CERE: LLVM based Codelet Extractor and REplayer for Piecewise Benchmarking and Optimization

Chadi Akel, P. de Oliveira Castro, M. Popov, E. Petit, W. Jalby

University of Versailles – Exascale Computing Research

EuroLLVM 2016

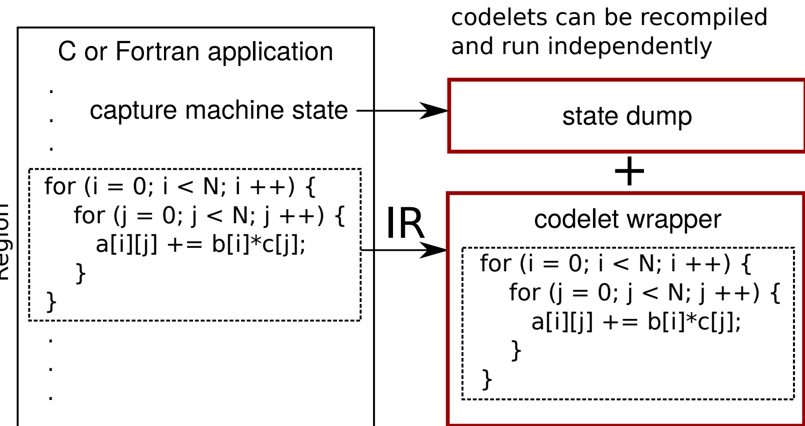
Motivation

- Finding best application parameters is a costly iterative process

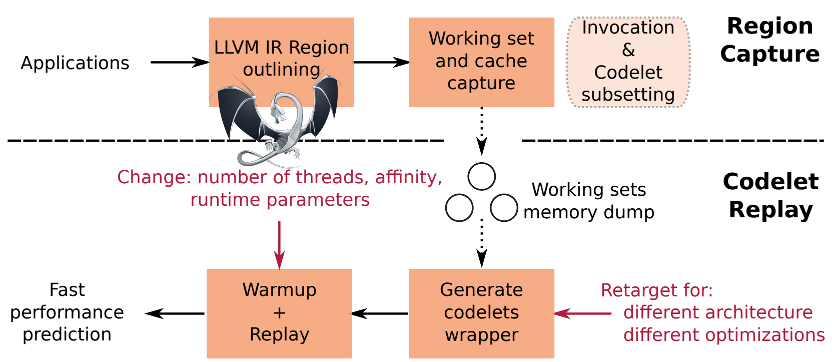
- Codelet Extractor and REplayer
  - Break an application into standalone codelets
  - Make costly analysis affordable:
    - Focus on single regions instead of whole applications
    - Run a single representative by clustering similar codelets

Codelet Extraction

- Extract codelets as standalone microbenchmarks



CERE Workflow



CERE can extract codelets from:

- Hot Loops
- OpenMP non-nested parallel regions [Popov et al. 2015]

Outline

Extracting and Replaying Codelets
- Faithful
- Retargetable

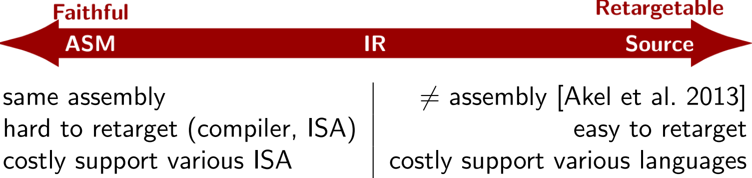
Applications
- Architecture selection
- Compiler flags tuning
- Scalability prediction

Demo
- Capture and replay in NAS BT
- Simple flag replay for NAS FT

Conclusion

Capturing codelets at Intermediate Representation

- Faithful: behaves similarly to the original region
- Retargetable: modify runtime and compilation parameters



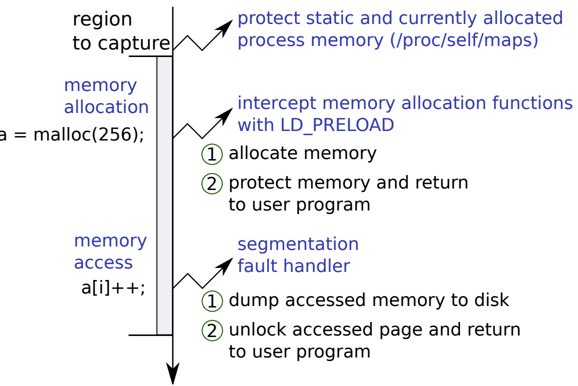
- LLVM Intermediate Representation is a good tradeoff

