Fine-grained Benchmark Subsetting for System Selection

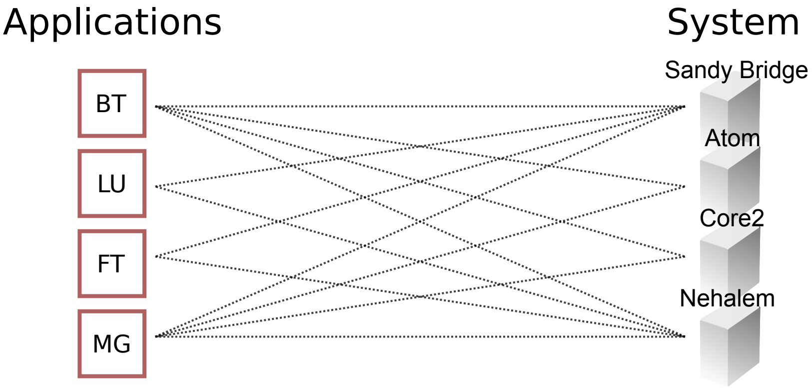
Pablo de Oliveira Castro, Y. Kashnikov,
C. Akel, M. Popov, W. Jalby

University of Versailles – Exascale Computing Research

CGO February 2014

Motivation

- Find system with the best performance on a set of applications?
- Reduce the cost of benchmarking



Key Idea

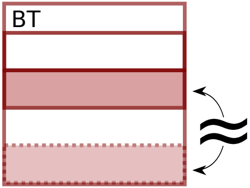
- Applications have redundancies
  - Similar code called multiple times
  - Similar code used in different applications
- Detect redundancies and keep only one representative

Previous Approaches

Remove similar applications

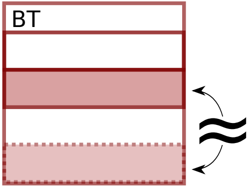
Joshi, Phansalkar, Eeckhout

Remove similar instruction blocks



Simpoint: Sherwood, Perelman, Calder

What can be improved?



Application subsetting

- Coarse grained: less similarity, less accuracy

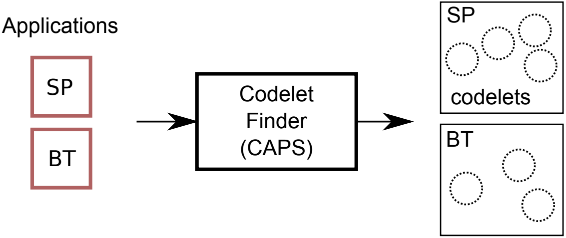
Instruction block subsetting

- Not portable, requires a simulator
- Cannot evaluate compilers

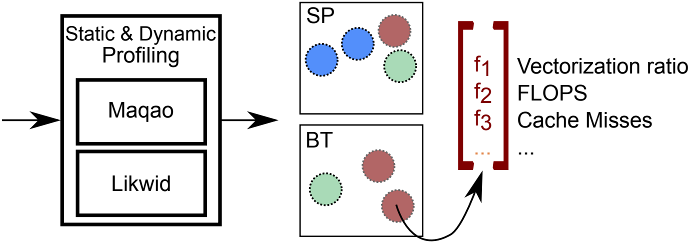
Source Code Subsetting

- Subset fine-grained source code fragments
  - Fine grained
  - Can be recompiled and executed on multiple architectures
- Codelets

