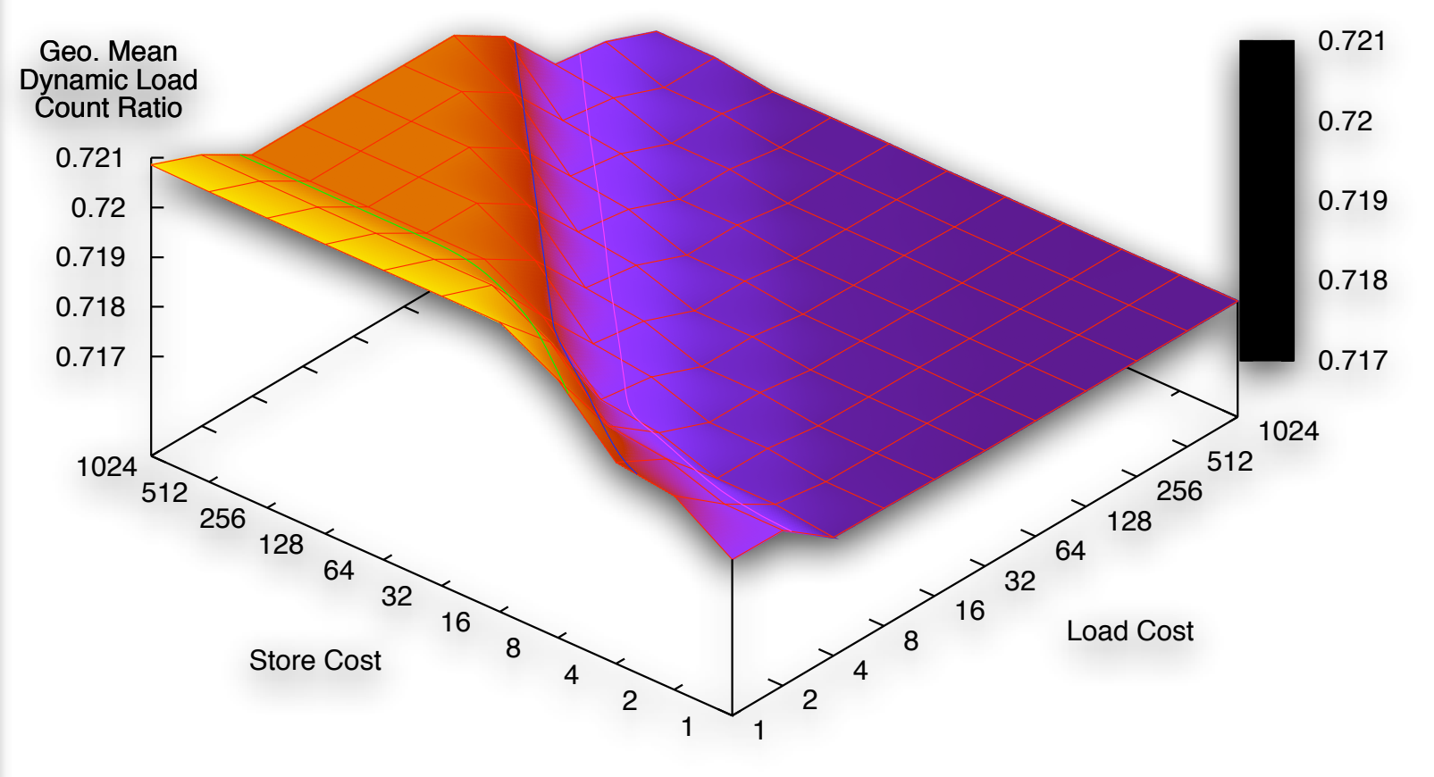
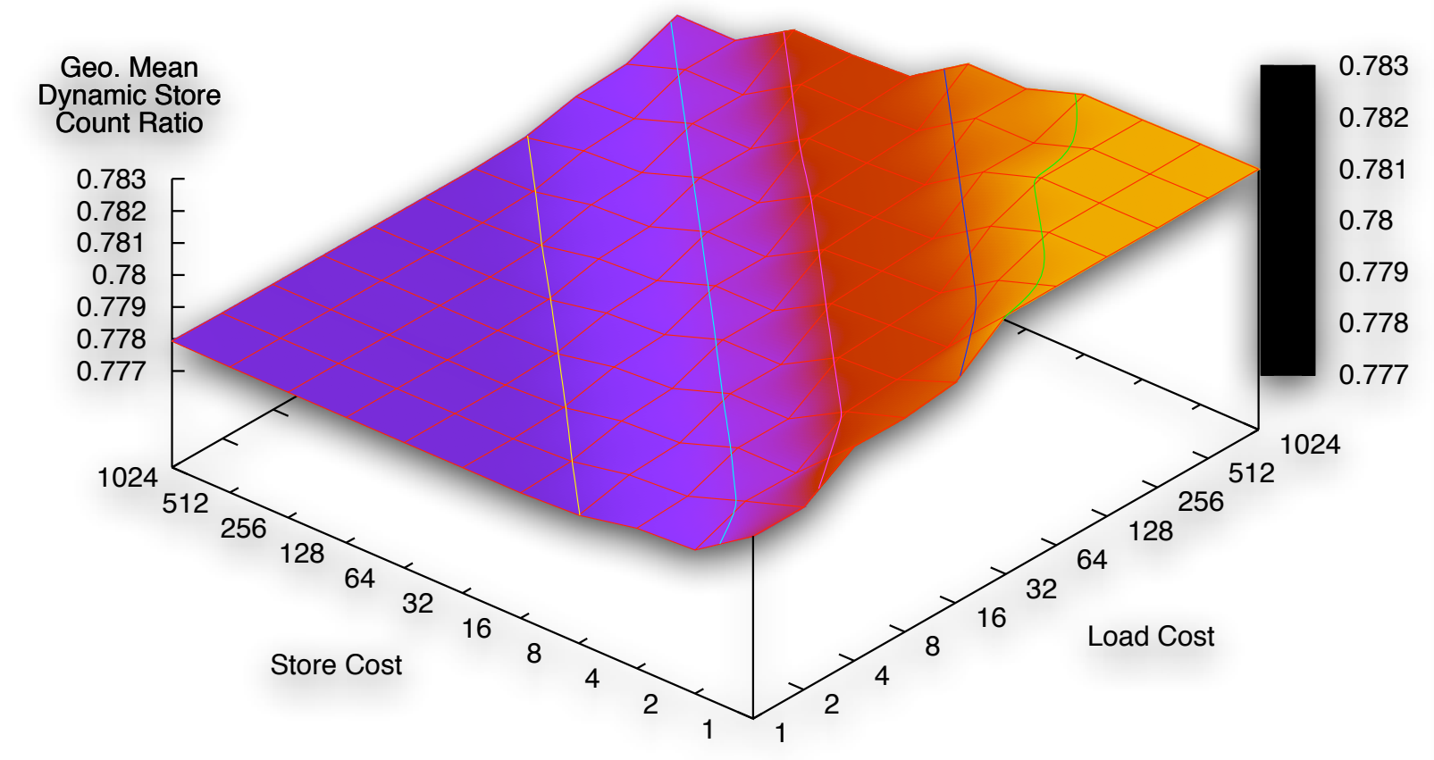


Performance Metrics for Optimal Register Allocation

David Koes
Advisor: Seth Copen Goldstein
Carnegie Mellon



Problem

Compiler optimizers that use an explicit model, such as an integer linear program, to find optimal solutions to classical optimization problems have become a feasible alternative to traditional, suboptimal heuristic optimizers. However, the actual optimality of these techniques depends on the accuracy of the model. Existing model-based optimizers that target processor performance do not attempt to accurately represent all the features of a modern desktop processor. Instead, simple metrics such as minimizing the dynamic total instruction count or memory instruction count are used. The goal of this work is to answer three questions in the context of optimal register allocation:

How effective are simple performance metrics? *How important is the choice of metric?*
What is missing from simple performance metrics?

