Minimizing MAXLIVE For Spill-Free Register Allocation

Shashi Deepa Arcot, Henry Gordon Dietz, &
Sarojini Priyadarshini Rajachidambaram

University of Kentucky
Electrical & Computer Engineering

Register Allocation Is Critical

- Memory ~1,000X slower than processor
- ILP often depends on register use
- Automated coding & compiler optimizations often build huge basic blocks
- Microcontrollers & nanocontrollers often don't have data memory beyond registers

Nanocontrollers

- This is **NOT** a talk about nanocontrollers... but they do make register allocation critical
- To minimize circuit complexity:
  - All data must fit in tens of 1-bit registers (there is no “main memory” to spill to)
  - Bit-serial computing... HUGE basic blocks
  - ALU is a 1-of-2 MUX... If-Then-Else trinary

Optimizing Register Allocation

- Minimize spill/reload cost
  - Shortest path & MIN algorithms
  - Graph coloring techniques
- Tune higher-level optimization interactions
- Directly try to **reduce MAXLIVE**

MAXLIVE

- MAXLIVE is the maximum number of values whose lifetimes (D-U chains) overlap
- Need ≥MAXLIVE registers to be spill-free
- MAXLIVE depends on access order; reordering alone can change MAXLIVE
- MAXLIVE depends on the operations; arithmetically equivalent expressions can yield different MAXLIVE

First Approach:
Genetic Algorithm Reordering

- Search legal orderings for min MAXLIVE
- Search space is factorial in block size; nanocontroller blocks often 1,000s of ops
- Use a “steady state” “island model” GA... other search techniques were less effective

