UNIVERSITEIT GENT

# Performance Prediction based on Inherent Program Similarity

## PACT-2006

***September 19th 2006***

Seattle, Washington

HiPEAC COMPILATION ARCHITECTURE

*Kenneth Hoste*[‡], Aashish Phansalkar[†], Lieven Eeckhout[‡],

Andy Georges[‡], Lizy K. John[†] and Koen De Bosschere[‡]

[‡] ELIS, Ghent University, Belgium

[†] ECE, The University of Texas at Austin

# The performance of a platform is evaluated using benchmarks


**MacBook Performance Benchmarks**
SPEC performance: Up to five times faster than the iBook G4.(2)

| iBook G4 1.42GHz | MacBook Core Duo 2.0GHz | Δ |
|---|---|---|
| 5.7 | 29.1 | 5.1X |
| SPECint_rate_base2000<br>Integer calculation (estimate) | | |

| iBook G4 1.42GHz | MacBook Core Duo 2.0GHz | Δ |
|---|---|---|
| 4.3 | 24.7 | 5.7X |
| SPECfp_rate_base2000<br>Floating-point calculation (estimate) | | |

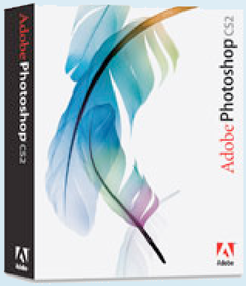
**www.apple.com/macbook/intelcoreduo.html, May 2006**

# How does that platform perform for my application(s) of interest?

ubiquitous problem in benchmarking


Excel (spreadsheet)


R (statistics)

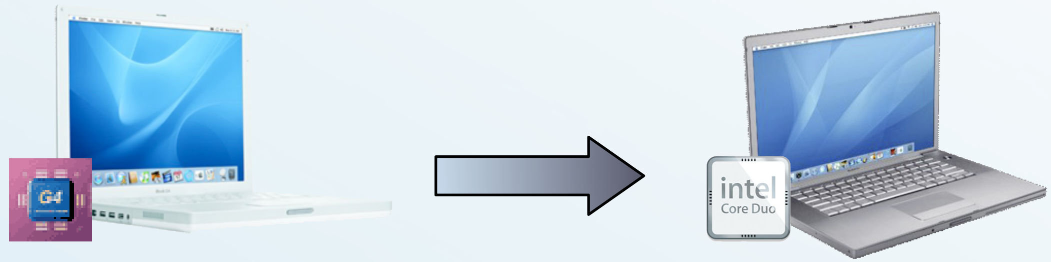



Photoshop (image processing)

Virtual PC (Windows on Mac)