Faithful capture

- Required for semantically accurate replay:
  - Register state
  - Memory state
  - OS state: locks, file descriptors, sockets
- No support for OS state except for locks. CERE captures fully from userland: no kernel modules required.
- Required for performance accurate replay:
  - Preserve code generation
  - Cache state
  - NUMA ownership
  - Other warmup state (eg. branch predictor)

Faithful capture: memory

Capture access at page granularity: coarse but fast



- Small dump footprint: only touched pages are saved
- Warmup cache: replay trace of most recently touched pages
- NUMA: detect first touch of each page

Outline

Extracting and Replaying Codelets
- Faithful
- Retargetable

Applications
- Architecture selection
- Compiler flags tuning
- Scalability prediction

Demo
- Capture and replay in NAS BT
- Simple flag replay for NAS FT

Conclusion

Selecting Representative Codelets

- Key Idea: Applications have redundancies
  - Same codelet called multiple times
  - Codelets sharing similar performance signatures
- Detect redundancies and keep only one representative

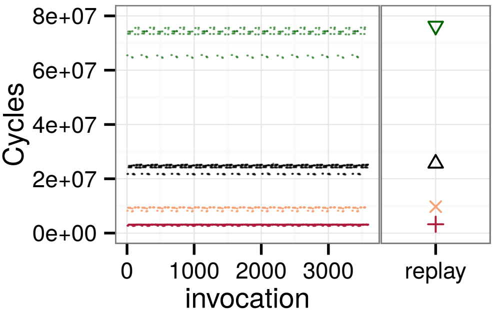
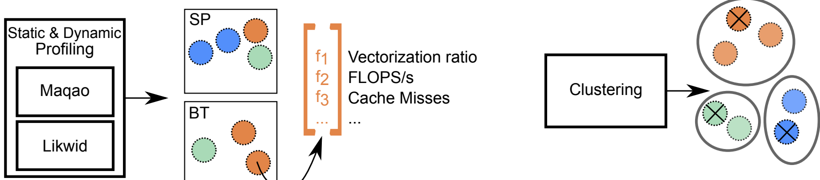


Figure : SPEC tonto `make_ft@shell2.F90:1133` execution trace. 90% of NAS codelets can be reduced to four or less representatives.

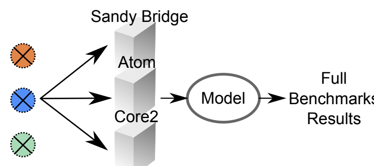
Performance Signature Clustering



Step A: Perform static and dynamic analysis on a reference architecture to capture codelet's feature vectors.

Step B: Using the proximity between feature vectors we cluster similar codelets and select one representative per cluster.

Step C: CERE extracts the representatives as standalone codelets. A model extrapolates full benchmark results.



[Oliveira Castro et al. 2014]

Codelet Based Architecture Selection

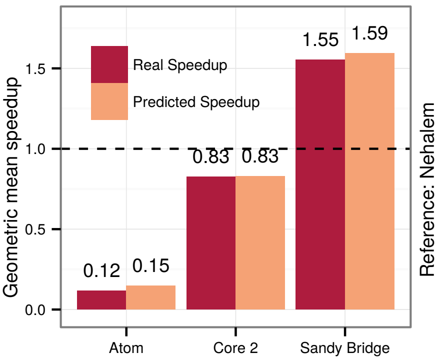


Figure : Benchmarking NAS serial on three architectures

- real: speedup when benchmarking original applications
- predicted: speedup predicted with representative codelets
- CERE 31× cheaper than running the full benchmarks.

Autotuning LLVM middle-end optimizations

- LLVM middle-end offers more than 50 optimization passes.
- Codelet replay enable per-region fast optimization tuning.