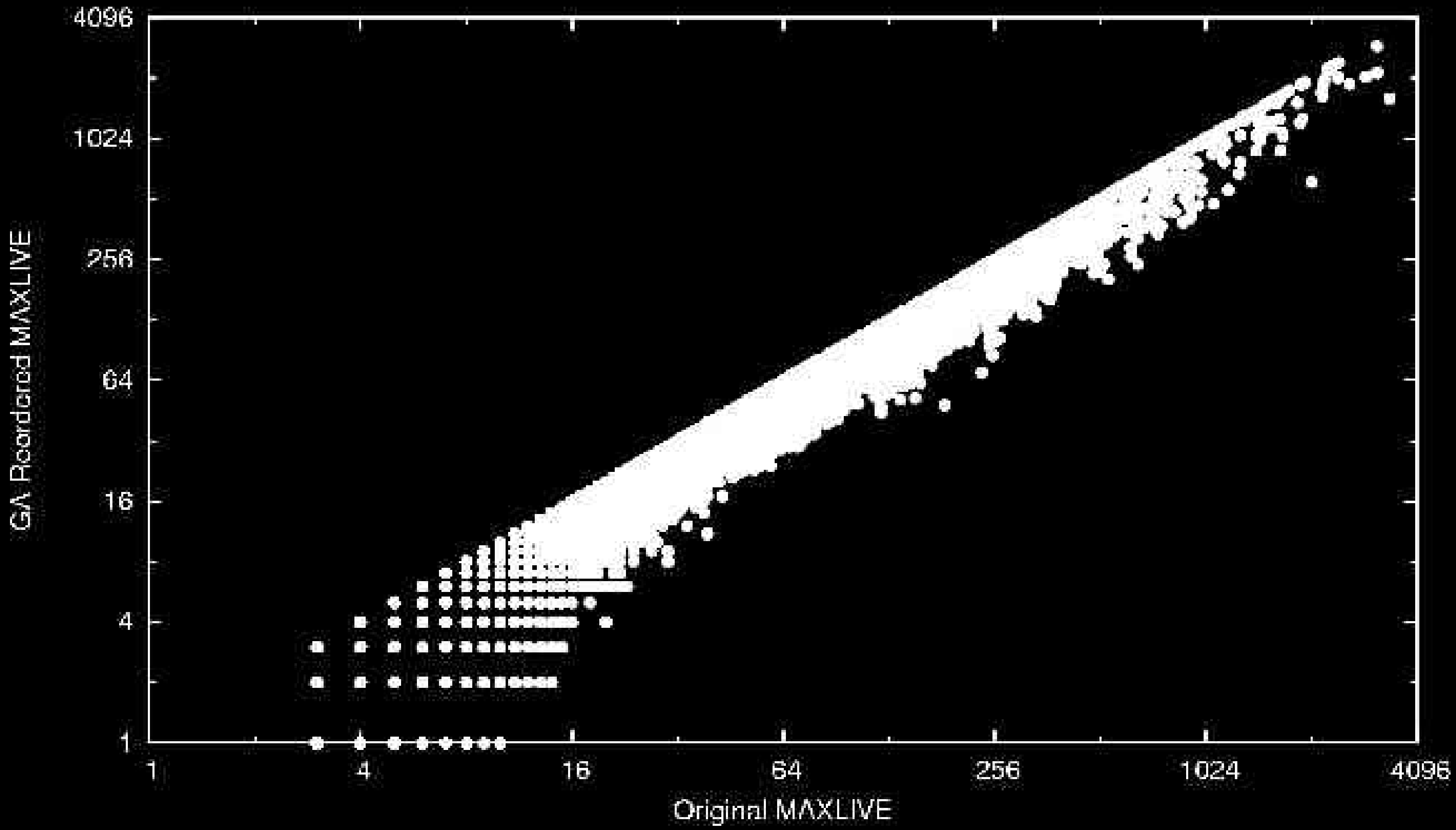
The Genetic Algorithm

- Trick is searching only valid schedules... genome is not a schedule, but a set of priorities to break ties in list scheduling
- A segmented population evolves:
  - Fitness: MAXLIVE, time at MAXLIVE
  - Selection: by tournament
  - Crossover: splices parts of 2 parents
  - Mutation: random change to 1 parent

Experimental Procedure

- Pseudo-random BitC block generator
- Optimizing BitC compiler generating ITEs
- Converter makes data structures from ITEs
- GA Reordering code (~300 lines C)
- Scripts & filters control test runs, discard duplicates & MAXLIVE<3 cases, collect and summarize results

GA-Reordered MAXLIVE Vs. Original MAXLIVE



Experimental Results

- Results from 32,912 accepted test cases
- Execution time was kept fast by:
  - Population of 50, 4 islands, cross 3X mut
  - Stop after evaluating 1,000 schedules
- Average **reduces MAXLIVE by 18%**
- Clearly worthwhile, but not sufficient...

Sethi-Ullman Numbering (SUN)

- Optimal technique for coding an assignment
- Fast, deterministic, algorithm
- Finds tree walk order that minimizes:
  - Number of generated instructions
  - MAXLIVE
- Published in 1970
- Many attempts to extend to DAGs...

SUN Labels Each Node With MAXLIVE For Its Subtree

1. If n is a leaf and a left descendant, L(n)=1. If it is a right descendant, L(n)=0;
2. If n has descendants with labels l1 and l2,
   (a) If $l1 \neq l2$, L(n)=max(l1, l2);
   (b) If $l1 = l2$, L(n)=l1+1

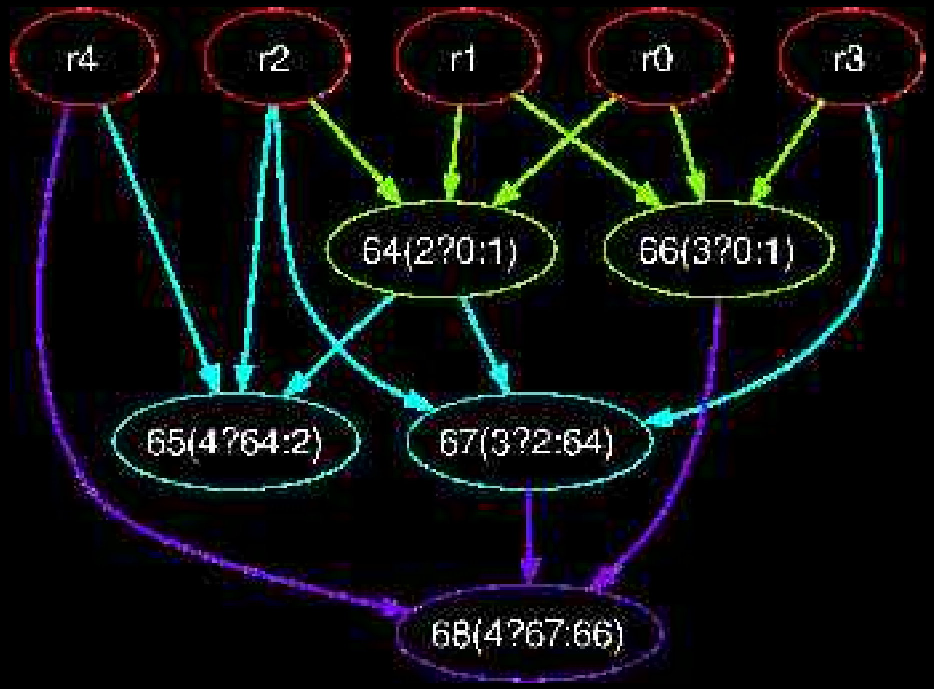
Our Modifications To SUN

- Common Subexpression Elimination (CSE) creates DAGs; disabling CSE yields trees
- Once a particular CSE is enabled, that register can be treated as “reserved” for as long as it is live using the SUN walk order
- Use a GA to selectively re-enable CSEs
- Must generalize SUN for trinary ops and modern instruction formats

