Dynamic Metrics for Java

Bruno Dufour, Karel Driesen, Laurie Hendren and Clark Verbrugge

Sable Research Group

McGill University

Outline

- Introduction and motivation
- Desirable qualities for dynamic metrics
- Kinds of dynamic metrics
- Examples of dynamic metrics
- Using dynamic metrics for optimization
- *J Framework
- Related work
- Future work & Conclusion

Motivation

- Compiler techniques are aimed at programs which are
  - numeric
  - memory-intensive
  - concurrent
  - pointer-intensive
  - . . .
- Necessary to assess the *relevance* of optimizations

Motivation

- Compiler techniques are aimed at programs which are
  - numeric
  - memory-intensive
  - concurrent
  - pointer-intensive
  - . . .
- Necessary to assess the *relevance* of optimizations
- **Problem**: How to determine which programs fit into these categories?

Motivation (2)

- **Answer #1**: Use a programmer's intuition

Motivation (2)

- **Answer #1**: Use a programmer's intuition
- **Answer #2**: Measure characteristics of the software

Motivation (2)

- **Answer #1**: Use a programmer's intuition
- **Answer #2**: Measure characteristics of the software
  - No well-established way of performing categorization based on measures

Motivation (2)

- **Answer #1**: Use a programmer's intuition
- **Answer #2**: Measure characteristics of the software
  - No well-established way of performing categorization based on measures
  - Measurements are often performed *statically*

Motivation (2)

- **Answer #1**: Use a programmer's intuition
- **Answer #2**: Measure characteristics of the software
  - No well-established way of performing categorization based on measures
  - Measurements are often performed *statically*
- **Answer #3**: Quantify *dynamic* behaviour of the software (dynamic metrics)

Goals

- Measure relevant runtime properties of programs
- Provide a methodology to compute dynamic metrics
- Provide a *concise* and *informative* set of metrics
- Provide a database of dynamic metrics

Desired Qualities for Dynamic Metrics

- Dynamic
- Unambiguous
- Robust
- Discriminating
- Machine-independent

} Not necessarily achieved

Desirable Qualities – Dynamic

"Dynamic metrics should measure an aspect of a program that can only be obtained by running it."

- Dynamic metrics should not be affected by dead code

Desirable Qualities – Unambiguous

"Ambiguous metric definitions lead to unusable metrics."

- e.g. Lines of Code (LOC) as a (static) measure of program size
  - comments?
  - blank lines?
  - programming style?

Desirable Qualities – Robust

"A small *relevant* change in behaviour should cause a correspondingly small change in the resulting metric."

- Example: # of *different* executed bytecode instructions instead of # of executed bytecode instructions
  - Example: Bubble Sort
  - Double input, quadruple # of executed bytecode instructions

Desirable Qualities – Discriminating

"A large *relevant* change in behaviour should cause a correspondingly large change in the resulting metric."

- Dual to robustness
  - Constant value metric is robust, but useless
- Example: # of loaded classes (typically 270-290)
  - **not** discriminating

Desirable Qualities – Platform-Independent

"Metric values should be the same if measured on different platforms."

- Very difficult to achieve
- Dictates some choices:
  - Time measurements in executed bytecode instructions instead of seconds
- Relevance depends on the characteristic to be quantified

Kinds of Dynamic Metrics

- Usual metrics belong to three categories:
  - Value
  - Percentile
  - Bin

Value metrics

- One value answers
- Typical examples
  - maximum
  - average
- Most suitable for:
  - observing differences before and after optimizations

Percentile metrics

- Pair consisting of percentage and threshold
- Typical example:
  - 15% of the allocation sites are responsible for 90% of the total allocated bytes
- Most suitable for:
  - determining how evenly distributed contributions are for a program characteristic

Bin metrics

- One value per subdivision of sample space
- Typical example:
  - Polymorphism (# of receivers/targets)

Bin #	# of receivers	% of call sites
1	1	73%
2	2	11%
3	$\geq$3	14%

- Most suitable for:
  - providing measurements in accordance with specific optimization cases

Naming scheme

`category.name.kind`

- **Category**: logical group of the metric
- **Name**: name of the metric. Prefix `app` denotes the application-only version (i.e. excluding libraries)
- **kind**: either `value`, `percentile` or `bin`

