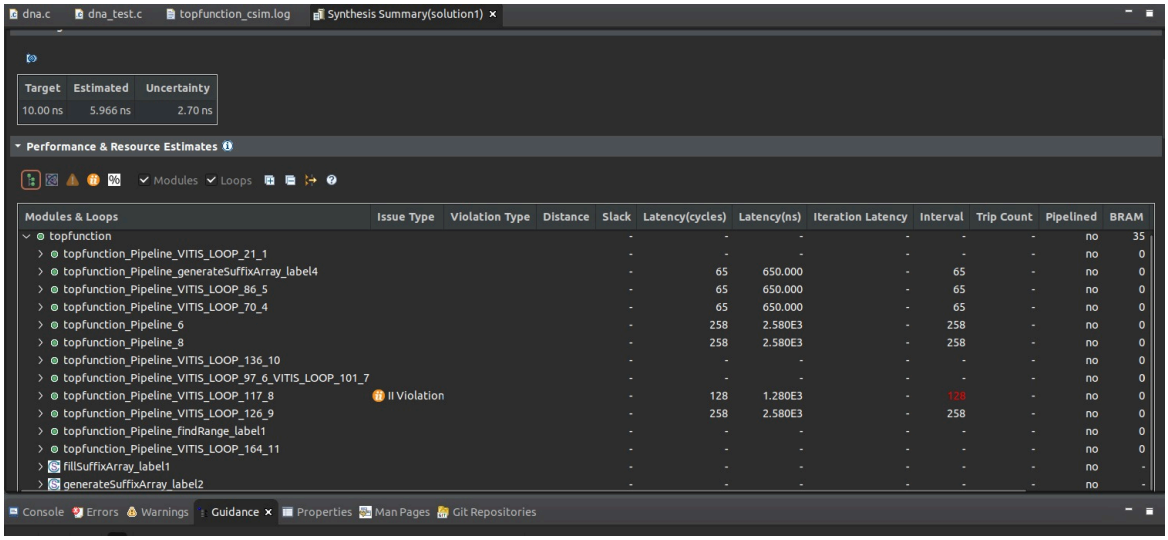
u4223374 (Member)
8 年前

Generally, if it runs C simulation in a few minutes and it hasn't finished cosimulation after a few hours, it's not going to finish.
Unfortunately, determining in what way it's broken is rarely straightforward. The generated HDL code is nearly unreadable for all but the most trivial modules. Even if you could find the bug, fixing it in the C code may be impossible - the cause may not actually be a bug in the C code, just an error in how HLS has done the translation.
The only way I've found to debug this sort of thing is to cut down the design until it works, then add things back in until it fails. Start by deleting all the dependence pragmas and all the cyclic and block partitioning pragmas. Change the stream/FIFO pragmas to have really massive lengths (eg. if your program processes 640*480 images, make the buffers big enough to store whole images). Resource usage and performance don't matter for now, so it doesn't matter if you end up at 4000% block RAM utilization.

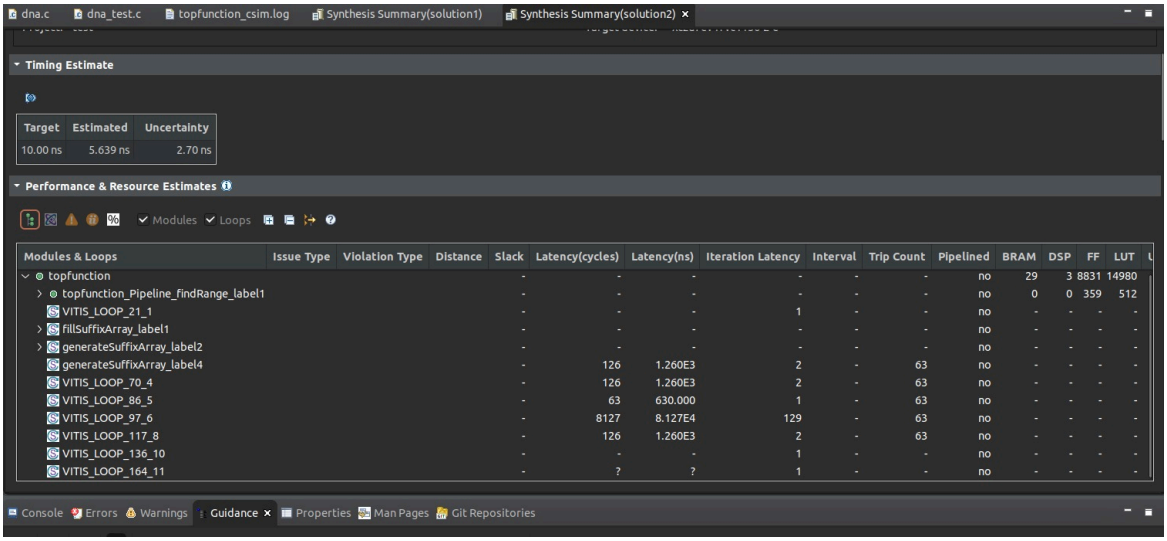
赞 · 回复

Initially we had some timing violations which were gone after using optimisation of the code with pragmas like pipelining and loop unrolling.

