# Exploration of Supervised Machine Learning Techniques for Runtime Selection of CPU vs.GPU Execution in Java Programs

Gloria Kim (Rice University)
Akihiro Hayashi (Rice University)
Vivek Sarkar (Georgia Tech)

# Java For HPC?

| 4K x 4K Matrix Multiply | GFLOPS | Absolute speedup |
|---|---|---|
| Python | 0.005 | 1 |
| **Java** | **0.058** | **11** |
| C | 0.253 | 47 |
| Parallel loops | 1.969 | 366 |
| Parallel divide and conquer | 36.168 | 6,724 |
| + vectorization | 124.945 | 23,230 |
| + AVX intrinsics | 335.217 | 62,323 |
| Strassen | 361.681 | 67,243 |

**Any ways to accelerate Java programs?**

Dual Socket Intel Xeon E5-2666 v3, 18cores, 2.9GHz, 60-GiB DRAM

Credits : Charles Leiserson, Ken Kennedy Award Lecture @ Rice University, Karthik Murthy @ PACT2015

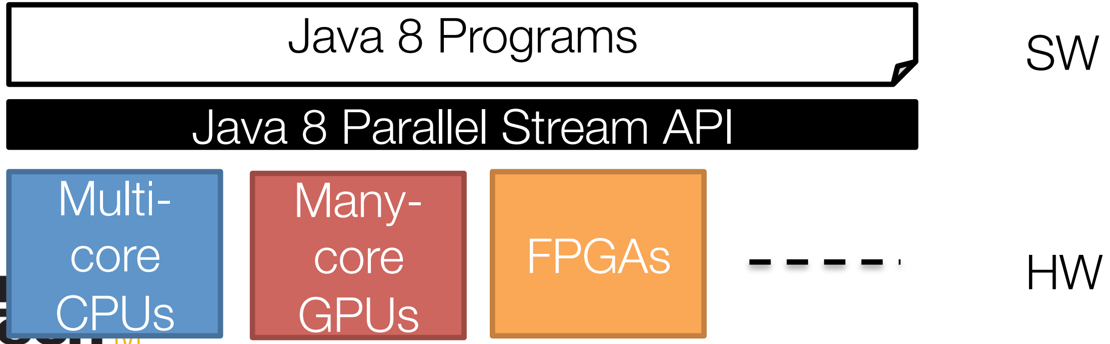
# Java 8 Parallel Streams APIs

- Explicit Parallelism with *lambda expressions*

```
IntStream.range(0, N)
             .parallel()
             .forEach(i ->
                         <lambda>);
```

# Explicit Parallelism with Java

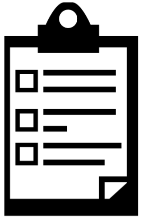
- High-level parallel programming with Java offers opportunities for
  - preserving portability
    - ✓ Low-level parallel/accelerator programming is not required
  - enabling compiler to perform parallel-aware optimizations and code generation



# Challenges for GPU Code Generation

**Standard Java API Call for Parallelism**

```
IntStream.range(0, N).parallel().forEach(i -> <lambda>);
```




**Challenge 1**
Supporting **Java Features** on GPUs

- ✓ Exception Semantics
- ✓ Virtual Method Calls

**Challenge 2**
**Accelerating** Java programs on GPUs

- ✓ Kernel Optimizations
- ✓ DT Optimizations

**Challenge 3**
**CPU / GPU Selection**

- ✓ Selection of a faster device from CPUs and GPUs

Credits : Checklist by Phil Laver from the Noun Project, Rocket by Luis Prado from the Noun Project
Rocket by Maxwell Painter Karpylev from the Noun Project, Running by Dillon Arloff from the Noun Project

