Stage of Improvement	Benchmark	User Time(u)	Total Instructions	User Time Relative to Initial Stage	User Relative to Previous Stage	Bottlenecks from previous phases
Initial Stage	Midmark	8.654	50,000,000	N/A	N/A	N/A
	Advent	70.240	N/A	N/A	N/A	
	Sandmark	228.941	N/A	N/A	N/A	
Compiling with -O1	Midmark	7.738	50,000,000	0.894	0.894	This phase ran faster because we changed the compilation flag. With this flag it reduces redundant syntax and common subexpressions.
				-10.58%	-10.58%	
	Advent	65.479	N/A	0.932	0.932	
				-6.78%	-6.78%	
	Sandmark	208.075	N/A	0.909	0.909	
				-9.11%	-9.11%	
Compiling with -O2	Midmark	7.711	50,000,000	0.891	0.997	This phase ran faster because we changed the compilation flag. This flag expanded on compilation flag from the previous stage.
				-10.90%	-89.02%	
	Advent	65.448	N/A	0.932	1.000	
				-6.82%	-0.05%	
	Sandmark	201.853	N/A	0.882	0.970	
				-11.83%	-2.99%	
Change underlying Data Structure	Midmark	2.712	50,000,000	0.313	0.352	Ths phase ran faster because we replaced our ADT (Seq of Seq). This is because this hanson ADT required a lot of pointer chasing and mallocing to make sure everything ran smooth. In ccachegrind we found that Seq_get accounted for a majority of our user time.
*in this stage we changed our ADT from a Sequence of Sequence to an C array of C array				-98.82%	-64.83%	
	Advent	21.418	N/A	0.327	0.327	
				-67.27%	-67.27%	
	Sandmark	65.080	N/A	0.284	0.322	
			N/A	-71.57%	-67.76%	
Remove calls from bitpack	Midmark	0.695	50,000,000	0.080	0.256	This phase ran faster because although we had redone our entire UM in main in the previous phase. We found ourselves still calling external functions in Bitpack.c. So we decided to implement that functionality in our main program and eliminate calls to the file.
*in this stage we removed all the calls to the Bitpack getu function in our program. we instead did field instructions in the same file				-91.97%	-74.37%	
	Advent	3.615	N/A	0.051	0.169	
				-94.85%	-83.12%	
	Sandmark	17.476	N/A	0.076	0.269	
				-92.37%	-73.15%	
Created a Stack Pointer	Midmark	0.386	50,000,000	0.045	0.555	After the last phase, we wanted to match the reference implementation. We looked at ccachegrind, and found that we were still using a lot of execution cycles by using the Hanson ADT for a stack. So, we decided to implement a stack using an array and stack pointer. In doing this, we were able to access memory without pointer chasing and costly mallocs and frees.
*in this stage we created an abstraction of the stack we use to keep track of our freed IDs. we used a calloc array and stack pointer to index where the most recently freed id is				-95.54%	-44.46%	
	Sandmark	10.356	N/A	0.045	0.593	
				-95.48%	-40.74%	
	Advent	2.740	N/A	0.039	0.758	
				-96.10%	-24.20%	