Benchmark Programs

- **EMPTY**: Empty program
- **COEFF**: Computes polynomial approximation of order 0 to n for a set of points
- **COMP**: SPECjvm98 compress, performs Lempel-Ziv compression
- **JAVAC**: SPECjvm98 javac, a java compiler

Program Size and Structure

- **size.[app]run.value**:

$$\#\ touched\ bytecode\ insts$$

Metric	EMPTY	COEFF	COMP	JAVAC
size.run.value	7343	12880	14514	37830
(increase)		175%	113%	261%
size.appRun.value	0	975	5084	26267
(increase)		n/a	521%	517%

Program Size and Structure

- **size.[app]run.value**:

$$\#\ touched\ bytecode\ insts$$

Metric	EMPTY	COEFF	COMP	JAVAC
size.run.value	**7343**	**12880**	**14514**	**37830**
(increase)		175%	113%	261%
size.appRun.value	0	975	5084	26267
(increase)		n/a	521%	517%

Program Size and Structure

- **size.[app]run.value**:

$$\#\ touched\ bytecode\ insts$$

Metric	EMPTY	COEFF	COMP	JAVAC
size.run.value	7343	12880	14514	37830
(increase)		**175%**	**113%**	**261%**
size.appRun.value	0	975	5084	26267
(increase)		n/a	521%	517%

Program Size and Structure

- **size.[app]run.value**:

$$\#\ touched\ bytecode\ insts$$

Metric	EMPTY	COEFF	COMP	JAVAC
size.run.value	7343	12880	14514	37830
(increase)		175%	113%	261%
size.appRun.value	**0**	**975**	**5084**	**26267**
(increase)		n/a	521%	517%

Program Size and Structure

- **size.[app]run.value**:

$$\#\ touched\ bytecode\ insts$$

Metric	EMPTY	COEFF	COMP	JAVAC
size.run.value	7343	12880	14514	37830
(increase)		175%	113%	261%
size.appRun.value	0	975	5084	26267
(increase)		**n/a**	**521%**	**517%**

Data Structures

- **data.[app]arrayDensity.value**:

$$\frac{\#\ array\ bytecode\ insts}{\#\ executed\ bytecode\ instructions} \text{x} 1000$$

Metric	EMPTY	COEFF	COMP	JAVAC
data.arrayDensity.value	73	151	52	38
data.appArrayDensity.value	n/a	160	52	15

- **pointer.[app]refFieldAccessDensity.value**:

$$\frac{\#\ reference\ field\ accesses}{\#\ executed\ bytecode\ instructions} \text{x} 1000$$

Metric	EMPTY	COEFF	COMP	JAVAC
data.refFieldAccessDensity.value	6	31	44	50
data.appRefFieldAccessDensity.value	n/a	33	44	76

Data Structures

- **data.[app]arrayDensity.value**:

$$\frac{\#\ array\ bytecode\ insts}{\#\ executed\ bytecode\ instructions} \text{x} 1000$$

Metric	EMPTY	COEFF	COMP	JAVAC
data.arrayDensity.value	**73**	**151**	**52**	**38**
data.appArrayDensity.value	n/a	160	52	15

- **pointer.[app]refFieldAccessDensity.value**:

$$\frac{\#\ reference\ field\ accesses}{\#\ executed\ bytecode\ instructions} \text{x} 1000$$

Metric	EMPTY	COEFF	COMP	JAVAC
data.refFieldAccessDensity.value	6	31	44	50
data.appRefFieldAccessDensity.value	n/a	33	44	76

Data Structures

- **data.[app]arrayDensity.value**:

$$\frac{\#\ array\ bytecode\ insts}{\#\ executed\ bytecode\ instructions} \text{x} 1000$$

Metric	EMPTY	COEFF	COMP	JAVAC
data.arrayDensity.value	73	151	52	38
data.appArrayDensity.value	**n/a**	**160**	**52**	**15**

- **pointer.[app]refFieldAccessDensity.value**:

$$\frac{\#\ reference\ field\ accesses}{\#\ executed\ bytecode\ instructions} \text{x} 1000$$

Metric	EMPTY	COEFF	COMP	JAVAC
data.refFieldAccessDensity.value	6	31	44	50
data.appRefFieldAccessDensity.value	n/a	33	44	76