# Related Work: Java + GPU

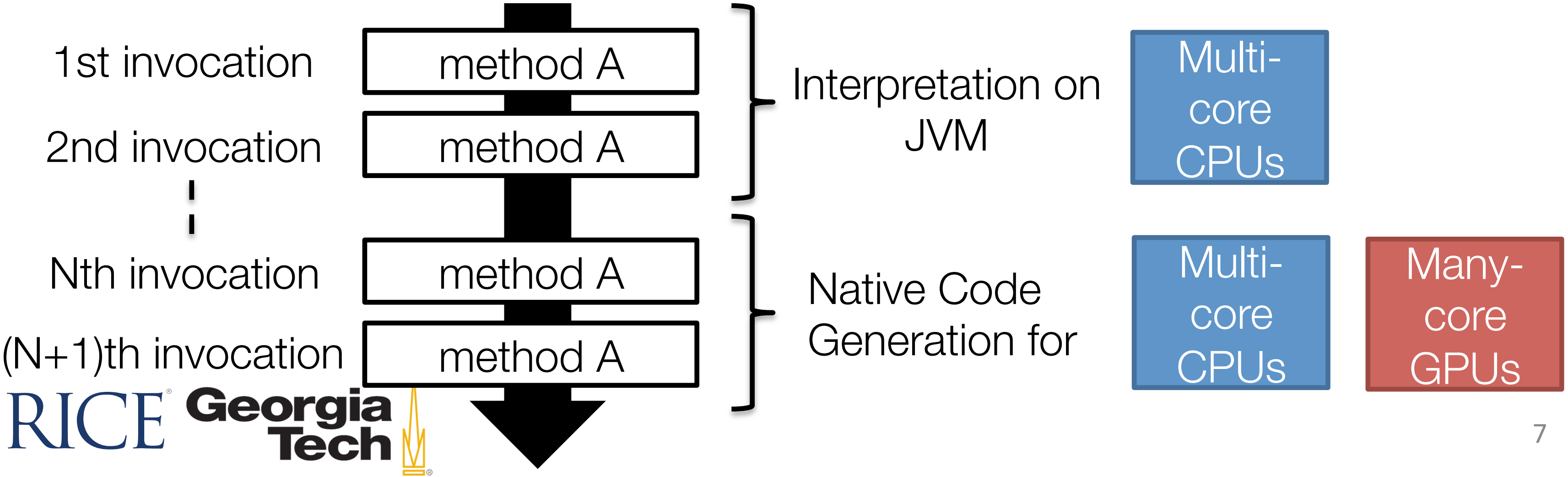
| | Lang | JIT | GPU Kernel | Device Selection |
|---|---|---|---|---|
| **JCUDA** | Java | - | CUDA | GPU only |
| **Lime** | Lime | ✔ | Override map/reduce | Static |
| **Firepile** | Scala | ✔ | reduce | Static |
| **JaBEE** | Java | ✔ | Override run | GPU only |
| **Aparapi** | Java | ✔ | map | Static |
| **Hadoop-CL** | Java | ✔ | Override map/reduce | Static |
| **RootBeer** | Java | ✔ | Override run | Not Described |
| **HJ-OpenCL** | HJ | - | forall / lambda | Static |
| **PPPJ09 (auto)** | Java | ✔ | For-loop | Dynamic with Regression |
| **Our Work** | Java | ✔ | Parallel Stream | Dynamic with Machine Learning |

RICE Georgia Tech

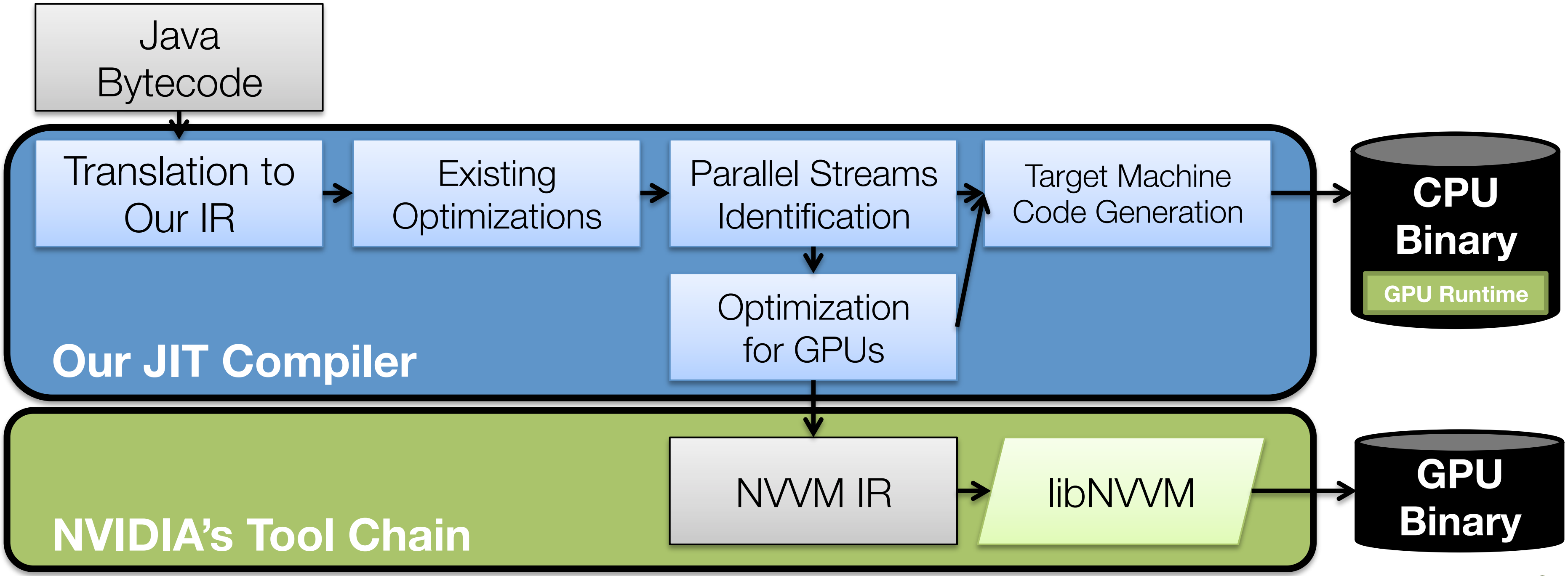
***None of these approaches considers Java 8 Parallel Stream APIs and a dynamic device selection with machine-learning***