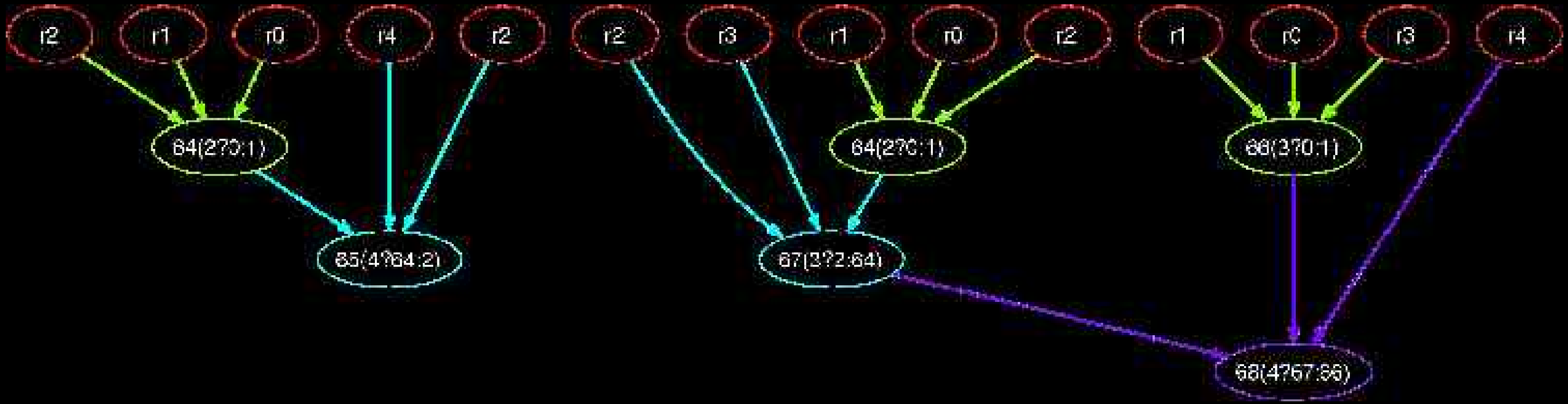
SUN For Trinary Ops (e.g., ITEs)

1. If n is a leaf, $L(n)=0$;
2. If n has descendants with labels l1, l2, & l3 and sorted such that $l1 \geq l2 \geq l3$
   - (a) If $l1>l2>l3$, $L(n)=l1$;
   - (b) If $l1>l2=l3=0$, $L(n)=l1$;
   - (c) If $l1>l2=l3\neq0$ & $l1-l2=1$, $L(n)=l1+1$;
   - (d) If $l1>l2=l3\neq0$ & $l1-l2>1$, $L(n)=l1$;
   - (e) If $l1=l2>l3$, $L(n)=l1+1$;
   - (f) If $l1=l2=l3\neq0$, $L(n)=l1+2$;
   - (g) If $l1=l2=l3=0$, $L(n)=1$