Methodology

A large space of simple performance metrics for an ILP-based optimal register allocator is explored using the Intel x86 architecture (1.8GHz Pentium 4 and 2GHz Core Duo) and gcc compiler (version 3.4.4, compiling with -O3).

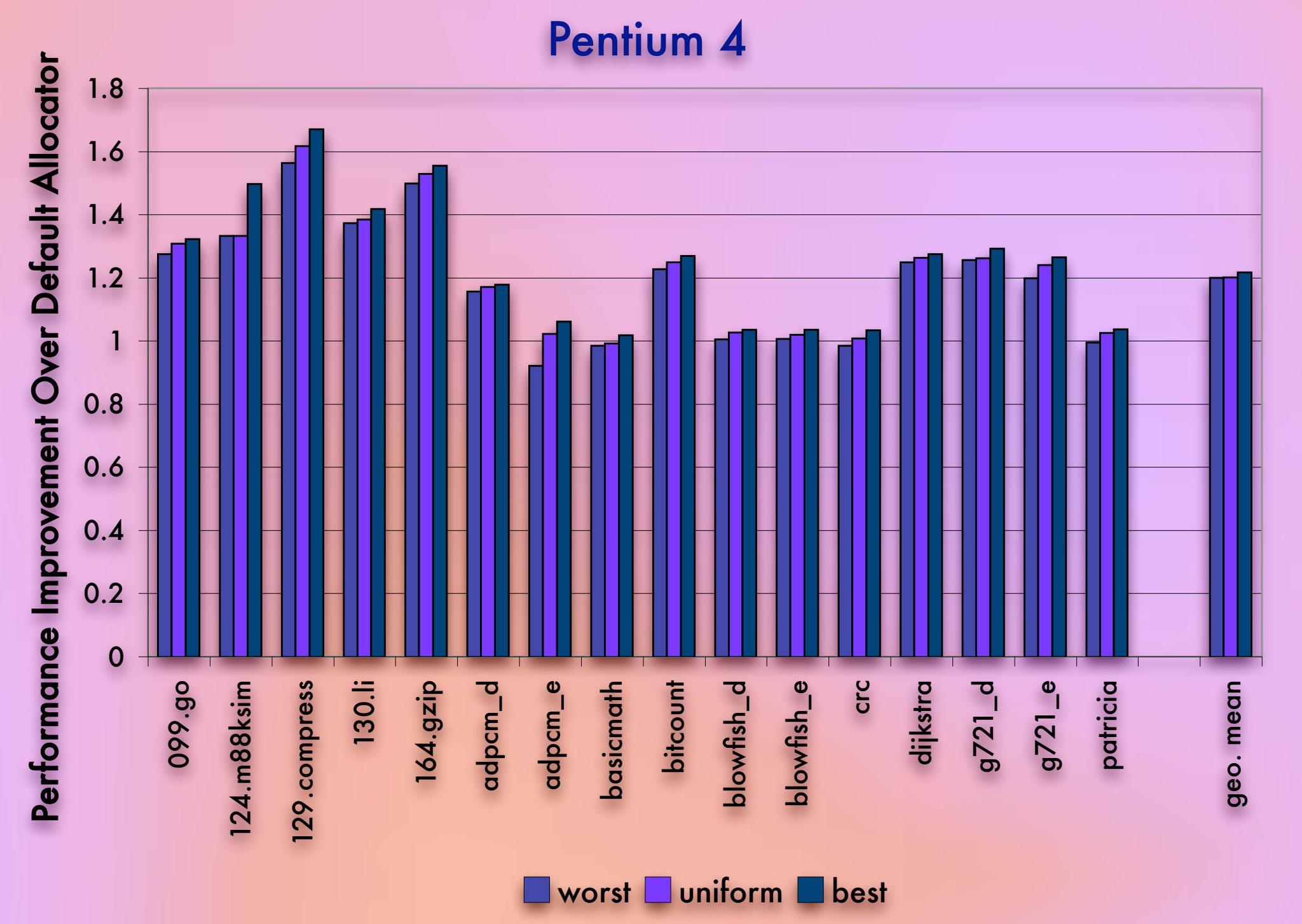
Performance Metric Framework:
Three separate costs:
moves, loads, and stores
Costs multiplied by execution frequency
Optimal allocation minimizes total cost

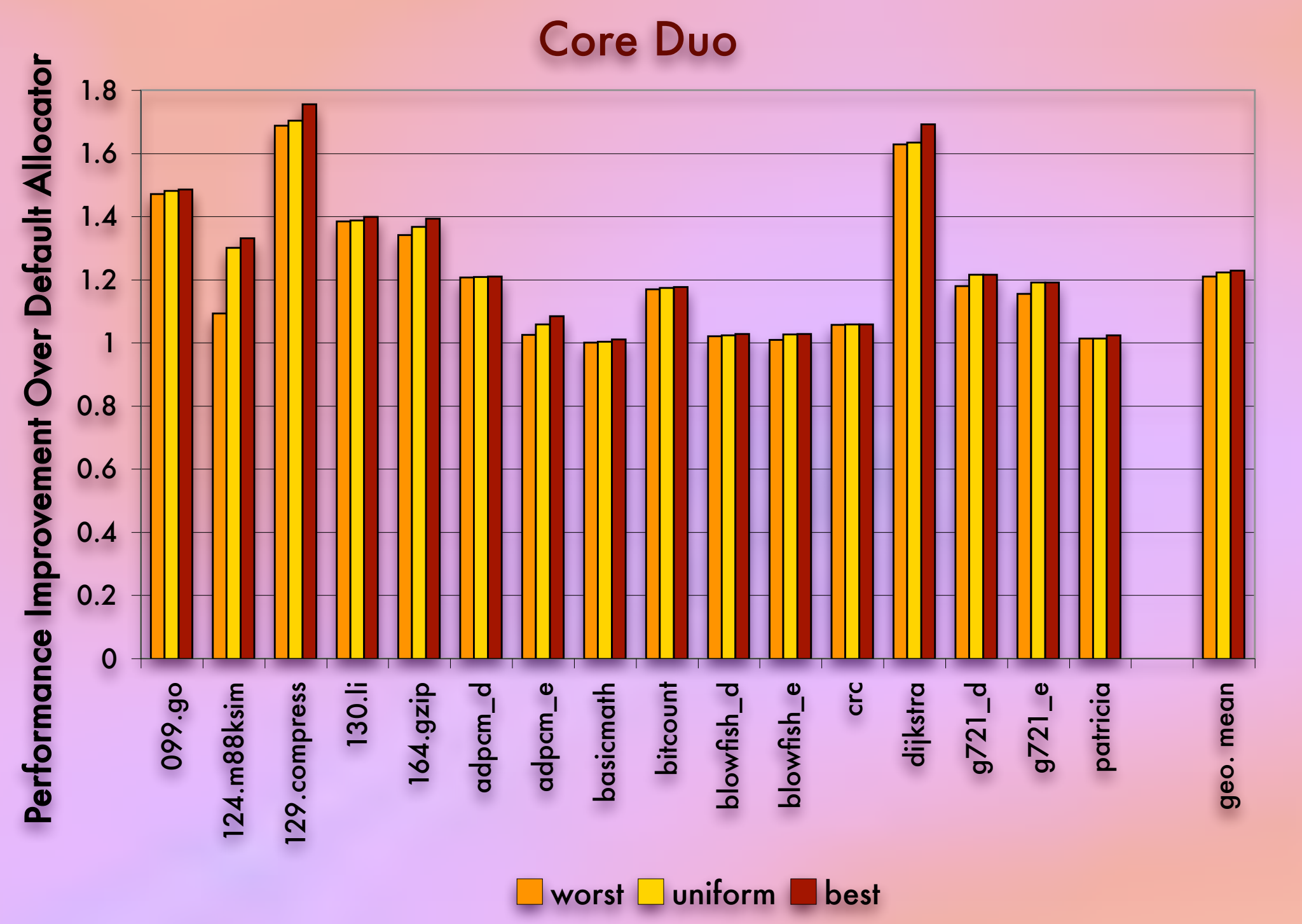
Evaluation:
Use profile data for exact execution frequencies
No optimization after register allocation
Measurements use hardware performance counters
Only fully optimal benchmarks evaluated

Expectation:
Meaningful and understandable correlations between various cost ratios and performance will be observed (e.g., a metric where moves cost more than loads will be slower).

