Quantitative Evaluation of SPEC Benchmarks

Jozo Dujmović and Wilson Kwok
Department of Computer Science
San Francisco State University

Contents

1. Black-box metrics for benchmark comparison
2. Utilization of program space
3. Density of benchmark suite
4. Evaluation of CFP95, CINT95, CFP2000, CINT2000, CFP2006, CINT2006, JVM98, MPIM2007, MPIL2007, OMPM2001, OMPL2001
5. Evolutionary adjustment of benchmark suites
6. Reduction of the cost of SPEC benchmarking
7. Conclusions

SPEC component-level benchmarks

- N = number of component benchmarks in a benchmark suite
- M = number of computers measured using the benchmark suite
- N as a function of time for SPEC CPU benchmarks:

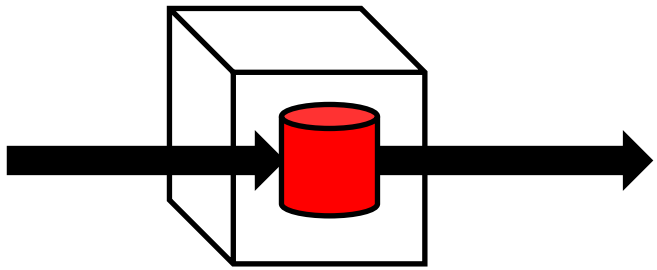
Year	1989	1992	1995	2000	2006
N(t)	**10**	**20**	**18**	**26**	**29**

Survey of analyzed SPEC benchmark suites

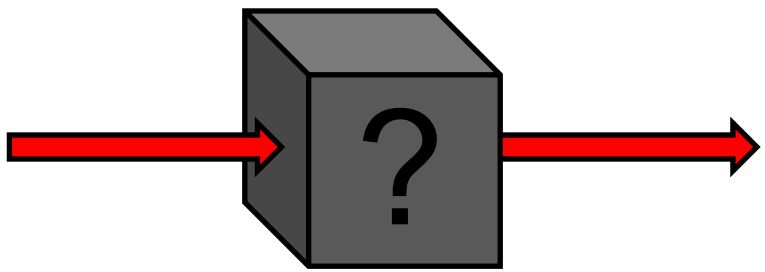
Benchmark Suite	N(# of Prog.)	Type of benchmark workload	M(# of Sys.)
CFP95	10	Compute-intensive floating point	603
CINT95	8	Compute-intensive integer oper.	547
CFP2000	14	Compute-intensive floating point	1384
CINT2000	12	Compute-intensive integer oper.	1373
CFP2006	17	Compute-intensive floating point	2551
CINT2006	12	Compute-intensive integer oper.	2626
JVM98	7	Java virtual machine client WL	84
MPIM2007	13	MPI-parallel, floating point, compute-intensive operations	205
MPIL2007	12	MPI-parallel, floating point, compute-intensive operations	51
OMPM2001	11	Compute-intensive parallel oper.	269
OMPL2001	9	Compute-intensive parallel oper.	103

Workload Difference Models

- **White-box models** (based on accessible internal resources)



- **Black-box models** (based on external [run time] measurements)



Run Times Table (SPEC measurements)

Measured System	Component Benchmarks						
	B_1	…	B_j	…	B_k	…	B_N
S_1	t_{11}	…	t_{1j}	…	t_{1k}	…	t_{1N}
…	…		…		…		…
S_i	t_{i1}	…	t_{ij}	…	t_{ik}	…	t_{iN}
…	…		…		…		…
S_M	t_{M1}	…	t_{Mj}	…	t_{Mk}	…	t_{MN}

SPECmark performance indicator

- R = reference computer
- C, D = analyzed computers
- Measured run times:

$$t_1(R),...,t_N(R) \qquad t_1(C),...,t_N(C) \qquad t_1(D),...,t_N(D)$$

- Performance comparison:

$$S(C,R) = \prod_{k=1}^{N} [t_k(R) / t_k(C)]^{1/N}$$

$$S(C,D) = \frac{S(C,R)}{S(D,R)} = \prod_{k=1}^{N} [t_k(D) / t_k(C)]^{1/N}$$

Black-box metrics for component benchmark comparison

- Metrics based on run times
- Basic concepts and indicators:
  - Distance (difference) between benchmarks
  - Program space (a space where each point represents a program)
  - Size of benchmark suite
  - Utilization (coverage) of program space
  - Density of benchmark suite

Program Space

Program space is defined as a space where each point represents a program, and the distance (difference) between programs is computed using a multidimensional non-Euclidean (semi)metric.

Program = computer workload of any complexity

d(A,B) = difference between programs A and B

$d(A,B) \geq 0$ (non-negativity)

$d(A,B) = 0, \quad A=B$ (non-degeneracy)

$d(A,B) > 0, \quad A \neq B$

$d(A,B) = d(B,A)$ (symmetry) [semimetric]

$d(A,B) \leq d(A,C)+d(C,B)$ (triangular inequality) [metric]

The Concept of Limited Difference

$0 \leq d(A, B) \leq 1$	Range of difference between programs A and B
$d_{min} = 0$	Identical programs
$d_{max} = 1$ (or 100%)	Completely different prog.
$s(A,B) = 1 – d(A,B)$	Similarity (complement of d)
$0 \leq s(A, B) \leq 1$	Range of similarity

Using the CLD, all differences can be conveniently interpreted as fractions of d_{max}

The uniform random number difference (distance) between benchmarks

If we have N benchmarks and M measurements of run time t_{ij}, then our first step is to normalize these measured values, so that all measurements are in the interval [0,1]:

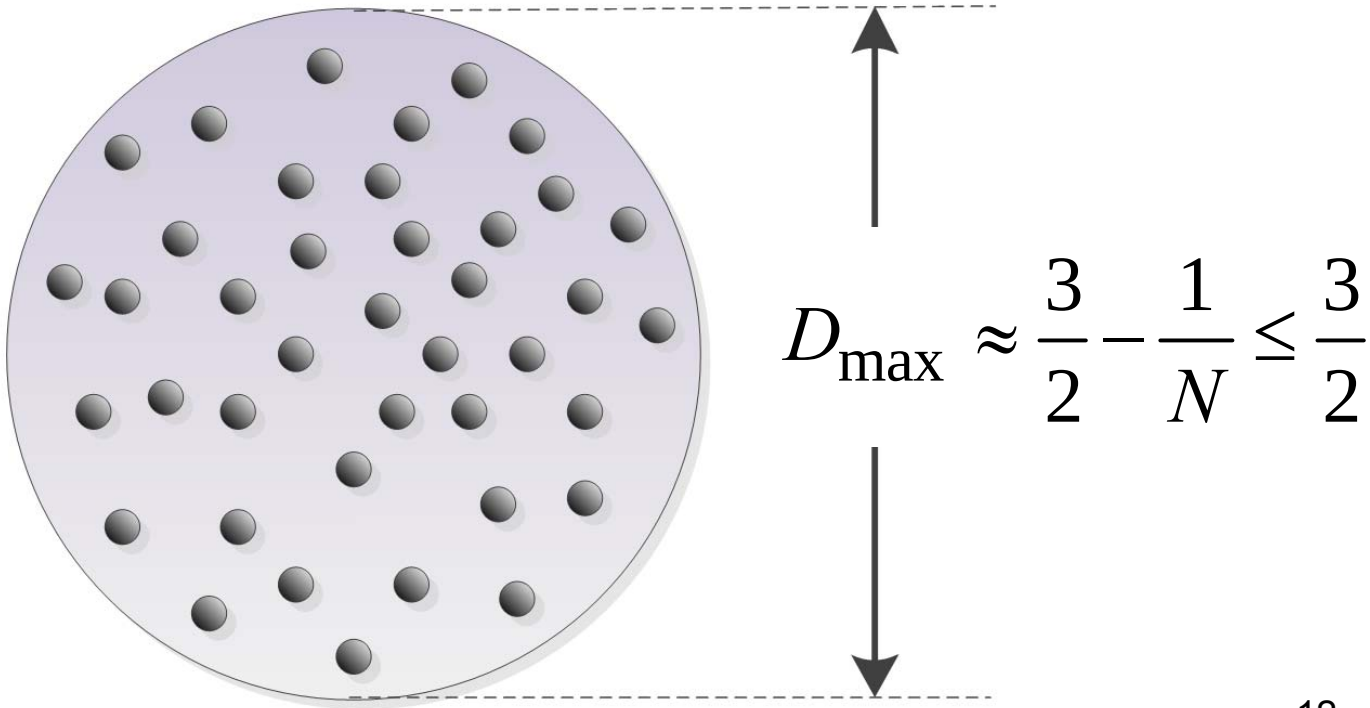
$$T_{ij} = \frac{t_{ij}}{\max_{1 \le i \le M} t_{ij}}, \quad 0 < T_{ij} \le 1, \quad i = 1, ..., M, \quad j = 1, ..., N.$$

A normalized difference between component benchmarks B_j and B_k $(1 \le j \le N,\ 1 \le k \le N)$ can be computed as follows:

$$d(B_j, B_k) = \min\left(1, \frac{3}{M} \sum_{i=1}^{M} | T_{ij} - T_{ik} |\right), \quad 0 \le d(B_j, B_k) \le 1$$

The size of program space

- The distance between two benchmarks (j and k) is limited to 1 ($0 \leq d_{jk} \leq 1$). Consequently, all *N* benchmarks fit in a finite closed program space:



$$D_{\max} \approx \frac{3}{2} - \frac{1}{N} \leq \frac{3}{2}$$

The Size of Benchmark Suite

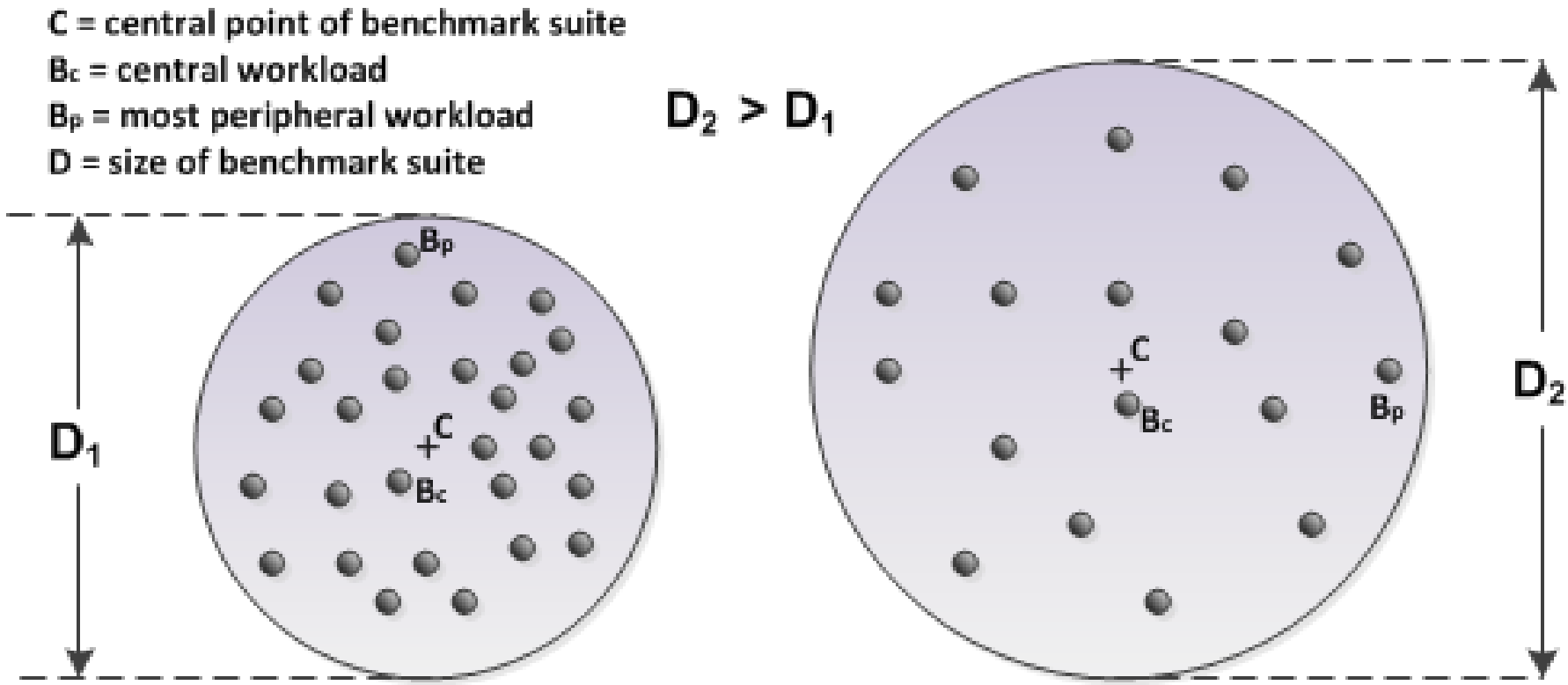
- Size of benchmark suite = diameter of the smallest circumscribed hypersphere that contains N benchmarks.
- Radius = distance between the most centrally located benchmark B_c (best representative of a benchmark suite), and the most peripheral benchmark, B_p.
- Approximate size of benchmark suite:

  $D \cong 2d(B_p, B_c)$

- Maximum possible size of benchmark suite:

  $D_{max} \leq 3/2 - 1/N$

Comparison of Benchmark Suite Size



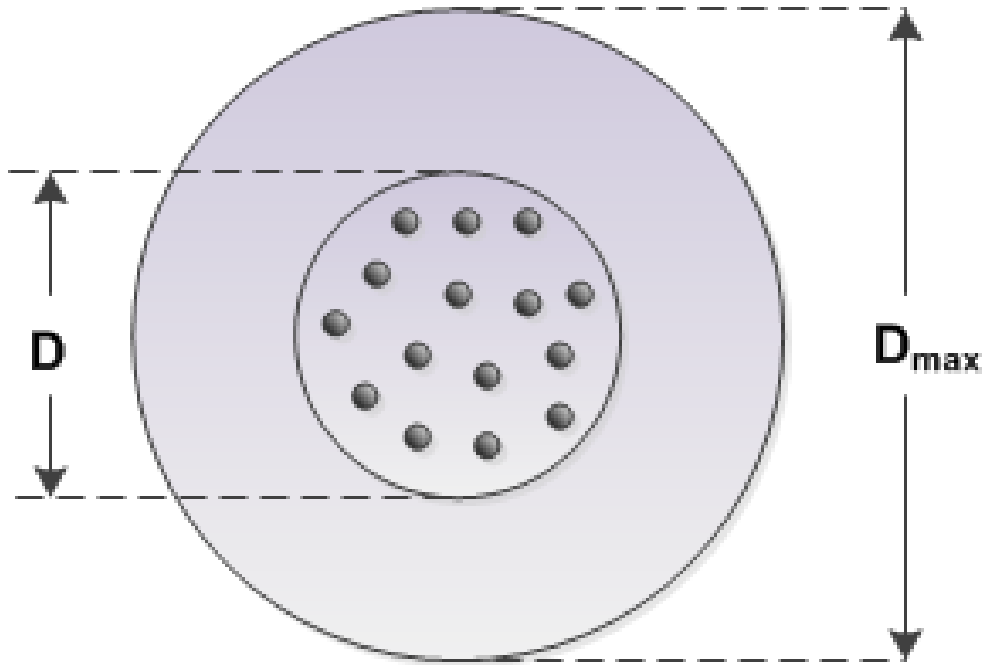
Can the size of benchmark suite be small?

- Yes! The primary reasons are:
  - Semantic reasons (focus on a narrow group of similar workloads, e.g. strictly numerical floating point intensive applications [lots of FADD, FSUB, FMUL, FDIV])
  - Constrained use of computing resources (e.g. only processor, bus and memory)
- The small size is neither the reason nor a valid excuse for excessive redundancy and exaggerated density of a benchmark suite.
- Strict control of density is necessary to select an appropriate number of component benchmarks and control the cost of benchmarking

Utilization (Coverage) of Program Space

- Utilization (U) is a ratio of the actual and the max size of a benchmark suite ($U = 100\ D/D_{max}$ [%])
- Max utilization is achieved when a benchmark suite includes a wide variety of workloads having various characteristics within the program space.
- In general, the goal of designing benchmark suites is to maximize utilization of program space by providing a spectrum of different programs.

