Dynamic Metrics for Java

Bruno Dufour, Karel Driesen, Laurie Hendren and Clark Verbrugge
McGill University

Presented by:
Kiran Nagaraja
Rutgers University

Motivation

Compiler optimization techniques span different aspects of program behavior

- Compile time
- Total runtime
- In-memory size

Optimization techniques work on specific characteristics of programs

- Numeric or compute intensive
- Memory intensive
- Pointer intensive
- High concurrency

Need representative benchmarks to test the particular optimizations

Basic question: How do u characterize program behavior? Are there a small set of representative metrics that can be defined and measured for a quantitative analysis?

Secondary question: How do u guide compiler optimization? (Which technique to apply?)

Approach

Intuitive qualitative techniques are not good enough

Static analysis of programs may not capture dynamic behavior well

- Conservative
- Certain behavior cannot be estimated at compile time
  - E.g., memory allocation behavior

So, measure dynamic metrics to capture dynamic behavior

- Pros: measures ‘relevance’
- Cons: optimistic

Use quantitative metrics to characterize what we qualitatively know about them

Desired Properties for Dynamic Metrics

Unambiguous

Dynamic

E.g. not affected by dead code

Robust

E.g., Anything which is O(n) is not a good metric!

Discriminating

Machine independence

Can be published with secondary information…

Kinds of Metrics

Value metrics

Single values such as average, median, total

Percentiles

X% of characteristic A accounts for Y% of B, rather than count(A) and count(B), for example.

Defines the distribution of contributions

Bins

Classes of behavior of special interest

Continuous metrics

Time series data of dynamic characteristic

Visual inspection - qualitative

Comparison with Systems Benchmarking Approaches

Static analysis across multiple components is incredibly hard

- Analytical models for single-tier systems possible
- Empirical approaches increasing in popularity

Goals: Resource utilization, performance/availability prediction, bottleneck evaluation

- Behavior can vary significantly with workloads

Metrics: CPU, memory allocation, Disk/NW bandwidth

Benchmark Programs

An array of standard programs qualitatively different in behavior

- See paper for list
- Import ones are: CA, COEFF, EMPTY, Volano, COMP, JAVAC

Try to quantitatively compare their behaviors along the following:

- Program size and control structure
- Data structure
- Polymorphism
- Memory use
- Concurrency and synchronization

Program Size and Structure

Metric	EMPTY	HELLO	Omst	LPACK	Operm	COEFF	COMP	SOOT	JAVAC
size.appLoadedClasses.value	1	1	6	1	10	6	22	531	175
size.appLoad.value	4	7	727	1056	863	2374	6555	45111	44664
size.appRun.value	0	4	600	749	785	975	5084	25666	26267
size.appHot.value	0	4	175	59	393	57	396	2549	2759
size.appHot.percentile	n/a	100%	29%	8%	50%	6%	8%	10%	11%
size.loadedClasses.value	275	275	281	278	285	286	310	818	471
size.load.value	71818	71821	76302	77932	76438	80292	90762	126566	133172
size.run.value	7343	7793	10112	10698	9991	12880	14514	37852	37830
size.hot.value	1014	1038	186	115	398	115	396	3097	2258
size.hot.percentile	14%	13%	2%	1%	4%	1%	3%	8%	6%

Metrics which include libraries are not discriminating

Bytecode level metrics are less ambiguous

'Actual use' metrics are most characterizing – robust to optimizations

'Hot' metrics are less robust to varying input

Program Control Structure

Measuring instructions that change the control flow

Example: (if, switch, invoke virtual)

Control density(value): total number of control byte codes touched

Changing control density(value):
byte codes change direction / # of control byte code touched

Changing control rate (value)

Most dynamic – Measures actual change in direction

Data Structures

Metric	COEFF	LPACK	CA	Obh	Opow	SBLCC	EMPTY
data.appArrayDensity.value	160.404	157.775	139.890	105.947	97.433	38.868	n/a
data.appCharArrayDensity.value	0.0	0.0	15.494	0.0	0.0	0.0	n/a
data.appNumArrayDensity.value	79.486	148.385	124.334	97.577	96.487	11.209	n/a
data.appRefArrayDensity.value	80.713	9.389	0.015	4.383	0.162	13.274	n/a
data.arrayDensity.value	150.939	152.170	30.881	105.891	93.418	43.551	73.496
data.charArrayDensity.value	1.513	2.077	10.208	0.012	0.016	6.547	32.549
data.numArrayDensity.value	75.033	140.929	14.603	97.513	92.491	9.416	35.025
data.refArrayDensity.value	73.772	8.890	0.400	4.380	0.156	15.636	1.874

Array intensive: Tracking array access bytecodes

Problems: Accesses within libraries; Multidimensional arrays

Metrics normalized by kbc (kilo bytecode), hence the numbers for EMPTY

Polymorphism

[app]CallSites (value):Total number of different call sites executed

Skew: does not include static invoke instr.