Data Structures

- **data.[app]arrayDensity.value**:

$$\frac{\#\ array\ bytecode\ insts}{\#\ executed\ bytecode\ instructions} \text{x}1000$$

Metric	EMPTY	COEFF	COMP	JAVAC
data.arrayDensity.value	73	151	52	38
data.appArrayDensity.value	n/a	160	52	15

- **pointer.[app]refFieldAccessDensity.value**:

$$\frac{\#\ reference\ field\ accesses}{\#\ executed\ bytecode\ instructions} \text{x}1000$$

Metric	EMPTY	COEFF	COMP	JAVAC
data.refFieldAccessDensity.value	6	31	44	50
data.appRefFieldAccessDensity.value	n/a	33	44	76

Data Structures

- **data.[app]arrayDensity.value**:

$$\frac{\#\ array\ bytecode\ insts}{\#\ executed\ bytecode\ instructions} \text{x} 1000$$

Metric	EMPTY	COEFF	COMP	JAVAC
data.arrayDensity.value	73	151	52	38
data.appArrayDensity.value	n/a	160	52	15

- **pointer.[app]refFieldAccessDensity.value**:

$$\frac{\#\ reference\ field\ accesses}{\#\ executed\ bytecode\ instructions} \text{x} 1000$$

Metric	EMPTY	COEFF	COMP	JAVAC
data.refFieldAccessDensity.value	6	31	44	50
data.appRefFieldAccessDensity.value	n/a	33	44	76

Data Structures

- **data.[app]arrayDensity.value**:

$$\frac{\#\ array\ bytecode\ insts}{\#\ executed\ bytecode\ instructions} \text{x} 1000$$

Metric	EMPTY	COEFF	COMP	JAVAC
data.arrayDensity.value	73	151	52	38
data.appArrayDensity.value	n/a	160	52	15

- **pointer.[app]refFieldAccessDensity.value**:

$$\frac{\#\ reference\ field\ accesses}{\#\ executed\ bytecode\ instructions} \text{x} 1000$$

Metric	EMPTY	COEFF	COMP	JAVAC
data.refFieldAccessDensity.value	**6**	**31**	**44**	**50**
data.appRefFieldAccessDensity.value	n/a	33	44	76

Data Structures

- **data.[app]arrayDensity.value**:

$$\frac{\#\ array\ bytecode\ insts}{\#\ executed\ bytecode\ instructions} \text{x} 1000$$

Metric	EMPTY	COEFF	COMP	JAVAC
data.arrayDensity.value	73	151	52	38
data.appArrayDensity.value	n/a	160	52	15

- **pointer.[app]refFieldAccessDensity.value**:

$$\frac{\#\ reference\ field\ accesses}{\#\ executed\ bytecode\ instructions} \text{x} 1000$$

Metric	EMPTY	COEFF	COMP	JAVAC
data.refFieldAccessDensity.value	6	31	44	50
data.appRefFieldAccessDensity.value	**n/a**	**33**	**44**	**76**

Polymorphism

- **polymorphism.[app]invokeDensity.value**:

$$\frac{\#\ executed\ \texttt{invokevirtual}\ or\ \texttt{invokeinterface}\ instructions}{\#\ executed\ bytecode\ instructions} \text{x}1000$$

Metric	EMPTY	COEFF	COMP	JAVAC
polymorphism.invokeDensity.value	15	61	17	39
polymorphism.appInvokeDensity.value	n/a	66	17	72

Polymorphism

- **polymorphism.[app]invokeDensity.value**:

$$\frac{\#\ executed\ \texttt{invokevirtual}\ or\ \texttt{invokeinterface}\ instructions}{\#\ executed\ bytecode\ instructions} \text{x}1000$$

Metric	EMPTY	COEFF	COMP	JAVAC
polymorphism.invokeDensity.value	**15**	**61**	**17**	**39**
polymorphism.appInvokeDensity.value	n/a	66	17	72

Polymorphism

- **polymorphism.[app]invokeDensity.value**:

$$\frac{\#\ executed\ \texttt{invokevirtual}\ or\ \texttt{invokeinterface}\ instructions}{\#\ executed\ bytecode\ instructions} \text{x}1000$$

Metric	EMPTY	COEFF	COMP	JAVAC
polymorphism.invokeDensity.value	15	61	17	39
polymorphism.appInvokeDensity.value	**n/a**	**66**	**17**	**72**