Utilization (Coverage) of Program Space (Cont.)



$$U(N) = \frac{D}{D_{\max}} \approx \frac{2d(B_c, B_p)}{1.5 - 1/N} \; ; \quad 0 \le U(N) \le 1$$

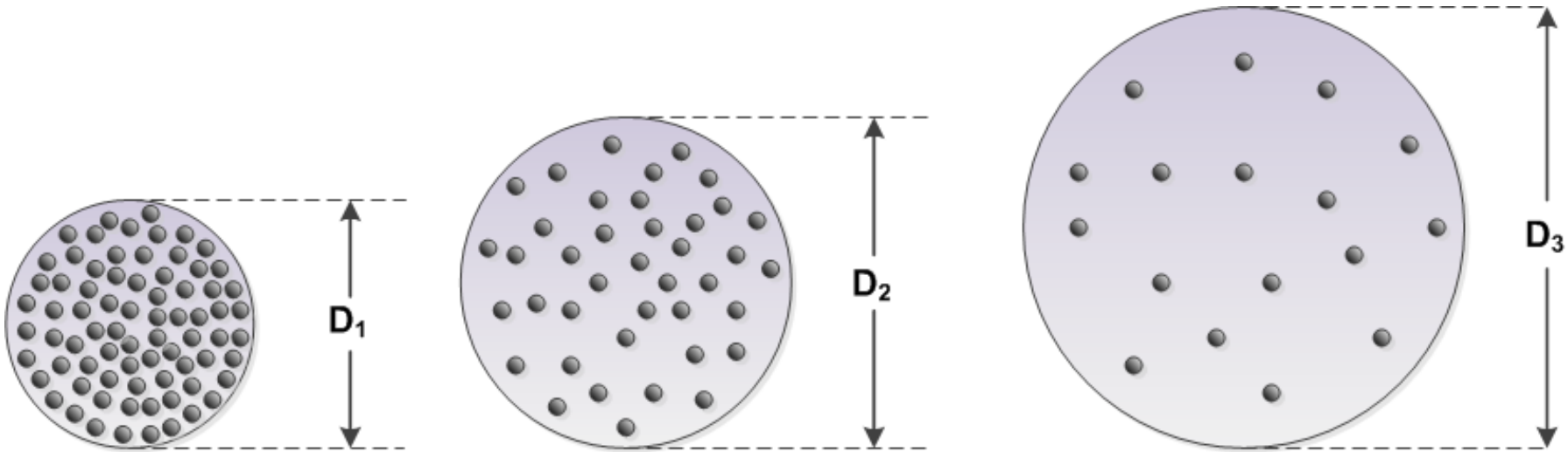
Density of Benchmark Suite

- Density can be defined as the number of workloads per unit of volume of occupied program space:

$$H(N) = N / U(N) , \quad 0 < U(N) \leq 1 , \quad N > 1$$

- High density means a higher precision of benchmark suite (less likely that some properties of the analyzed system are incompletely tested), but also a higher cost.
- High density does not increase the quality of benchmark suite unless the utilization of program space is also high.

Size and Density of Benchmark Suite



Suggested standard density $H \approx 20$

- The density of component-level benchmark suites is a primary parameter for selecting the appropriate size of benchmark suites.
- The density level $\boldsymbol{H \approx 20}$ can be used to provide sufficient accuracy and a reasonable cost of standard industrial benchmarking.
- In such cases each uniformly distributed component benchmark covers approximately 5% of the available program space.

Cost of standard benchmarking

- N = number of component benchmarks in a benchmark suite
- M = number of computers measured using the benchmark suite
- $Cost = a + bNM$ = cost of benchmarking
  - a = cost of benchmark project initialization and administration
  - b = cost of preparing, executing, and documenting a benchmark measurement

Benchmark Suite Design Criteria

- The size of benchmark suite (it should be larger than a selected minimum value)
- The utilization/coverage of program space (it should be larger than a selected minimum value)
- The density of benchmark suite (it should be close to a standard constant value)
- The redundancy of component benchmarks (it should be low)
- The distribution of component benchmarks (it should be approximately uniform)
  - Elimination of excessive redundancy
  - Elimination or supplementing of outliers
- The cost of benchmarking (it should be low)

The role of component benchmark in a benchmark suite

- **Regular contributor** (component benchmarks that are moderately different from other component benchmarks)
- **Overrepresented** (exceedingly similar workloads)
- **Underrepresented** (located far apart from all other benchmarks in a suite)

Evaluation of SPEC Benchmark Suites

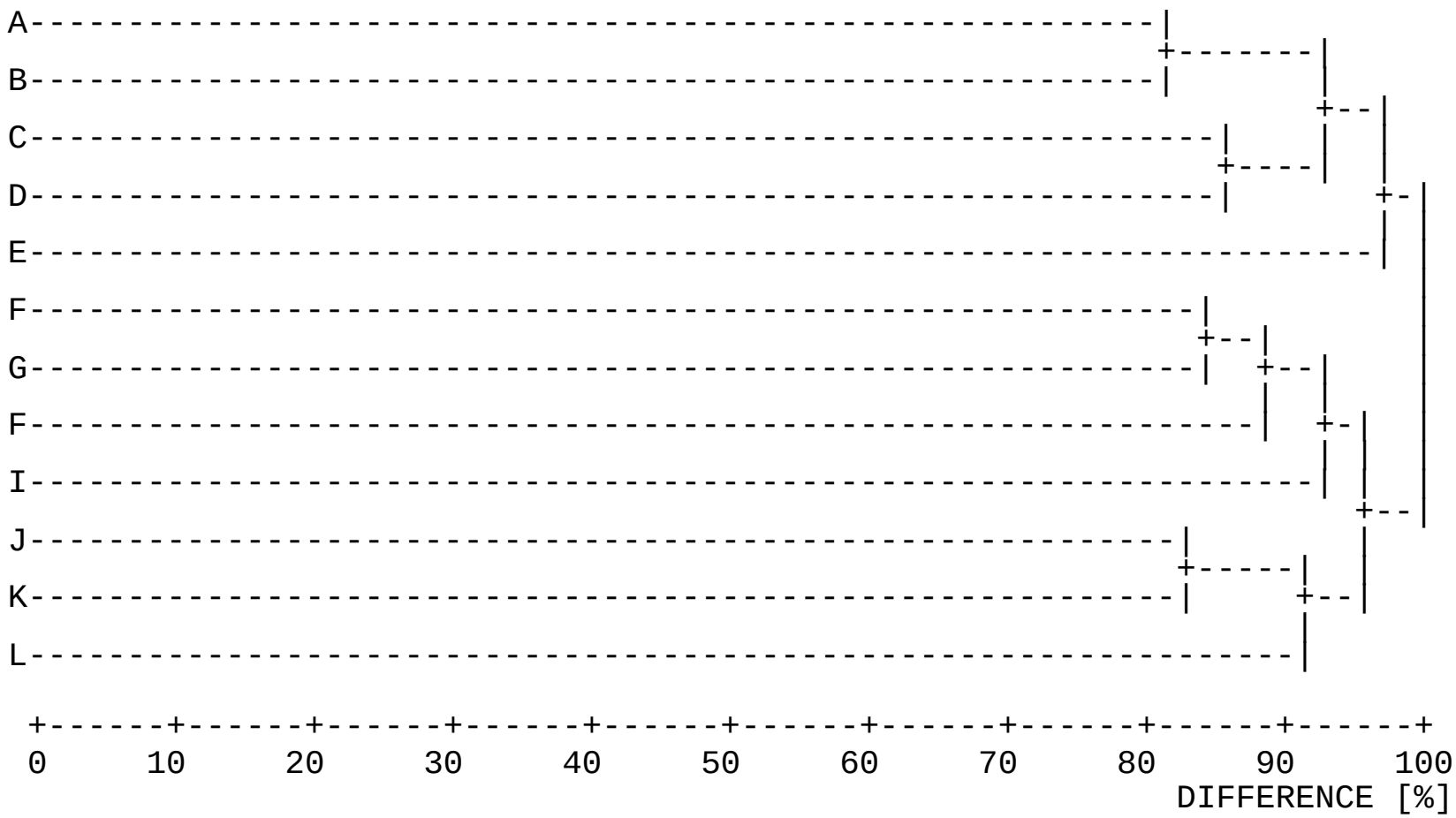
- SPEC benchmark suites:
  - **CPU**: CPU95 (CFP95 & CINT95), CPU2000 (CFP2000 & CINT2000), CPU2006 (CFP2006 & CINT2000)
  - **Java**: JVM98,
  - **High performance**: OMP2001 (OMPL & OMPM), and MPI2007 (MPIL & MPIM)
- Workload difference model: Black-box model (based on run time data from SPEC)
- Difference between benchmarks used as an input for hierarchical clustering
- Use of cluster analysis to evaluate benchmark suite distribution, coverage of program space, and density

Cluster Analysis

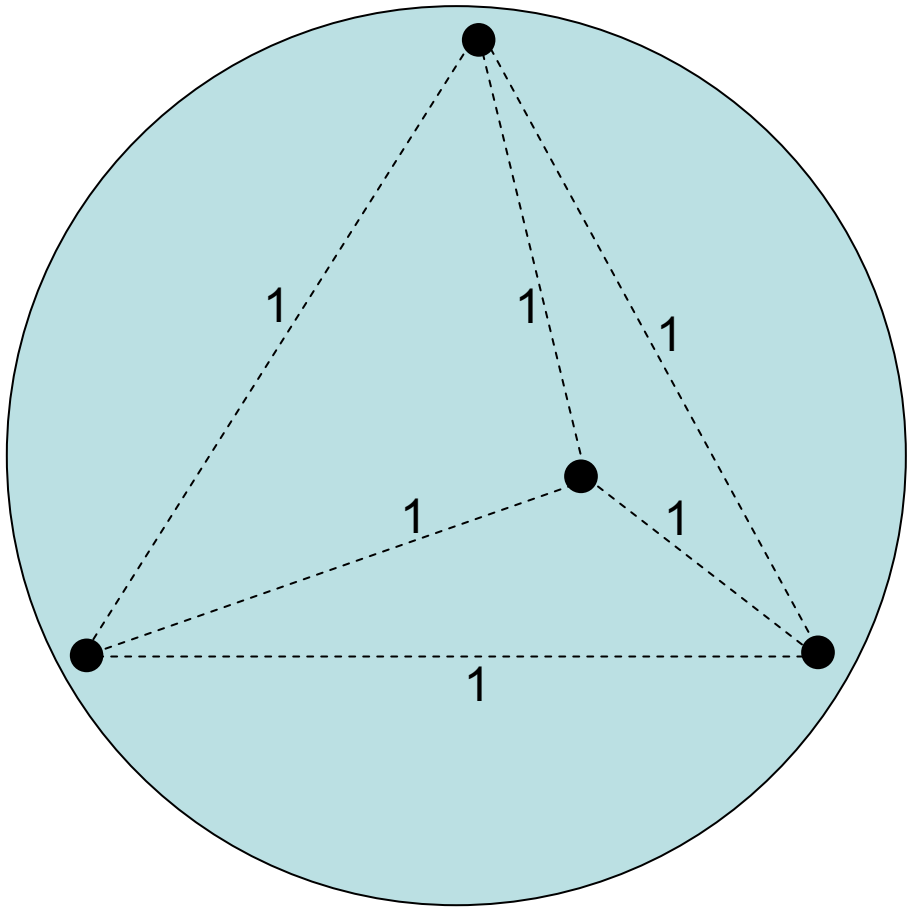
- Goals:
  - Determine the differences and similarities between objects (i.e. component benchmarks)
  - Group a collection of objects into clusters (subsets) such that each cluster contains objects that are similar to one another but different from the objects that are in other clusters

- Visualization or relationships between objects:
  - Dendrogram (binary tree where the distance of each node that merges two clusters from the left edge denotes the average distance between merging clusters)
  - Covergram (binary tree where the distance of each node that merges two clusters from the left edge denotes the average utilization of program space by the merged clusters)

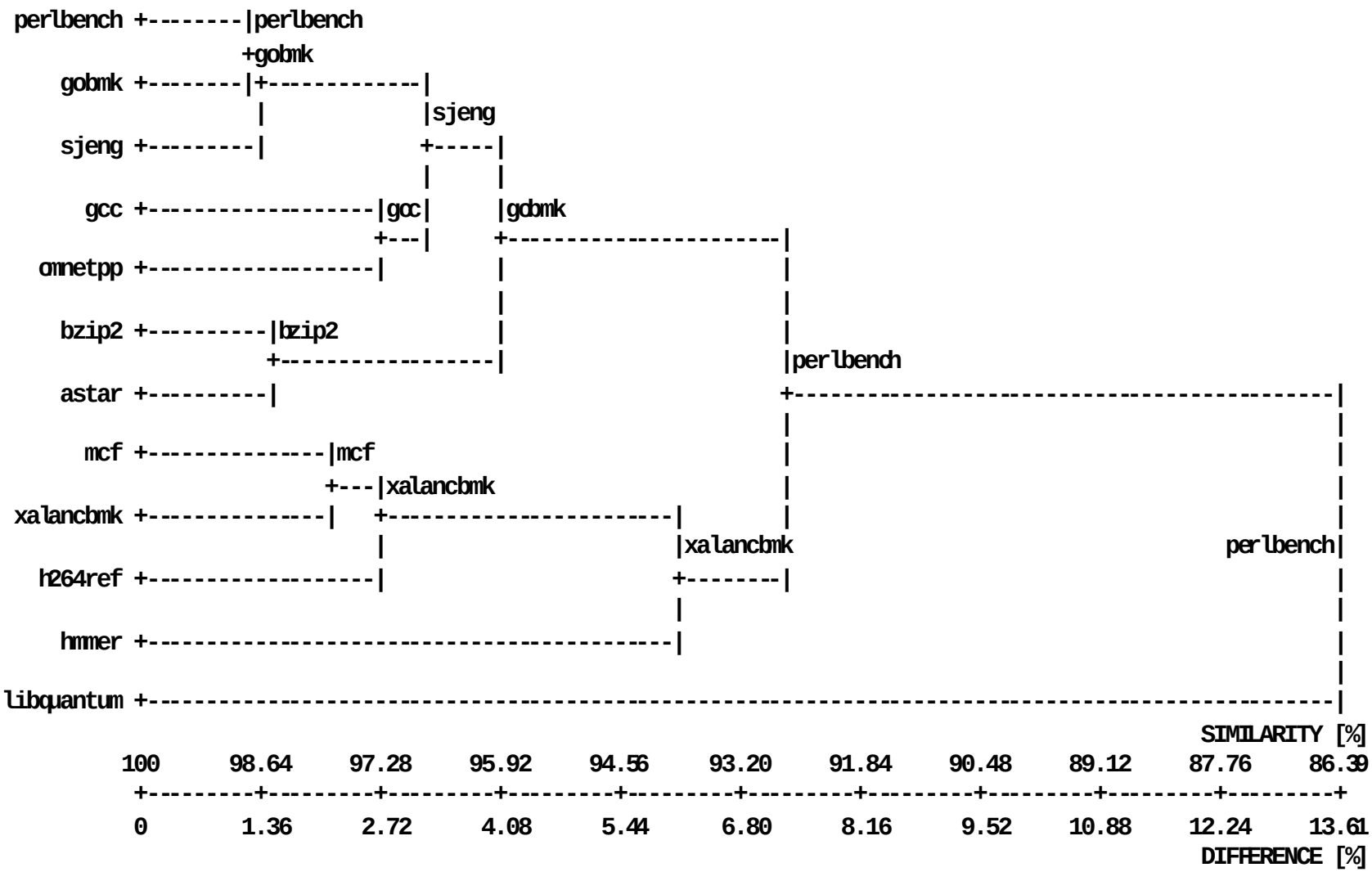
A characteristic pattern of dendrogram for uniformly distributed random objects