DAGs To Trees: A Sample DAG



DAGs To Trees:
The Corresponding Trees



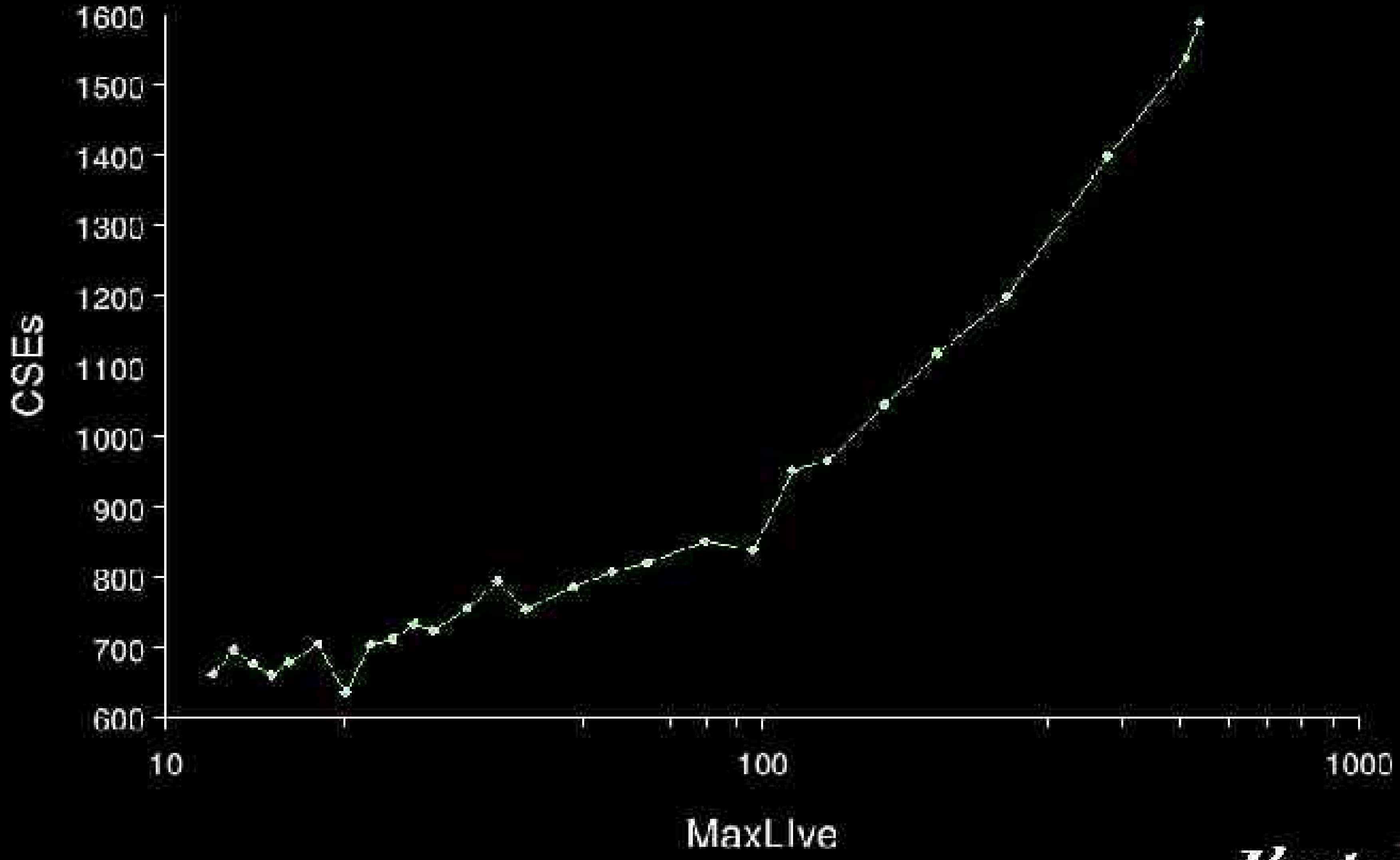
The SUN-Based GA

- K potential CSEs means 2**K search space; again use “steady state” “island model” GA
- Genome is a traditional bit vector in which each potential CSE is a bit, 1 if enabled
- The population is initialized to include both all CSEs enabled and all disabled