Polymorphism (2)

- **polymorphism.[app]targetArity.bin**: measures the percentage of all *call sites* which have 1,2 or $\geq$3 different target methods:

Metric	EMPTY	COEFF	COMP	JAVAC
polymorphism.targetArity.bin(1)	98.4%	98.8%	98.3%	91.2%
polymorphism.targetarity.bin(2)	1.4%	1.0%	1.5%	3.4%
polymorphism.targetarity.bin(3+)	0.2%	0.1%	0.2%	5.4%
polymorphism.apptargetarity.bin(1)	n/a	100.0%	98.1%	89.7%
polymorphism.apptargetarity.bin(2)	n/a	0.0%	1.9%	3.8%
polymorphism.apptargetarity.bin(3+)	n/a	0.0%	0.0%	6.5%

Polymorphism (2)

- **polymorphism.[app]targetArity.bin**: measures the percentage of all *call sites* which have 1,2 or $\geq$3 different target methods:

Metric	EMPTY	COEFF	COMP	JAVAC
polymorphism.targetArity.bin(1)	**98.4%**	**98.8%**	**98.3%**	**91.2%**
polymorphism.targetarity.bin(2)	1.4%	1.0%	1.5%	3.4%
polymorphism.targetarity.bin(3+)	0.2%	0.1%	0.2%	5.4%
polymorphism.apptargetarity.bin(1)	n/a	100.0%	98.1%	89.7%
polymorphism.apptargetarity.bin(2)	n/a	0.0%	1.9%	3.8%
polymorphism.apptargetarity.bin(3+)	n/a	0.0%	0.0%	6.5%

Polymorphism (2)

- **polymorphism.[app]targetArity.bin**: measures the percentage of all *call sites* which have 1,2 or $\geq$3 different target methods:

Metric	EMPTY	COEFF	COMP	JAVAC
polymorphism.targetArity.bin(1)	98.4%	98.8%	98.3%	91.2%
polymorphism.targetarity.bin(2)	1.4%	1.0%	1.5%	3.4%
polymorphism.targetarity.bin(3+)	0.2%	0.1%	0.2%	5.4%
polymorphism.apptargetarity.bin(1)	**n/a**	**100.0%**	**98.1%**	**89.7%**
polymorphism.apptargetarity.bin(2)	n/a	0.0%	1.9%	3.8%
polymorphism.apptargetarity.bin(3+)	n/a	0.0%	0.0%	6.5%

Memory Use

- **memory.byteAllocationDensity.value**:

$$\frac{\#\ allocated\ \ bytes}{\#\ executed\ \ bytecode\ \ instructions} \text{x} 1000$$

Metric	EMPTY	COEFF	COMP	JAVAC
memory.byteAllocationDensity.value	1750	109	11	132

Memory Use

- **memory.byteAllocationDensity.value**:

$$\frac{\#\ allocated\ \ bytes}{\#\ executed\ \ bytecode\ \ instructions} \text{x} 1000$$

Metric	EMPTY	COEFF	COMP	JAVAC
memory.byteAllocationDensity.value	1750	109	11	132

Concurrency and Synchronization

- **concurrency.lock.percentile**:

$$\frac{\#\ locks\ responsible\ for\ 90\%\ of\ locking\ ops}{\#\ touchedlocks} \text{x}100$$

Metric	EMPTY	COEFF	COMP	JAVAC
concurrency.lock.percentile	66.7%	10.3%	56.7%	6.2%

Concurrency and Synchronization

- **concurrency.lock.percentile**:

$$\frac{\#\ locks\ responsible\ for\ 90\%\ of\ locking\ ops}{\#\ touchedlocks} \text{x}100$$

Metric	EMPTY	COEFF	COMP	JAVAC
concurrency.lock.percentile	66.7%	10.3%	56.7%	6.2%

Using Metrics for Optimization

- Benchmark: JOlden Voronoi

Metric	Orig	Inline	PT+CSE
base.executedInstructions.value	445.13 M	287.86 M	282.08 M
size.appRun.value	1008	1449	1425
poly.appInvokeDensity.value	116.17	10.84	11.06
poly.appTargetArity.bin(1)	1	1	1
pointer.appFieldAccessDensity.value	126.7	196.1	177.8