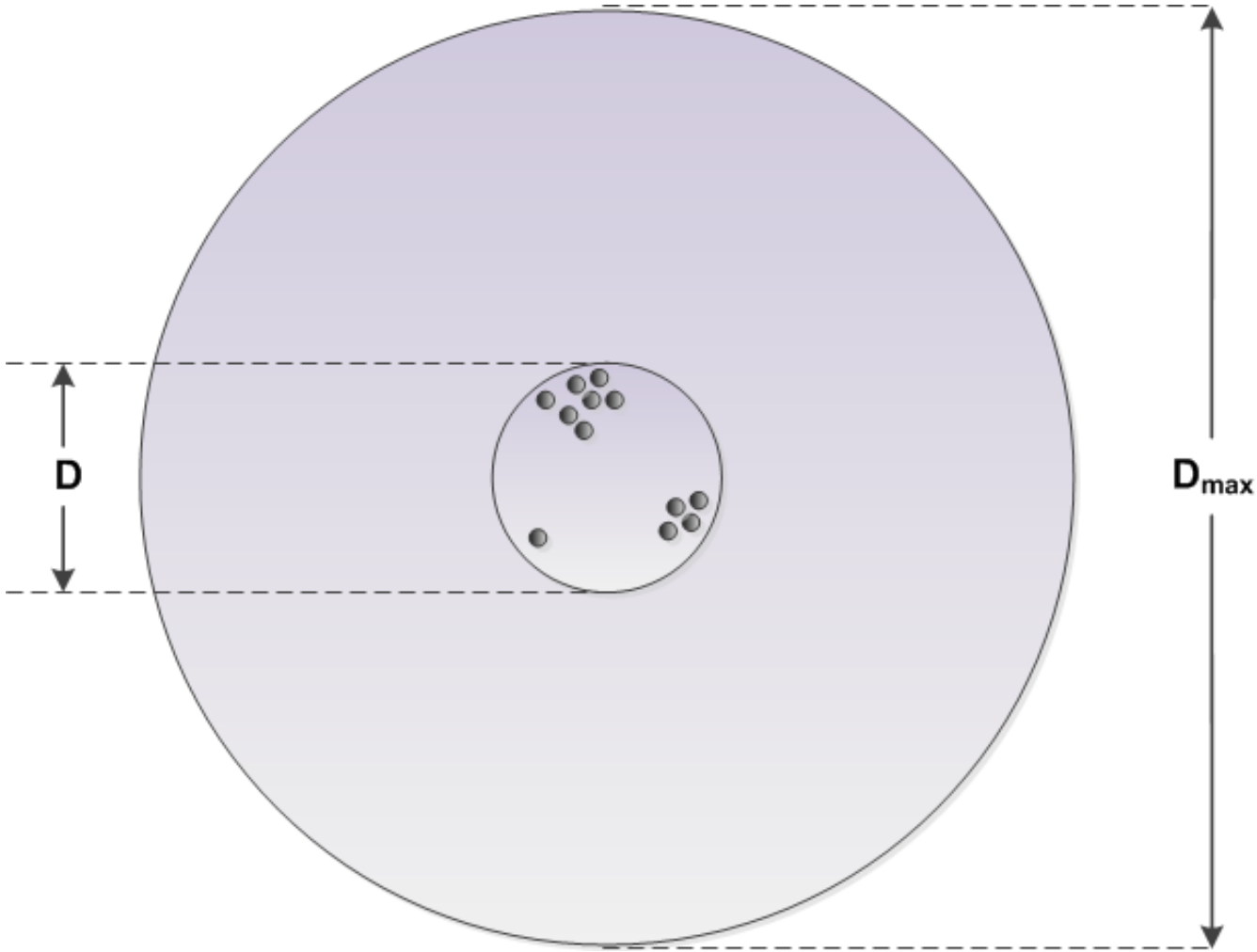
Visual Interpretation of uniform distribution



Nonuniformly distributed objects: a dendrogram of CINT2006

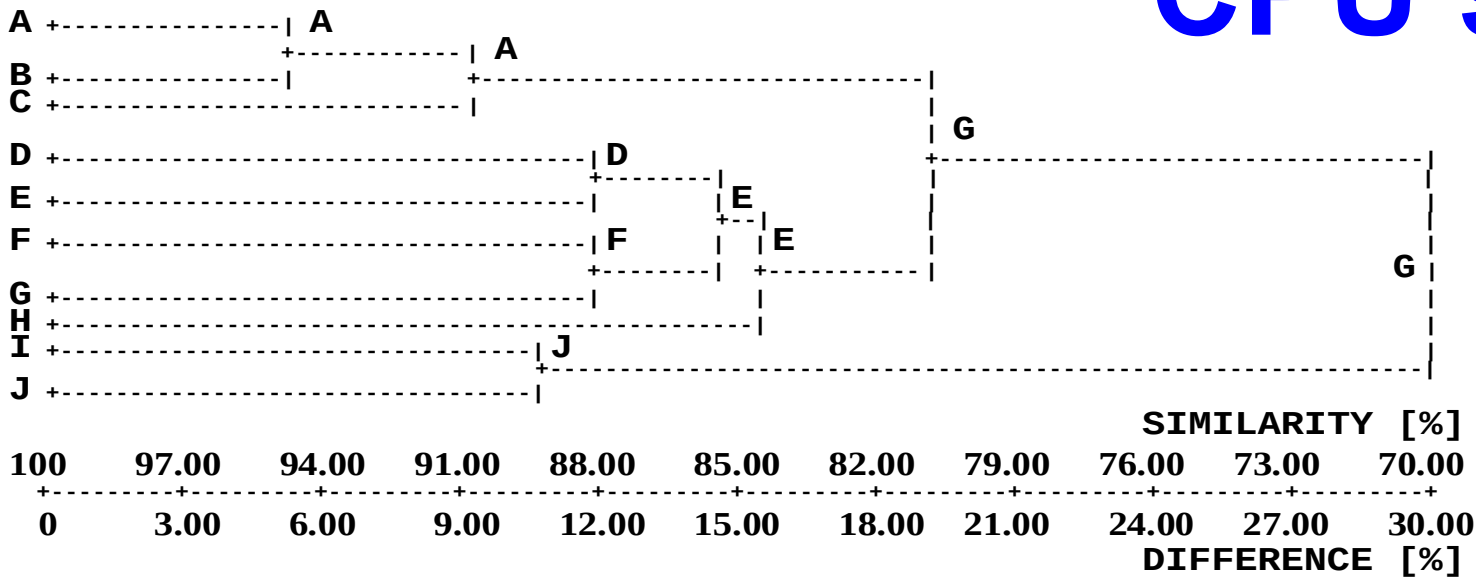


Visual Interpretation of CINT2006

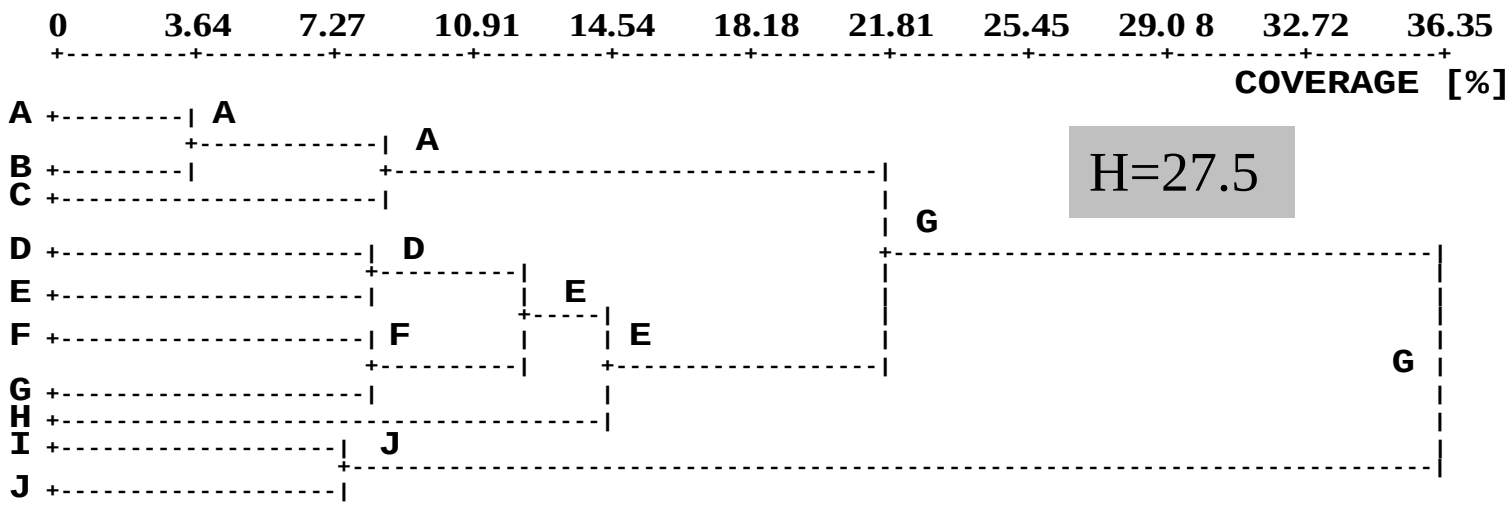


CFP95

CPU 95



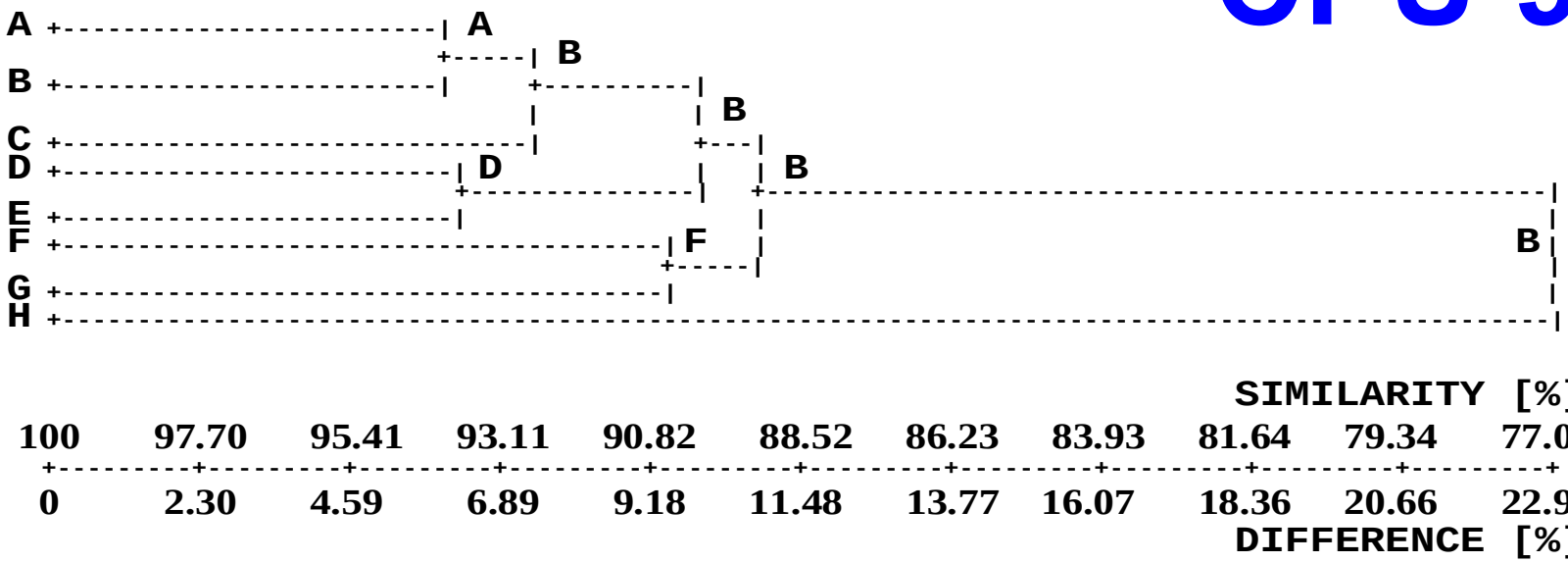
A=tomcatv, B=swim, C=fpppp, D=mgrid, E=applu, F=apsi, G=wave5, H=turb3d, I=su2cor, J=hydro2d



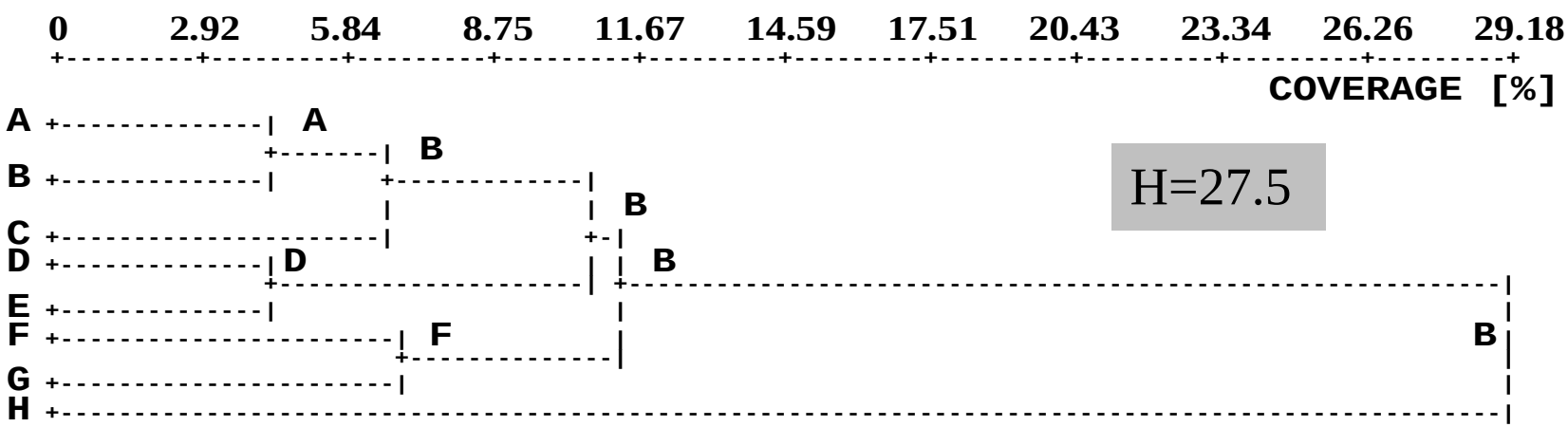
H=27.5

CINT95

CPU 95



A=go, B=gcc, C=m88ksim, D=li, E=vortex, F=compress, G=ijpeg, H=perl



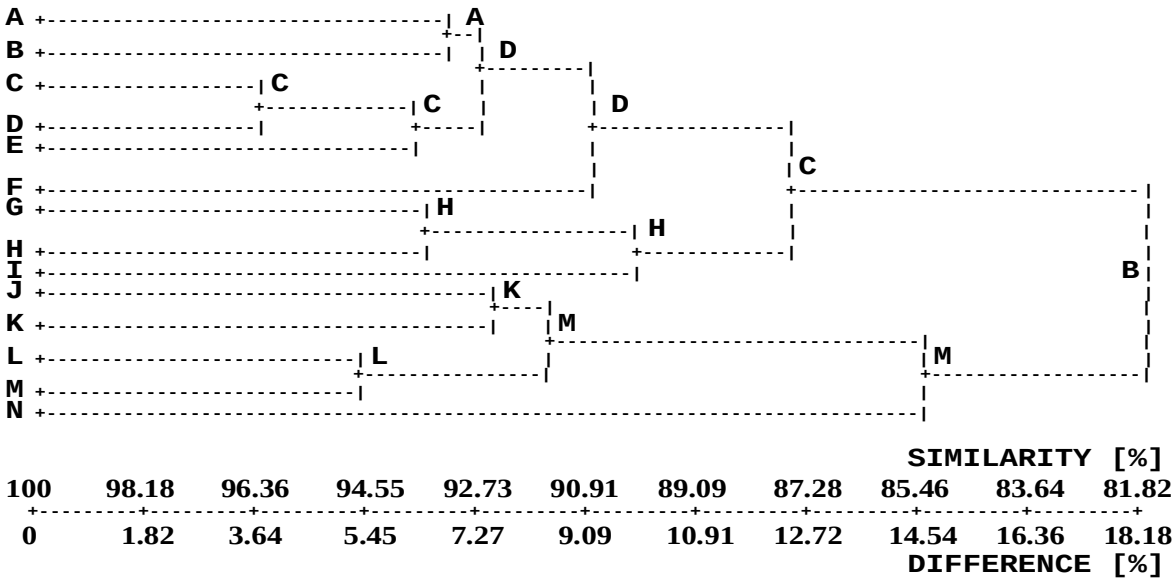
H=27.5

Evaluation of CPU95

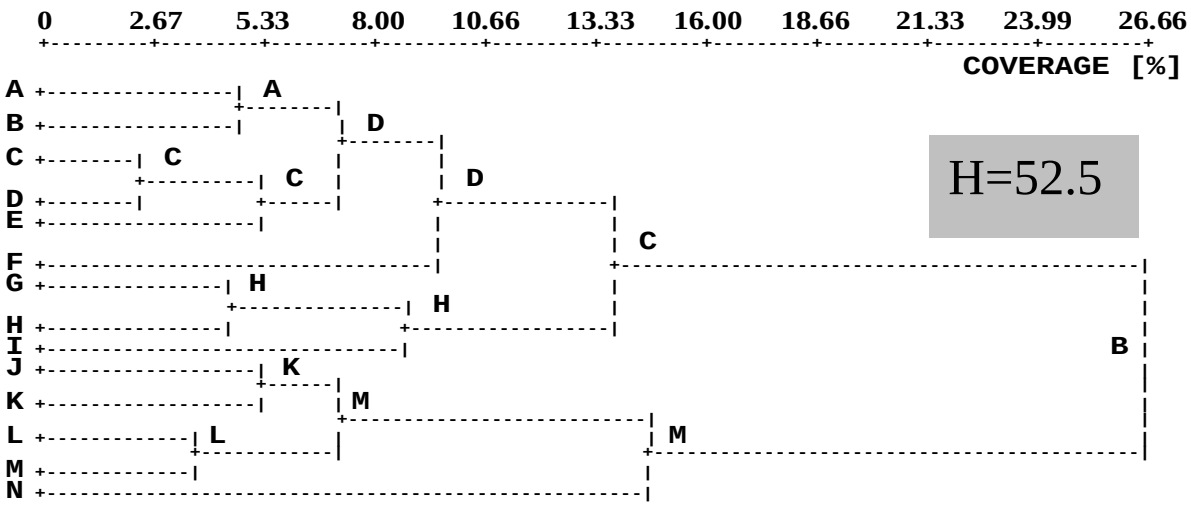
- CFP95
  - Appropriate density
  - {A,B,C} and {I,J} seem to be underrepresented
  - {D,E,F,G,H} seems to be overrepresented
  - H is a local outlier
- CINT95
  - The central group {A,B,C,D,E,F,G} has an exaggerated density H=60
  - {A,B} and {D,E} differ at about 6%
  - H is a remote outlier