Our Approach

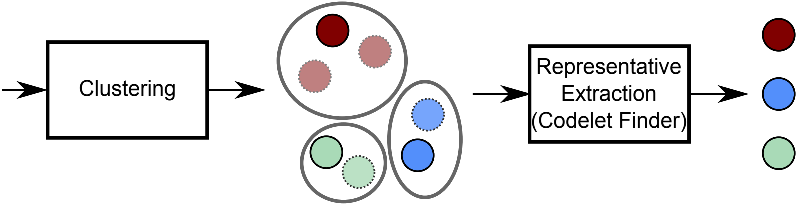


Step A: Detect codelets

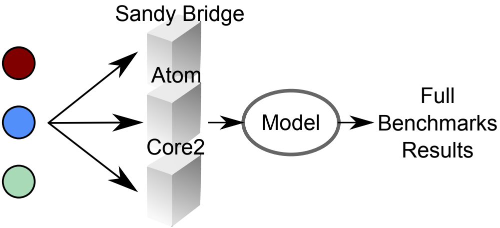


Step B: Build profile on a reference system

Our Approach



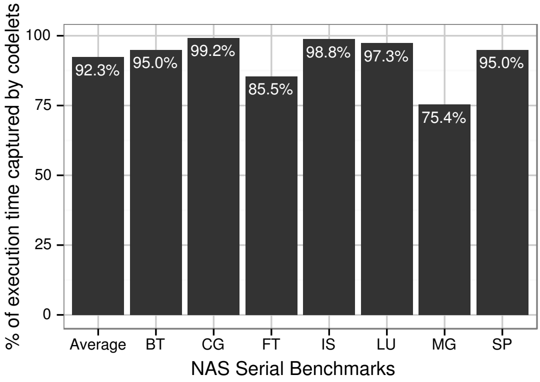
Step C: Cluster similar codelets **Step D**: Extract representative set



Step E: Benchmark representatives

Breaking the Application into Codelets

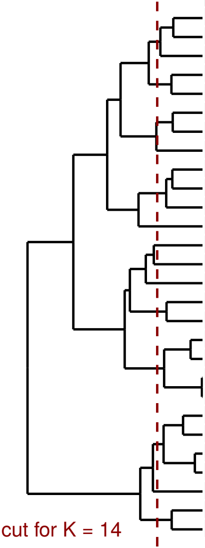
- Codelet: source code fragment
  - Functions: too big, mixes different computation patterns
  - Innerloops: too small, hard to warmup and to measure
  - Outerloops (sweetspot)
- Capture most of the performance in HPC applications



Profiling and Clustering

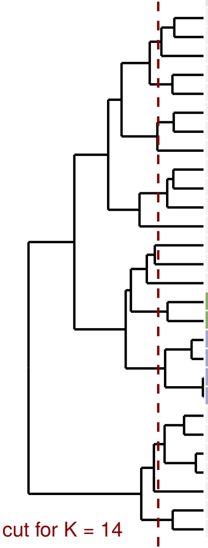
- Automatically group similar codelets
  - Profile codelets on a *reference* system
  - Memory/Cache bandwidth, Instruction mix, Vectorization, ...
- Cluster codelets using feature distance
- We expect that:
  - Clusters capture similar computation patterns
  - Clusters react similarly to architecture change

Clustering NR Codelets



Codelet	Computation Pattern
toeplz_1	DP: 2 simultaneous reductions
rstrct_29	DP: MG Laplacian fine to coarse mesh transition
mprove_8	MP: Dense Matrix x vector product
toeplz_4	DP: Vector multiply in asc./desc. order
realft_4	DP: FFT butterfly computation
toeplz_3	DP: 3 simultaneous reductions
svbksb_3	SP: Dense Matrix x vector product
lop_13	DP: Laplacian finite difference constant coefficients
toeplz_2	DP: Vector multiply element wise in asc./desc. order
four1_2	MP: First step FFT
tridag_2	DP: First order recurrence
tridag_1	DP: First order recurrence
ludcmp_4	SP: Dot product over lower half square matrix
hqr_15	SP: Addition on the diagonal elements of a matrix
relax2_26	DP: Red Black Sweeps Laplacian operator
svdcmp_14	DP: Vector divide element wise
svdcmp_13	DP: Norm + Vector divide
hqr_13	DP: Sum of the absolute values of a matrix column
hqr_12_sq	SP: Sum of a square matrix
jacobi_5	SP: Sum of the upper half of a square matrix
hqr_12	SP: Sum of the lower half of a square matrix
svdcmp_11	DP: Multiplying a matrix row by a scalar
elmhes_11	DP: Linear combination of matrix rows
mprove_9	DP: Substracting a vector with a vector
matadd_16	DP: Sum of two square matrices element wise
svdcmp_6	DP: Sum of the absolute values of a matrix row
elmhes_10	DP: Linear combination of matrix columns
balanc_3	DP: Vector multiply element wise