1. How effective are simple performance metrics?

More than 100 performance metrics were evaluated with load and store costs ranging between 1 and 1024 and move costs ranging between 1 and 8. The performance of the best and worse performing metric for each benchmark, as well as the performance of a uniform cost metric (which minimizes the dynamic instruction count) is shown below. Overall, an average performance improvement of about 20% is achieved with some benchmarks demonstrating a 50% or better performance improvement. Although for many benchmarks the performance difference between the best and worst metric is small, for several the difference is larger than 5%.

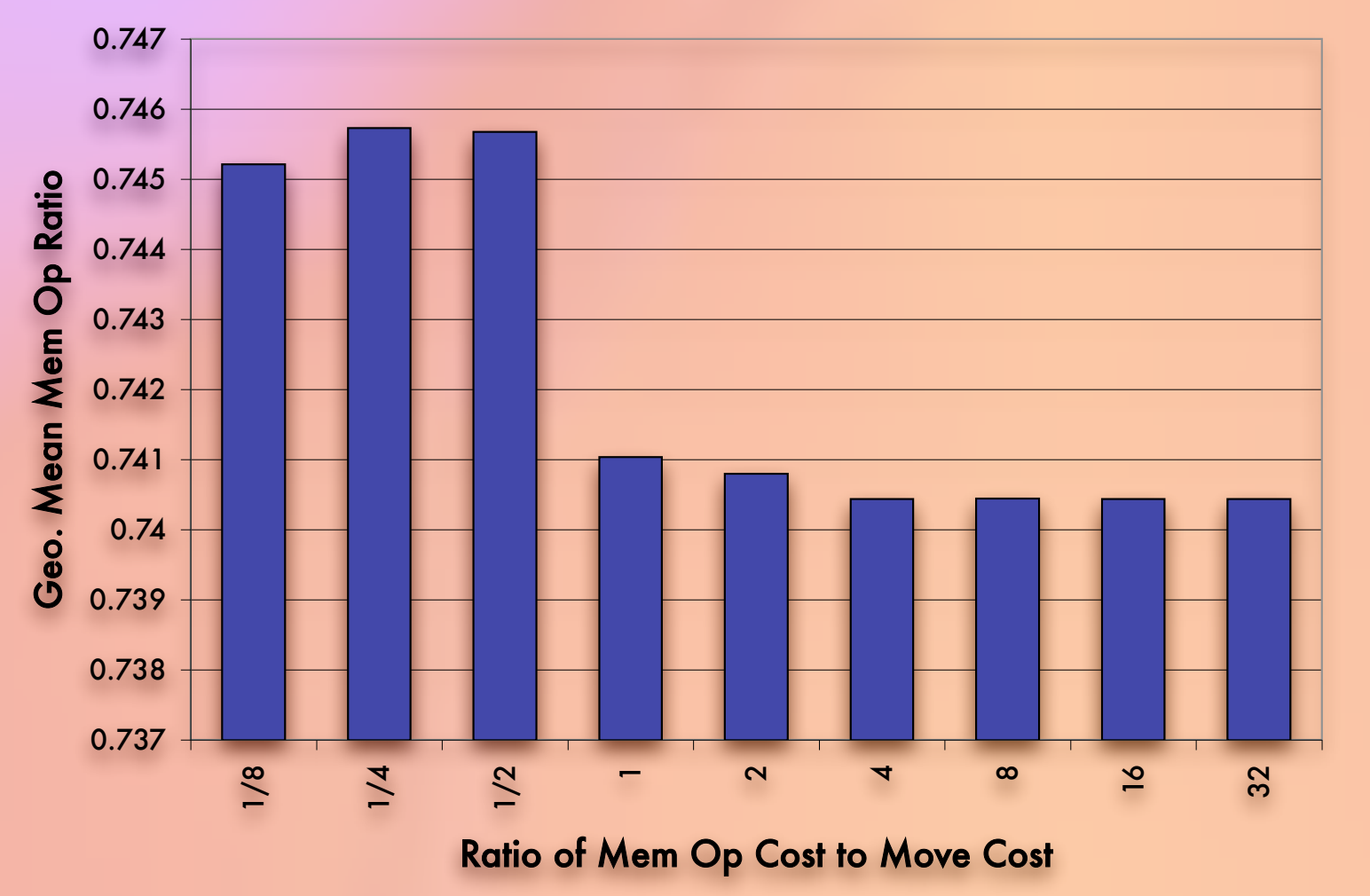


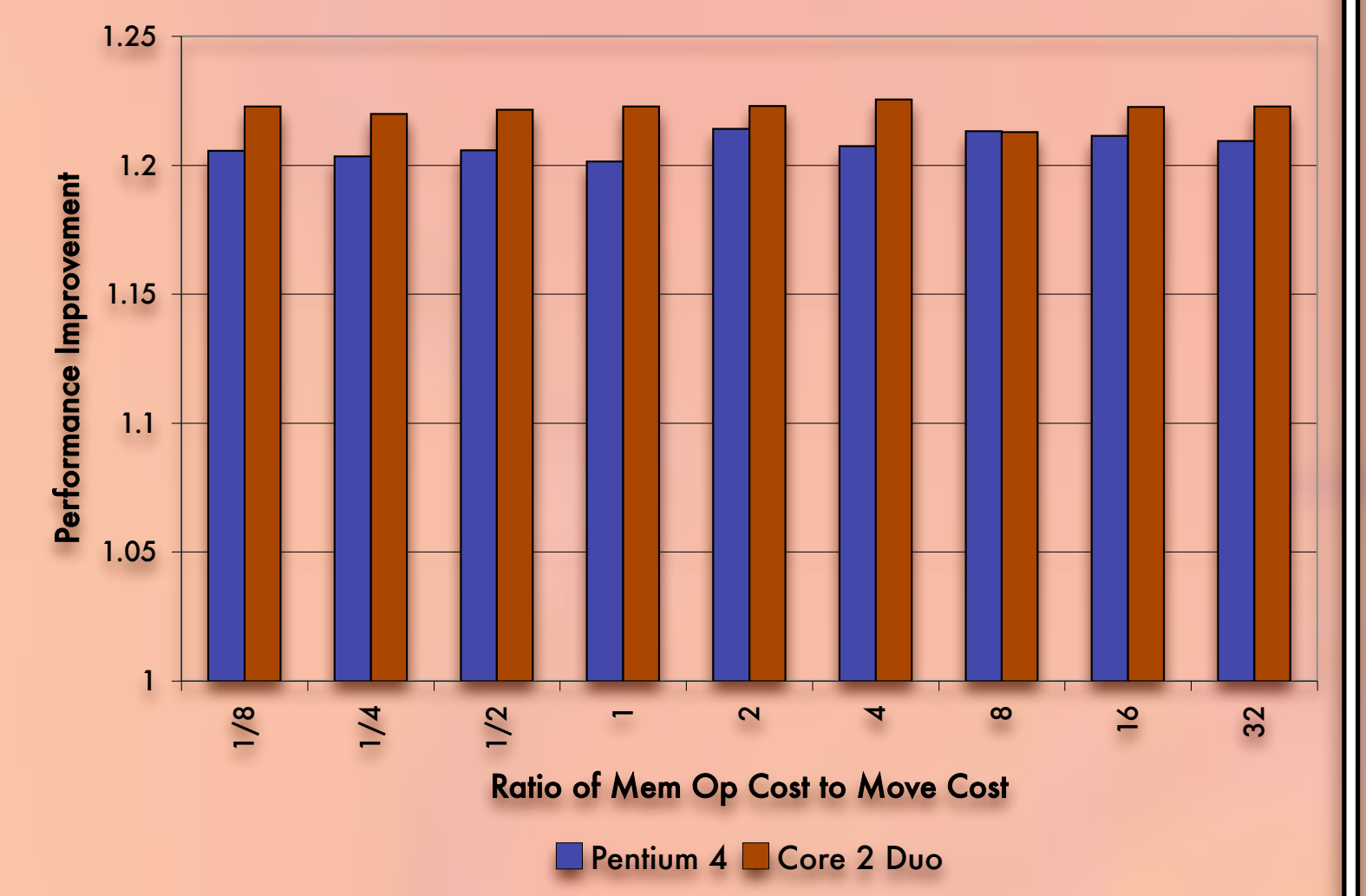


Conclusion: Simple performance metrics can be very effective at improving performance. In addition, the performance difference between the best and worst metrics implies that the choice of metric may sometimes play a meaningful role.