CFP2000

CPU 00



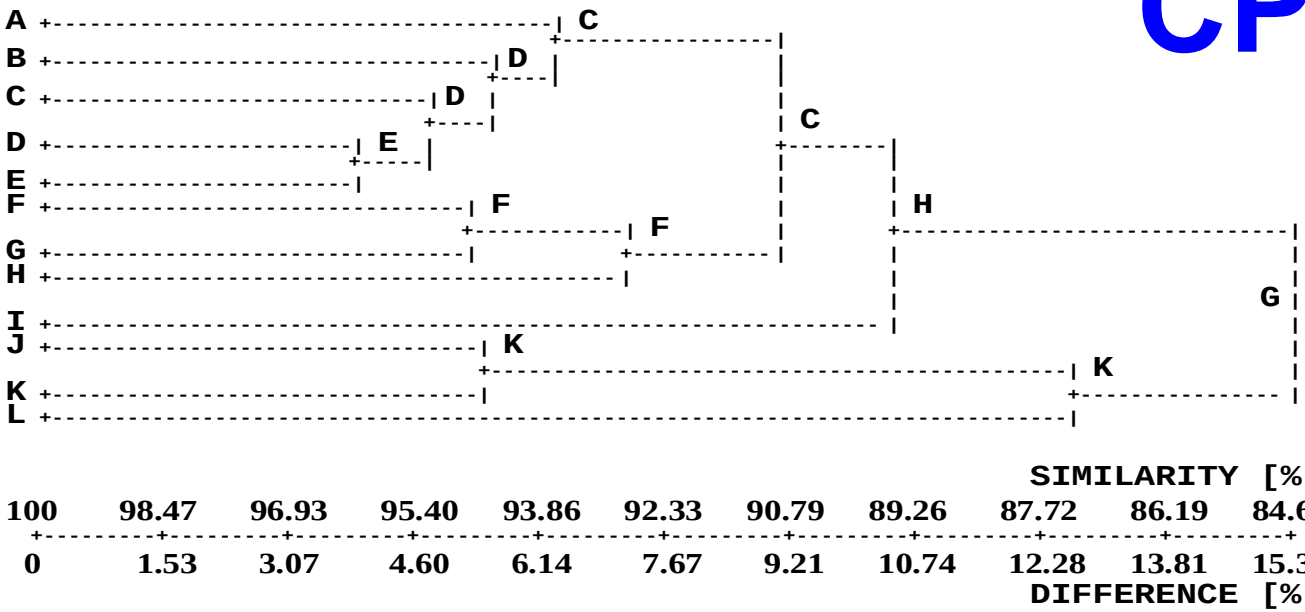
A=wupwise, B=lucas, C=mgrid, D=applu, E=facerec, F=equake, G=swim, H=galgel, I=art, J=mesa, K=fma3d, L=ammp, M=apsi, N=sixtrack



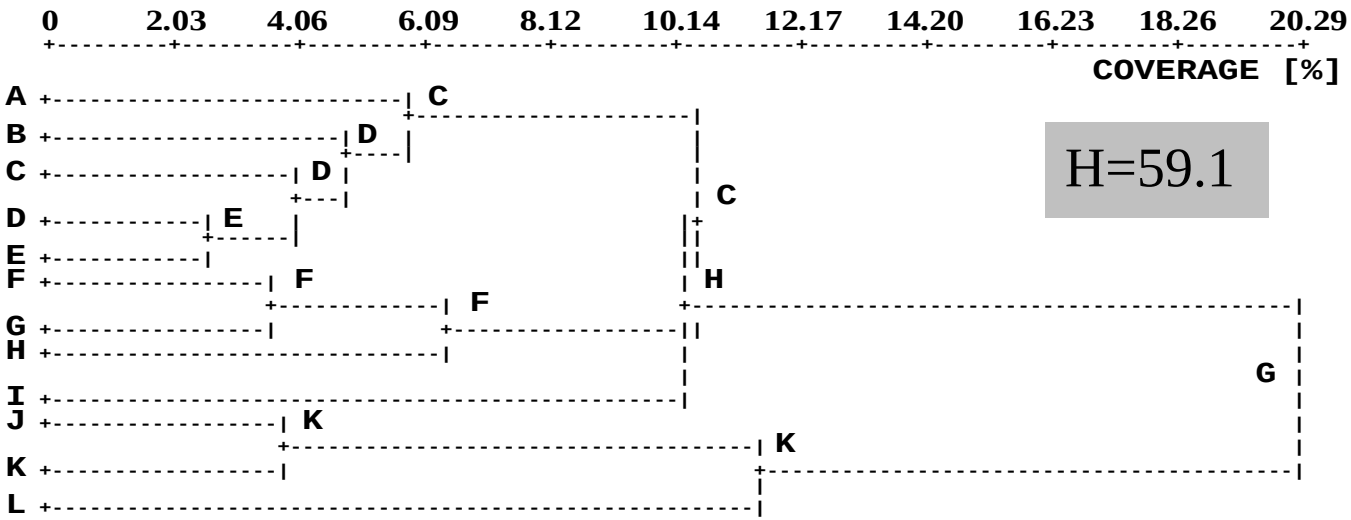
H=52.5

CINT2000

CPU 00



A=gzip, B=gcc, C=parser, D=perlbmk, E=vortex, F=crafty, G=bzip2, H=eon, I=gap, J=vpr, K=twolf, L=mcf



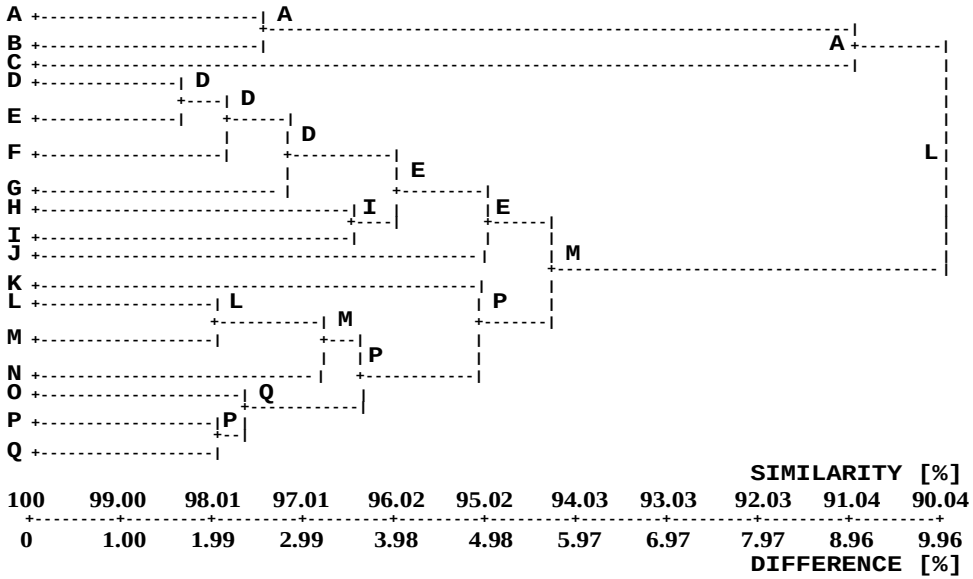
H=59.1

Evaluation of CPU 2000

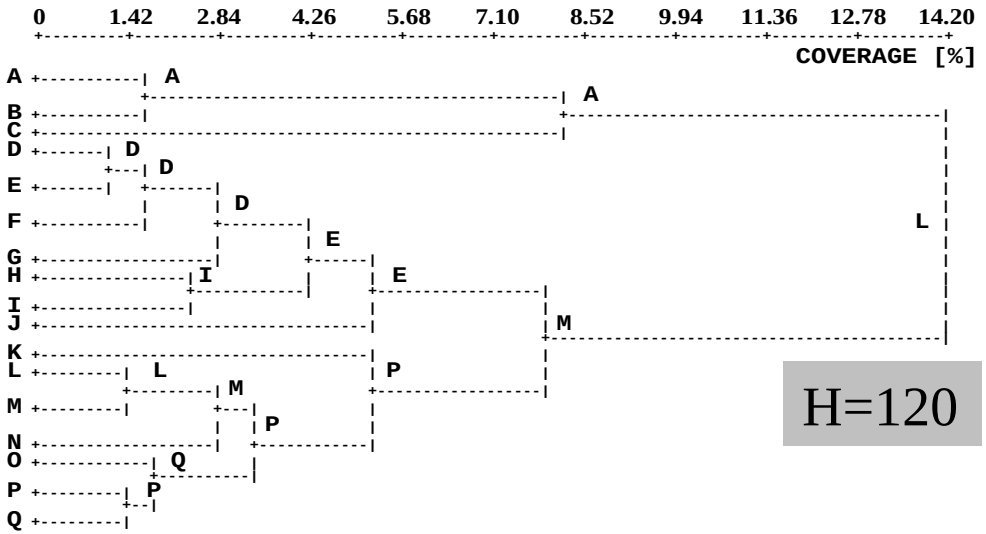
- CFP2000
  - The workload consists of two insufficiently balanced groups: a large group {A,B,C,D,E,F,G,H,I,J} and a group {J,K,L,M,N} that is two times smaller
  - Benchmarks F and N are outliers
- CINT2000
  - Benchmarks D and E are overly redundant and I and L are outliers. The group {J,K,L} is obviously underrepresented.
- Density is to high

CPU 06

CFP2006



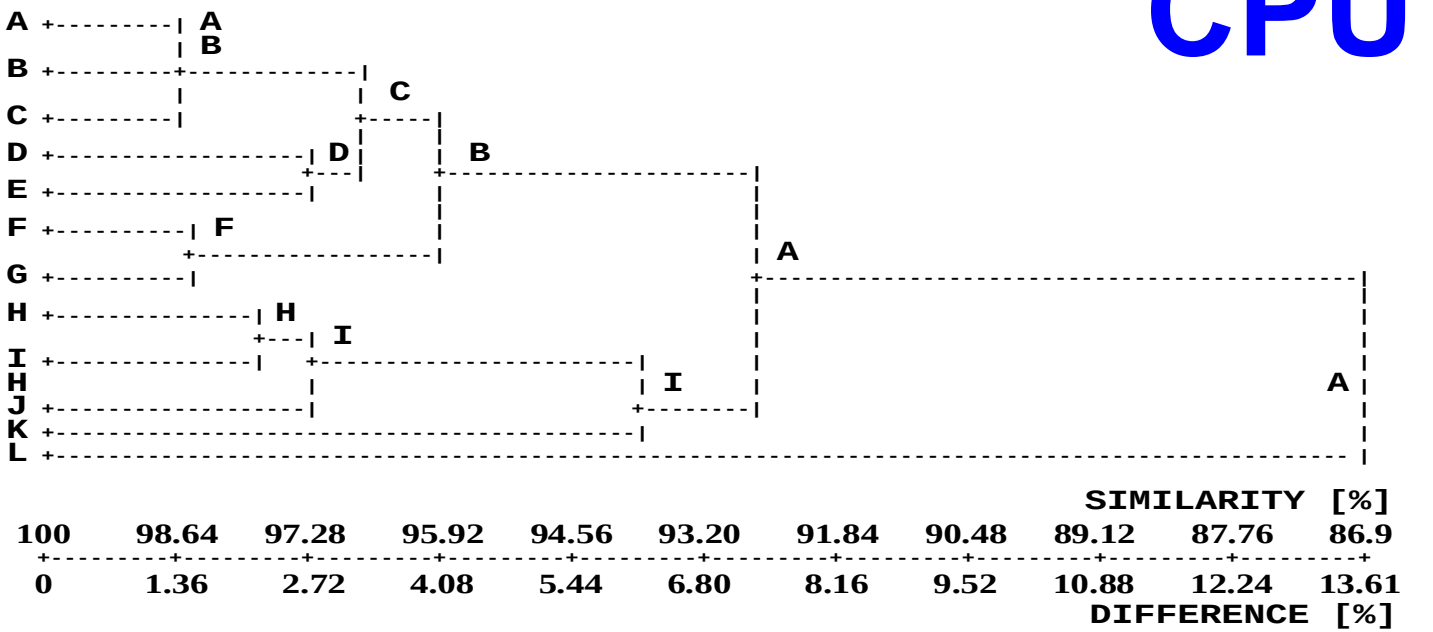
A=bwaves, B=cactusADM, C=lbm, D=gamess, E=tonto, F=gromacs, G=povray, H=soplex, I=calculix, J=namd, K=milc, L=zeusmp, M=leslie3d, N=GemsFDTD, O=dealII, P=wrf, Q=sphinx3



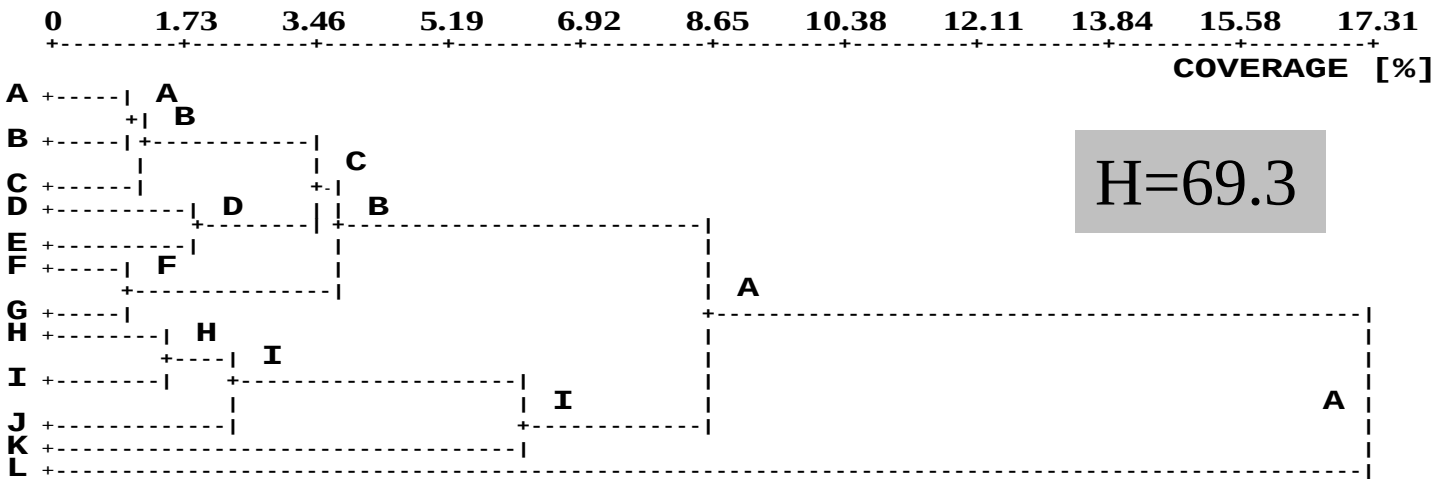
H=120

CINT2006

CPU 06



A=perlbench, B=gobmk, C=sjeng, D=gcc, E=omnetpp, F=bzip2, G=astar, H=mcf, I=xalancbmk, J=h264ref, K=hmer, L=libquantum