Clustering NR Codelets



Codelet	Computation Pattern
toeplz_1	
rstrct_29	
mprove_8	
toeplz_4	
realft_4	
toeplz_3	
svbksb_3	
lop_13	
toeplz_2	
four1_2	
tridag_2	
tridag_1	
ludcmp_4	
hqr_15	
relax2_26	
svdcmp_14	DP: Vector divide element wise
svdcmp_13	DP: Norm + Vector divide
hqr_13	DP: Sum of the absolute values of a matrix column
hqr_12_sq	SP: Sum of a square matrix
jacobi_5	SP: Sum of the upper half of a square matrix
hqr_12	SP: Sum of the lower half of a square matrix
svdcmp_11	
elmhes_11	
mprove_9	
matadd_16	
svdcmp_6	
elmhes_10	
balanc_3	

Similar computation patterns

Long Latency Operations

Reduction Sums

Capturing Architecture Change

Nehalem (Ref)
Freq: 1.86 GHz
LLC: 12 MB

Core 2
→ 2.93 GHz
→ 3 MB

Atom
→ 1.66 GHz
→ 1 MB

LU/erhs.f : 49

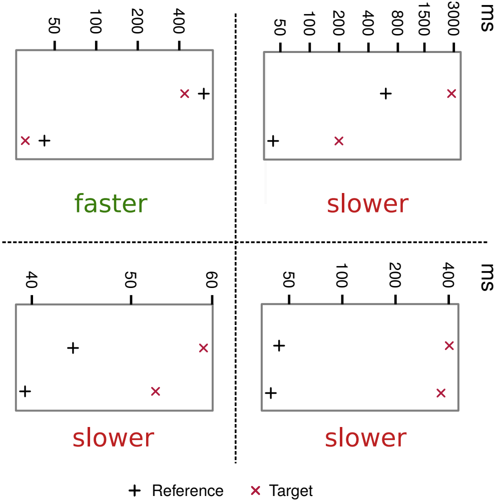
FT/appft.f : 45

Cluster A: triple-nested high latency operations (div and exp)

BT/rhs.f : 266

SP/rhs.f : 275

Cluster B: stencil on five planes (memory bound)



Same Cluster = Same Speedup

Nehalem (Ref)
Freq: 1.86 GHz
LLC: 12 MB

Core 2
→ 2.93 GHz
→ 3 MB

Atom
→ 1.66 GHz
→ 1 MB

LU/erhs.f : 49

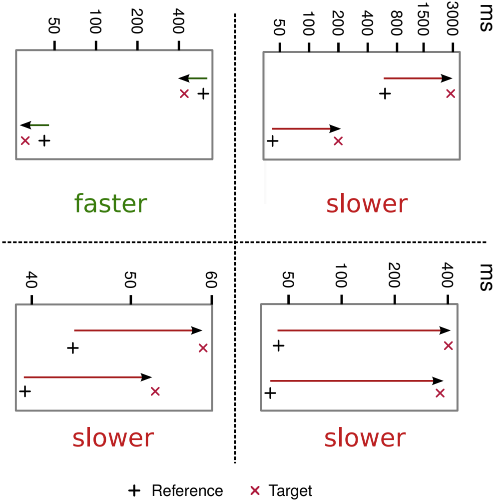
FT/appft.f : 45

Cluster A: triple-nested high latency operations (div and exp)

BT/rhs.f : 266

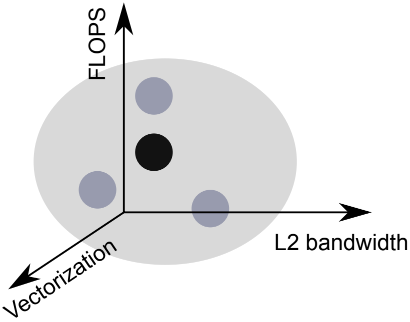
SP/rhs.f : 275

Cluster B: stencil on five planes (memory bound)



