The character of the instruction scheduling problem

Darko Stefanović
Department of Computer Science
University of Massachusetts

March 1997

Abstract

Here I present some measurements that serve to characterize the nature of the problem of basic block instruction scheduling, as it is encountered in practice. Finding optimal schedules is known to be NP-hard [Hennessy and Gross, 1983]; but it is worth knowing how hard the task is in an average sense, where the average is with respect to an input space of problems (basic blocks) found in the everyday practice of compiling. The space I consider is the set of all basic blocks in all of SPEC95 benchmarks, produced by compiling on the Digital Alpha architecture [Bhandarkar, 1996]. It is also worth knowing how close a heuristic scheduler comes to the optimum, especially when one wants to design a new heuristic scheduler; I present one such evaluation for a scheduler made available by Digital and show that it is almost optimal (with respect to its own cost measure).

1 SPEC95 properties

The SPEC95 benchmark suite comprises 18 programs, of which 10 are written in FORTRAN and 8 in C. The programs are listed in Table 1. The distinction between languages and compilers should be made, as it is possible that there are inherent differences between object codes produced by the native FORTRAN and C compilers with respect to basic block structure.

2 Search space size

The number of permutations that can be arranged out of n instructions is $n!$, a very large number. It is quite clear that if no systematic way to explore this space efficiently is available, and that is in the nature of NP-hardness, then finding an optimal schedule requires one to generate a very large number of schedules and simulate each one. Now, not all permutations are legal schedules: data dependence constraints rule out some. One would like to know how many schedules are legal out of the possible $n!$, and whether this remaining number is perhaps sufficiently small to allow finding optima by brute force. Since the nature of the data dependence constraints is not *a priori* known, it is best to explore this question empirically. Hence I performed an exhaustive search of the space of legal schedules, without simulating the costs of each, with the express purpose of finding how many legal schedules there are. It turns out that the number of legal schedules still grows very fast, perhaps exponentially, in absolute terms, even though this number in relative terms (as a fraction of the maximum possible, or $n!$) falls rapidly.

Let us observe this behavior in the plot of distribution of this fraction in Figure 1, and in the summary in Table 2. Consider the median size of the search space: for blocks of size 5, it is 6, or

Benchmark	SPEC95 classification	SPEC95 description	Source lines	Number of blocks	Number of instructions	Average size of block (instructions)
FORTRAN programs						
110.applu		Parabolic and elliptic partial differential equations	3817	25475	129853	5.097
141.apsi	FORTRAN scientific benchmark with double precision floating point arithmetic	Solves for the mesoscale and synoptic variations of potential temperature, wind, velocity, and distribution of pollutants	4211	29077	159482	5.485
145.fpppp	FORTRAN scientific benchmark with double precision floating point arithmetic	Quantum chemistry	2122	25693	132139	5.143
104.hydro2d	a vectorizable FORTRAN program with double precision floating-point arithmetic	Astrophysics: hydrodynamical Navier-Stokes equations are solved to compute galactical jets	2522	26789	129568	4.837
107.mgrid		Multi-grid solver in a 3D potential field	368	25555	121750	4.764
103.su2cor	a vectorizable FORTRAN program with double precision floating-point arithmetic	Quantum physics: Monte Carlo calculation of elementary particle masses	1614	26972	135837	5.036
102.swim	a FORTRAN scientific benchmark with single precision floating point arithmetic	Shallow water model with 512×512 grid	259	25109	119333	4.753
101.tomcatv	a highly vectorizable double precision floating point FORTRAN benchmark	A mesh-generation program	107	23856	117515	4.926
125.turb3d		Simulates isotropic, homogeneous turbulence in a cube	1280	26285	127884	4.865
146.wave5	FORTRAN scientific benchmark with double precision floating point arithmetic	Plasma physics: solves Maxwell's equations and particle equations of motion on a Cartesian mesh with a variety of field and particle boundary conditions	6430	28932	152655	5.276
Subtotal FORTRAN			22730	263743	1326016	5.028
C programs						
129.compress	a CPU-intensive integer benchmark with a significant I/O component	Reduces the size of files using adaptive Lempel-Ziv coding	1422	4596	20152	4.385
126.gcc	a CPU-intensive integer benchmark written in C	based on the GNU C compiler version 2.5.3, builds SPARC code	133049	77269	332184	4.299
099.go	a CPU-bound integer benchmark	Artificial intelligence: plays the game of *go*	25362	16095	80900	5.026
132.ijpeg	C	Graphic compression and decompression	17449	12033	70928	5.894
130.li	a CPU-intensive integer benchmark written in C. The benchmark performs minimal I/O	LISP interpreter running the Gabriel benchmarks	4323	8056	36668	4.552
124.m88ksim	essentially an integer program	Motorola 88100 microprocessor simulator: runs test program	12026	10121	46438	4.588
134.perl	C	Manipulates strings (anagrams) and prime numbers in Perl	21078	22590	111849	4.951
147.vortex	a single-user object-oriented database transaction benchmark which which exercises a system kernel coded in integer C	Subset of a full object oriented database program called VORTEx (Virtual Object Runtime EXpository)	41034	32624	180331	5.528
Subtotal C			255743	183384	879450	4.796
Total				447127	2205466	4.933