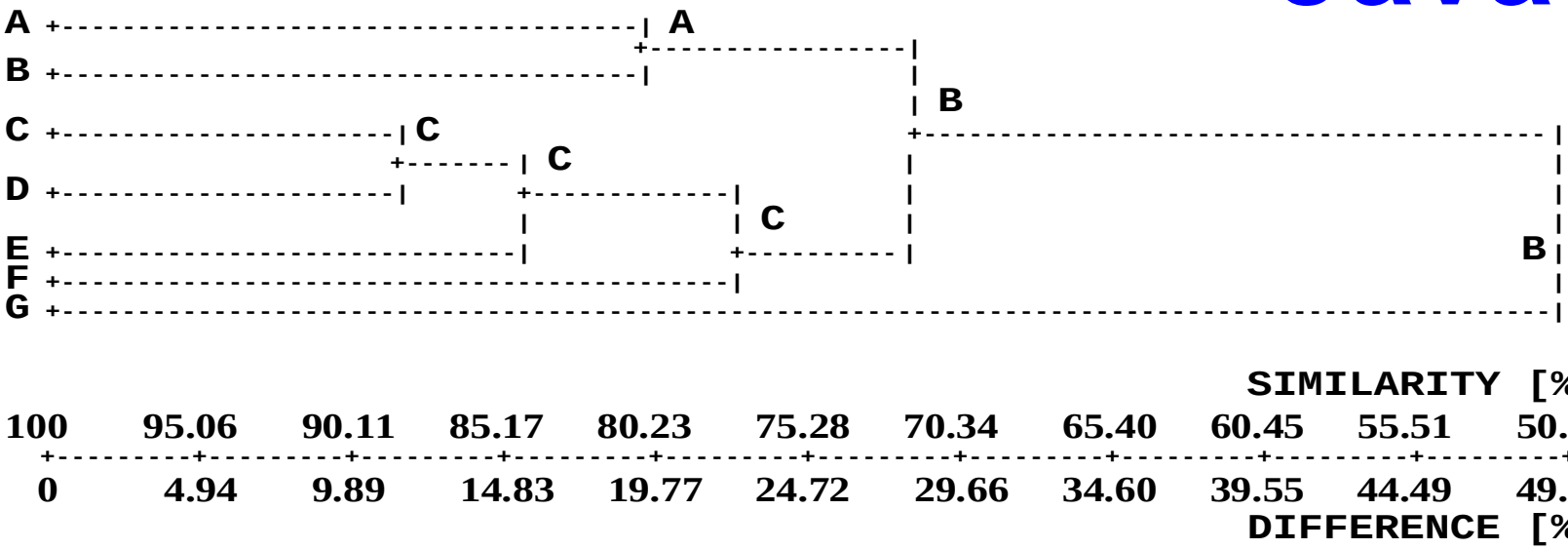
H=69.3

Evaluation of CPU 2006

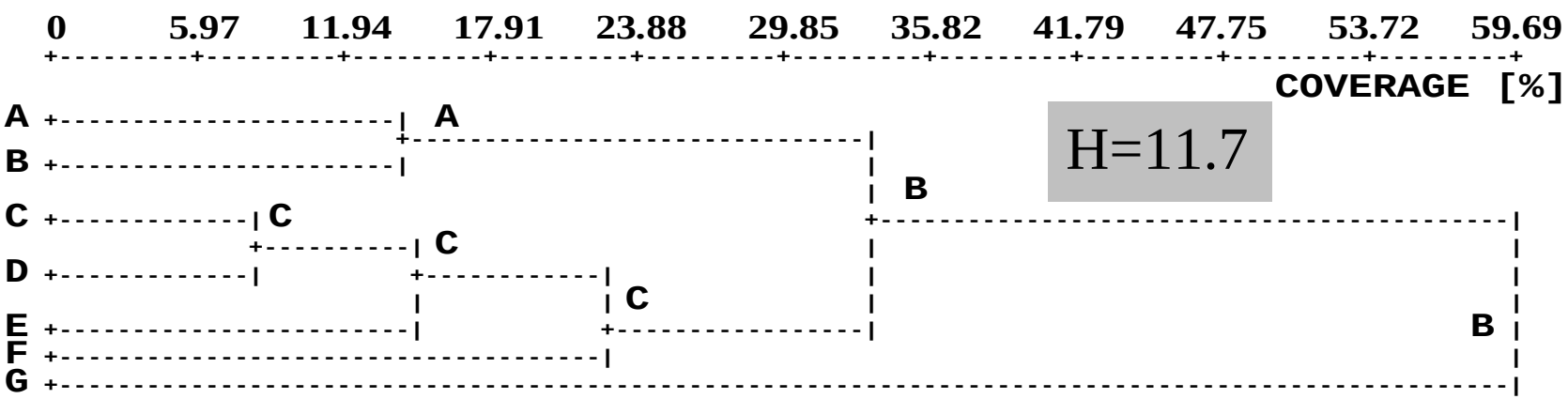
- Excessive density (29 component benchmarks)
- Applied to more than 2500 computers
- CFP2006
  - Component benchmarks differ for less than 10% (and frequently less then 5%)
  - Plutonium mean density: H=120
  - The main group of 14 highly redundant benchmarks {D,E,…,Q} differs for less that 6% and attains a record local density of approximately 240
- CINT2006
  - Similar problems as CFP2006. The main group with 11 highly redundant benchmarks covers only 8.6% of program space
  - There are five benchmarks that differ for less than 2%.

JVM98

Java 98



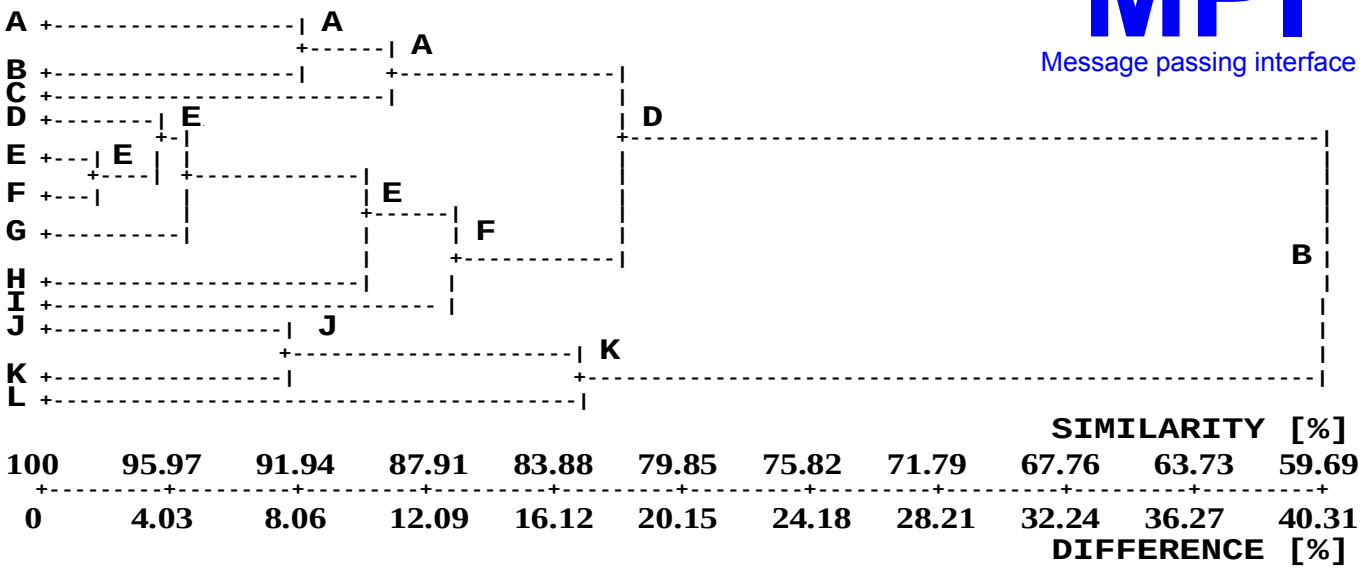
A=mtrt, B=javac, C=jess, D=jack, E=mpegaudio, F=db, G=compress



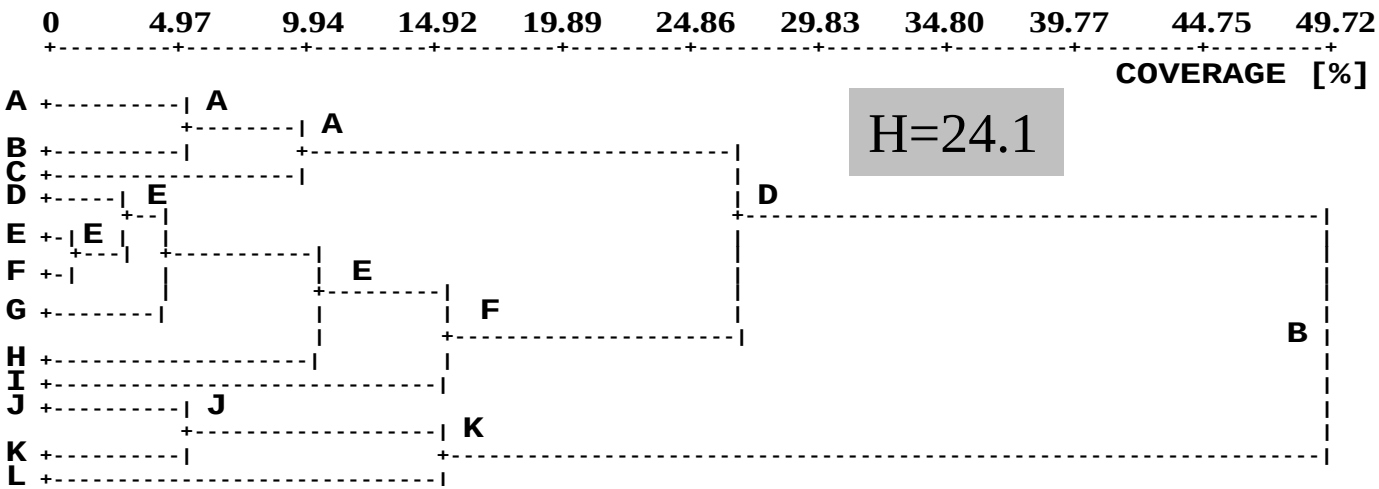
Evaluation of JVM98

- The main cluster of JVM98 includes six well-distributed component benchmarks that cover 33% of program space with an appropriate density of 18.2.
- The outlier benchmark G expands the coverage to approximately 60%, and dramatically reduces the density.
- Consistent with the machine-independent white-box clustering (*javac* is always the central benchmark)

MPIL2007



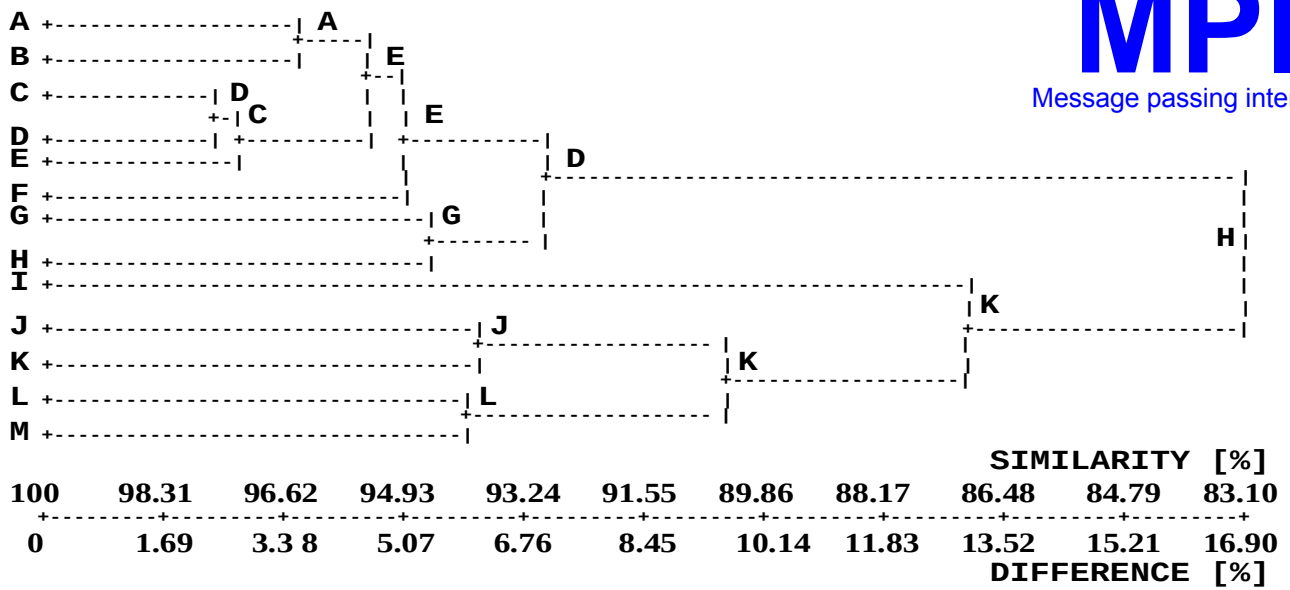
A=pop2, B=zeusmp2, C=lammps, D=GAPgeofem, E=lGemsFDTD, F=l2wrf2, G=lu, H=dleslie, I=dmilc, J=tachyon, K=tera_tf, L=RAxML



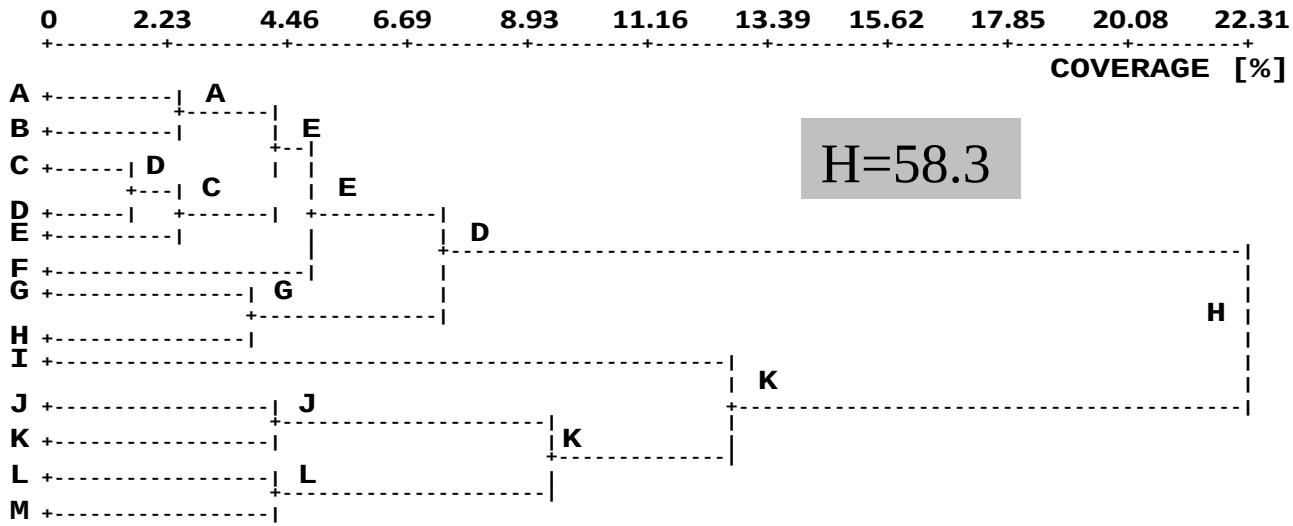
H=24.1

MPIM2007

MPI
Message passing interface



A=milc, B=socorro, C=leslie3d, D=GAPgeofem, E=fds4, F=lu, G=GemsFDTD, H=wrf2, I=pop2, J=tachyon, K=tera_tf, L=lammps, M=zeusmp2