Latency before optimisation

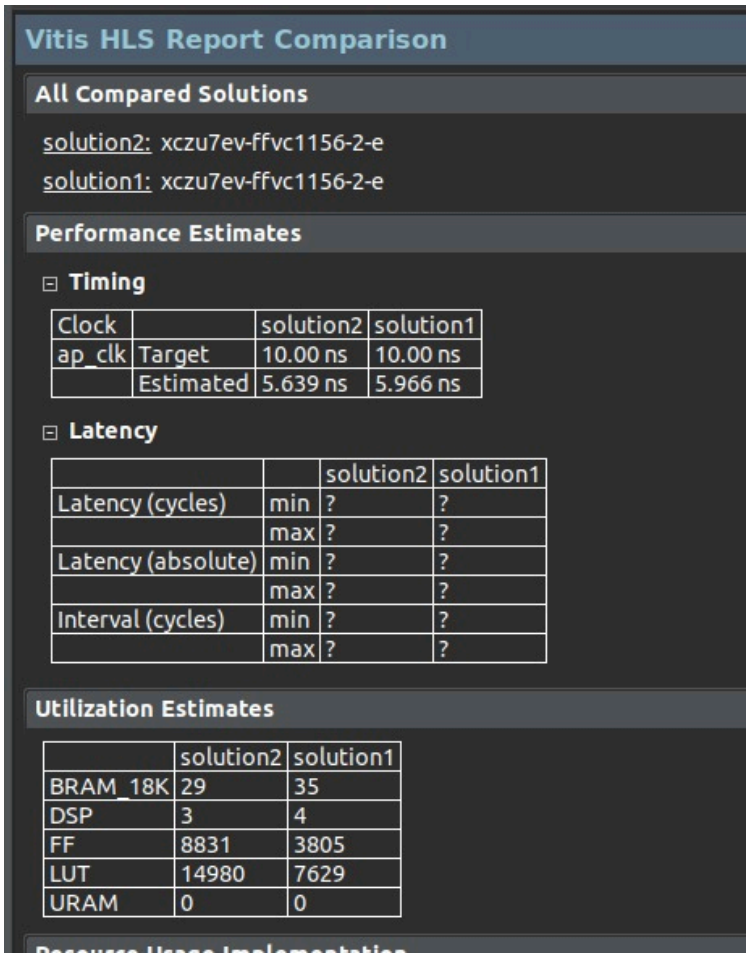


Latency after optimisation



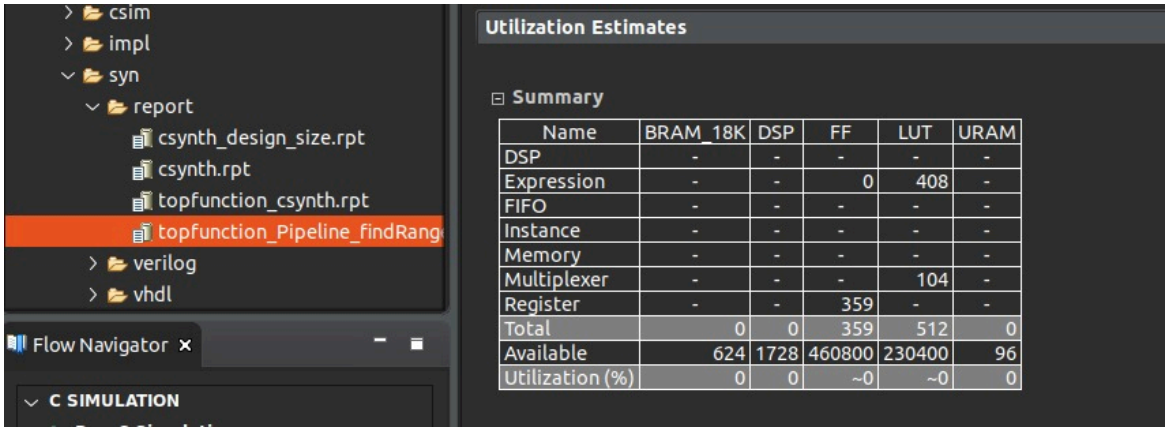
Latency of the circuit is the time taken by the input to generate the output.

We can see the individual latency of each function in the above diagram .So the latency of the circuit after optimization is 5.639 ns.