# Evaluating several platforms for applications of interest is troublesome

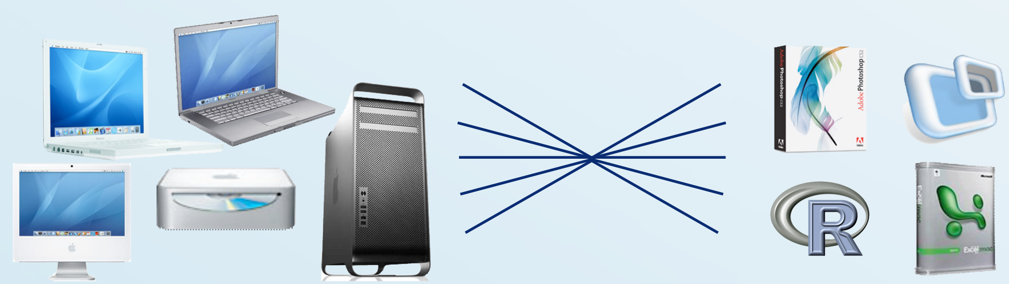
- Porting



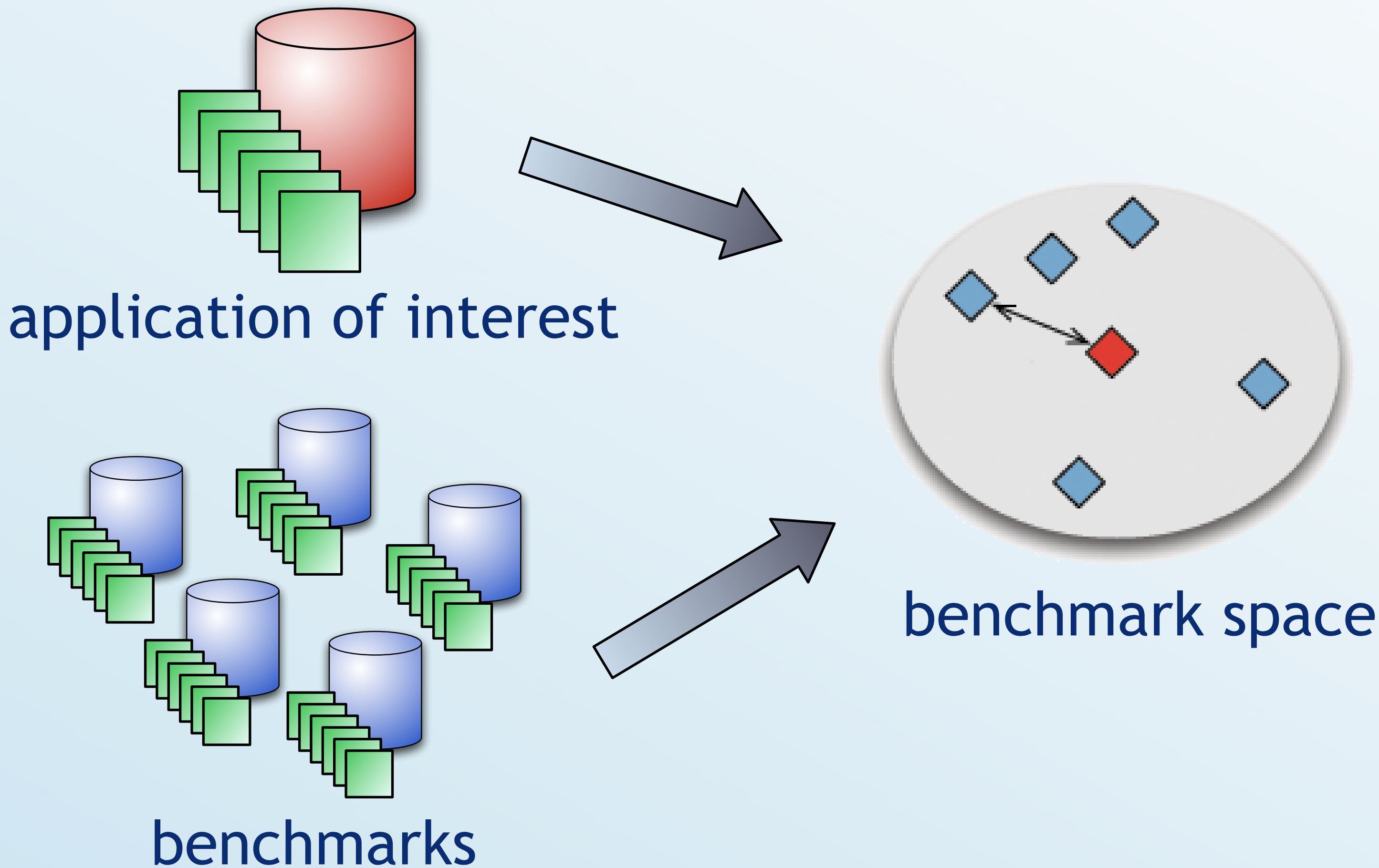

- Hardware availability

Playstation 3

- Time constraints

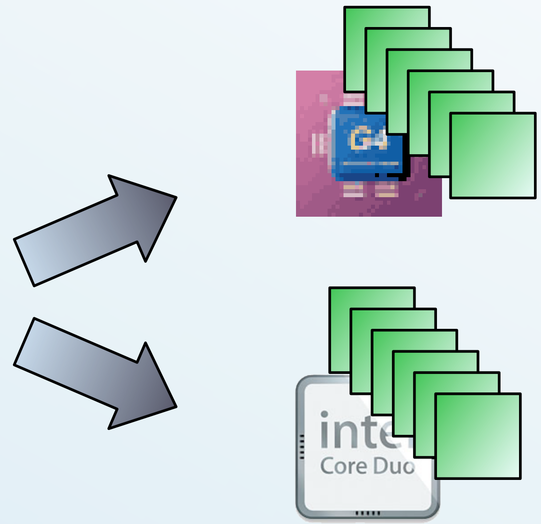


# We estimate performance based on program similarity



# How do we characterize a program without executing it on each platform?

on




application of interest

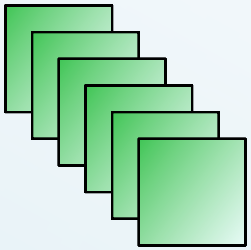
**microarchitecture-independent program characteristics**