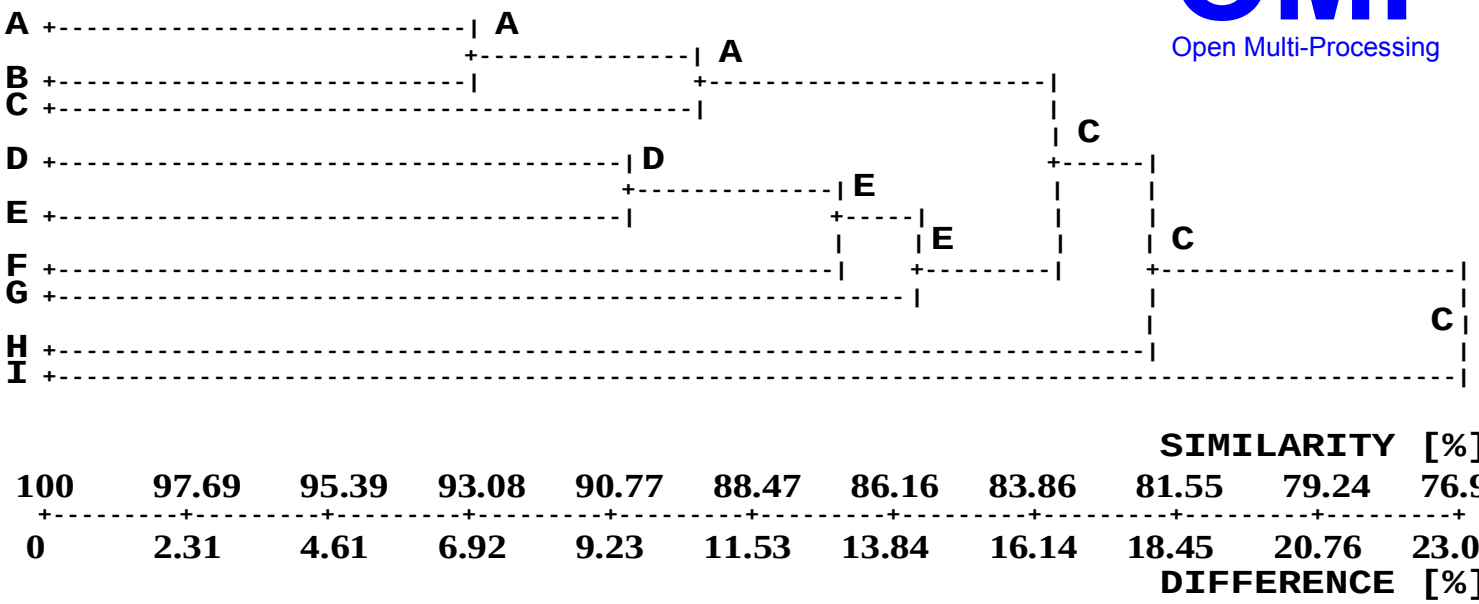
H=58.3

Evaluation of MPI2007

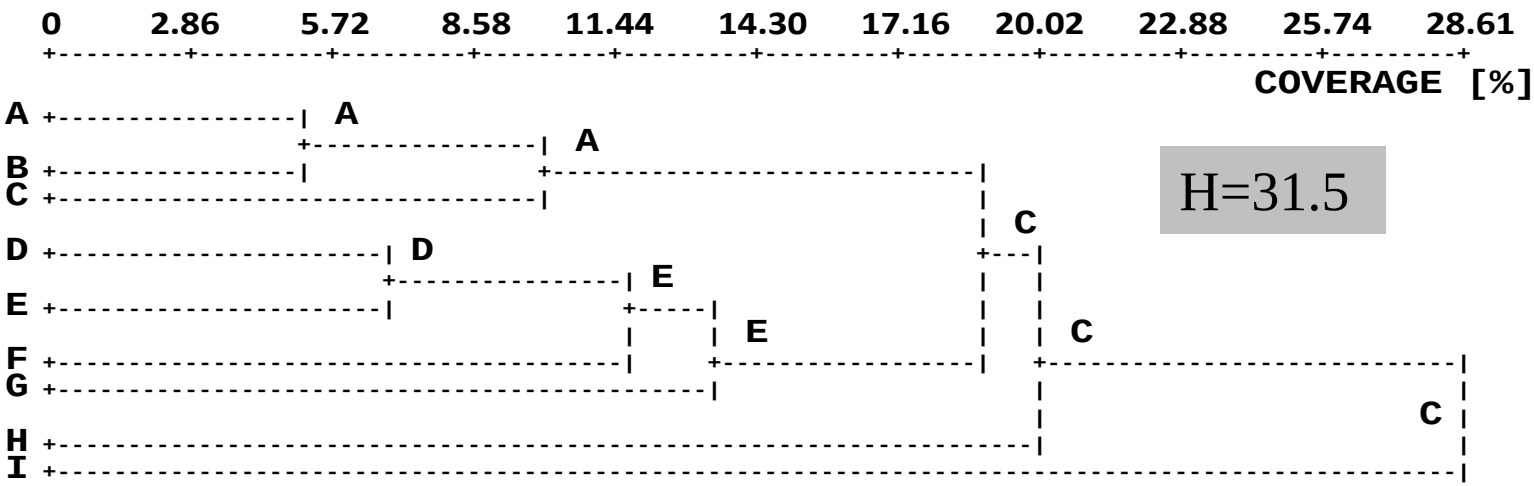
- MPIL2007
  - Completely appropriate density
  - The distribution of component benchmarks can be significantly improved because it consists of two remote and imbalanced subclusters
- MPIM2007
  - The coverage is very low yielding an excessively high density.
  - Similarly to MIPL2007, the distribution of MPIM2007 includes two disconnected and overrepresented subclusters.

OMPL2001

OMP
Open Multi-Processing



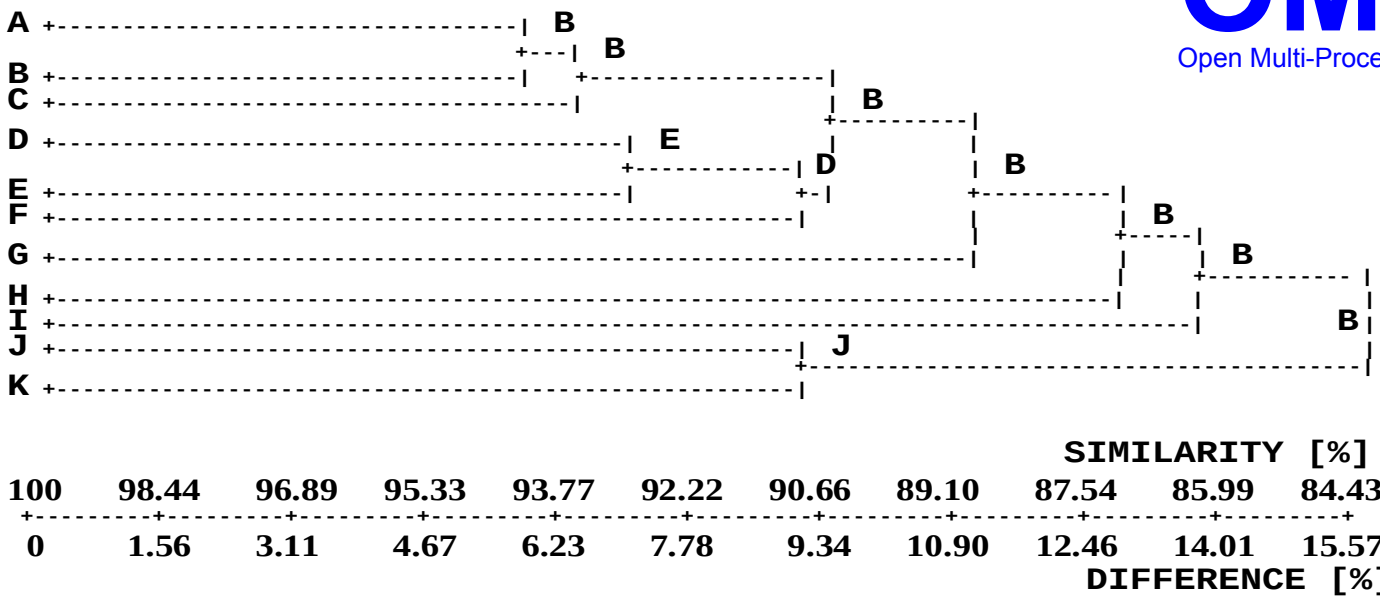
A=wupwise_l, B=gafort_l, C=mgrid_l, D=swim_l, E=fma3d_l, F=apsi_l, G=equake_l, H=applu_l, I=art_l



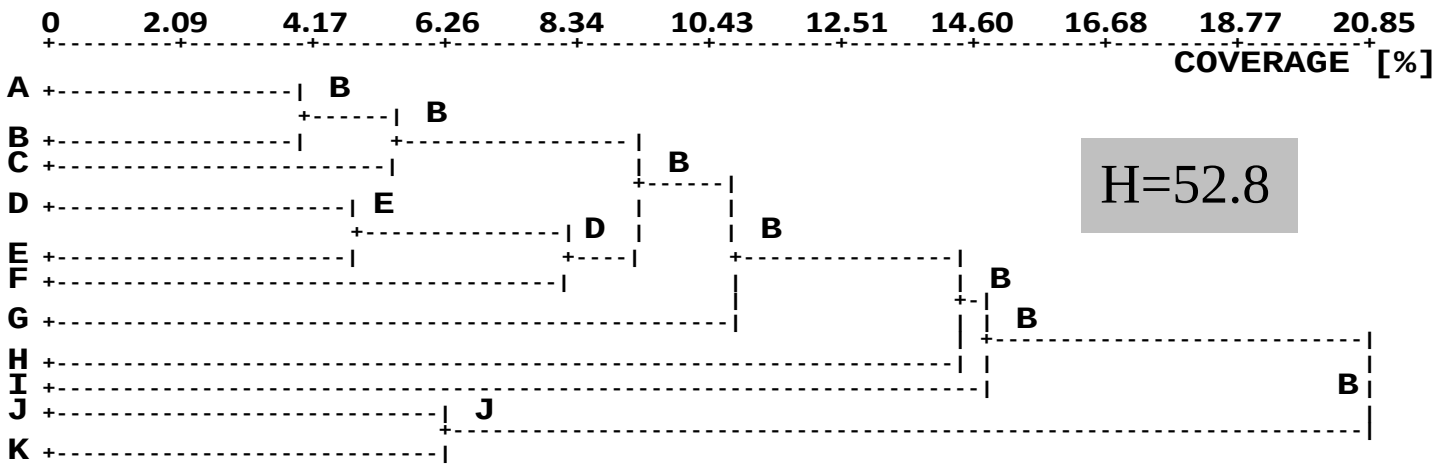
H=31.5

OMPM2001

OMP
Open Multi-Processing



A=wupwise_m, B=gafort_m, C=apsi_m, D=swim_m, E=mgrid_m, F=applu_m, G=equake_m, H=art_m, I=galgel_m, J=fma3d_m, K=ammp_m



H=52.8

Evaluation of OMP2001

- OMPL2001
  - High density
  - Low coverage
  - The outlier I
- OMPM2001
  - Excessive density
  - Low coverage
  - J and K : detached and underrepresented group

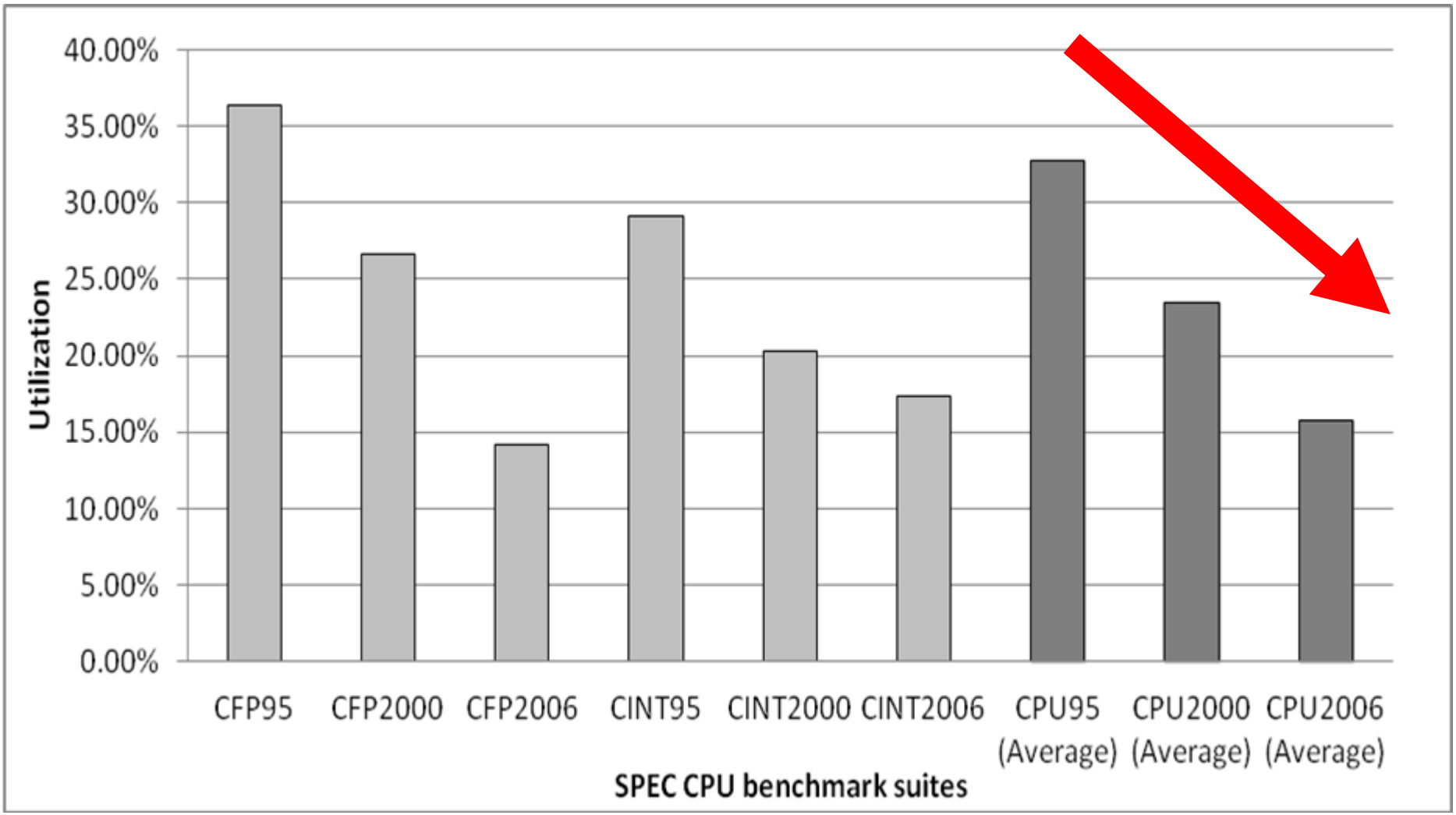
Summary of cluster analysis results

- According to cluster analysis, current SPEC component-level benchmark suites have the following drawbacks [Oct'11] :
  - Use of overrepresented workloads
  - Use of underrepresented workloads
  - Insufficient coverage of program space
  - Excessive density of component benchmarks
  - Waste of benchmarking effort

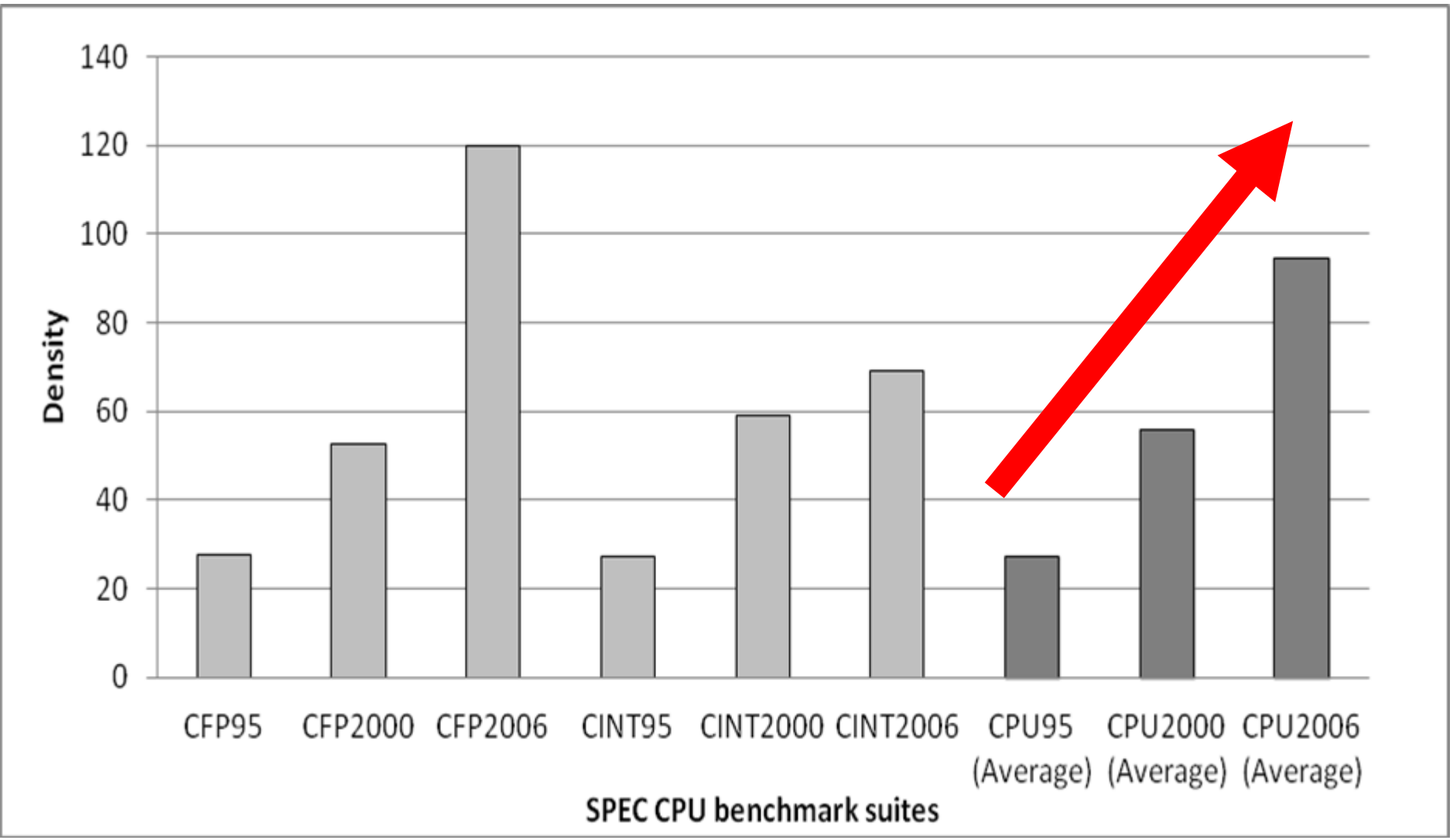
Trends

- Decreasing program space utilization
- Increasing density of SPEC CPU benchmark suites
- Increasing number of measured computer systems (and increasing cost)
- Decreasing average difference between individual workloads