Representative Selection

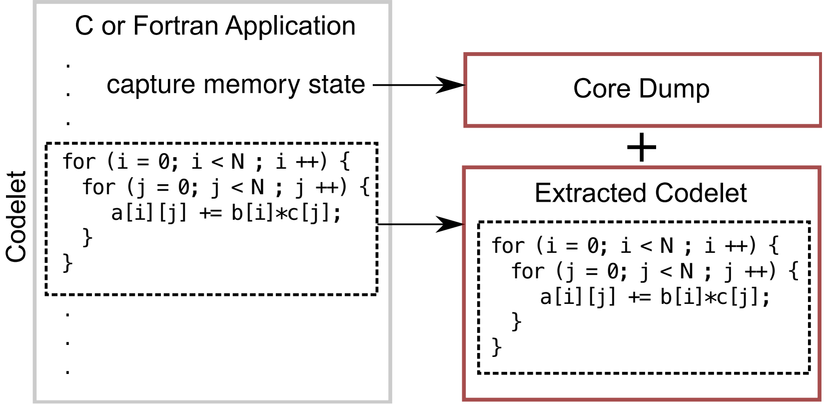
Choose central codelet as representative



- Prediction model: Codelets from the same cluster have the same speedup when changing architectures

Representative Extraction: Codelet Finder

- Extract representatives as standalone microbenchmarks
- Can be recompiled and run outside of the original application



Is Source-code Isolation Viable for Performance Characterization? *PSTI'13*

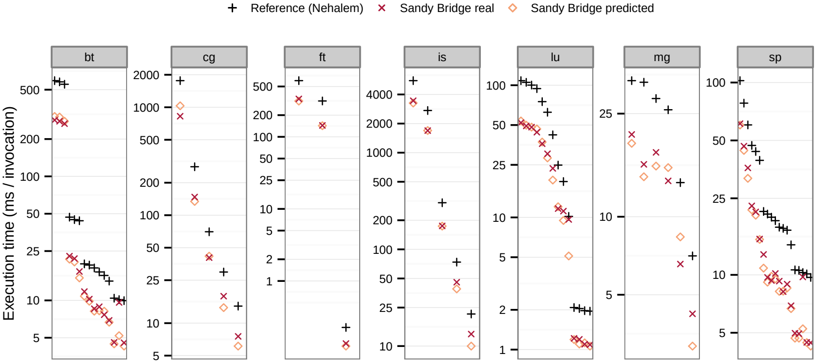
Validation

- Trained and selected feature set on Numerical Recipes + Atom + Sandy Bridge
- Validated approach on NAS Serial and a new architecture, Core 2

	Reference	**Target**		
	Nehalem	Atom	Core 2	Sandy Bridge
CPU	L5609	D510	E7500	E31240
Frequency (GHz)	1.86	1.66	2.93	3.30
Cores	4	2	2	4
L1 cache (KB)	4×64	2×56	2×64	4×64
L2 cache (KB)	4×256	2×512	3 MB	4×256
L3 cache (MB)	12	-	-	8
Ram (GB)	8	4	4	6

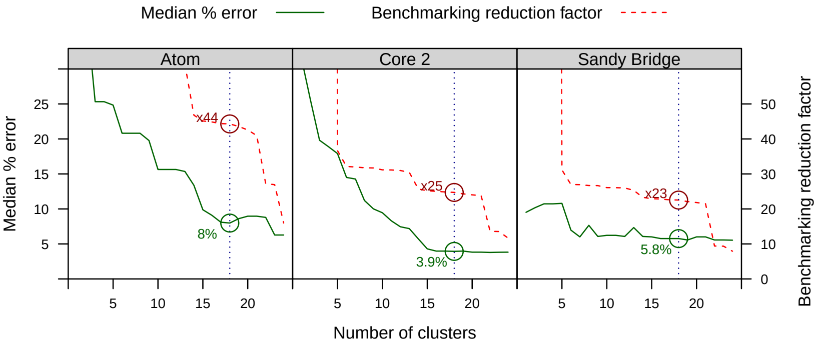
Table : Test architectures.