- Fitness computes MAXLIVE, but dynamically adjusts a cutoff threshold (“terrible”)

Variations On One Test Case

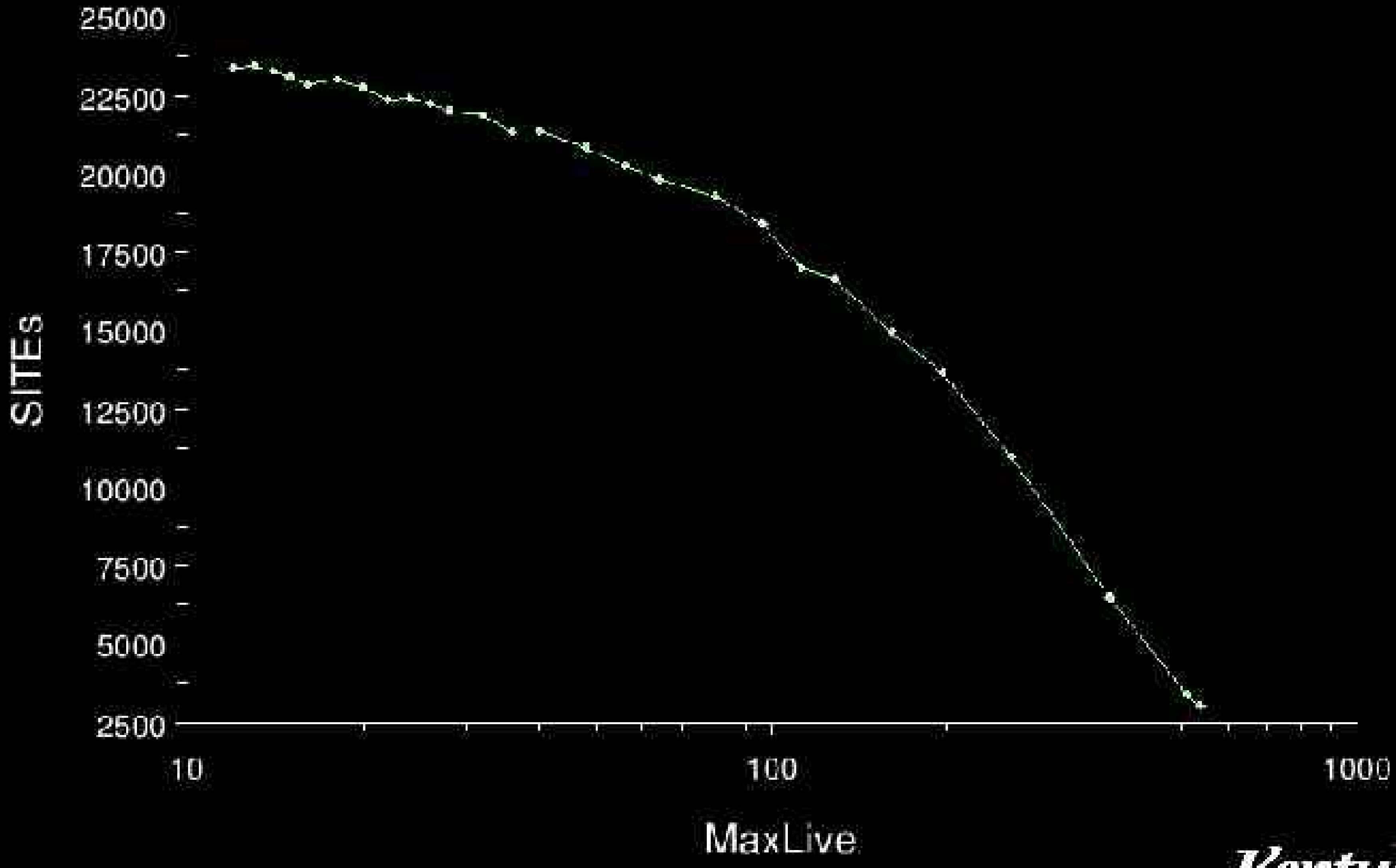
- A **large** nanocontroller basic block
- Initial parameters:
  - Number of SITEs = 3,041
  - MAXLIVE = 561
- With MAXLIVE minimized by SUN GA:
  - Number of SITEs = 23,819; 1:7.8 increase
  - MAXLIVE = 12; 47:1 reduction
- What about less extreme MAXLIVE targets?