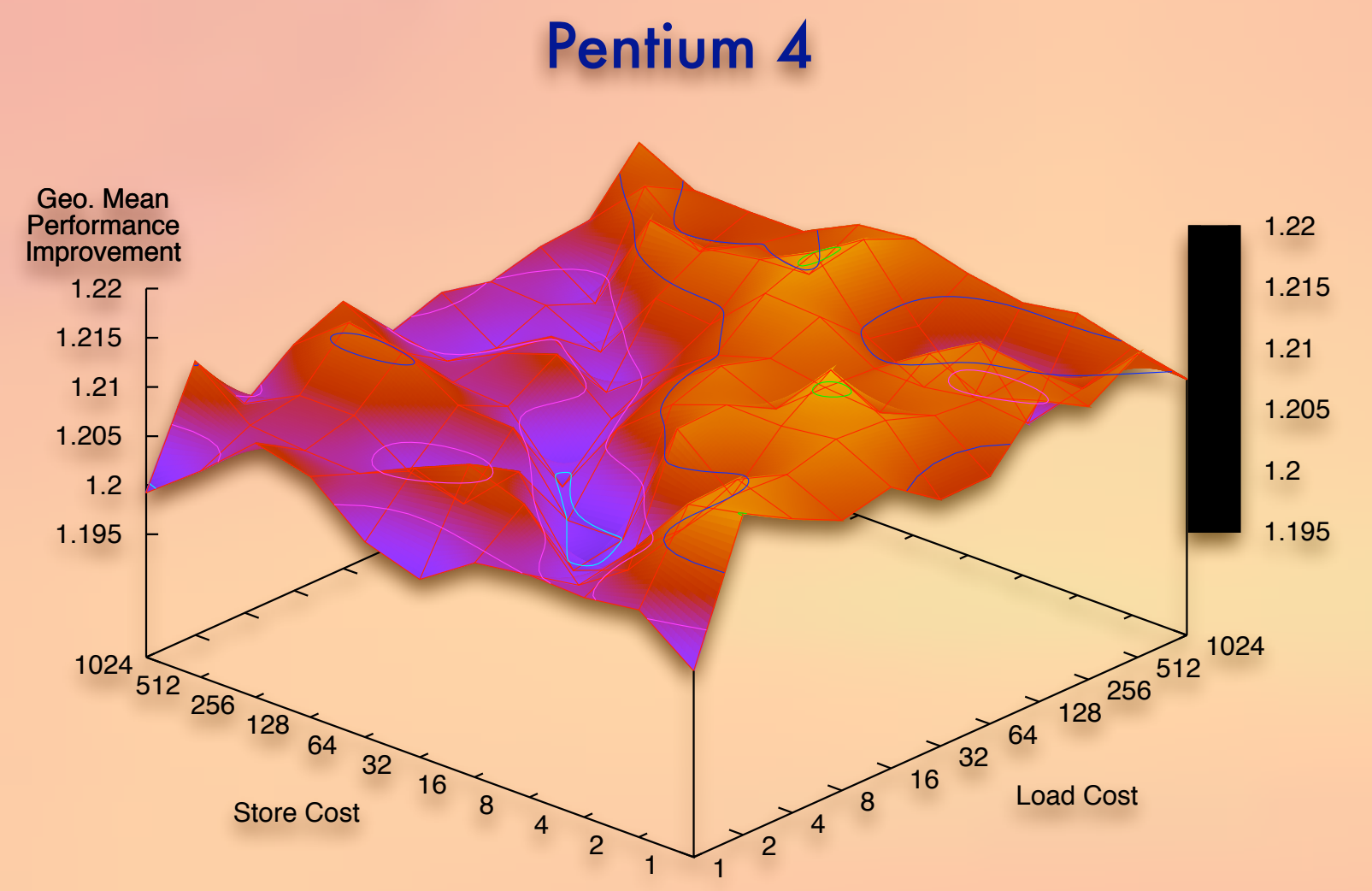
2. How important is the choice of metric?

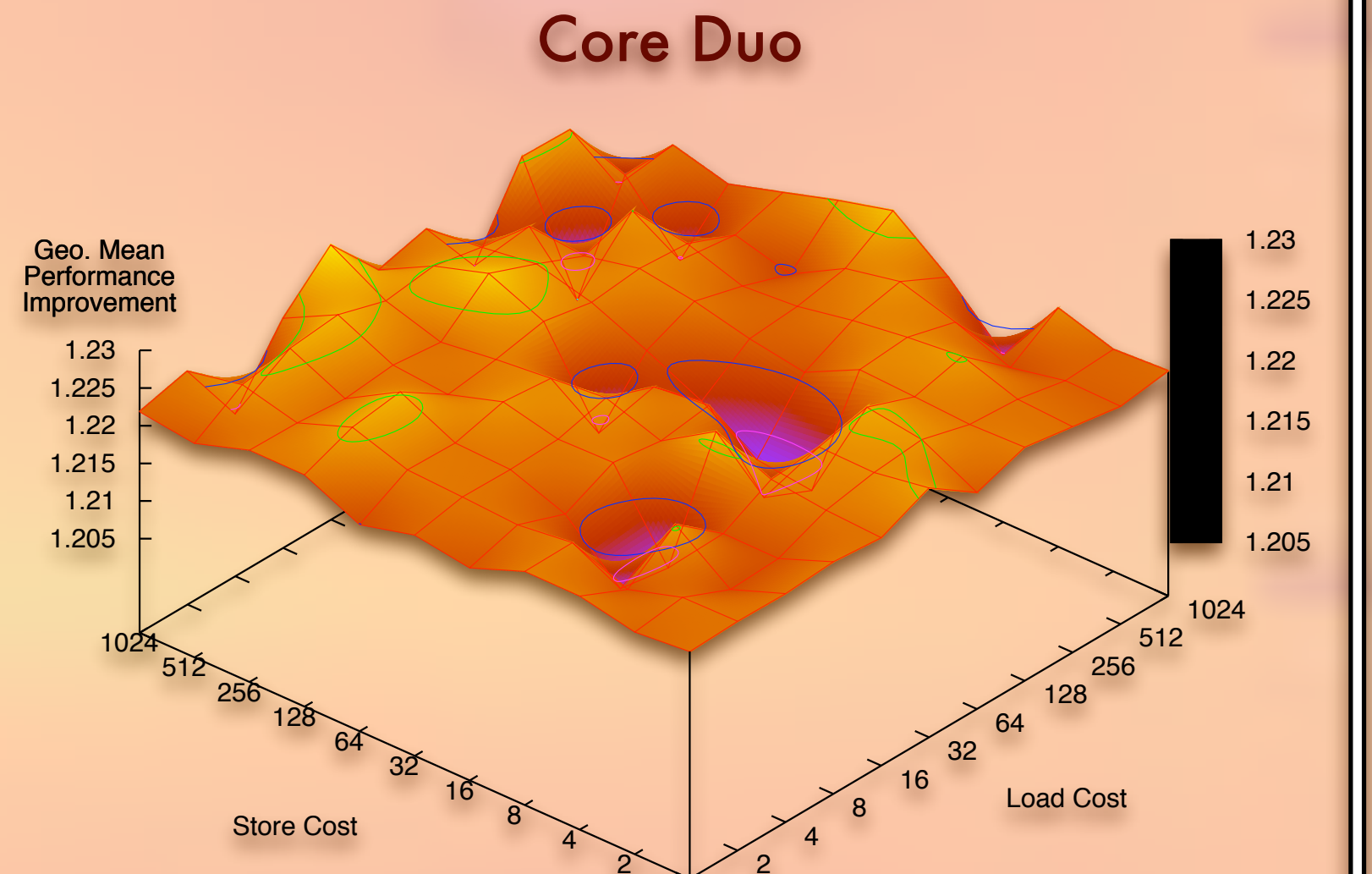
The importance of move costs in a simple performance metric is evaluated by fixing the costs of loads and stores while varying the cost of a move instruction. Although increasing the cost of a move does result in more memory operations being performed (see graph at immediate right), this does not discernibly effect the performance (see graph at far right).