in our setup: characterization on Alpha using ATOM

# Microarchitecture-independent program characteristics

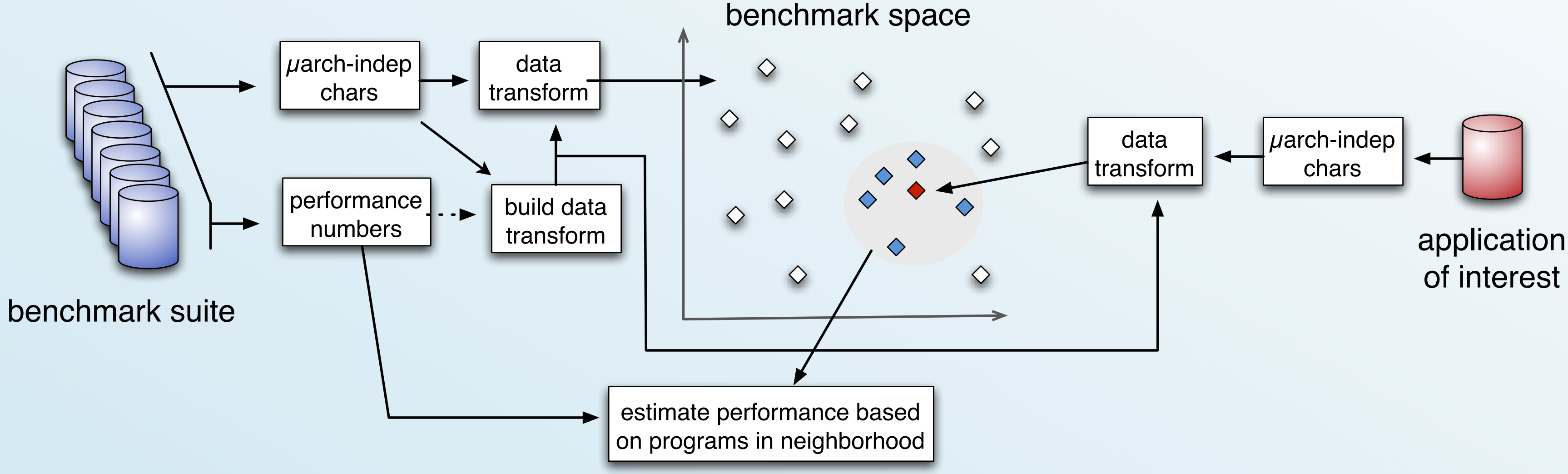
6 categories:

- instruction level parallelism (ILP)
- instruction mix
- branch predictability
- register traffic
- data stream strides
- working set size



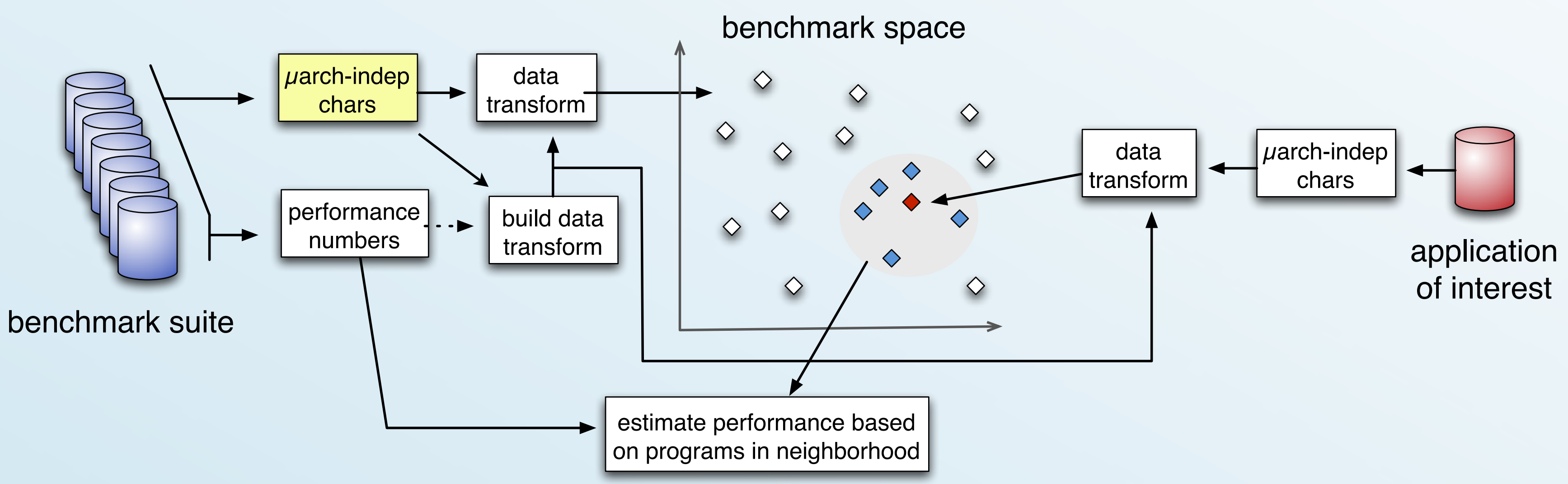
**totaling 47 program characteristics**
more details are available in the paper