Table 1: Properties of SPEC95 benchmarks.

$0.05 \times 5!$; for blocks of size 11, it is 3600, or only $0.0002 \times 11!$. The cumulative distribution curves in the log-lin plot are certainly not exactly linear, but they seem to be similar to linear. That is, all degrees of magnitude (of the relative number of legal schedules) are approximately equally represented. This is illustrated in the scatter plot of Figure 2, and is an indication that the instruction scheduling problem remains intractable in spite of the data dependence constraints imposed by typical basic blocks.

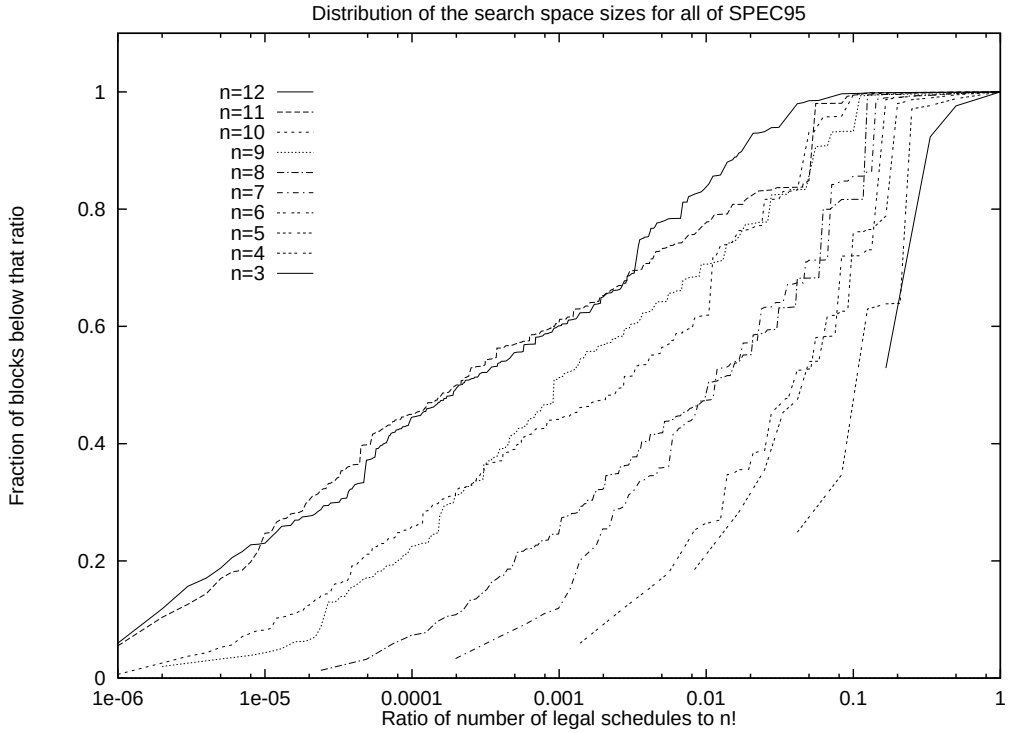

Figure 1: Relative search space size (cumulative distributions).