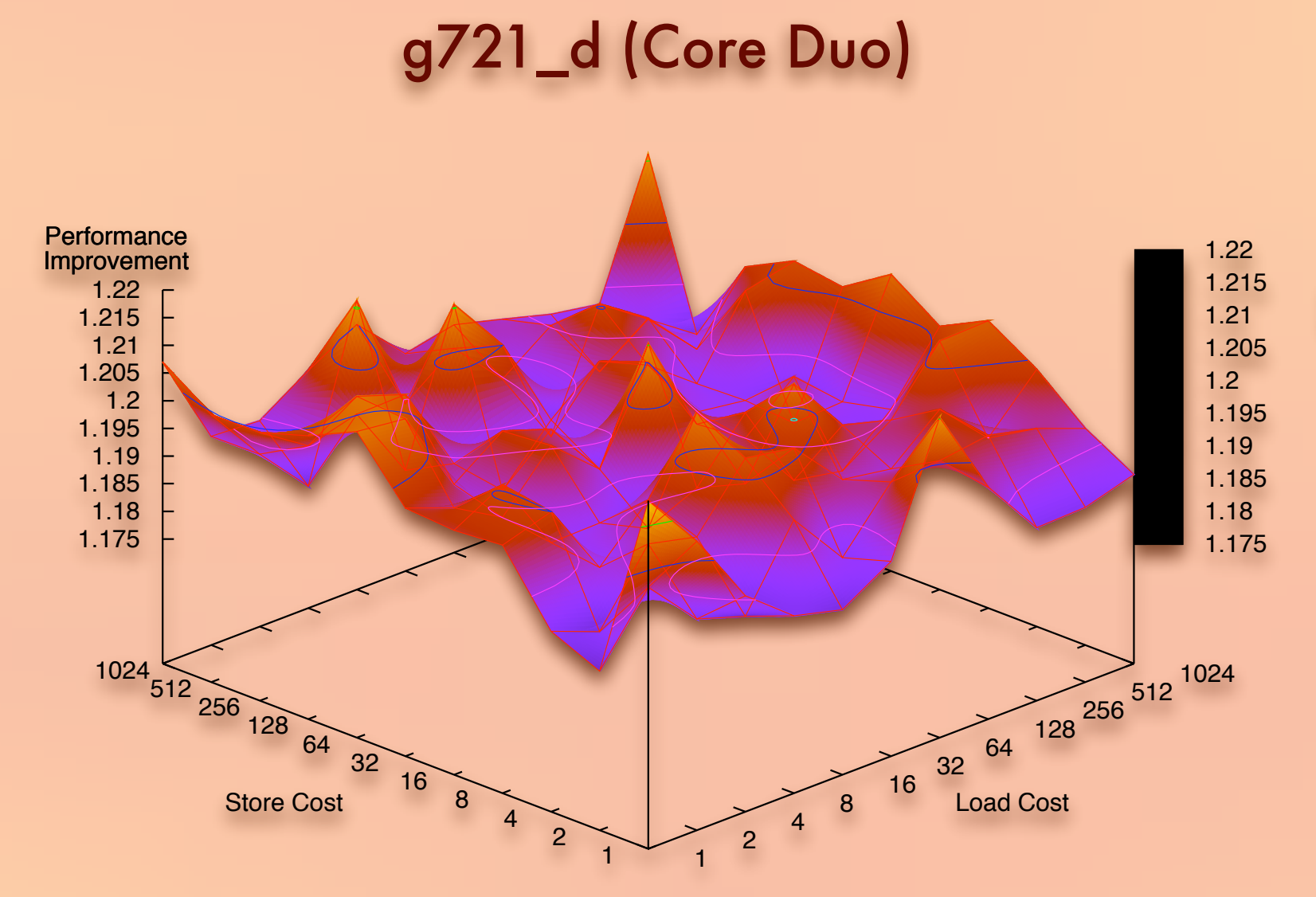


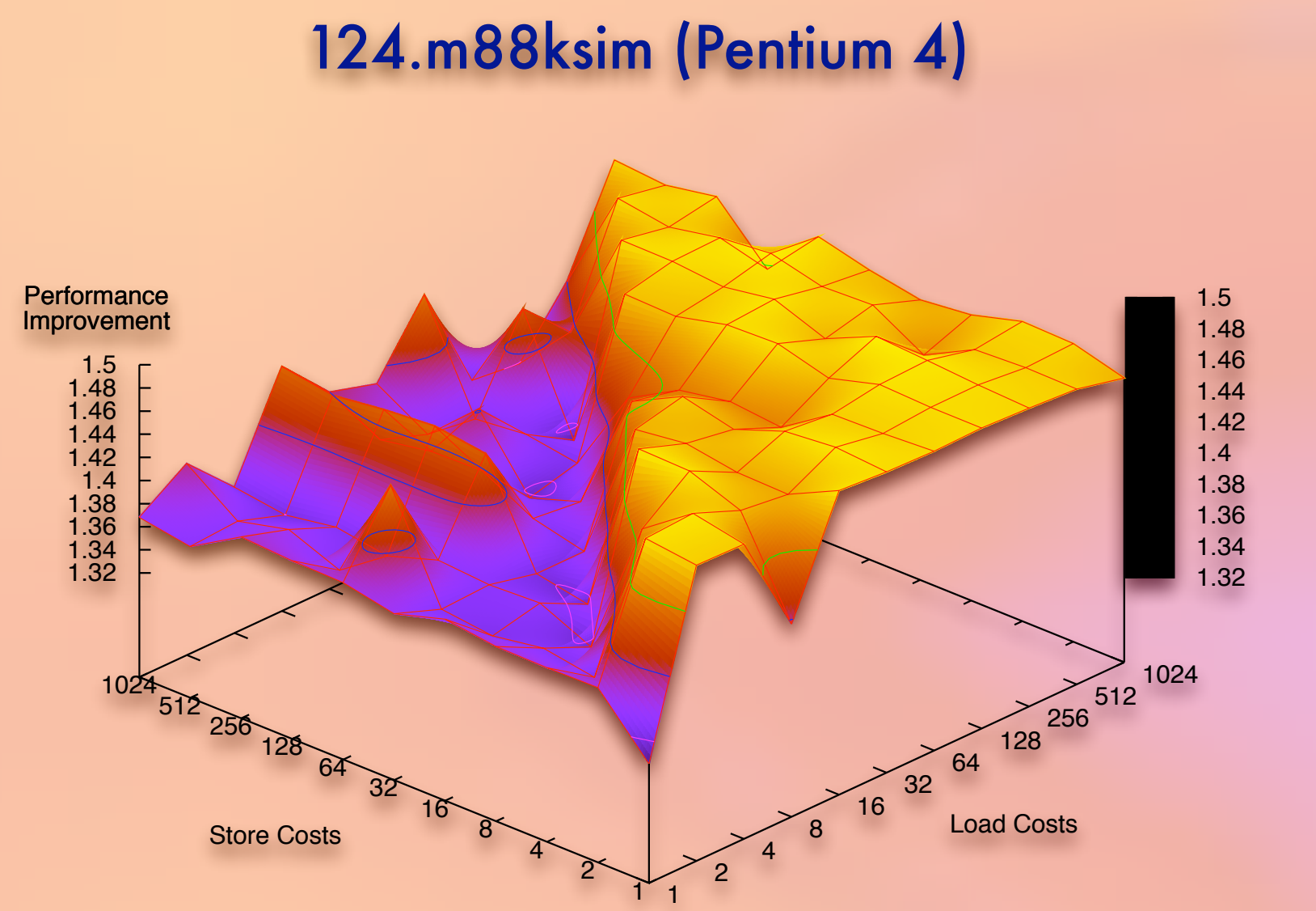


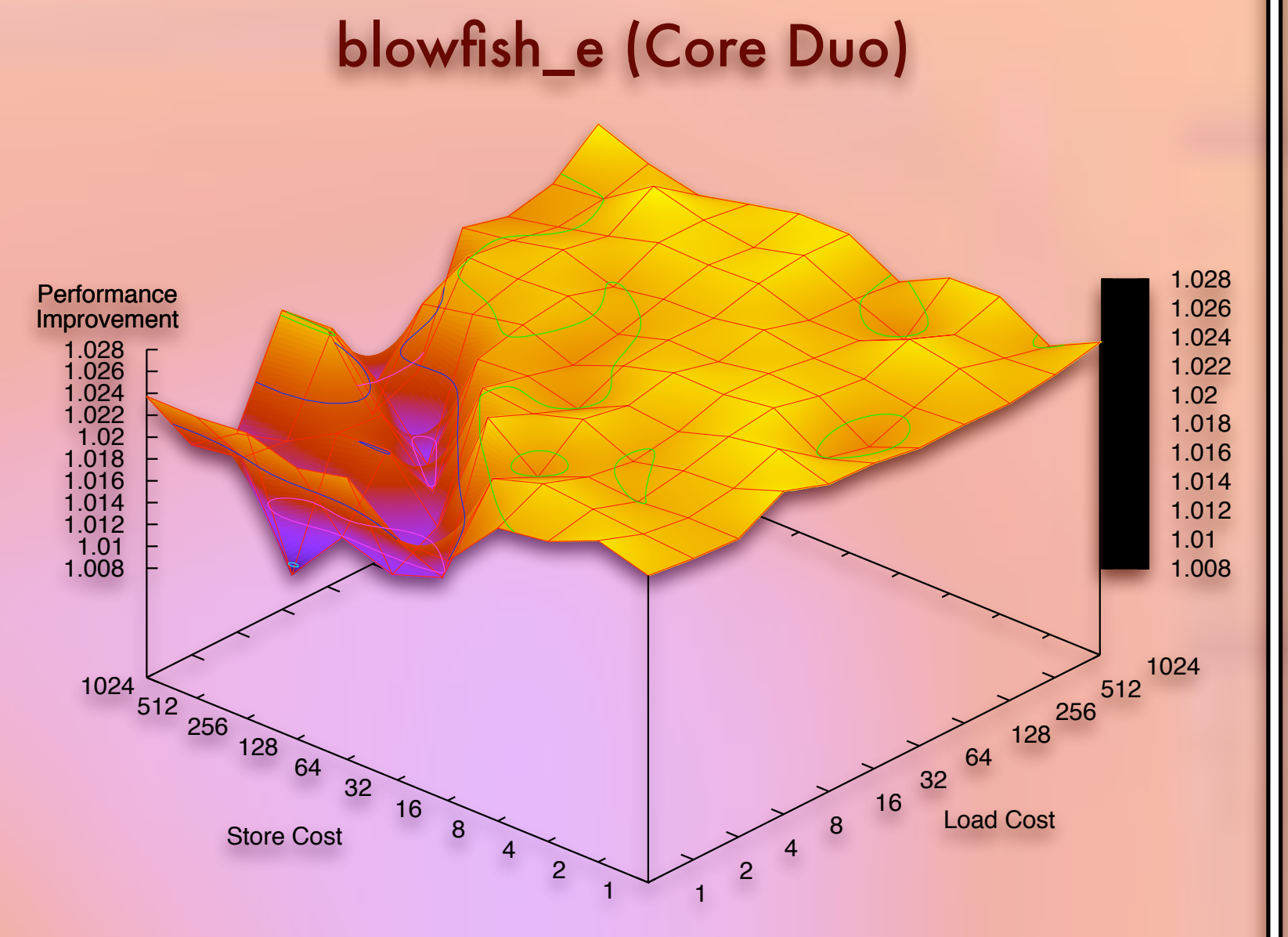
The importance of load and store costs is evaluated by varying these costs with respect to a unit move cost. The average performance improvement across this space for both architectures is shown at right. Although the metrics have the expected effect on the number of dynamic loads and stores (see graphs in title), neither result displays a discernible pattern. For example, g721_d displays no obvious correlation between performance and metric (below left) despite the strong, expected, correlation between dynamic memory operation count and metric (below left, second row). In fact, only two benchmarks exhibit a definite relationship between metrics, dynamic memory operations, and performance. 124.m88ksim on the Pentium 4 (below center) and blowfish_e on the Core Duo (below right) clearly prefer load cost dominant metrics.