Enabled CSEs Vs. MAXLIVE



UNIVERSITY OF KENTUCKY

SITEs Vs. MAXLIVE

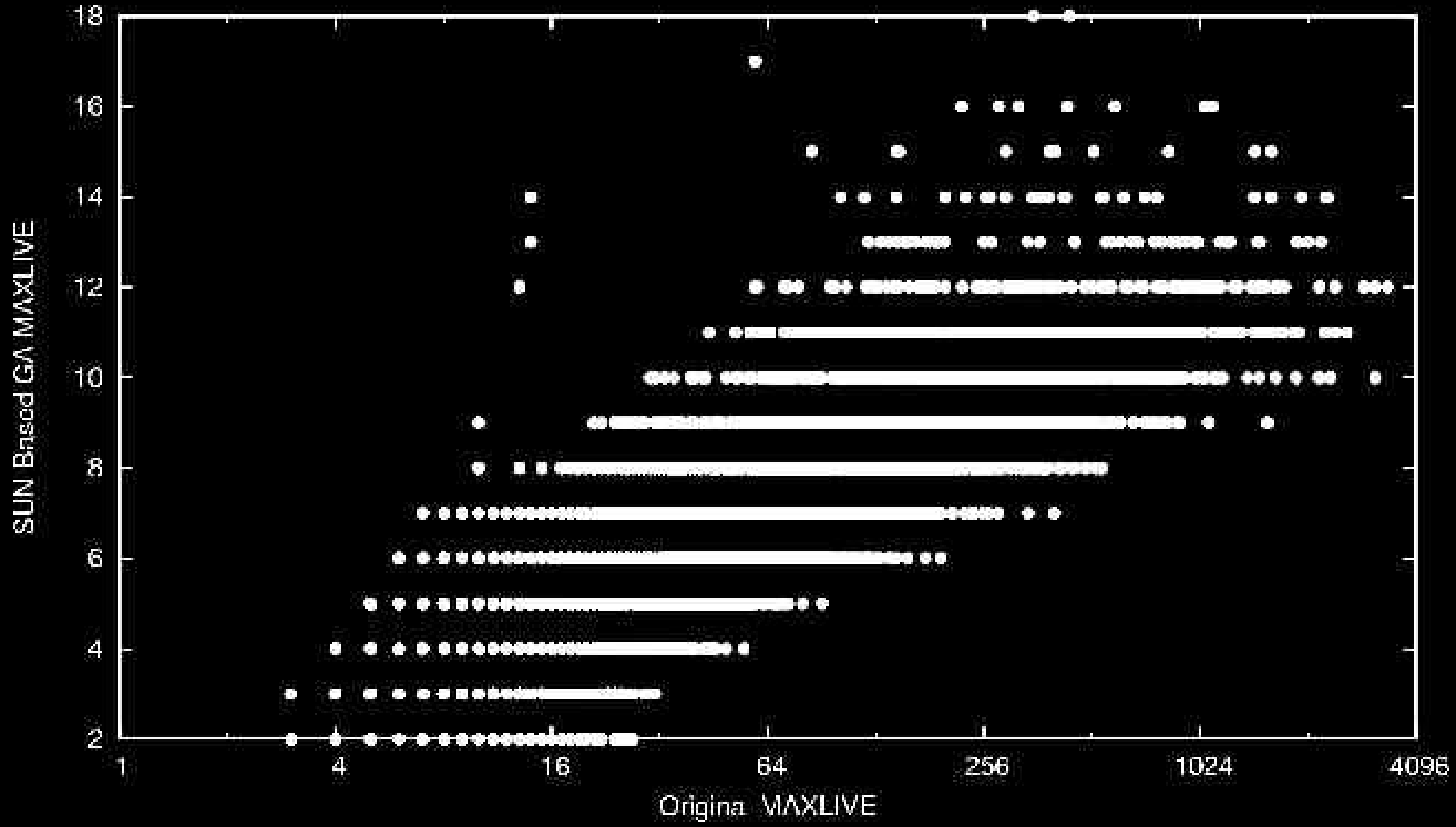


Kentucky UNBRIDLED SPIRIT

More Experimental Results

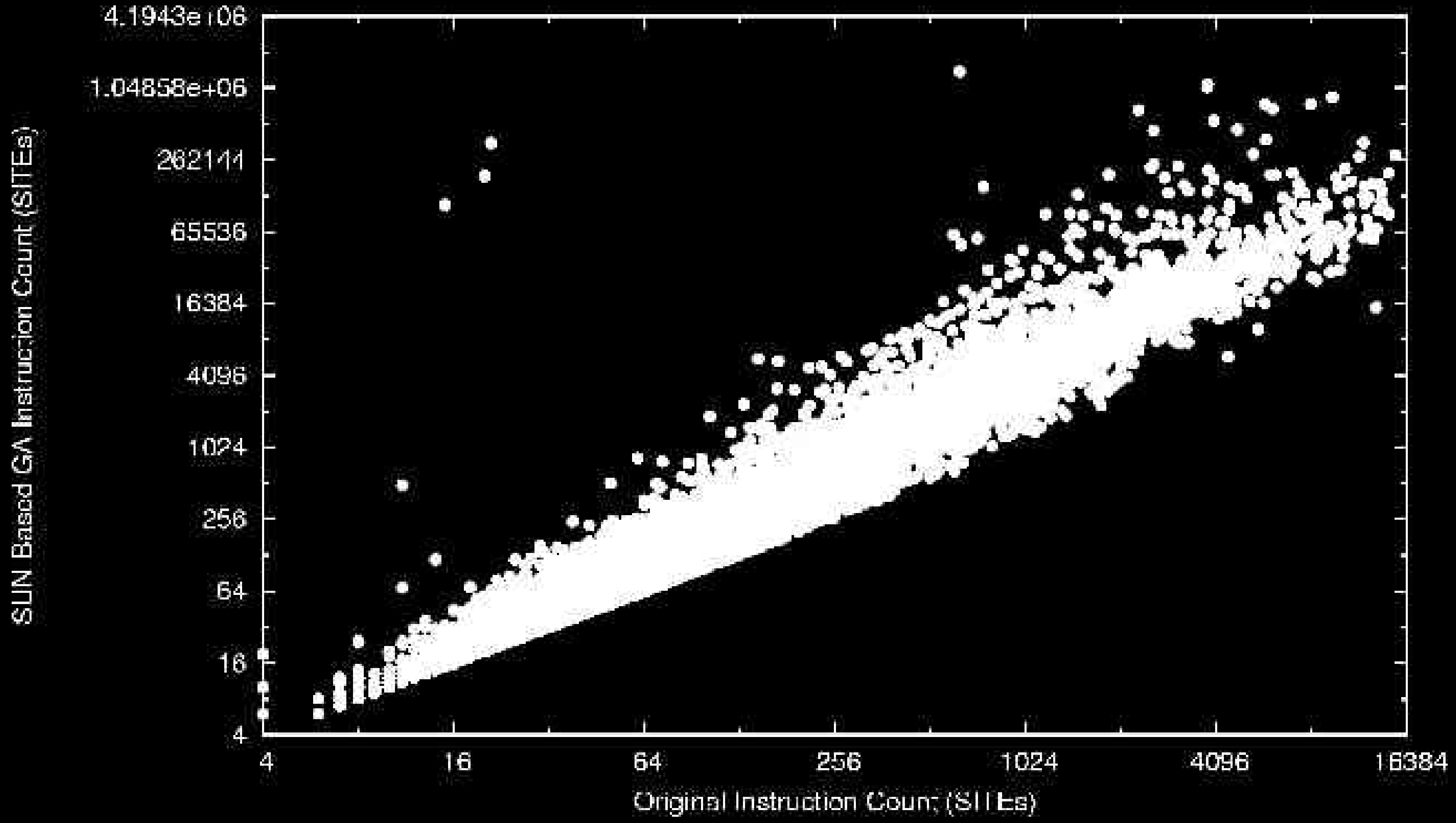
- Results from 32,912 accepted test cases... the same ones used for the reordering GA, so direct comparison of results is valid
- The goal was to minimize MAXLIVE, secondarily minimizing number of SITEs
- Execution time was limited to about 1 minute per test case on an Athlon XP