Executing VM	Orig	Inline	PT+CSE
Interpreter (sec)	51.40	33.38	32.17
JIT-noinlining (sec)	11.06	8.61	8.64
JIT (sec)	8.81	8.21	8.23

Using Metrics for Optimization

- Benchmark: JOlden Voronoi

Metric	Orig	Inline	PT+CSE
base.executedInstructions.value	**445.13 M**	287.86 M	282.08 M
size.appRun.value	**1008**	1449	1425
poly.appInvokeDensity.value	116.17	10.84	11.06
poly.appTargetArity.bin(1)	1	1	1
pointer.appFieldAccessDensity.value	126.7	196.1	177.8

Executing VM	Orig	Inline	PT+CSE
Interpreter (sec)	51.40	33.38	32.17
JIT-noinlining (sec)	11.06	8.61	8.64
JIT (sec)	8.81	8.21	8.23

Using Metrics for Optimization

- Benchmark: JOlden Voronoi

Metric	Orig	Inline	PT+CSE
base.executedInstructions.value	445.13 M	287.86 M	282.08 M
size.appRun.value	1008	1449	1425
poly.appInvokeDensity.value	**116.17**	10.84	11.06
poly.appTargetArity.bin(1)	**1**	1	1
pointer.appFieldAccessDensity.value	126.7	196.1	177.8

Executing VM	Orig	Inline	PT+CSE
Interpreter (sec)	51.40	33.38	32.17
JIT-noinlining (sec)	11.06	8.61	8.64
JIT (sec)	8.81	8.21	8.23

Using Metrics for Optimization

- Benchmark: JOlden Voronoi

Metric	Orig	Inline	PT+CSE
base.executedInstructions.value	445.13 M	287.86 M	282.08 M
size.appRun.value	1008	1449	1425
poly.appInvokeDensity.value	116.17	**10.84**	11.06
poly.appTargetArity.bin(1)	1	1	1
pointer.appFieldAccessDensity.value	126.7	196.1	177.8

Executing VM	Orig	Inline	PT+CSE
Interpreter (sec)	51.40	33.38	32.17
JIT-noinlining (sec)	11.06	8.61	8.64
JIT (sec)	8.81	8.21	8.23

Using Metrics for Optimization

- Benchmark: JOlden Voronoi

Metric	Orig	Inline	PT+CSE
base.executedInstructions.value	445.13 M	**287.86 M**	282.08 M
size.appRun.value	1008	**1449**	1425
poly.appInvokeDensity.value	116.17	10.84	11.06
poly.appTargetArity.bin(1)	1	1	1
pointer.appFieldAccessDensity.value	126.7	196.1	177.8

Executing VM	Orig	Inline	PT+CSE
Interpreter (sec)	51.40	33.38	32.17
JIT-noinlining (sec)	11.06	8.61	8.64
JIT (sec)	8.81	8.21	8.23

Using Metrics for Optimization

- Benchmark: JOlden Voronoi

Metric	Orig	Inline	PT+CSE
base.executedInstructions.value	445.13 M	287.86 M	282.08 M
size.appRun.value	1008	1449	1425
poly.appInvokeDensity.value	116.17	10.84	11.06
poly.appTargetArity.bin(1)	1	1	1
pointer.appFieldAccessDensity.value	126.7	196.1	177.8

Executing VM	Orig	Inline	PT+CSE
Interpreter (sec)	**51.40**	**33.38**	32.17
JIT-noinlining (sec)	11.06	8.61	8.64
JIT (sec)	8.81	8.21	8.23

Using Metrics for Optimization

- Benchmark: JOlden Voronoi

Metric	Orig	Inline	PT+CSE
base.executedInstructions.value	445.13 M	287.86 M	282.08 M
size.appRun.value	1008	1449	1425
poly.appInvokeDensity.value	116.17	10.84	11.06
poly.appTargetArity.bin(1)	1	1	1
pointer.appFieldAccessDensity.value	126.7	196.1	177.8

Executing VM	Orig	Inline	PT+CSE
Interpreter (sec)	51.40	33.38	32.17
JIT-noinlining (sec)	**11.06**	**8.61**	8.64
JIT (sec)	**8.81**	**8.21**	8.23