3 The infeasibility of exhaustive search

I found that the practical limit of scheduling by exhaustive search is at $L = 10$ or 11 instructions, in the sense that it is possible to construct and simulate all legal schedules of all blocks in SPEC95 up to the size L within a reasonable amount of time: for $L = 10$, within several hours on an Alpha 21164.[1]

In SPEC95, blocks of size 10 or smaller account for 92.34% of all blocks, as shown in Figure 3. One might be tempted to conclude that exhaustive instruction scheduling is feasible after all. However, the blocks of size 10 or smaller account for only 30.47% of total execution cost, calculated as the number of cycles dynamically executed (ignoring inter-block effects and memory system effects), as shown in Figure 4. Hence, only about 30% of the execution cost can be reliably covered by exhaustive scheduling; *eo ipso*, exhaustive basic block instruction scheduling is truly not feasible in practice.[2]

[1] As it happens, constructing schedules without simulation raises the L barrier by just one.

[2] A somewhat greater portion can be covered, if one includes well-behaved larger blocks.

Block size n	Number of blocks	Total legal schedules for blocks of size n	Average number of legal schedules	Average number of legal schedules as fraction of $n!$	Median number of legal schedules
1	70576	70576	1.000	1.000	1
2	92724	96868	1.045	0.522	1
3	64098	103799	1.619	0.270	1
4	50993	193069	3.786	0.158	3
5	39909	405731	10.166	0.0847	6
6	33130	1728894	52.185	0.0725	30
7	20711	4415816	213.211	0.0423	60
8	18147	28669077	1579.825	0.0392	420
9	10809	79020734	7310.642	0.0201	336
10	11762	687775949	58474.405	0.0161	9072
11	6465	2824527921	436895.270	0.0109	3600
12	5888	13162975344	2235559.671	0.00467	11088

Table 2: Search space size measures (all of SPEC95).

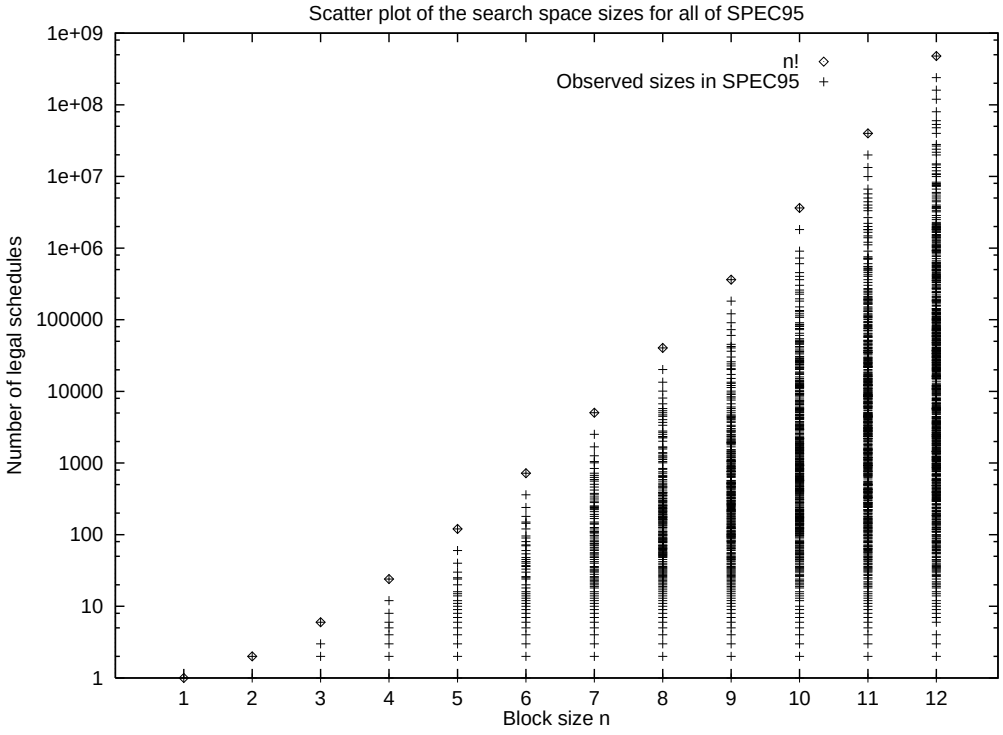

Figure 2: Observed search space sizes (scatter plot).