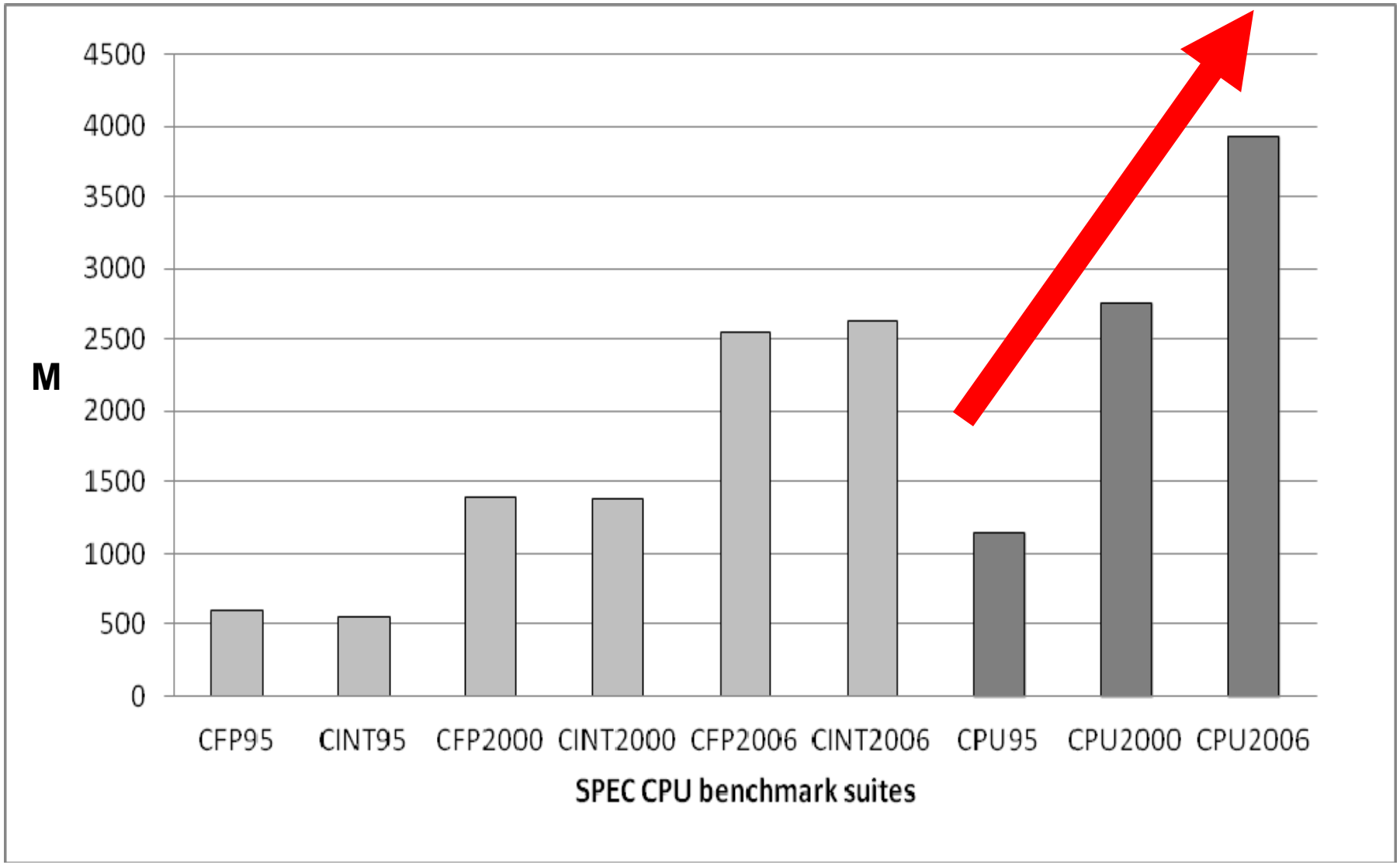
Program space utilization of SPEC CPU benchmark suites using all performance measurement results



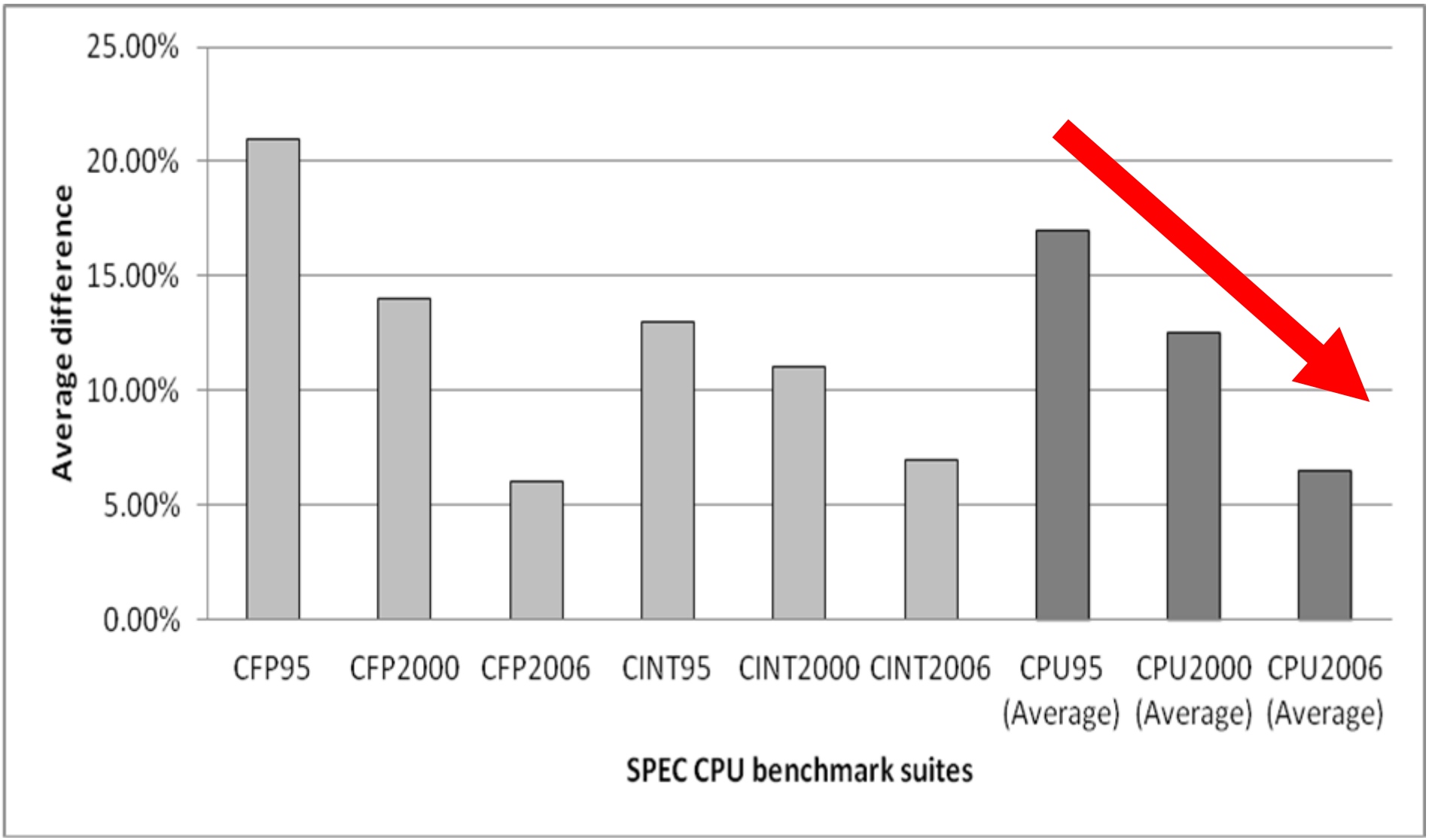
Density of SPEC CPU benchmark suites based on all performance measurement data



Number of computer systems measured with each version of SPEC CPU benchmark suite



Average difference between individual workloads within SPEC CPU benchmark suites based on all performance measurement data



Evolutionary Adjustment of Benchmark Suites

- Current SPEC approach: complete replacement of benchmark suites after several years
- A proposed more efficient approach: incremental evolutionary adjustment of benchmark suites
  - Continuous evaluation of benchmark suites
  - Removing too redundant workloads
  - Adding workloads that improve the uniformity of distribution
  - Supplementing/elimination/justification of outliers

Method #1: Increasing the uniformity of distribution

- Start using benchmark suite with all component benchmarks that are initially developed.
- Compute the level of redundancy between component benchmarks after collecting measurements from sufficient number of diverse computers.
- Remove redundant component benchmarks when redundancy (expressed as similarity) reaches a selected threshold level (e.g. 90-95%).
- Identify outlier benchmarks and investigate whether they are representatives of underrepresented groups of benchmarks that need to be expanded; in such cases introduce additional component benchmarks in the underrepresented groups.

Predictive power of reduced benchmark suite

1. Suppose that we have M computers that are measured using a benchmark suite. Then there are $M^* = M(M-1)/2$ pairs of computers that can be compared using the corresponding SPECmark indicator.

2. Compute the SPECmark indicators for computers C and D using the original N component benchmarks:

$$S_N(C,R) = \prod_{k=1}^{N} [t_k(R)/t_k(C)]^{1/N}$$
$$S_N(D,R) = \prod_{k=1}^{N} [t_k(R)/t_k(D)]^{1/N}$$

3. Computers C and D are one of M^* pairs for comparison. Assign to this pair the following values: 1 denotes that C is better than D, and -1 denotes that D is better than C:

$$Q_N(C,D) = \begin{cases} 1, \text{ if } S_N(C,R) > S_N(D,R) \\ -1, \text{ if } S_N(C,R) \le S_N(D,R) \end{cases}$$

4. Repeat the same procedure for $n \le N$ selected representatives of the original benchmark suite:

$$S_n(C,R) = \prod_{k=1}^{n} [t_k(R)/t_k(C)]^{1/n}$$
$$S_n(D,R) = \prod_{k=1}^{n} [t_k(R)/t_k(D)]^{1/n}$$

$$Q_n(C,D) = \begin{cases} 1, \text{ if } S_n(C,R) > S_n(D,R) \\ -1, \text{ if } S_n(C,R) \le S_n(D,R) \end{cases}$$

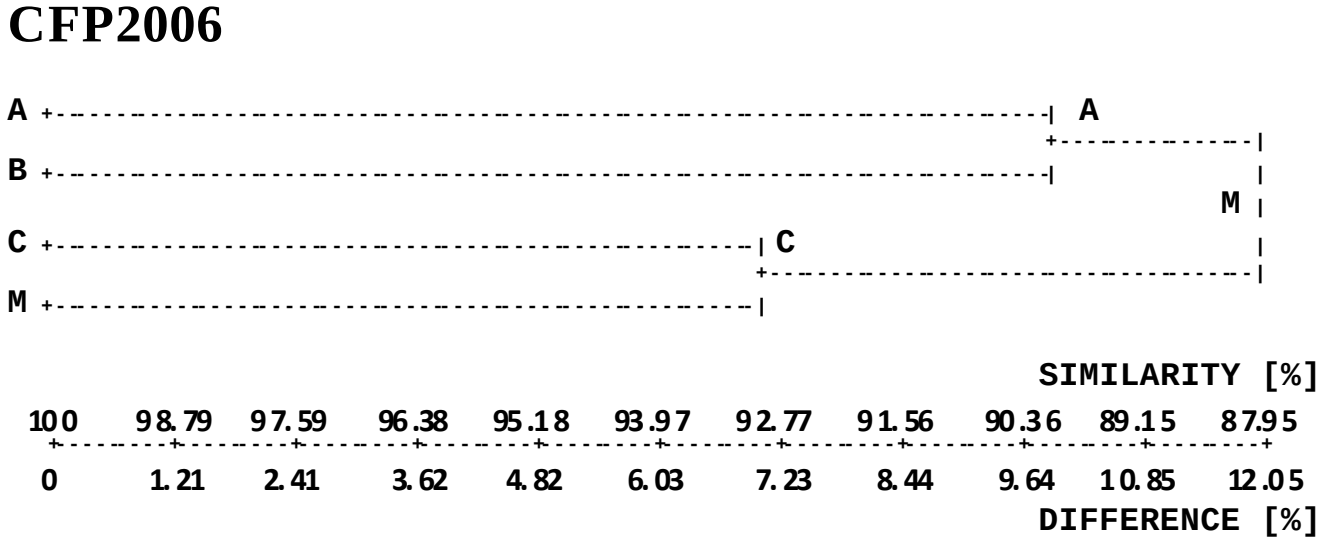
Compare arrays $Q_N[1..M^*]$ and $Q_n[1..M^*]$ and compute the fraction $P(N,n)$ of components that are the same. This percentage denotes the fraction of correct ranking and reflects the predictive power of the reduced benchmark suite.