Using Metrics for Optimization

- Benchmark: JOlden Voronoi

Metric	Orig	Inline	PT+CSE
base.executedInstructions.value	445.13 M	287.86 M	282.08 M
size.appRun.value	1008	1449	1425
poly.appInvokeDensity.value	116.17	10.84	11.06
poly.appTargetArity.bin(1)	1	1	1
pointer.appFieldAccessDensity.value	126.7	**196.1**	177.8

Executing VM	Orig	Inline	PT+CSE
Interpreter (sec)	51.40	33.38	32.17
JIT-noinlining (sec)	11.06	8.61	8.64
JIT (sec)	8.81	8.21	8.23

Using Metrics for Optimization

- Benchmark: JOlden Voronoi

Metric	Orig	Inline	PT+CSE
base.executedInstructions.value	445.13 M	287.86 M	282.08 M
size.appRun.value	1008	1449	1425
poly.appInvokeDensity.value	116.17	10.84	11.06
poly.appTargetArity.bin(1)	1	1	1
pointer.appFieldAccessDensity.value	126.7	196.1	**177.8**

Executing VM	Orig	Inline	PT+CSE
Interpreter (sec)	51.40	33.38	32.17
JIT-noinlining (sec)	11.06	8.61	8.64
JIT (sec)	8.81	8.21	8.23