# Estimating performance based on inherent program similarity



# Estimating performance based on inherent program similarity

## *Step 1*

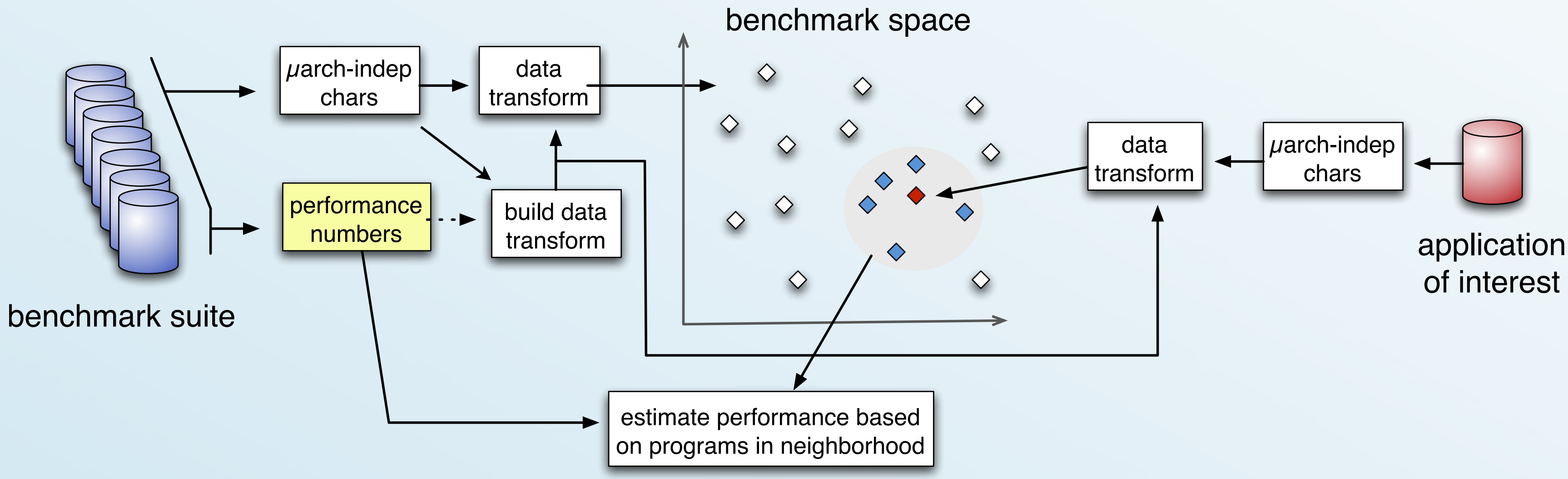



collect program characteristics for the benchmark suite

by instrumenting the benchmarks using ATOM/PIN

# Estimating performance based on inherent program similarity

## *Step 2*

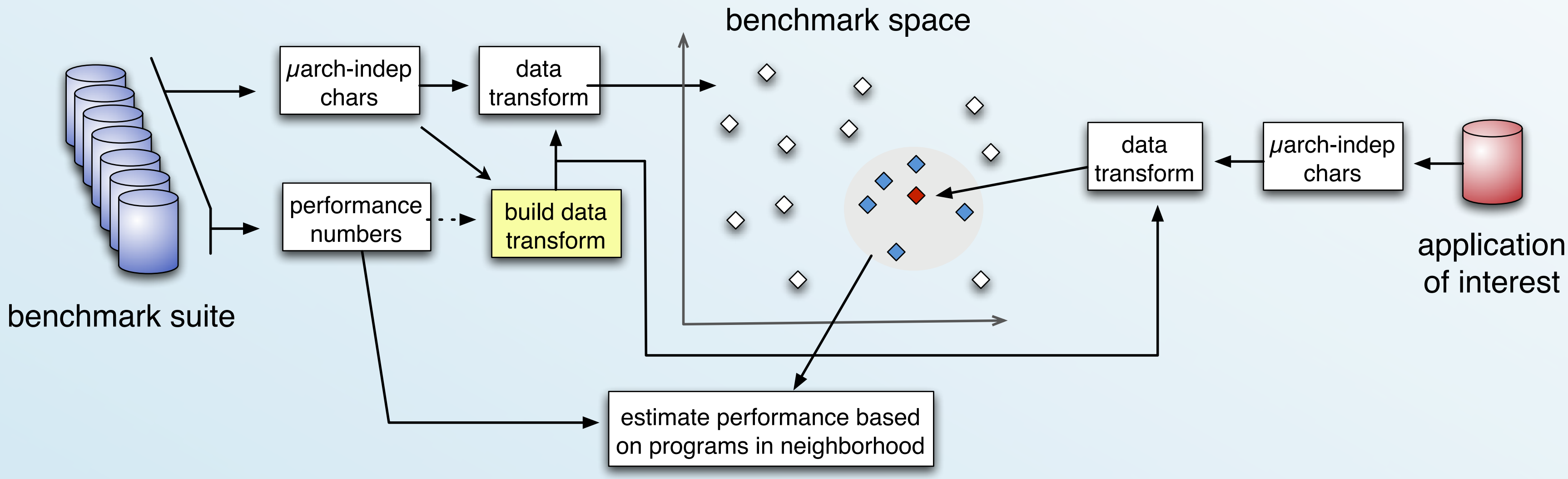



obtain performance numbers for each benchmark