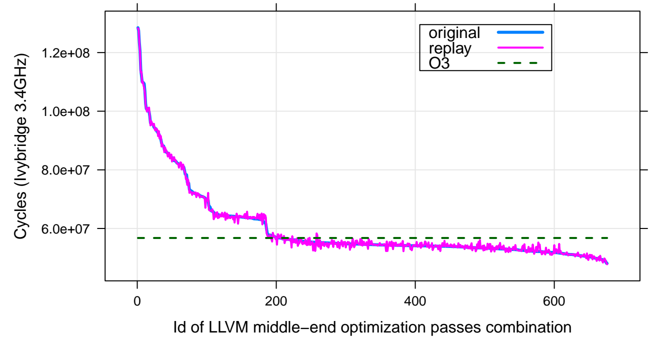
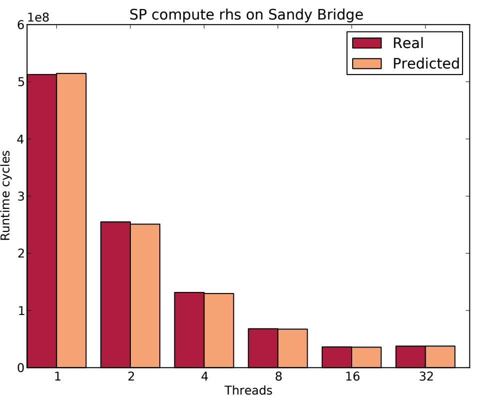


Figure : NAS SP ysolve codelet. 1000 schedules of random passes combinations explored based on O3 passes.

CERE 149× cheaper than running the full benchmark
(27× cheaper when tuning codelets covering 75% of SP)

Fast Scalability Benchmarking with OpenMP Codelets



	Core2	Nehalem	Sandy Bridge	Ivy Bridge
Accuracy	98.2%	97.1%	92.6%	97.2%
Acceleration	× 25.2	× 27.4	× 23.7	× 23.7

Figure : Varying thread number at replay in SP and average results over OMP NAS [Popov et al. 2015]

Outline

Extracting and Replaying Codelets
- Faithful
- Retargetable

Applications
- Architecture selection
- Compiler flags tuning
- Scalability prediction

Demo
- Capture and replay in NAS BT
- Simple flag replay for NAS FT

Conclusion

Conclusion

- CERE breaks an application into faithful and retargetable codelets
- Piece-wise autotuning:
  - Different architecture
  - Compiler optimizations
  - Scalability
  - Other exploration costly analysis ?
- Limitations:
  - No support for codelets performing IO (OS state not captured)
  - Cannot explore source-level optimizations
  - Tied to LLVM
- Full accuracy reports on NAS and SPEC'06 FP available at `benchmark-subsetting.github.io/cere/#Reports`

Thanks for your attention!

https://benchmark-subsetting.github.io/cere/

distributed under the LGPLv3

FT

Exec Time (%)	Codelet Name	Error (%)
5.7	APPFT_APPFT__40	5.9
▼ 68.0	appft_appft__1	1.3
▼ 28.6	FFT3D_FFTXYZ__152	0.1
70.3	fft3d_swarztrauber__27	4.2
5.1	AUXFNCT_EVOLVE__167	0.6
▸ 28.6	FFT3D_FFTXYZ__112	1.4
▸ 27.4	FFT3D_FFTXYZ__137	0.0
2.2	auxfnct_compute_initial_conditions__148	25.2

Exec Time : 1.257131e+10 cycles

Matching : 95.4%

100
75
50
ne Cumulated (%)

Clustering Code

5e+05
4e+05
3e+05
2e+05
Cycles
0
10000
20000
30000
invocation

__invivo__fft3d_swarztrauber__27

Real time: 8.847104e+09 -- Predicted time: 9.241310e+09

Callcount: 32768

Invocation	Cluster	Part	Invitro (cycles)	Invivo (cycles)	Error (%)
14276	1	7156.8835542227	4.192960e+05	4.134800e+05	1.3
4180	2	8195.06987453204	3.568270e+05	3.344280e+05	6.2
17561	3	1029.32478321909	5.200480e+05	4.843600e+05	6.8

Bibliography I

- Akel, Chadi et al. (2013). "Is Source-code Isolation Viable for Performance Characterization?" In: *42nd International Conference on Parallel Processing Workshops*. IEEE.
- Gao, Xiaofeng et al. (2005). "Reducing overheads for acquiring dynamic memory traces". In: *Workload Characterization Symposium, 2005. Proceedings of the IEEE International*. IEEE, pp. 46–55.
- Liao, Chunhua et al. (2010). "Effective source-to-source outlining to support whole program empirical optimization". In: *Languages and Compilers for Parallel Computing*. Springer, pp. 308–322.
- Oliveira Castro, Pablo de et al. (2014). "Fine-grained Benchmark Subsetting for System Selection". In: *Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization*. ACM, p. 132.
- Popov, Mihail et al. (2015). "PCERE: Fine-grained Parallel Benchmark Decomposition for Scalability Prediction". In: *Proceedings of the 29th IEEE International Parallel and Distributed Processing Symposium IPDPS*. IEEE.