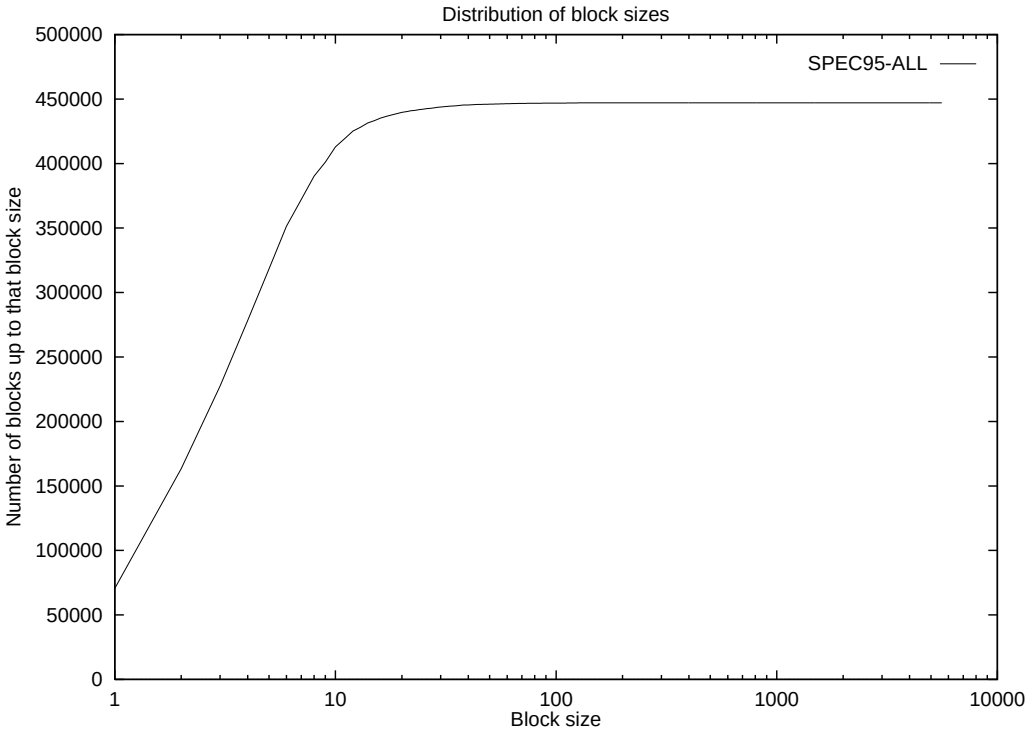

Figure 3: Block size for all SPEC95 basic blocks, not weighted (cumulative distribution).