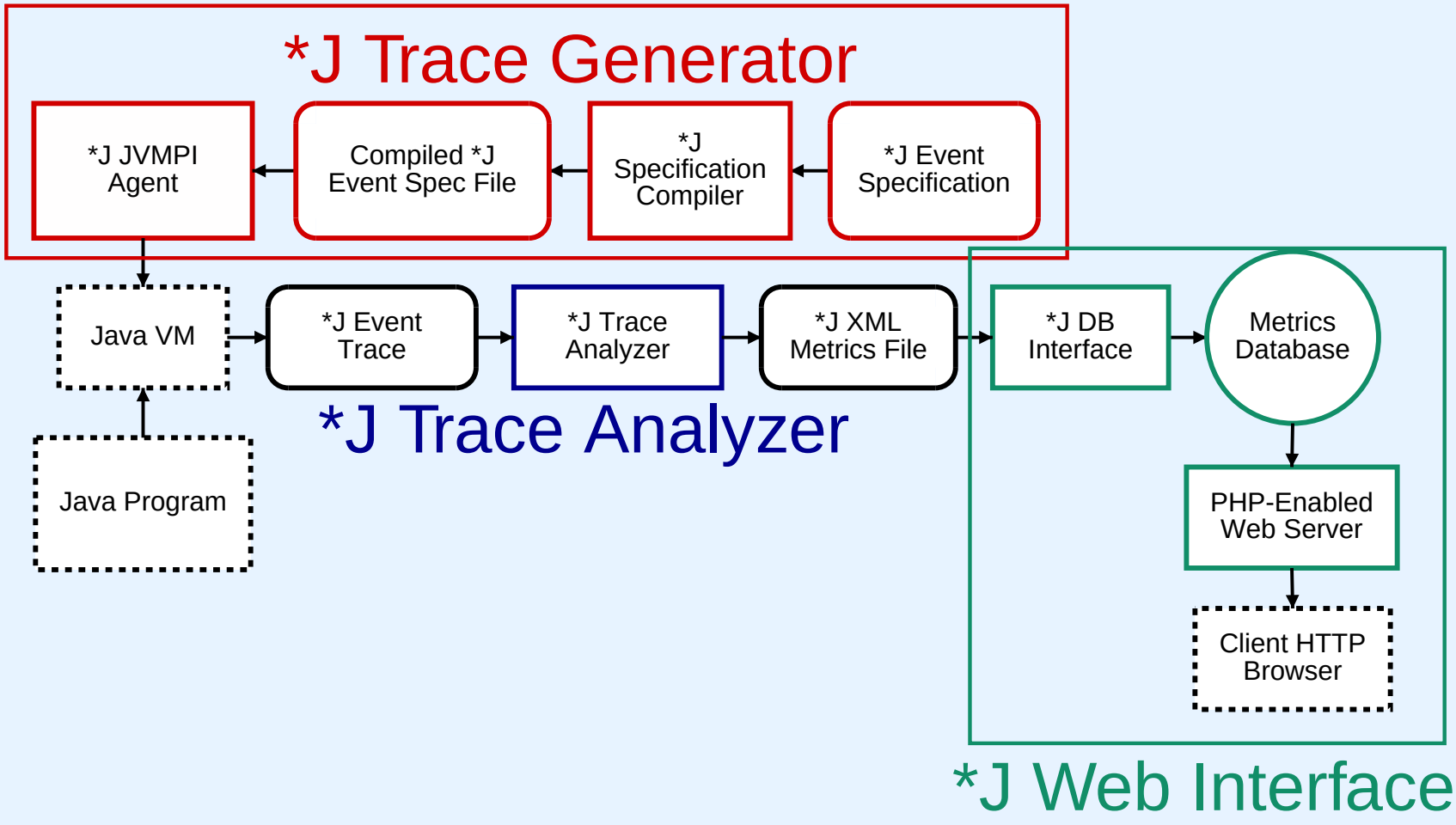
Using Metrics for Optimization

- Benchmark: JOlden Voronoi

Metric	Orig	Inline	PT+CSE
base.executedInstructions.value	445.13 M	287.86 M	282.08 M
size.appRun.value	1008	1449	1425
poly.appInvokeDensity.value	116.17	10.84	11.06
poly.appTargetArity.bin(1)	1	1	1
pointer.appFieldAccessDensity.value	126.7	196.1	177.8

Executing VM	Orig	Inline	PT+CSE
Interpreter (sec)	51.40	33.38	**32.17**
JIT-noinlining (sec)	11.06	8.61	**8.64**
JIT (sec)	8.81	8.21	**8.23**