for example, use the ones published on `spec.org` for SPEC CPU2000

# Estimating performance based on inherent program similarity

## *Step 3*

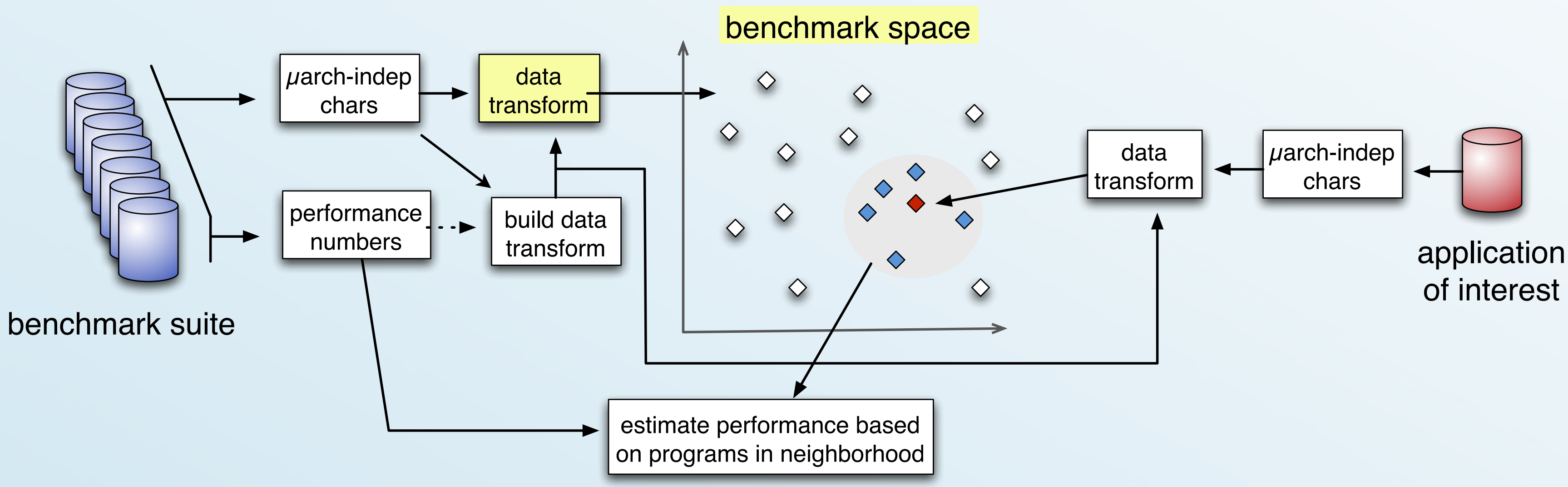



build the data transformation matrix

we evaluate 3 approaches: normalization, PCA and a genetic algorithm

# Estimating performance based on inherent program similarity

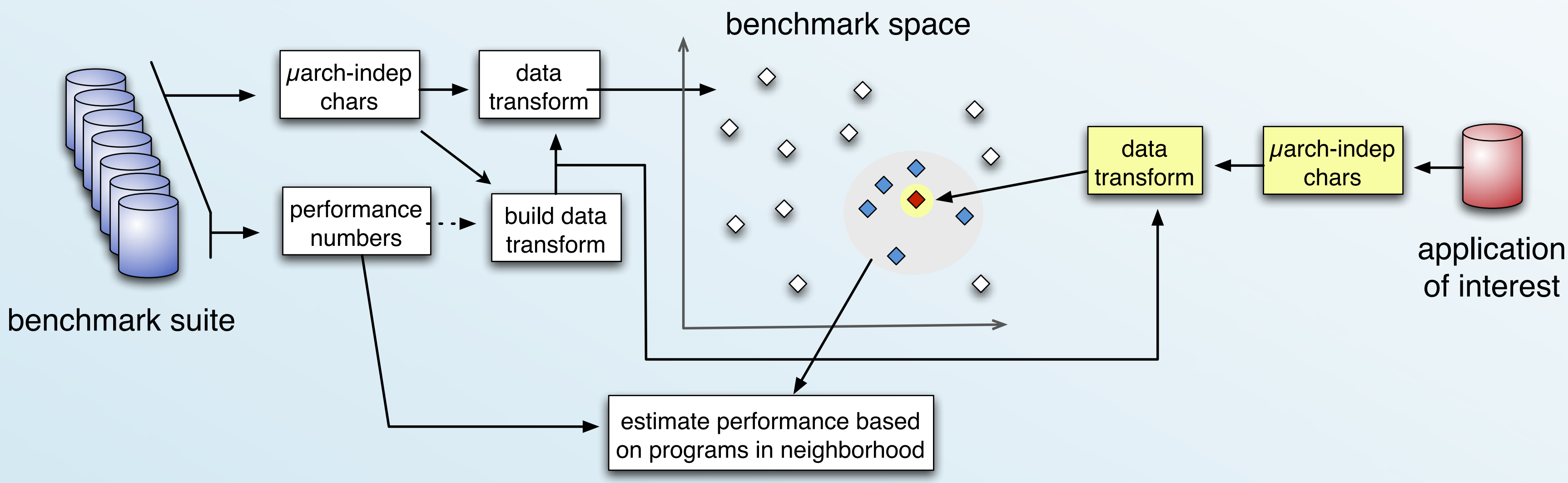