Retargetable replay: register state

- Issue: Register state is non-portable between architectures.
- Solution: capture at a function call boundary
  - No shared state through registers except function arguments
  - Get arguments directly through portable IR code
- Register agnostic capture

- Portable across Atom, Core 2, Haswell, Ivybridge, Nehalem, Sandybridge
- Preliminar portability tests between x86 and ARM 32 bits

Retargetable replay: outlining regions

Step 1: Outline the region to capture using *CodeExtractor* pass

```
original:
  %0 = load i32* %i, align 4
  %1 = load i32* %s.addr, align 4
  %cmp = icmp slt i32 %0, %1
  br i1 %cmp, ; loop branch here
  label %for.body,
  label %for.exitStub ...
```

```
define internal void @outlined(
  i32* %i, i32* %s.addr,
  i32** %a.addr) {
  call void @start_capture(i32* %i,
    i32* %s.addr, i32** %a.addr)
  %0 = load i32* %i, align 4
    ...
  ret void
}
original:
  call void @outlined(i32* %i,
  i32* %s.addr, i32** %a.addr)
```

Step 2: Call `start_capture` just after the function call
Step 3: At replay, reinlining and variable cloning [Liao et al. 2010]
steps ensure that the compilation context is close to original

Comparison to other Code Isolating tools

	CERE	Code Isolator	Astex	Codelet Finder	SimPoint
Support					
Language	C(++), Fortran, ...	Fortran	C, Fortran	C(++), Fortran	assembly
Extraction	IR	source	source	source	assembly
Indirections	yes	no	no	yes	yes
Replay					
Simulator	yes	yes	yes	yes	yes
Hardware	yes	yes	yes	yes	no
Reduction					
Capture size	reduced	reduced	reduced	full	-
Instances	yes	manual	manual	manual	yes
Code sign.	yes	no	no	no	yes

Accuracy Summary: NAS SER

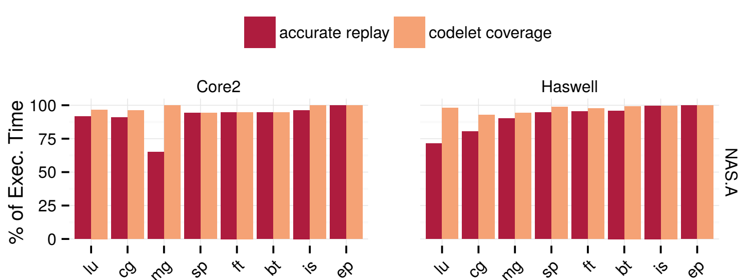


Figure : The Coverage the percentage of the execution time captured by codelets. The Accurate Replay is the percentage of execution time replayed with an error less than 15%.

Accuracy Summary: SPEC FP 2006

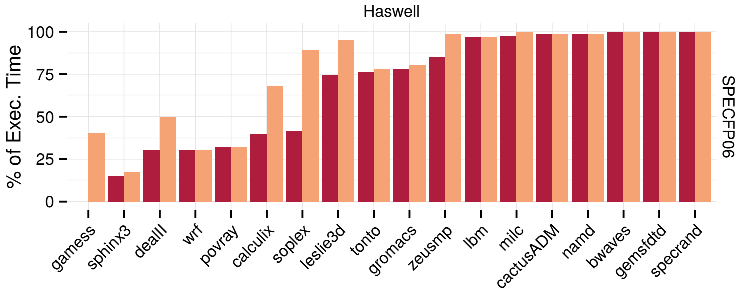


Figure : The Coverage is the percentage of the execution time captured by codelets. The Accurate Replay is the percentage of execution time replayed with an error less than 15%.