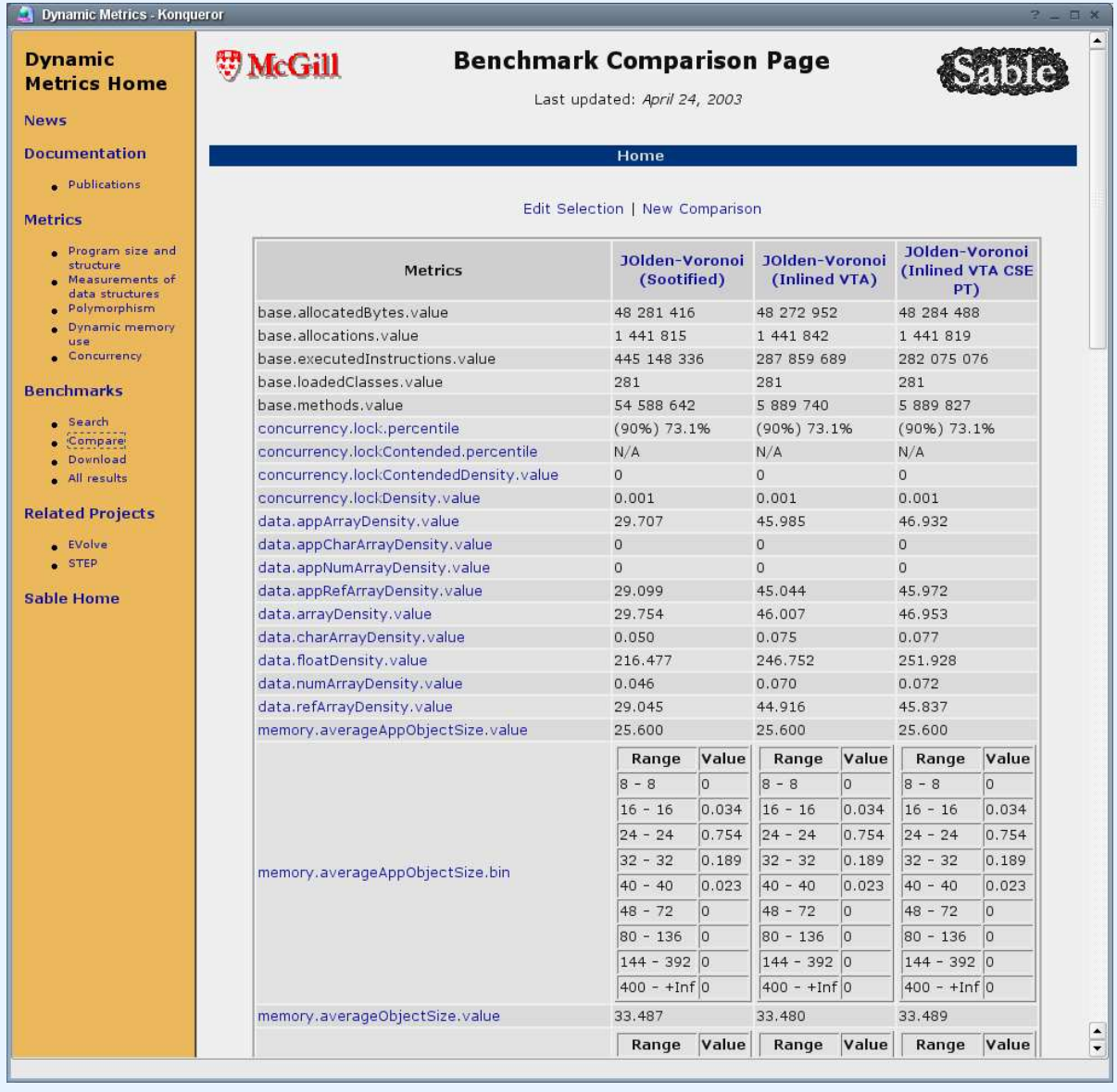
*J Framework at a glance



(Poster on display)

`http://www.sable.mcgill.ca/~bdufou1/starj/`

Website Usage



Metrics	JOlden-Voronoi (Sootified)	JOlden-Voronoi (Inlined VTA)	JOlden-Voronoi (Inlined VTA CSE PT)
base.allocatedBytes.value	48 281 416	48 272 952	48 284 488
base.allocations.value	1 441 815	1 441 842	1 441 819
base.executedInstructions.value	445 148 336	287 859 689	282 075 076
base.loadedClasses.value	281	281	281
base.methods.value	54 588 642	5 889 740	5 889 827
concurrency.lock.percentile	(90%) 73.1%	(90%) 73.1%	(90%) 73.1%
concurrency.lockContended.percentile	N/A	N/A	N/A
concurrency.lockContendedDensity.value	0	0	0
concurrency.lockDensity.value	0.001	0.001	0.001
data.appArrayDensity.value	29.707	45.985	46.932
data.appCharArrayDensity.value	0	0	0
data.appNumArrayDensity.value	0	0	0
data.appRefArrayDensity.value	29.099	45.044	45.972
data.arrayDensity.value	29.754	46.007	46.953
data.charArrayDensity.value	0.050	0.075	0.077
data.floatDensity.value	216.477	246.752	251.928
data.numArrayDensity.value	0.046	0.070	0.072
data.refArrayDensity.value	29.045	44.916	45.837
memory.averageAppObjectSize.value	25.600	25.600	25.600
memory.averageAppObjectSize.bin	Range / Value	Range / Value	Range / Value
	8 - 8 / 0	8 - 8 / 0	8 - 8 / 0
	16 - 16 / 0.034	16 - 16 / 0.034	16 - 16 / 0.034
	24 - 24 / 0.754	24 - 24 / 0.754	24 - 24 / 0.754
	32 - 32 / 0.189	32 - 32 / 0.189	32 - 32 / 0.189
	40 - 40 / 0.023	40 - 40 / 0.023	40 - 40 / 0.023
	48 - 72 / 0	48 - 72 / 0	48 - 72 / 0
	80 - 136 / 0	80 - 136 / 0	80 - 136 / 0
	144 - 392 / 0	144 - 392 / 0	144 - 392 / 0
	400 - +Inf / 0	400 - +Inf / 0	400 - +Inf / 0
memory.averageObjectSize.value	33.487	33.480	33.489
	Range / Value	Range / Value	Range / Value

Related Work

- Static metrics & Measurement Theory
  - Fenton & Pfleeger
  - Chidamber & Kemerer
- Dynamic Program Analysis
  - Dieckmann & Hölzle
  - Shuf et al.
  - Daly et al.
- Dynamic metrics
  - Aggarwal
  - Mitchell & Power

Future Work

- Implement a general-purpose profiling framework in SableVM
  - Extend metric possibilities
  - Improve performance
- Extend the set of metrics
- Formalize the desirable qualities

Conclusions

- We have defined a concise set of metrics suitable for compiler optimizations
- We have shown the usefulness of those metrics
- We have built a framework that allows to measure most of the defined metrics

`http://www.sable.mcgill.ca/metrics/`