[app]invokeDensity (value):

executed invokevirtual or invokeinterface / kbc

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

Polymorphism

polymorphism.[app]targetArity.bin:

measures the percentage of all call sites which have 1,2 or 3 different target methods:

Similar bin metric of receiver

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

Allocated byte density:

# of allocated bytes / kbc

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

Concurrency and Synchronization

Concurrency lock percentile:

# of locks in 90% of locking operations / kbc

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

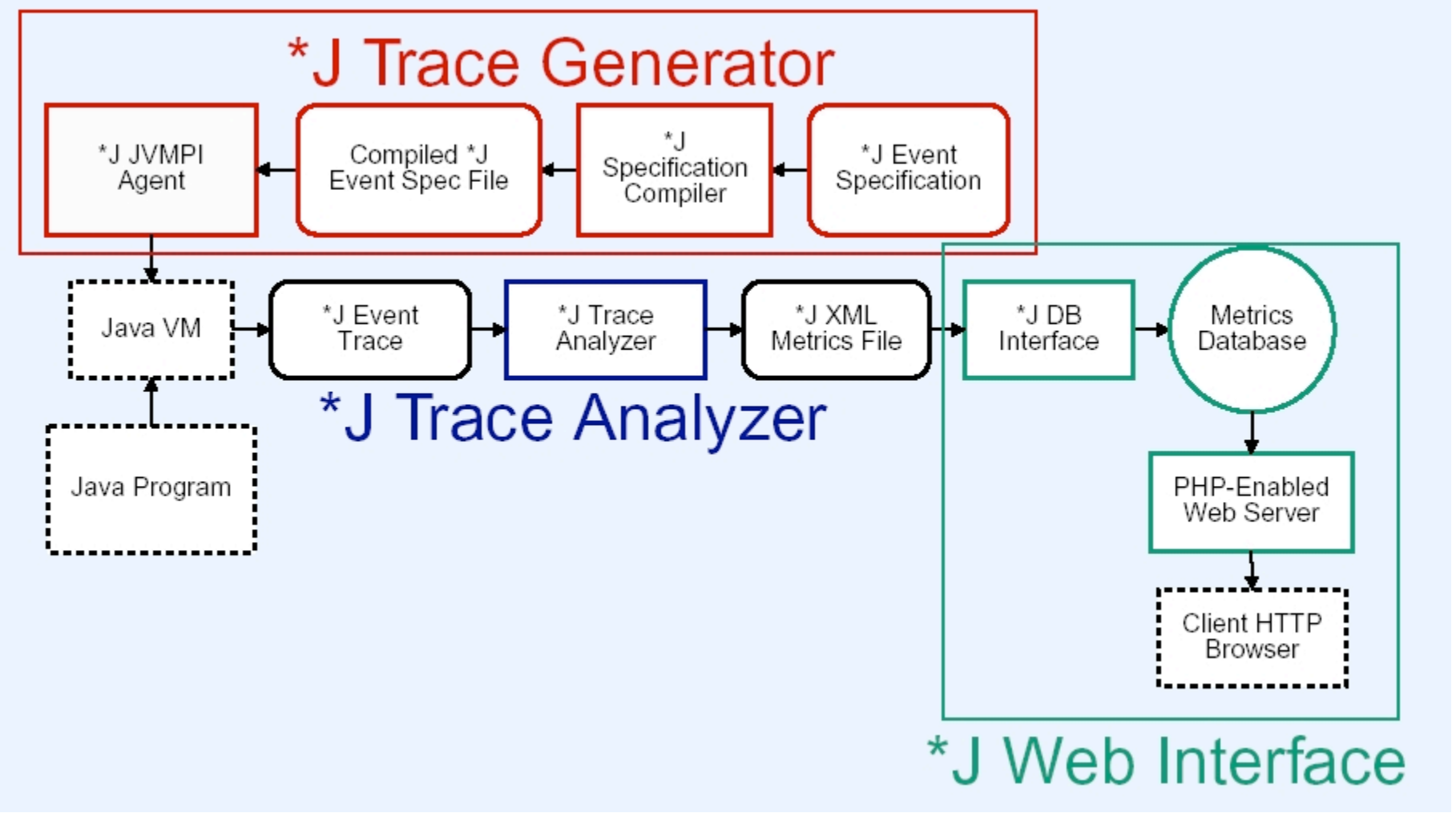
Guiding Compiler Optimizations

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

Applying optimizations requires knowledge about program characteristic

J* Framework


*J Trace Generator
*J JVMPI Agent
Compiled *J Event Spec File
*J Specification Compiler
*J Event Specification
Java VM
*J Event Trace
*J Trace Analyzer
*J XML Metrics File
*J DB Interface
Metrics Database
Java Program
*J Trace Analyzer
PHP-Enabled Web Server
Client HTTP Browser
*J Web Interface

Conclusions

Uniform presentation of the set of dynamic metrics that can be used to characterize the qualitative properties of programs used as benchmarks

- Can be used to quantitatively compare program behavior

Can be used as a guide for applying compiler optimization techniques

Thank you!

Questions?

For more information

http://vivo.cs.rutgers.edu