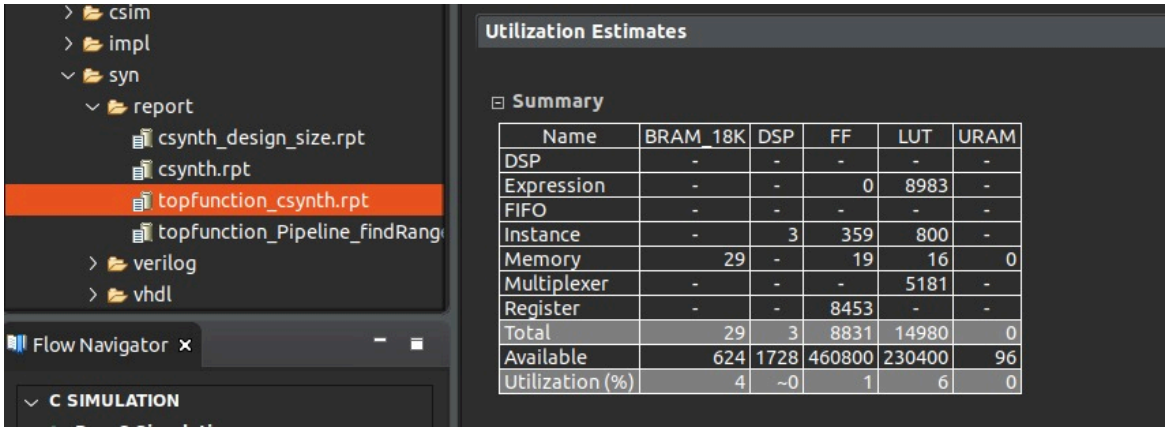
Comparison report of two solutions

Utilization report of function (findRange)



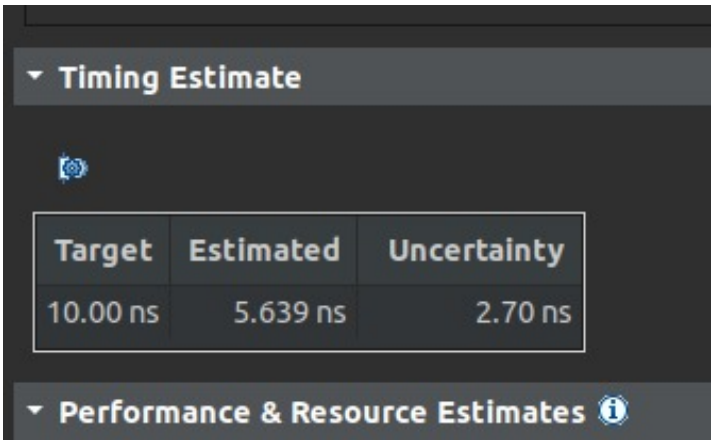
Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	408	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	104	-
Register	-	-	359	-	-
Total	0	0	359	512	0
Available	624	1728	460800	230400	96
Utilization (%)	0	0	~0	~0	0

Utilization report of function (Topfunction)



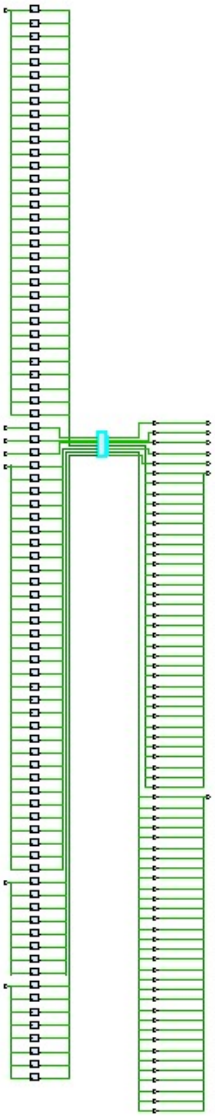
Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	8983	-
FIFO	-	-	-	-	-
Instance	-	3	359	800	-
Memory	29	-	19	16	0
Multiplexer	-	-	-	5181	-
Register	-	-	8453	-	-
Total	29	3	8831	14980	0
Available	624	1728	460800	230400	96
Utilization (%)	4	~0	1	6	0

Max Clk Frequency Report



We can that the min clock period is 5.639 ns + 2.70 ns =8.339 ns so the max clock frequency is
1/t_period. So the achieved max frequency is 1/8.339ns = 119.9MHZ.

Layout design



Github Link to the code:

https://github.com/Vadlamudi04/dna_sequencing

References:

1. https://www.cs.jhu.edu/~langmea/resources/lecture_notes/bwt_and_fm_index.pdf
2. https://ieeexplore.ieee.org/abstract/document/5771277?casa_token=c6pxv7GcrcMAAAAA:YXWEw9T2NoYs4aol0DbKkq9KaoofpqFIeP-f5FM6F2-SYKsFChiRF5BmcTBJxsJip196N_gQEro
3. https://github.com/BenLangmead/comp-genomics-class/tree/master
4. P. Ferragina and G. Manzini, “Opportunistic data structures with

applications,” in Proc. 41st Annu. Symp. Found. Comput. Sci., 2000, pp. 390–398.