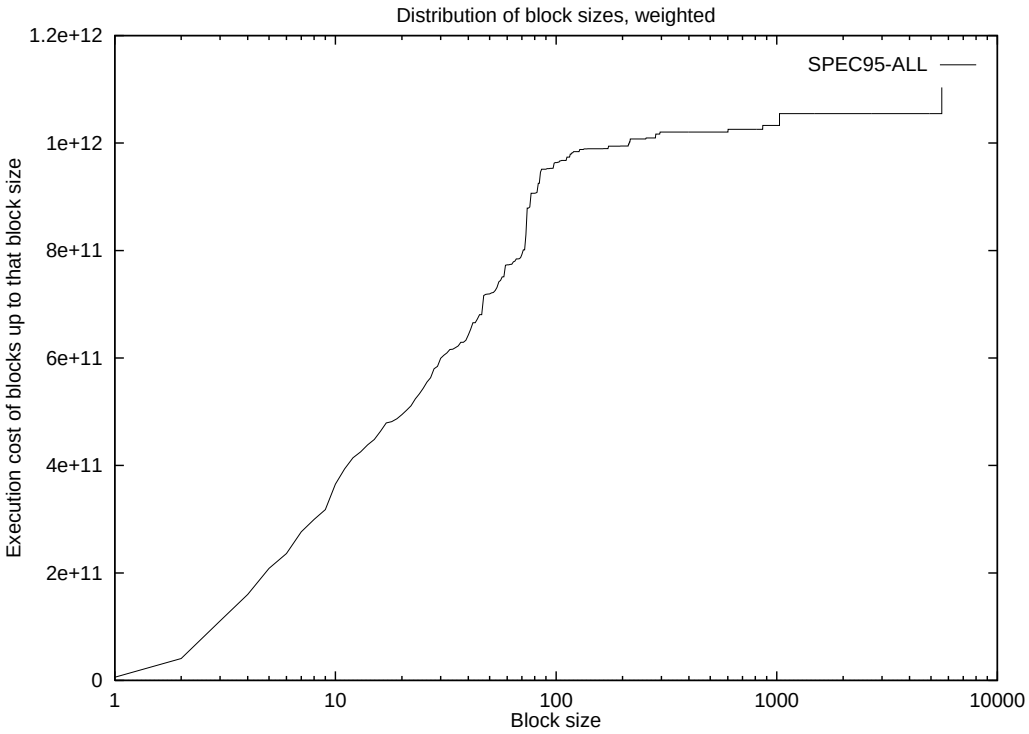

Figure 4: Block size for all SPEC95 basic blocks, weighted by execution cost (cumulative distribution).

4 Assessment of the DEC scheduler

The DEC scheduler is a basic block instruction scheduler for the Alpha architecture and its 21064 implementation [Bhandarkar, 1996]. It was made available by Digital together with a basic block execution simulator: the simulator calculates how many cycles the presented sequence of instructions takes to execute without reordering. This scheduler is not, as far as I know, the same as that found in Digital's compilers.

As I mentioned above, I carried out an exhaustive search of legal schedules for all basic blocks of up to 11 instructions. I simulated each of these schedules using the DEC simulator, and thus I found the optimal schedules and optimal costs for all such blocks. I also scheduled all basic blocks using the DEC scheduler.

The DEC scheduler comes very close to optimum in almost all examined blocks. Indeed, it produces optimal schedules for 99.13% of the examined blocks. It never produces a schedule longer than 150% of the optimal schedule, and only in one case longer by more than 3 cycles than the optimal schedule. In Figures 5 and 6 I show how the suboptimal schedules are distributed. Note that the quality of schedules diminishes as the blocks grow larger, but it remains quite good; 94.04% of blocks with exactly 11 instructions still receive optimal schedules.