## *Step 4*




use the program characteristics and data transformation matrix to build the benchmark space

# Estimating performance based on inherent program similarity

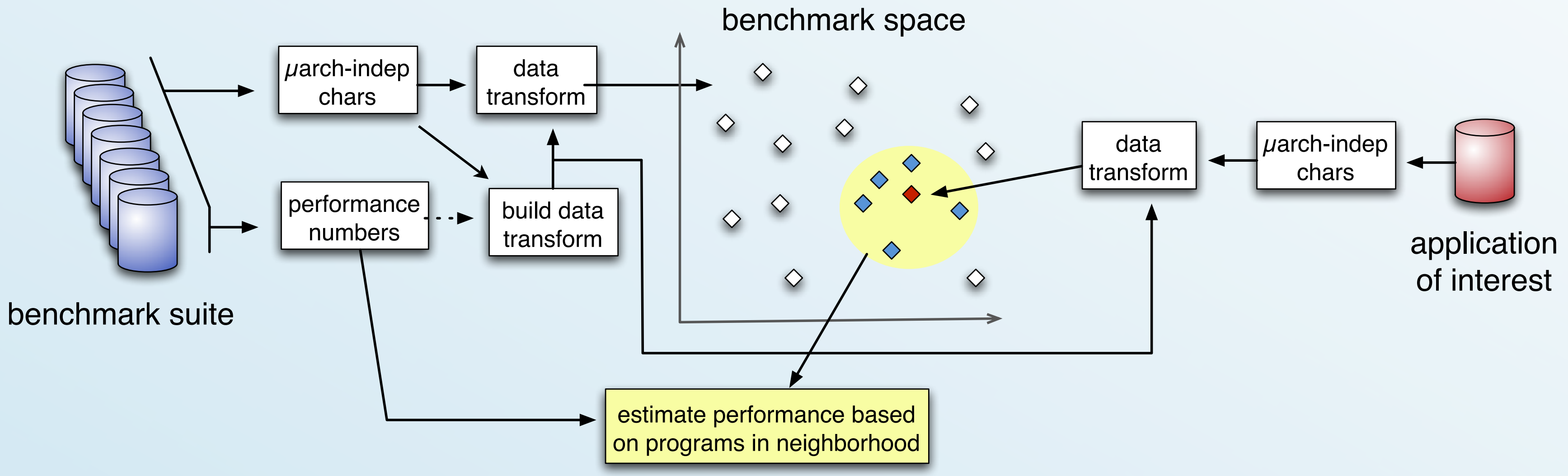
## *Step 5*




locate the application of interest
in the benchmark space

# Estimating performance based on inherent program similarity

## *Step 6*




estimate performance of the application of interest using the benchmarks in the neighborhood