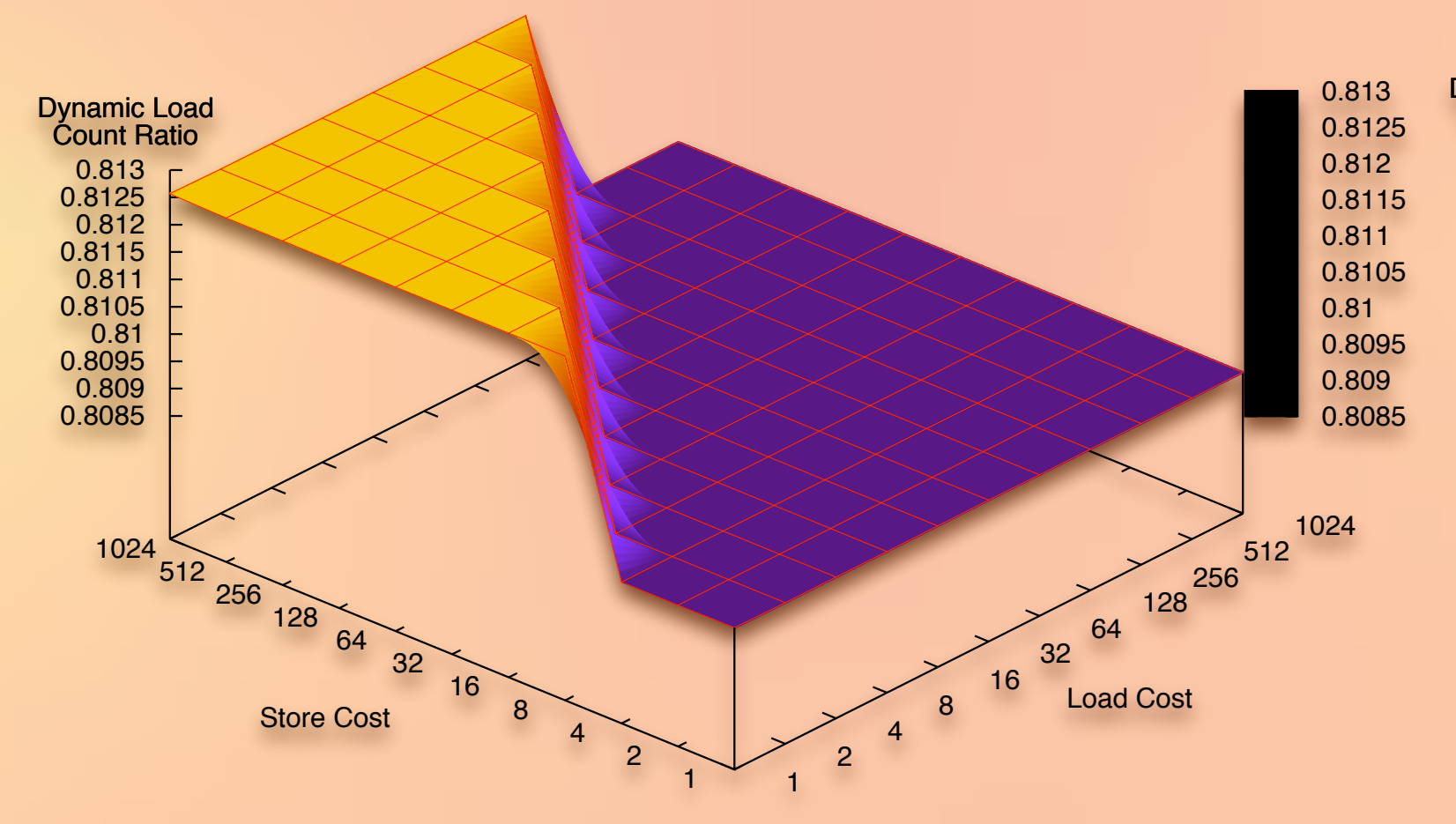


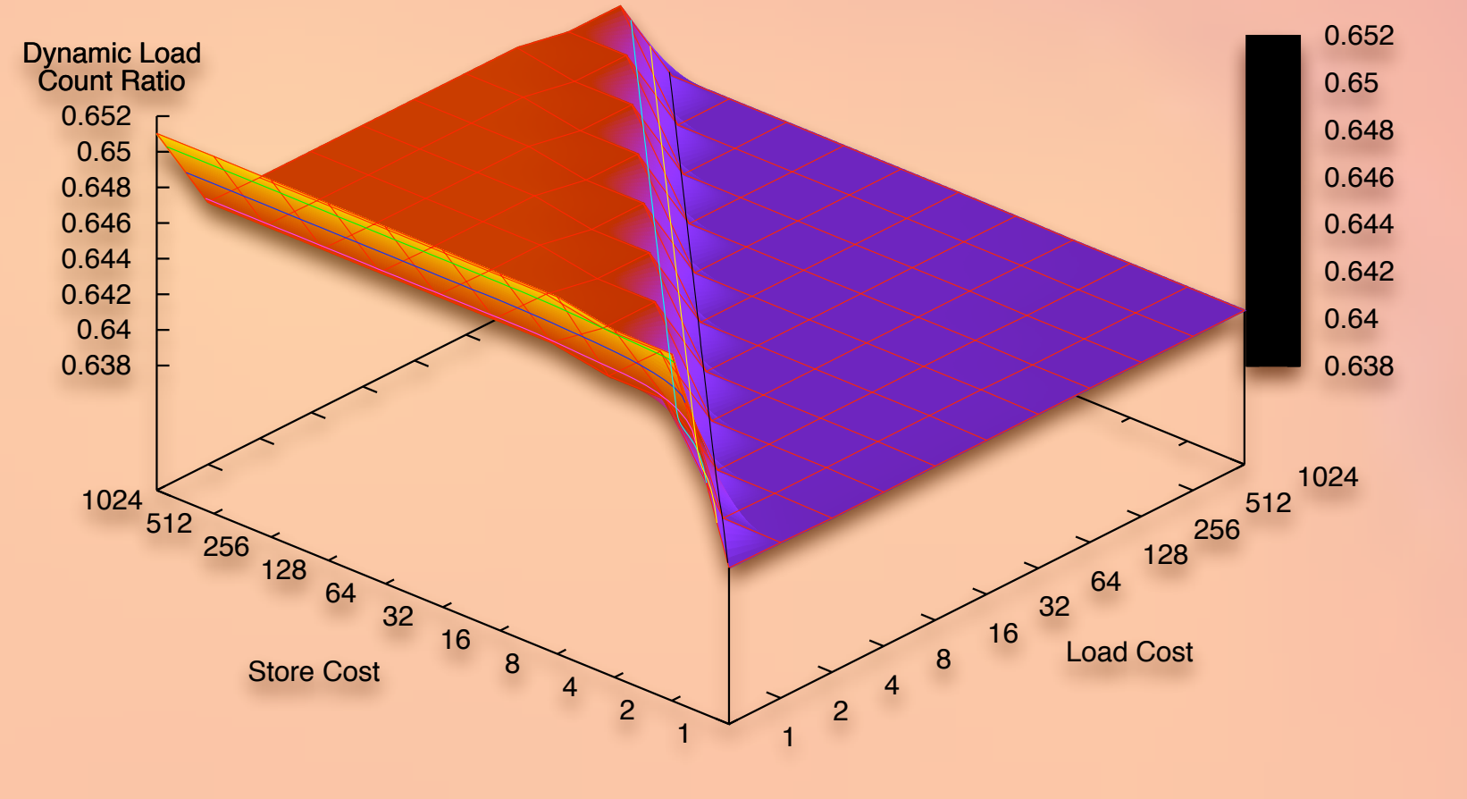


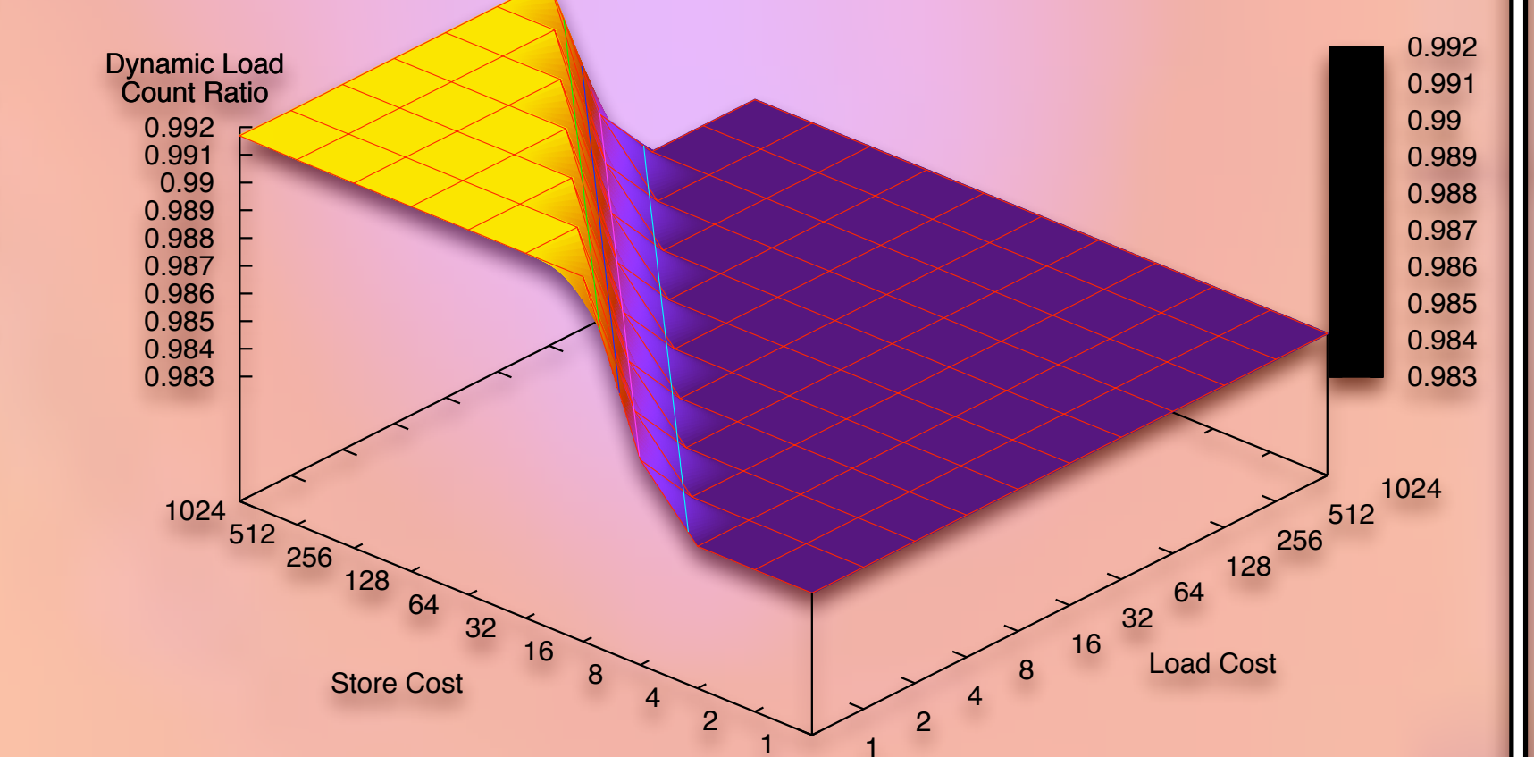










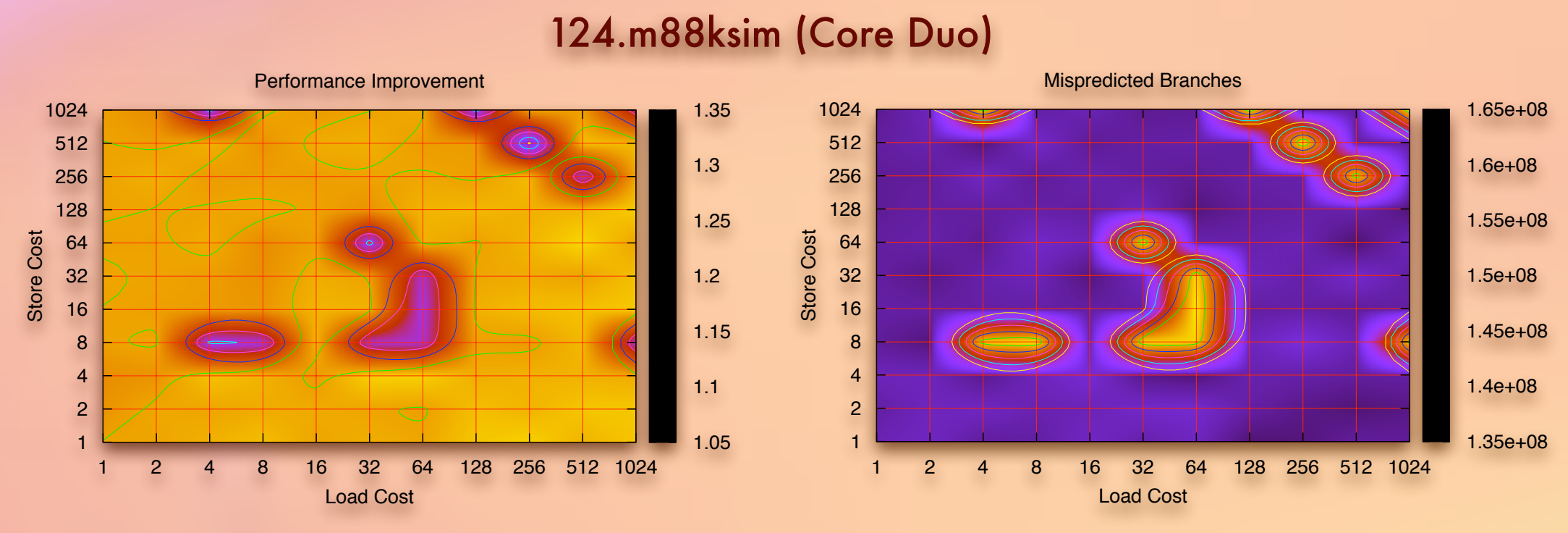


Conclusion: The cost of moves in a simple performance metric has a negligible impact on performance. Although in some cases the load/store cost ratio has a demonstrable effect on performance, in most cases other factors appear to have a more significant impact. This implies that an optimal register allocator targeting processor performance would have to adopt a more complex model than simply minimizing the number of dynamic memory operations in order to fully maximize performance.

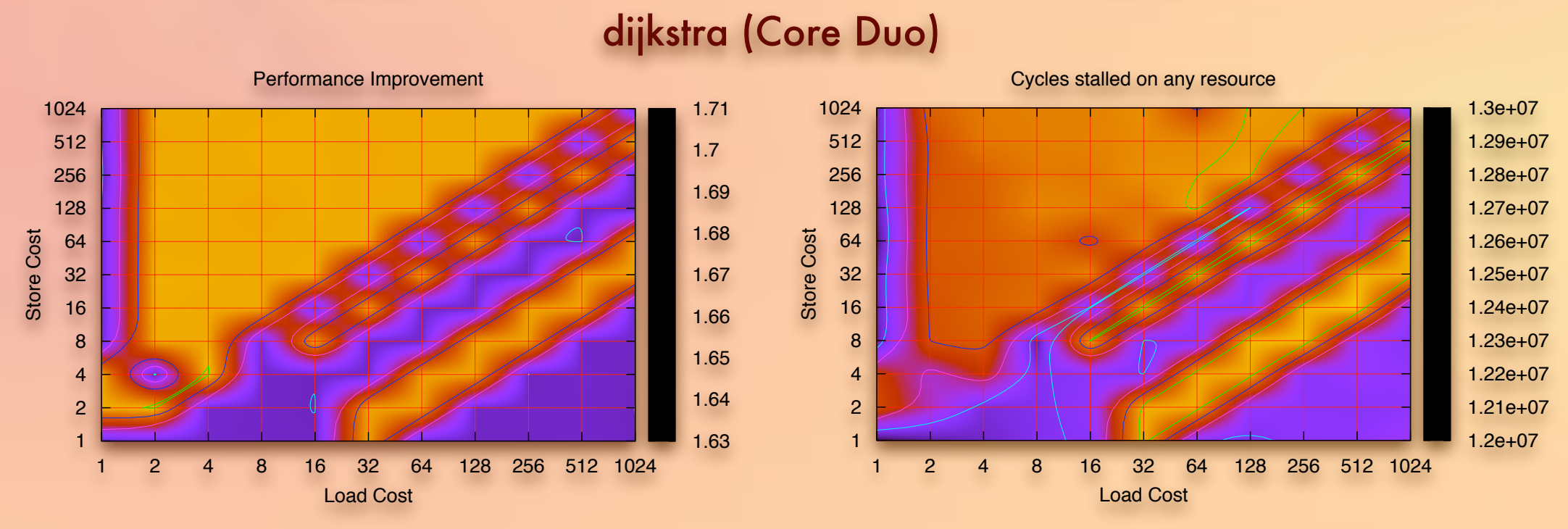
3. What is missing from simple performance metrics?

Simple metrics can satisfactorily represent a uniform cost model of memory operations, but this is not sufficient for maximum performance. A benchmark compiled with different metrics may still exhibit the same dynamic memory operation behavior, yet display significant differentials in performance. In this case the two versions may be optimal with respect to both metrics, but some other factor that is not present in the optimization model effects performance. These differences reveal additional factors that need to be taken into account when optimizing for maximum performance.