- low coverage (sphinx3, wrf, povray): $<$ 2000 cycles or IO
- low matching (soplex, calculix, gamess): warmup “bugs”

CERE page capture dump size

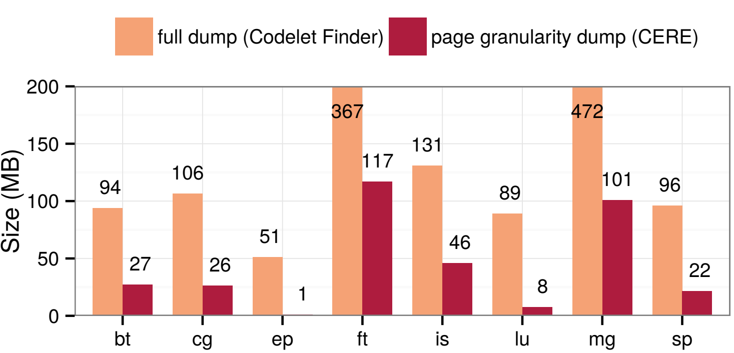


Figure : Comparison between the page capture and full dump size on NAS.A benchmarks. CERE page granularity dump only contains the pages accessed by a codelet. Therefore it is much smaller than a full memory dump.

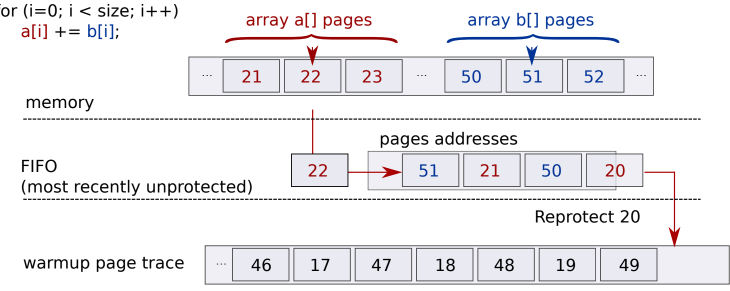
CERE page capture overhead

CERE page capture is coarser but faster

	CERE	ATOM 3.25	PIN 1.71	Dyninst 4.0
cg.a	19.4	98.82	222.67	896.86
ft.a	24.1	44.22	127.64	1054.70
lu.a	62.4	80.72	153.46	301.4
mg.a	8.9	107.69	168.61	989.53
sp.a	73.2	67.56	93.04	203.66

Slowdown of a full capture run against the original application run. (takes into account the cost of writing the memory dumps and logs to disk and of tracing the page accesses during the whole execution.). We compare to the overhead of memory tracing tools as reported by [Gao et al. 2005].

CERE cache warmup

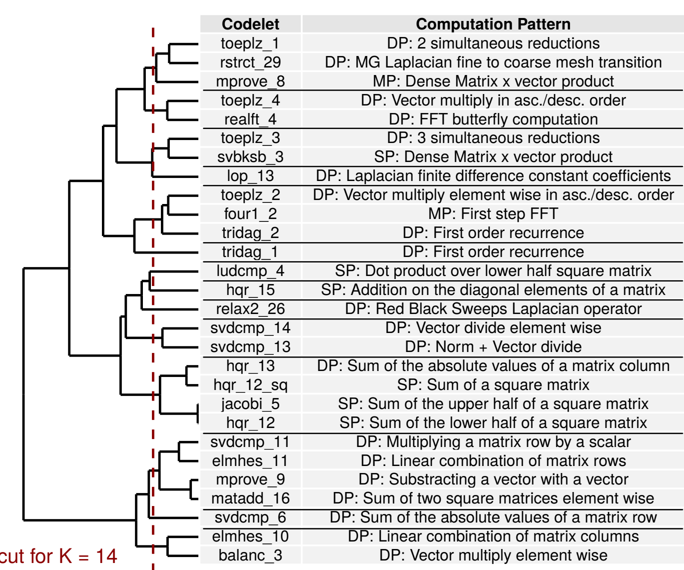


Test architectures

	Atom	Core 2	Nehalem	Sandy Bridge	Ivy Bridge	Haswell
CPU	D510	E7500	L5609	E31240	i7-3770	i7-4770
Frequency (GHz)	1.66	2.93	1.86	3.30	3.40	3.40
Cores	2	2	4	4	4	4
L1 cache (KB)	2×56	2×64	4×64	4×64	4×64	4×64
L2 cache (KB)	2×512	3 MB	4×256	4×256	4×256	4×256
L3 cache (MB)	-	-	12	8	8	8
Ram (GB)	4	4	8	6	16	16