SUN GA Vs. Original MAXLIVE



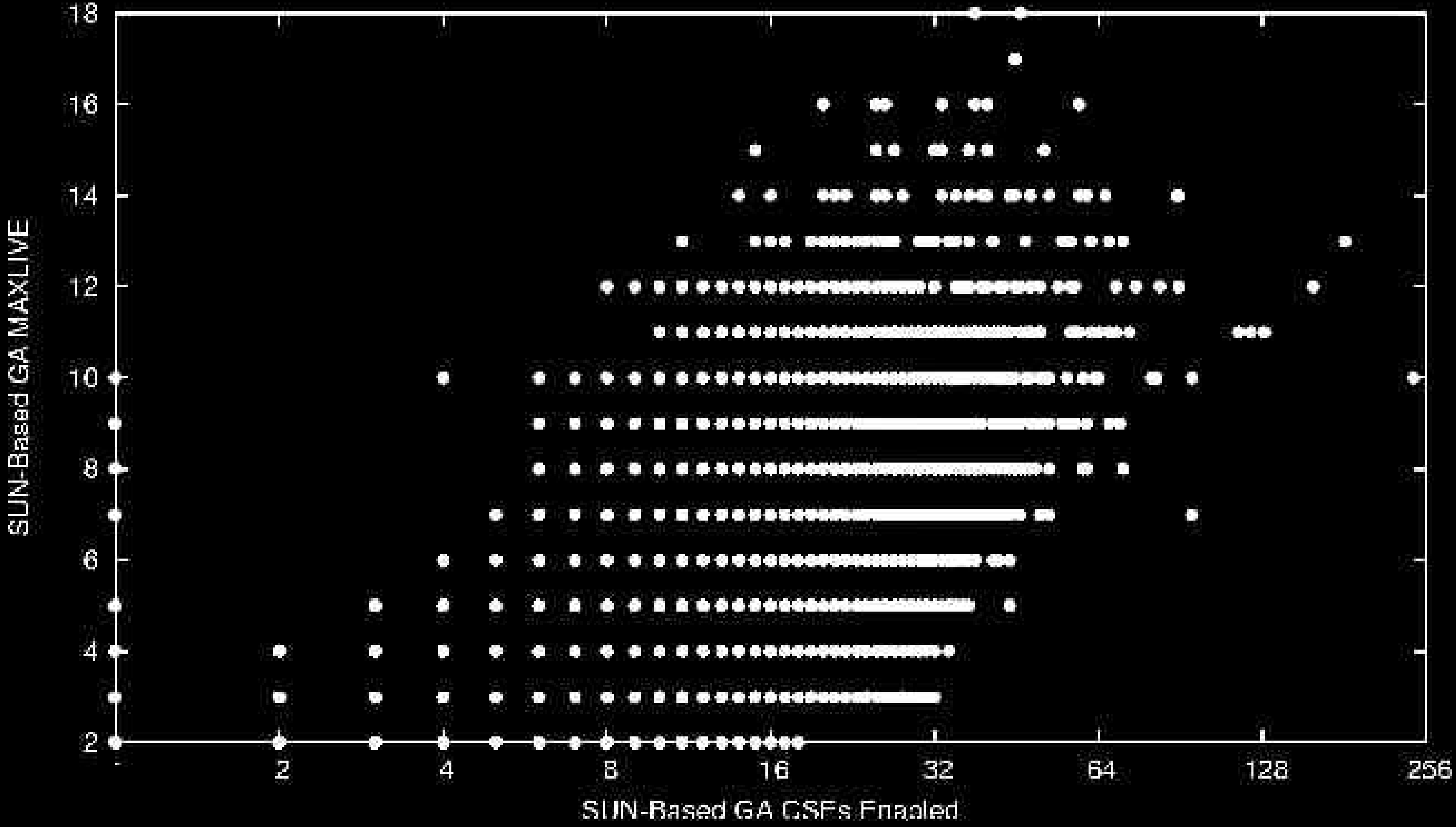
UNIVERSITY OF KENTUCKY

SUN GA Vs. Original SITEs



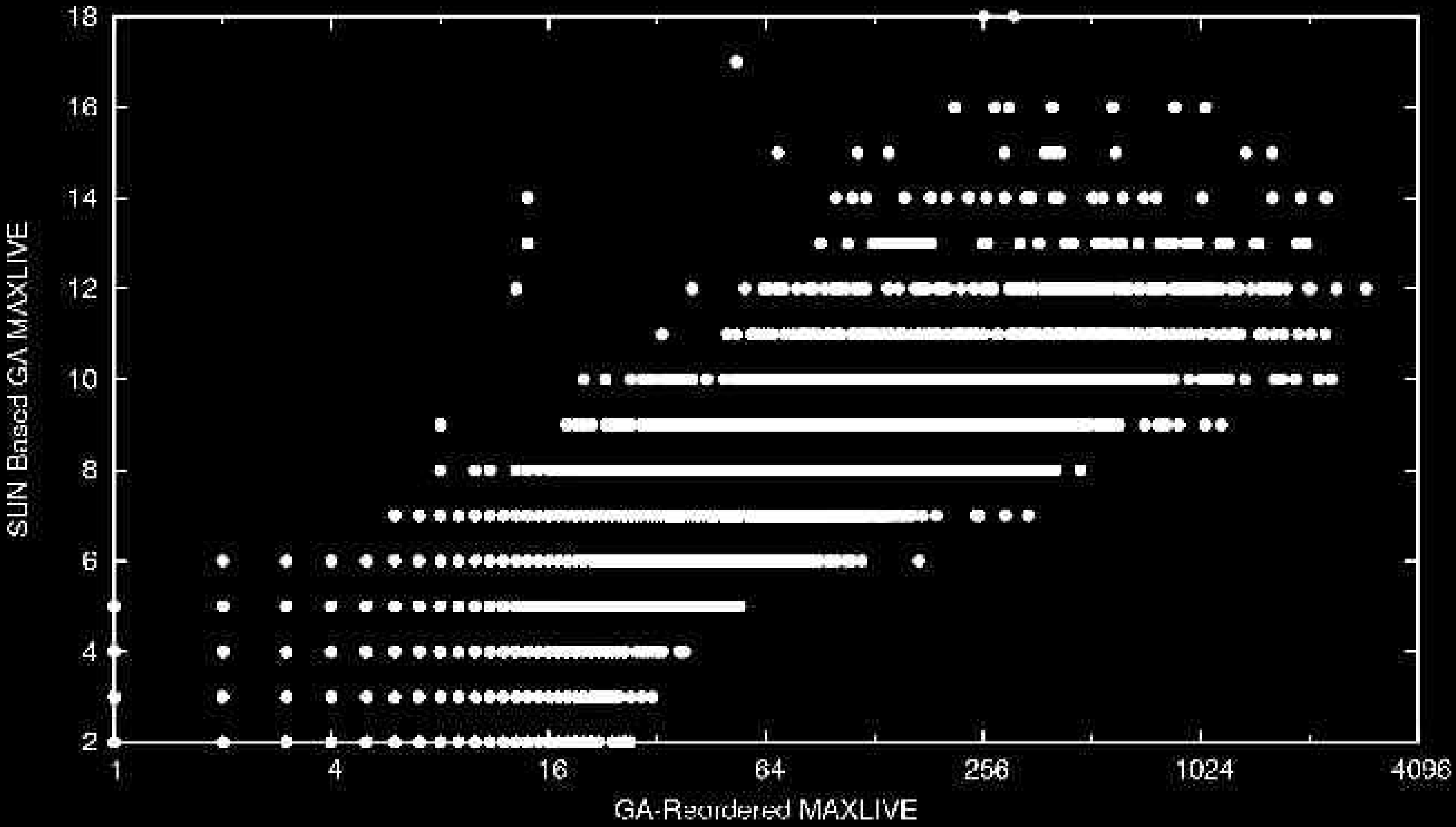
Kentucky UNBRIDLED SPIRIT

SUN GA MAXLIVE Vs. CSEs Enabled



UNIVERSITY OF KENTUCKY

SUN GA MAXLIVE Vs. Reorder GA MAXLIVE



Kentucky
UNBRIDLED SPIRIT

Summary

- The Reordering GA should be widely used
- The SUN-Based GA is very aggressive:
  - 8X increase in SITEs was common, worst was 15,309 and became 1,431,548
  - MAXLIVE reduction also was huge, from a maximum over all test cases of 3,409 to 18 (a 189:1 improvement!)
  - Fortunately, targeting a specific MAXLIVE can greatly reduce SITE count

Future Work

- SUN GA uses modified SUN order within trees; how should we order across trees?
- How well will SUN GA work for conventional processors?
- Can we incorporate substitution of equivalent arithmetic expressions?

Questions?

Kentucky
UNBRIDLED SPIRIT ™

Aggregate.Org
UNBRIDLED COMPUTING