NAS results



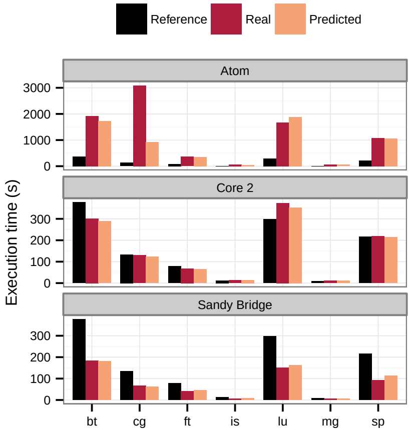
- 18 representatives
- 23 times faster benchmark
- 5.8% median error

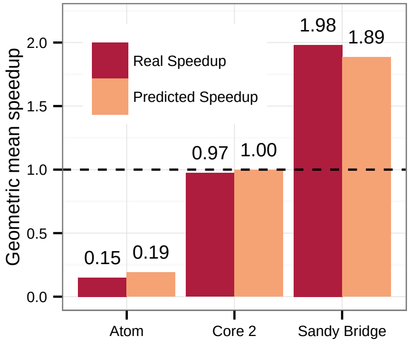
Tradeoff Reduction / Accuracy (NAS)



- More clusters:
  - ↗ accuracy
  - ↗ benchmarking cost
- Automatically select good tradeoff using Elbow method

Overall results (NAS)





- Accurately evaluate architectures
- Choose the best architecture-benchmark pairs

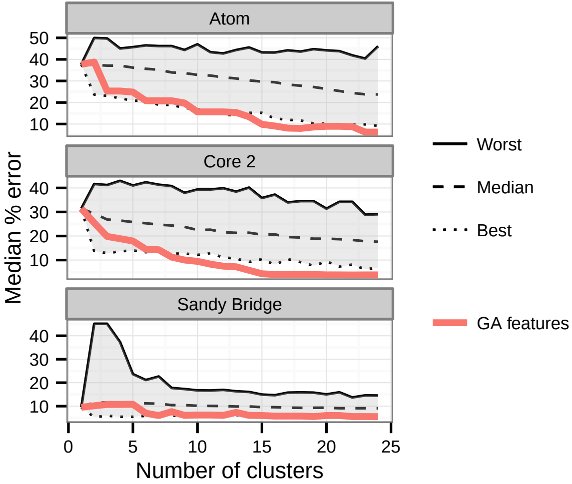
Conclusion

- Take advantage of source loops redundancies to reduce benchmarking time
  - Generate portable compressed benchmarks
  - Accurate ($< 10\%$) and Faster ($> \times 23$)
- Applications
  - System Selection (this)
  - Fast compiler performance regression tests
  - Iterative Compilation
- `http://benchmark-subsetting.github.io/fgbs/`
  - data and analysis code available as a reproducible IPython notebook

Thanks for your attention!

Feature Selection

- Genetic Algorithm: find best set of features on Numerical Recipes + Atom + Sandy Bridge
- The feature set is still among the best on NAS



Reduction Factor Breakdown

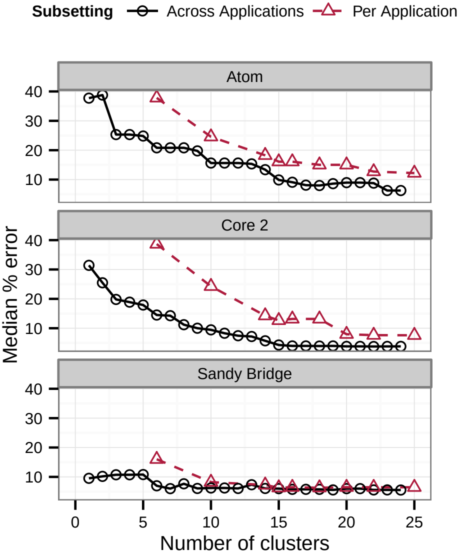
Reduction	Total	Reduced invocations	Clustering
Atom	44.3	$\times 12$	$\times 3.7$
Core 2	24.7	$\times 8.7$	$\times 2.8$
Sandy Bridge	22.5	$\times 6.3$	$\times 3.6$

Table : Benchmarking reduction factor breakdown with 18 representatives.

Same working set?

- NAS: regular codes.
  - Only 19% of codelets have different behavior accross invocations.
  - Detect *ill-behaved codelets*. Exclude them from representatives.
- SPEC: different working set per invocation.
  - Ongoing: Cluster codelets across working sets

Across Applications Similarities



Profiling Features