For a few benchmarks, the anomalous performance results are clearly dominated by a single non-memory factor. The performance of the 124.m88ksim benchmark on the Core Duo is dominated by mispredicted branches (see right). The removal of a single low-cost move instruction at the end of a function results in the addresses of all successive functions being bumped down to the next alignment boundary. This change of addresses apparently triggers some degenerate behavior in the branch predictor resulting in >10% differences in performance.



The performance of the dijkstra benchmark on the Core Duo is strongly correlated with cycles stalled on a resource (see right). These cycles include stalls due to register renaming buffers, memory buffers, and branch misprediction. The performance does not correlate with the branch misprediction rate nor with the cache miss rate (not shown), and the only difference between benchmark versions is the choice of register assignment. It appears that the interaction between these differing register assignments and the register renamer accounts for a ~4% difference in peformance.



Other benchmarks are not so clearly dominated by a single factor, but appear to be influenced by several factors such as the dynamic instruction count, the amount of speculation, and data cache misses in addition to branch misprediction and resource stalls.

Conclusion: Many factors impact performance and are influenced by register allocation. In many cases, these factors are complex and involve counterintuitive interprocedural interactions that depend on microarchitectural details. This complexity makes it unlikely that such factors could be meaningfully incorporated into a performance metric for an optimal allocator.