# Building the data transformation matrix

## problem:

program characteristics vary in range

*ILP vs instruction mix*

Euclidean distance measure will be biased

## solution:

normalize characteristics

mean = 0, variance = 1

**data transformation:**

subtract mean and divide by standard deviation

# Building the data transformation matrix

**problem:**

program characteristics are correlated

➡ Euclidean distance gives a higher weight to correlated characteristics

**solution:**

obtain uncorrelated characteristics using Principal Components Analysis (PCA)

**data transformation:**

perform PCA on norm. chars, and normalize PCs

# Building the data transformation matrix

**problem:**

some program characteristics are more important for estimating performance than others

*branch predictability n vs % multiply operations*

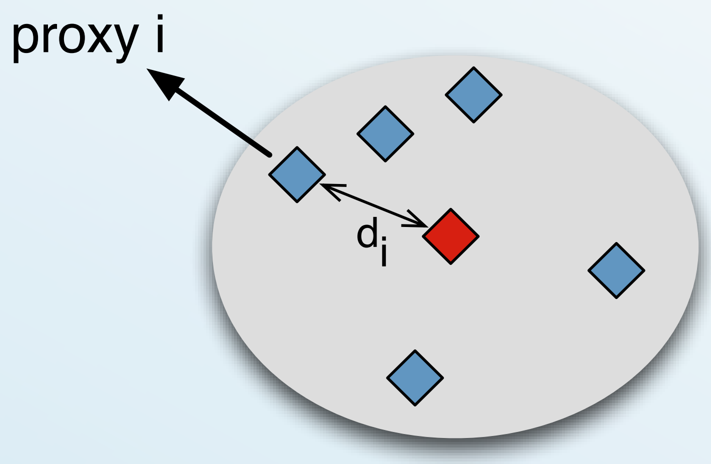
**solution:** genetic algorithm

learning how to scale the characteristics so that distance in the benchmark space correlates with difference in performance

**data transformation:**

weigh normalized characteristics

# Estimating performance using application proxies




$$w_i = \frac{\sum_{i=1}^{n} \frac{1}{d_i}}{d_i}$$

weight for each proxy i

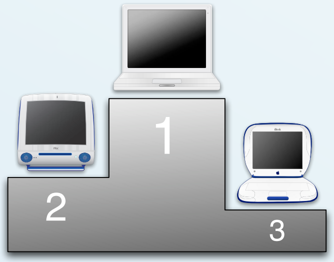
$$S = \frac{1}{\sum_{i=1}^{n} \frac{w_i}{S_i}}$$

weighted harmonic mean

# Determine the hardware platform ranking based on estimated performance

current practice:

rank machines based on average performance




**our approach:**

- estimate performance of the application of interest for each machine considered
- rank machines based on estimated performance

# Experimental setup

- full SPEC CPU2000 benchmark suite (26 benchmarks)

  methodology is evaluated using *crossvalidation*

  *n-1* benchmarks form benchmark suite,
  *n*th benchmark is application of interest

- speedup numbers for 36 machines
  - different ISAs, processor, configurations and manufacturers
  - taken from SPEC website (`http://www.spec.org`)

# Comparing the estimated ranking with the actual ranking of the machines

Spearman rank correlation coefficient

a value between -1 and 1

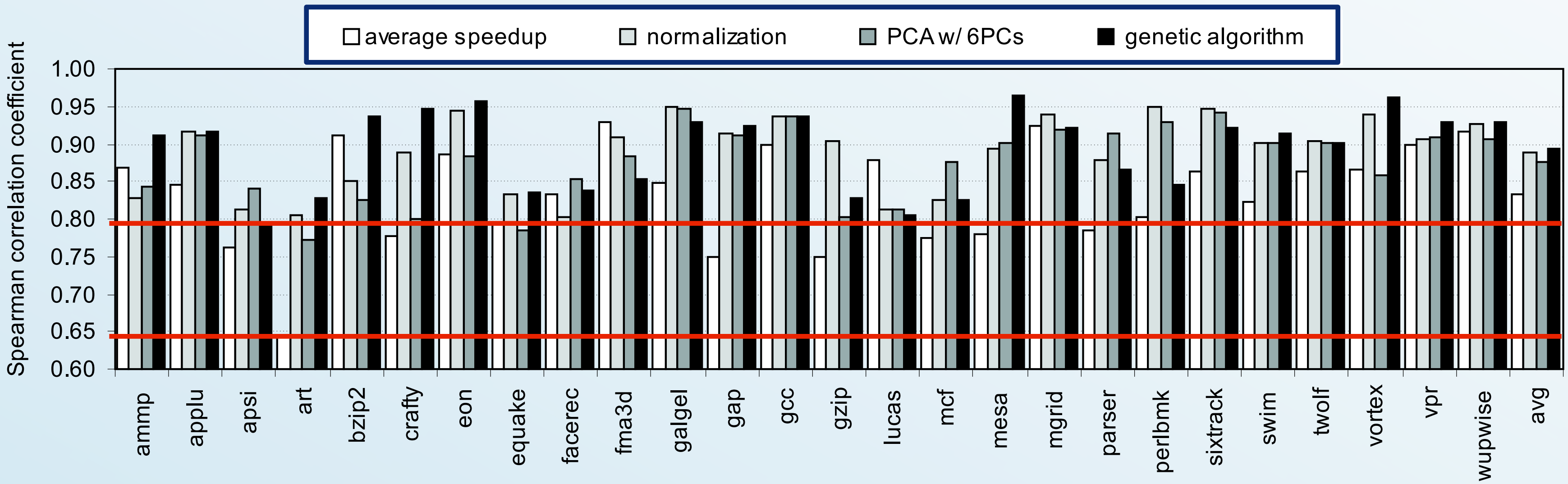
quantifies the quality of the estimated ranking

