

Why We Have Benchmarks: Restoring Perspectives

Andrew B. Kahng

UCLA Computer Science Department

<http://vlsicad.cs.ucla.edu/>

Benchmarks Capture Use Models and Context

- Heuristic engines/algorithms are tuned to prevailing benchmarks
- Benchmarks must capture the driving application
 - e.g., hypergraph bipartitioning driven by top-down cell placement
 - community does not accept generated benchmarks
- Benchmarks can be fatally flawed by details
 - hidden or previously unrealized
 - previously believed insignificant

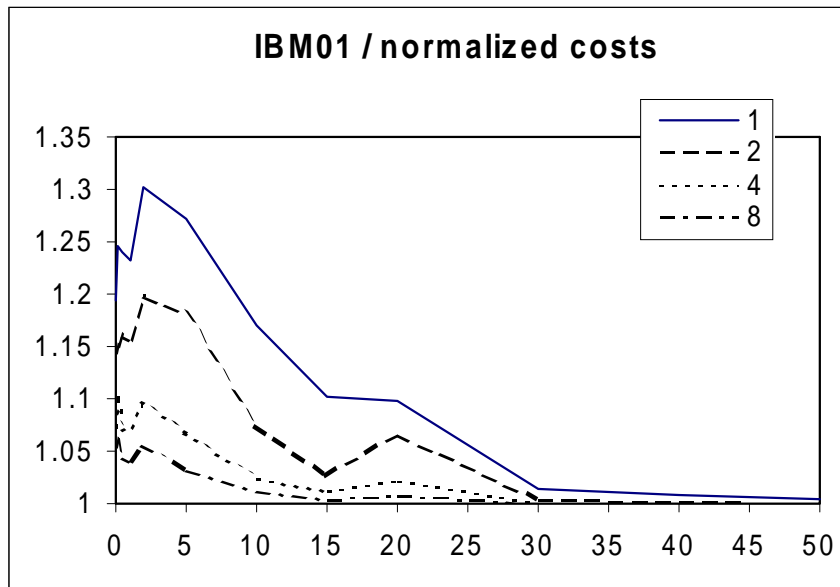
Weaknesses of Previous Benchmarks

- MCNC
 - only small cells, no signal directions, no terminal locations
- ISPD98
 - no large nets, no terminal locations
- Steps to take as a community:
 - augmented standard formats, complete benchmarks
 - e.g., (.fix, .blk) for terminal locations in partitioning
 - replay evolution of heuristic technology when new benchmarks appear
 - e.g., CLIP-FM implementations fail on ISPD98 suite

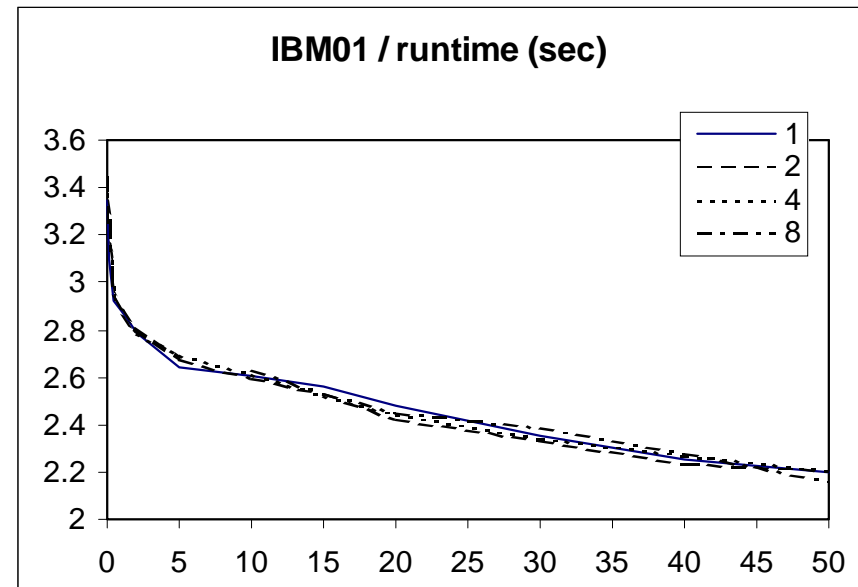
Researchers Must Use Benchmarks in Context

- Match application, benchmarking use models
- Top-down placement use model for partitioning
 - 2 sec CPU for 25K cells (200MHz Sun Ultra-2), 60 sec for 500K cells
 - cells have widely varying areas
 - *every* partitioning instance in top-down placement has fixed terminals
- Non-problems in top-down placement
 - unit-area partitioning of standard-cell netlists
 - partitioning "free" hypergraphs without fixed terminals

Effect of Fixed Terminals



Normalized Cost for IBM01



Runtime for IBM01

Benchmark Abuse

- Excuse to work on non-problems
- Report experimental results in a useless way
 - report only on a biased selection of benchmarks
 - interpret/transform benchmarks in non-standard way
 - irrelevant use model (e.g., best of 100 starts, 22000 sec runtime on Primary2)
 - over-tune to benchmark set, beat up on strawmen, ...
- Steps to take as a community:
 - clear "attached" context for benchmarks and for published algorithms
 - reporting requirements enforced by reviewers
 - larger set of benchmarks

Why Reference Implementations Are Needed

- Papers do not give details necessary to replicate results
 - amazingly, true even for "classic" algorithms
- Published reference implementations will raise quality
 - minimum standard for algorithm quality
 - **new community standard:** reinforced by reviewers, editorial policies
 - will reduce barrier to entry for new researchers
- Example: partitioning
 - only one reference implementation (Dutt/Deng LIFO-FM)
 - should be used more often
 - but is outdated (only does unit-area, flat LIFO-FM)

Magnitude of the "Barrier to Entry"

- Code development barrier
 - bare-bones self-contained partitioner: 800 lines
 - not leading-edge (Dutt/Deng LIFO-FM)
 - modern partitioner requires much more code
- Expertise barrier
 - very small details can have stunning impact
 - must not only know what to do, but also what not to do
 - impossible to estimate knowledge/expertise required to do research at leading edge

Summary of Steps to Take as a Community

- Remember why we have benchmarks in the first place
- Usable benchmarks
 - complete (e.g., .blk/.fix files for partitioning in top-down placement)
 - attached use-model context
 - wider selection
- Correct use of benchmarks
 - match application, benchmarking use models
 - complete reporting
 - community-enforced standards
- Benchmark implementations
 - portable, extendible, written in standard language
 - must be compared with