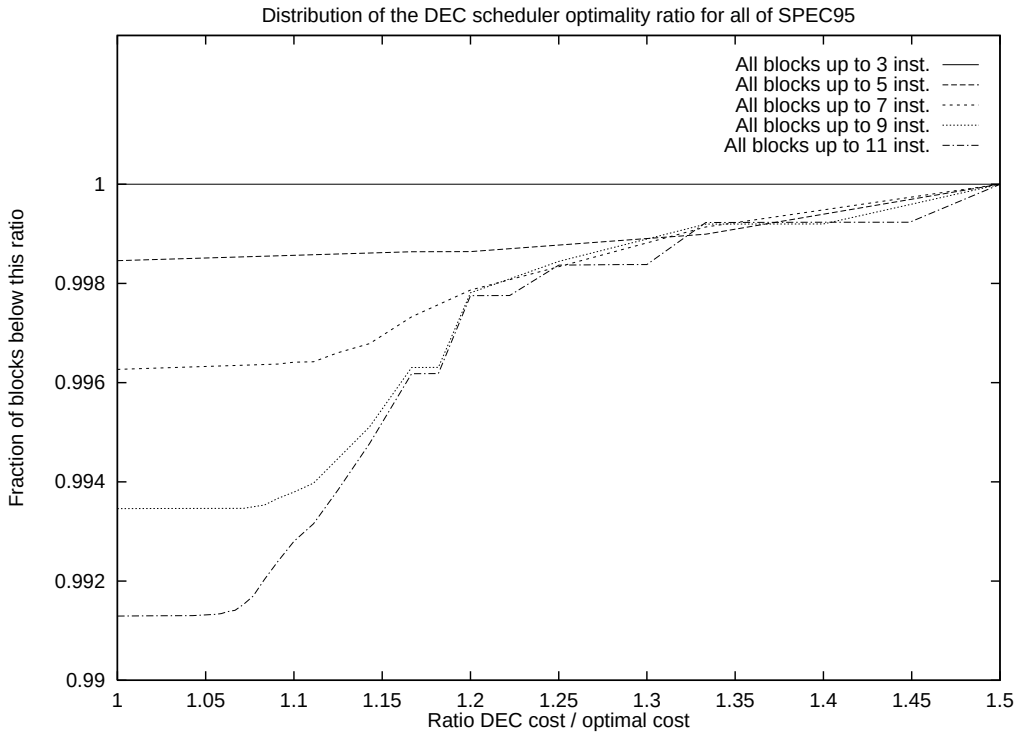

Figure 5: Suboptimality of the DEC scheduler (cumulative distribution of ratio).

In summary, the DEC scheduler is quite close to optimal in the range that permitted comparison. One can only speculate about its optimality for larger basic blocks. Although the observed quality of produced schedules decreases with increasing block size, it probably remains sufficiently high to allow one to use the DEC scheduler as the benchmark against which other heuristic scheduler are compared.

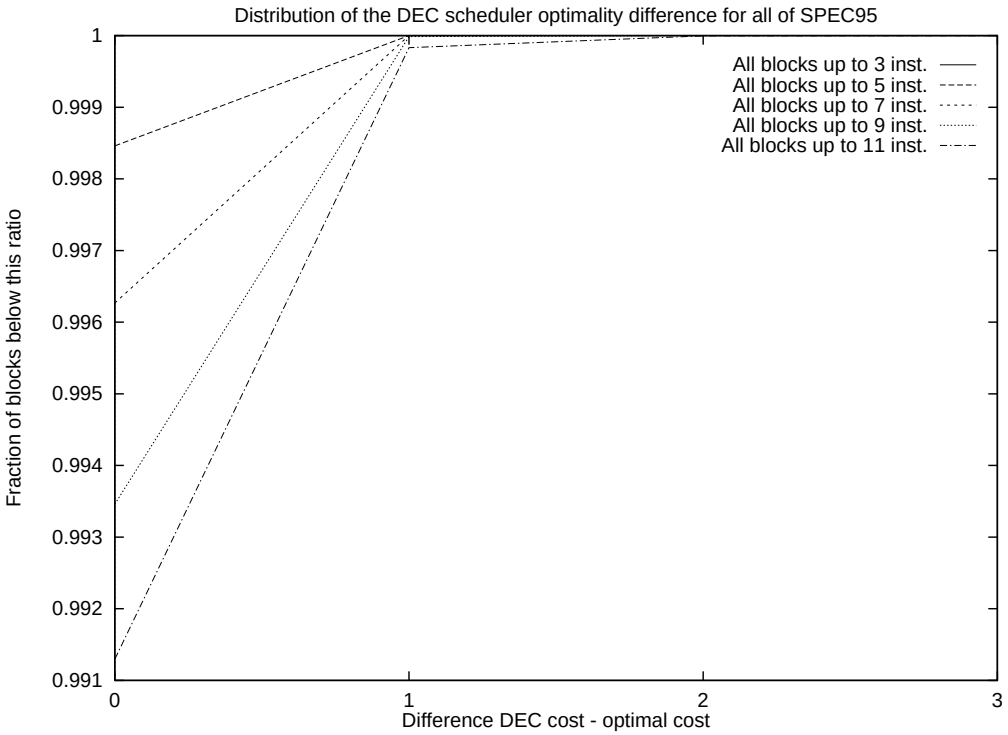

Figure 6: Suboptimality of the DEC scheduler (cumulative distribution of difference).