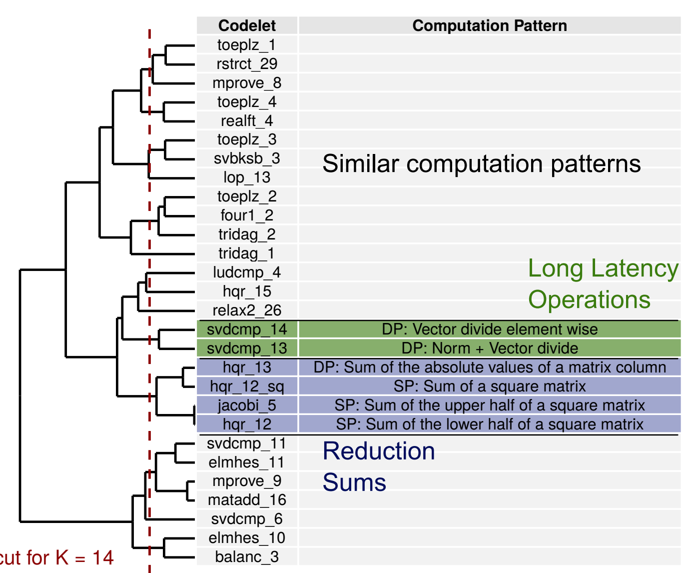
32 bits portability test: ARM1176JZF-S on a Raspberry Pi Model B+

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



Capturing Architecture Change

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)

50 100 200 400

faster

50 100 200 400 800 1500 3000 ms

slower

BT/rhs.f : 266

SP/rhs.f : 275

Cluster B: stencil on five planes (memory bound)

40 50 60

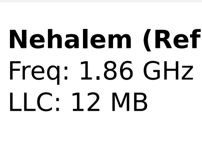
slower

50 100 200 400 ms

slower

+ Reference × Target

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)

50 100 200 400

faster

50 100 200 400 800 1500 3000 ms

slower

BT/rhs.f : 266

SP/rhs.f : 275

Cluster B: stencil on five planes (memory bound)

40 50 60

slower

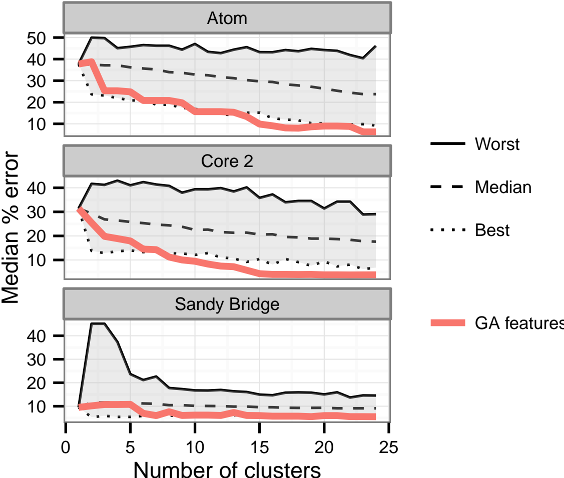
50 100 200 400 ms

slower

+ Reference × Target

Feature Selection

- Genetic Algorithm: train on Numerical Recipes + Atom + Sandy Bridge
- Validated on NAS + Core 2
- The feature set is still among the best on NAS

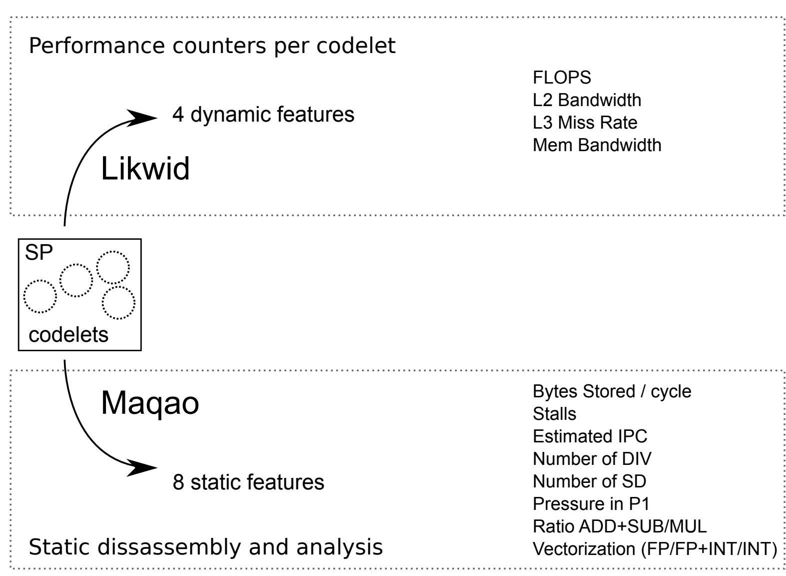


Reduction Factor Breakdown

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.

Profiling Features



Clang OpenMP front-end