1: perfect

0: random

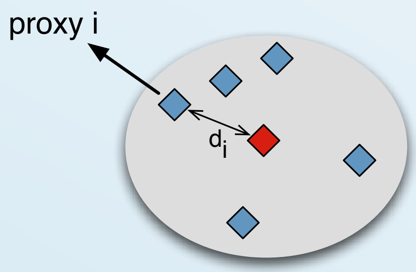
-1: worst possible (reverse)

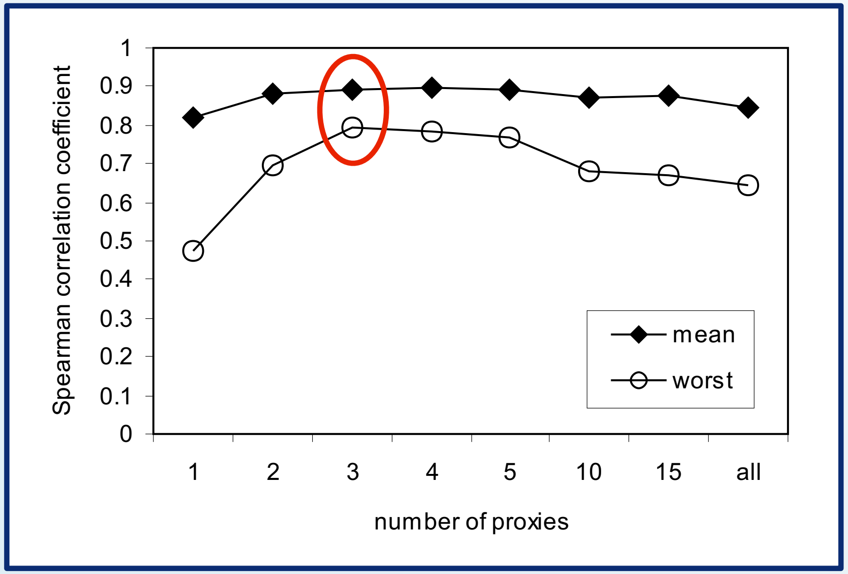
# Our methodology outperforms current practice




| | current practice | norm. | PCA |
|---|---|---|---|
| genetic algorithm | 23/26 | 16/26 | 16/26 |

# How many proxies should we retain?



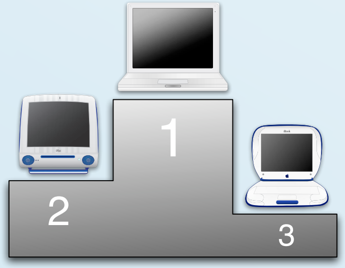

- a single proxy is not enough
- too many proxies degrades performance
- 3 proxies seems optimal
  (and is used in further evaluation)

# Evaluating the average performance loss

best machine:

current practice: 20%

**our approach: 13.6%**


top 3 machines:

current practice: 19.7%

**our approach: 11.1%**

small differences in rank correlation coefficient can make a big difference in practice

# Some interesting observations

- `gcc`, `mcf` and `swim` *never* appear as proxy
- `bzip2` is a *frequent* proxy
- microarchitecture-independent characteristics seem to be able to capture program behavior across ISAs

# Conclusions

- identifying the best machine should be done using the characteristics of the application of interest
- our methodology using the genetic algorithm is able to scale the benchmark space according to performance differences
- some benchmarks have very specific behavior (`gcc`, `mcf`, `swim`), others exhibit average behavior (`bzip2`)
- although program characteristics are measured on one particular ISA, they seem to capture program behavior over different ISAs fairly well

# Questions?



UNIVERSITEIT GENT