5 The variation of schedule costs

When the instruction scheduling of a basic block is viewed as a search task of minimizing the execution cost of the block, and a heuristic needs to be devised to guide this search, then it is well worth knowing what variation there is among possible costs. In particular, if there is little variation in the cost of legal schedules, then the (automated) construction of such heuristics on the basis of the cost as the optimality criterion becomes difficult. The same can be said if a large number of different schedules are tied for the optimum. In Figure 7, I show the measure of suboptimality of the schedule cost averaged over all legal schedules of a block. There is considerable variation in cost. Looking at all examined blocks (the lowest curve), one can see that for 68% of them, the average schedule cost is the same as the optimal cost. In other words, all schedules are equally good. If a heuristic is to be inferred (learned) from the schedule cost alone, then none of these blocks contributes to the learning. However, the remaining 32% of the blocks produce a fairly wide range of outcomes—almost an octave wide. This variation at the outcome side should be sufficient for learning techniques: how the input side should be devised is beyond the scope of this report.

References

[Bhandarkar, 1996] Bhandarkar, D. P. (1996). *Alpha implementations and architecture : complete reference and guide*. Digital Press, Boston, Mass.

[Hennessy and Gross, 1983] Hennessy, J. and Gross, T. (July 1983). Postpass code optimization of pipeline constraints. *ACM Transactions on Programming Languages and Systems*, 5(3):422–448.

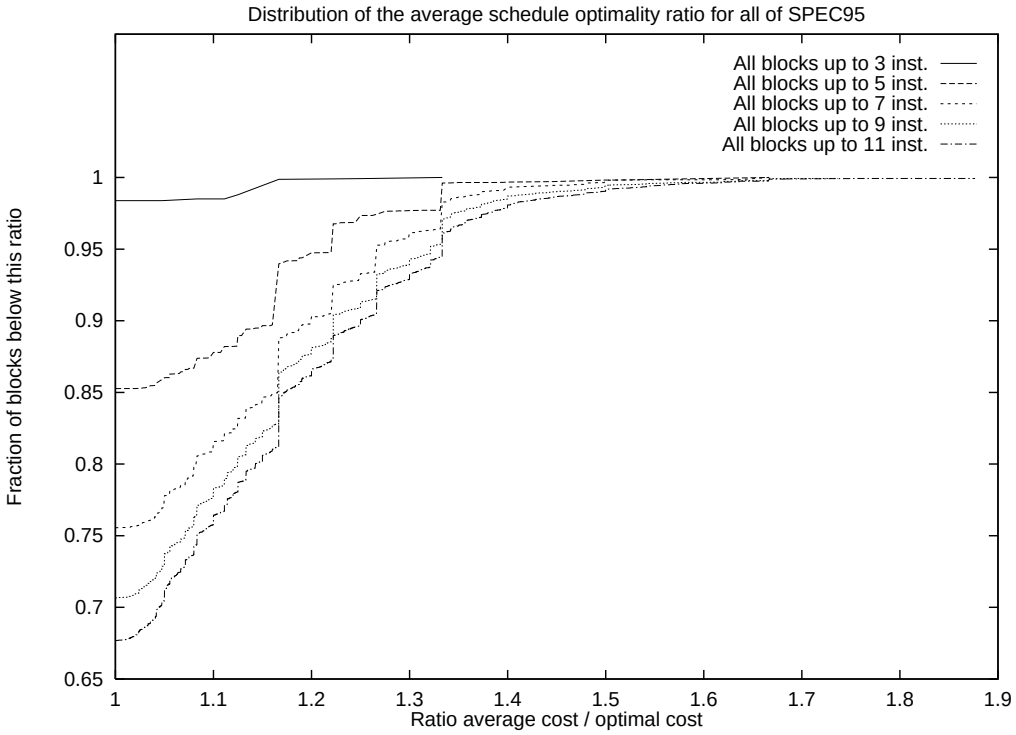

Figure 7: Suboptimality of average schedules (cumulative distribution of ratio).