Example of reducing CFP 2006

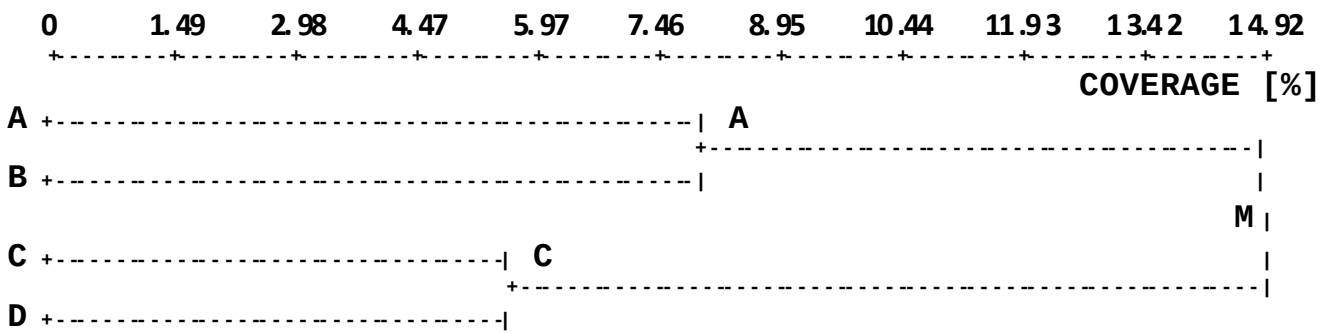
- Initial benchmarks N = 17
- Initial density H = 120
- Initial coverage U = 14.2%

- Reduced benchmarks N = 4
- Reduced density H = 26.8
- Reduced coverage U = 14.92%
- Correct ranking (compared to N=17): 95%
- Cost reduction: 4 times

Reduced CFP2006 benchmark suite



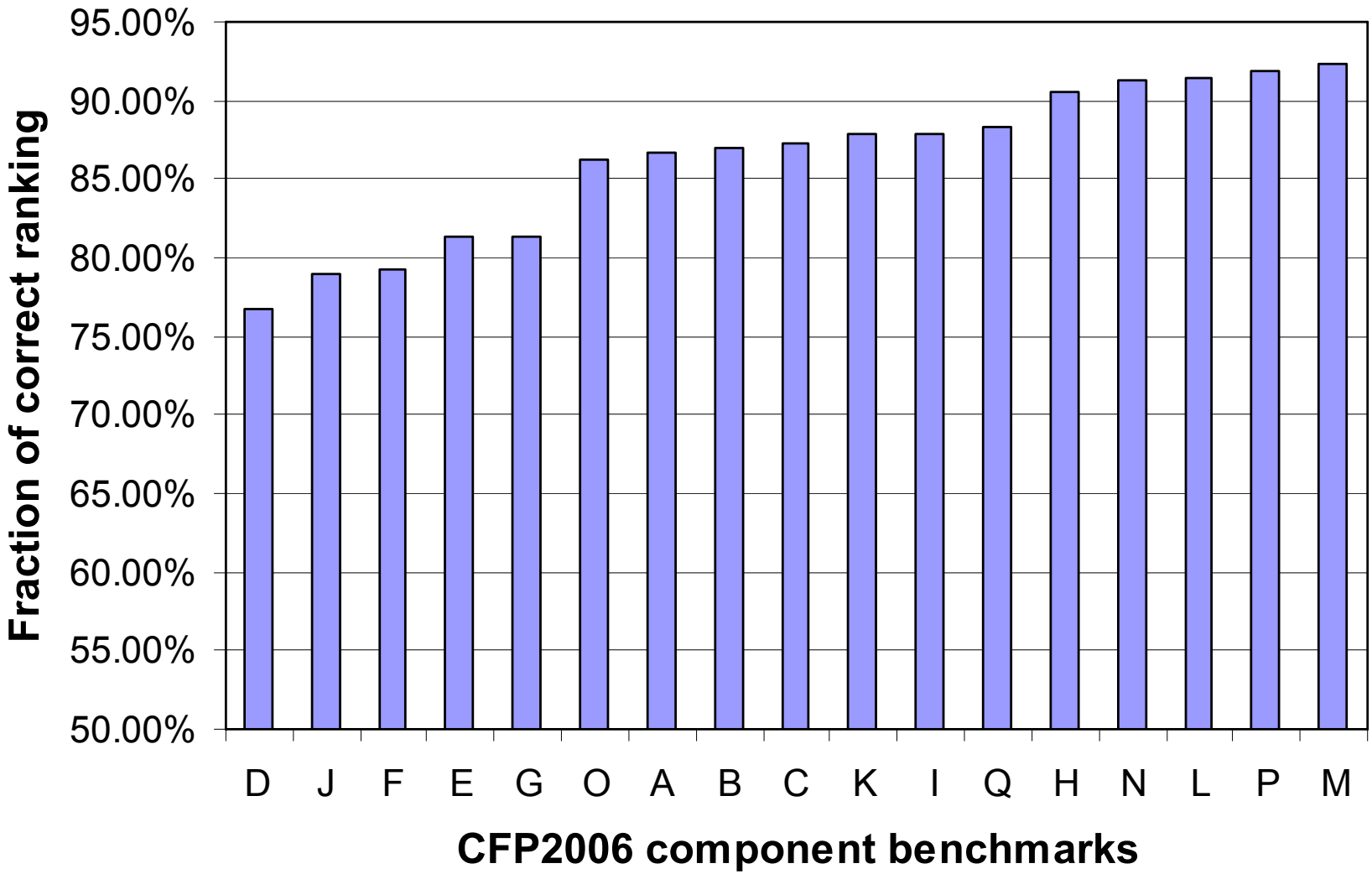
A=lbm, B=cactusADM, C=namd, M=leslie3d



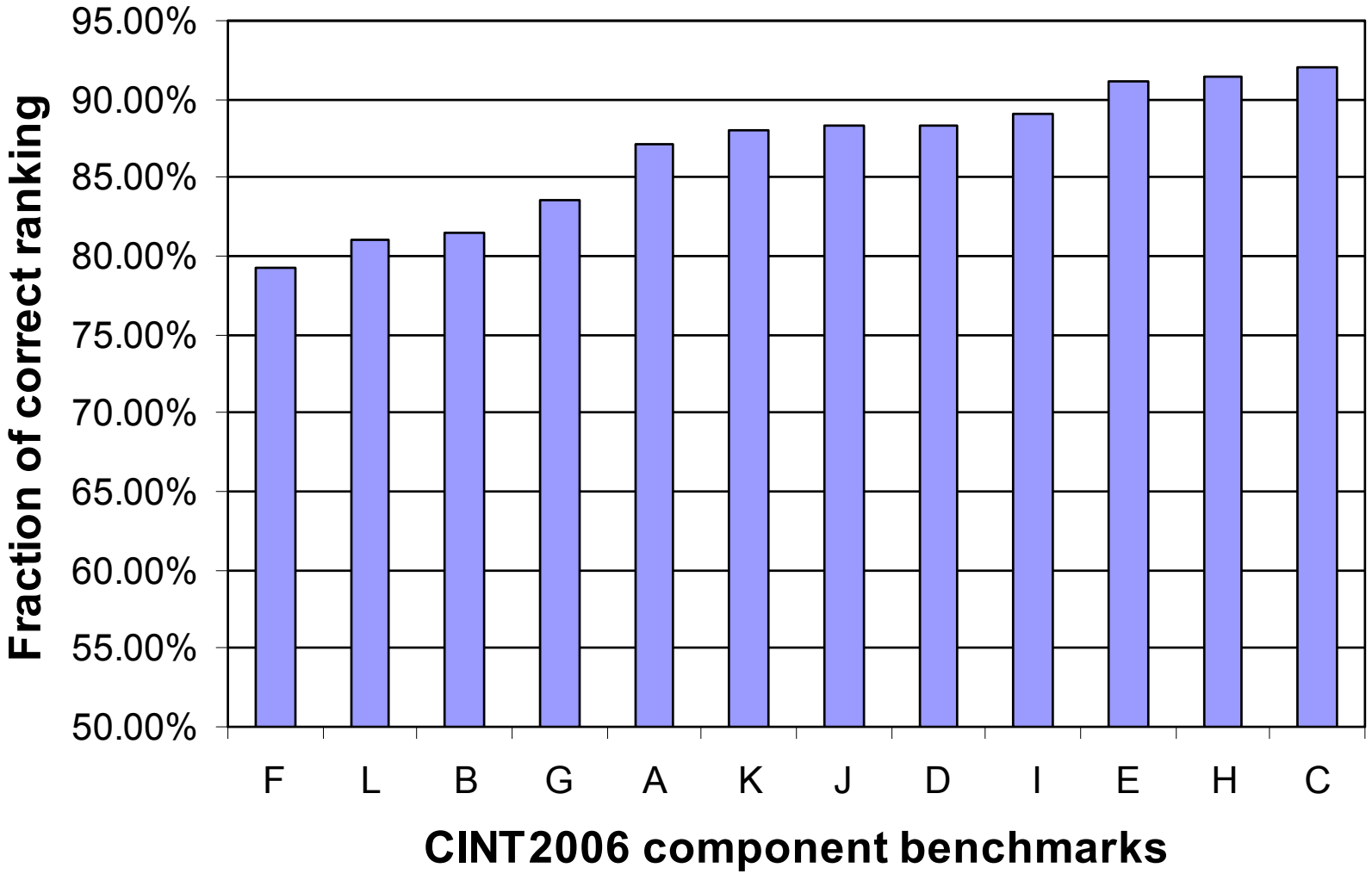
Method #2: Proximity to SPECmark

- Sort component benchmarks according to the quality of ranking (similarity between SPECmark ranking and ranking obtained by component benchmark)
- Select n best component benchmarks (n≥1) and use them to compute a SPECmark approximation with high accuracy (e.g. 95%)

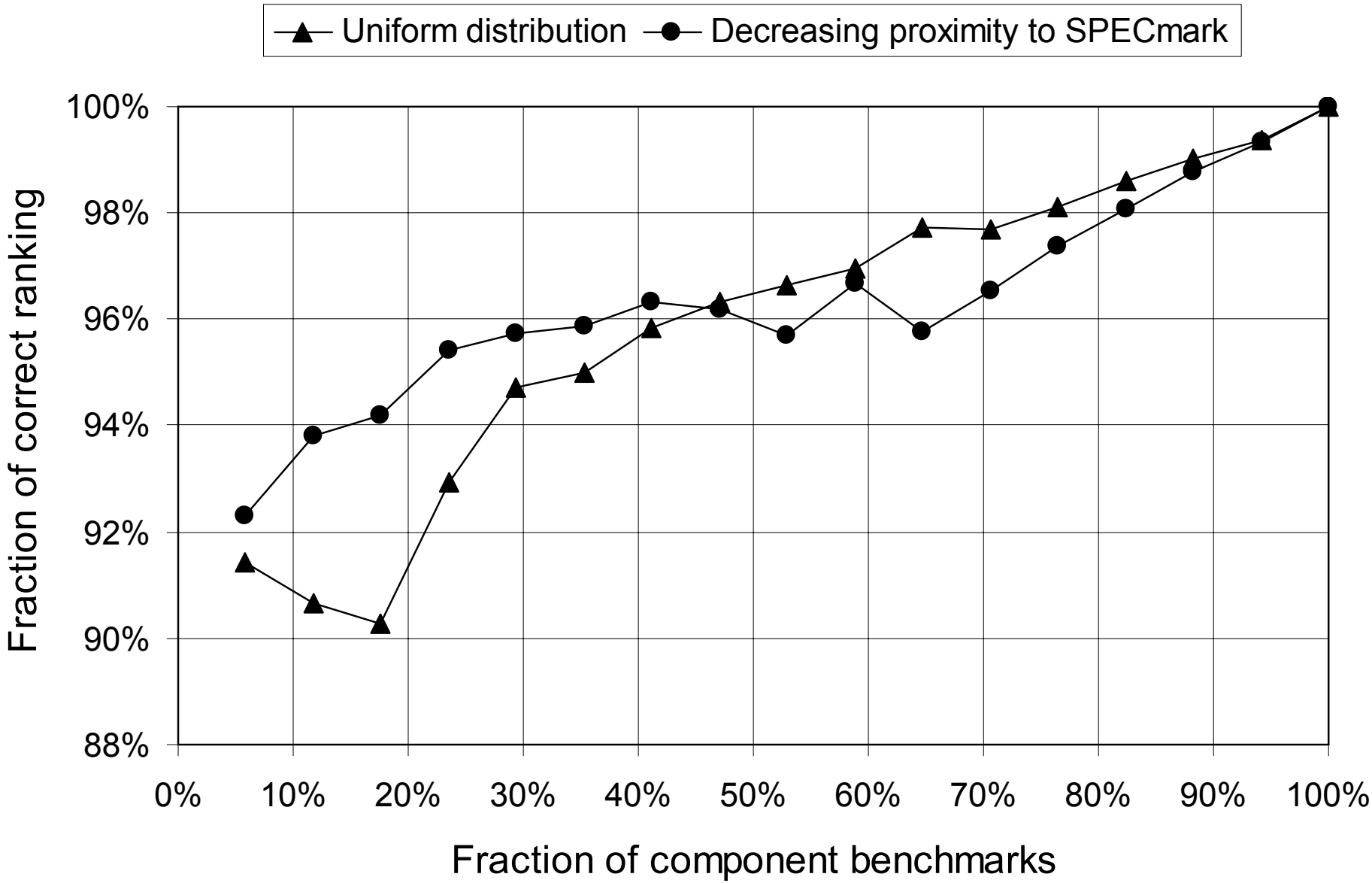
Predictive power of CFP2006 component benchmarks



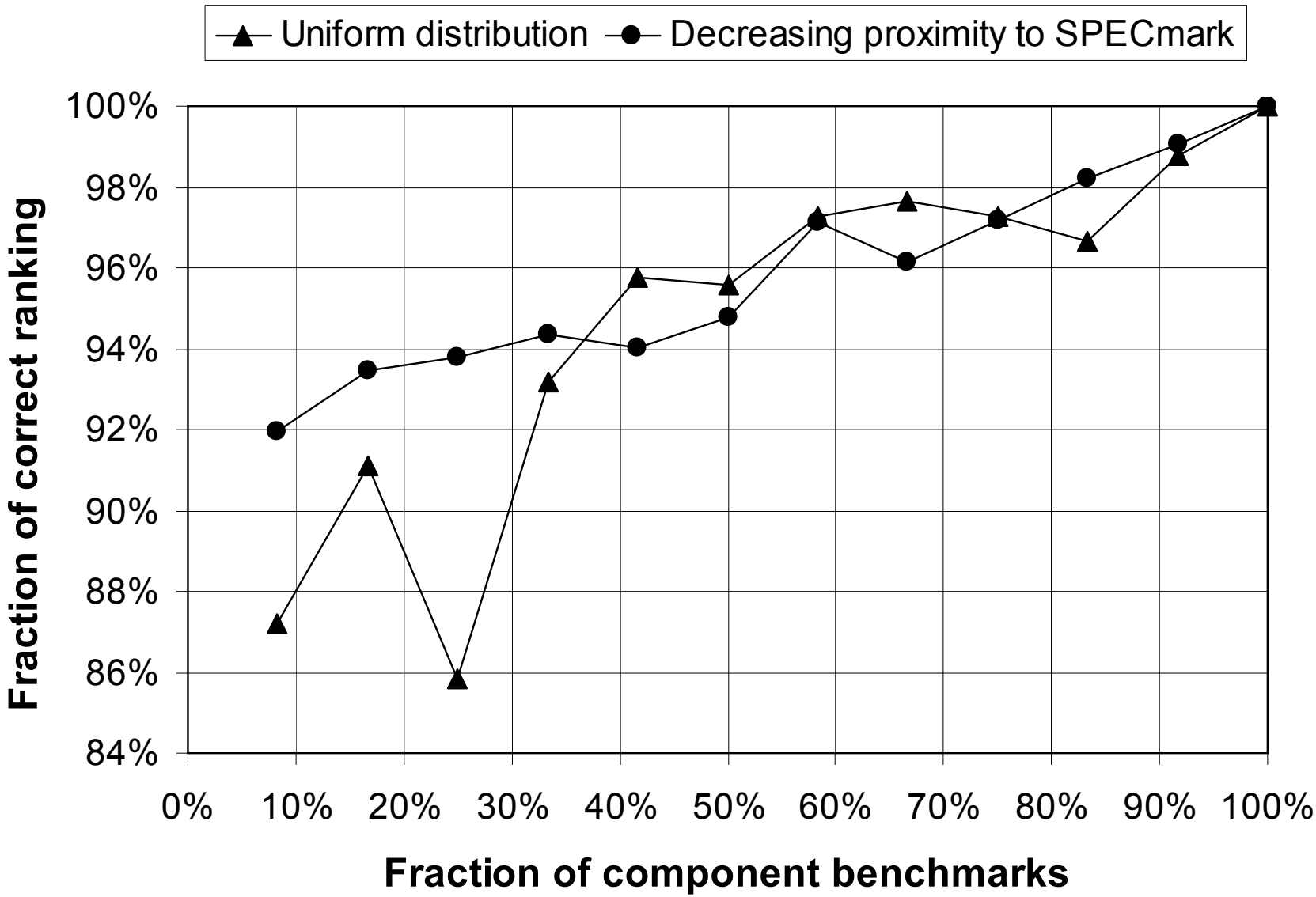
Predictive power of CINT2006 component benchmarks



Fraction of correct ranking for CFP2006



Fraction of correct ranking for CINT2006



Results

- With 50% of all of the component benchmarks and only 50% of the cost of benchmarking, SPEC can generate more than 95% of correct rankings. This shows a dramatic potential for reducing both the number of component benchmarks and the cost of benchmarking.
- In extreme cases of using only one best representative, SPEC can achieve about 92% of correct rankings.
- In the case of CFP2006, 95% of correct rankings can be achieved using only 25% of component benchmarks

Conclusions

- The redundancy of component-level SPEC benchmarks is too high (i.e. insufficient diversity of workloads).
- The density of SPEC component-level benchmark suites is generally too high
- SPEC benchmark suites include both overrepresented and underrepresented component benchmark programs
- The cost of standard SPEC benchmarking can be significantly reduced
- Evolutionary adjustment of benchmark suites is a way to control and reduce the cost of benchmarking