# JIT Compilation for GPU

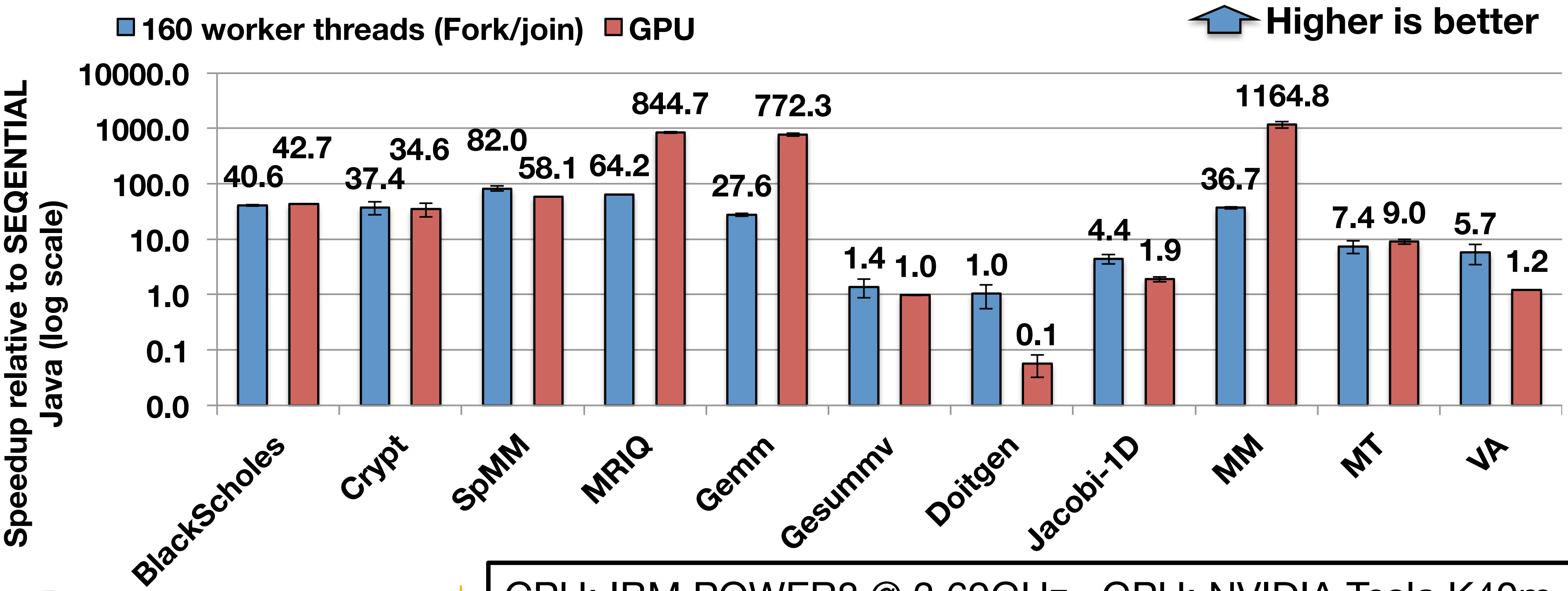
- IBM Java 8 Compiler
  - Built on top of the production version of the IBM Java 8 runtime environment (J9 VM)



# The JIT Compilation Flow



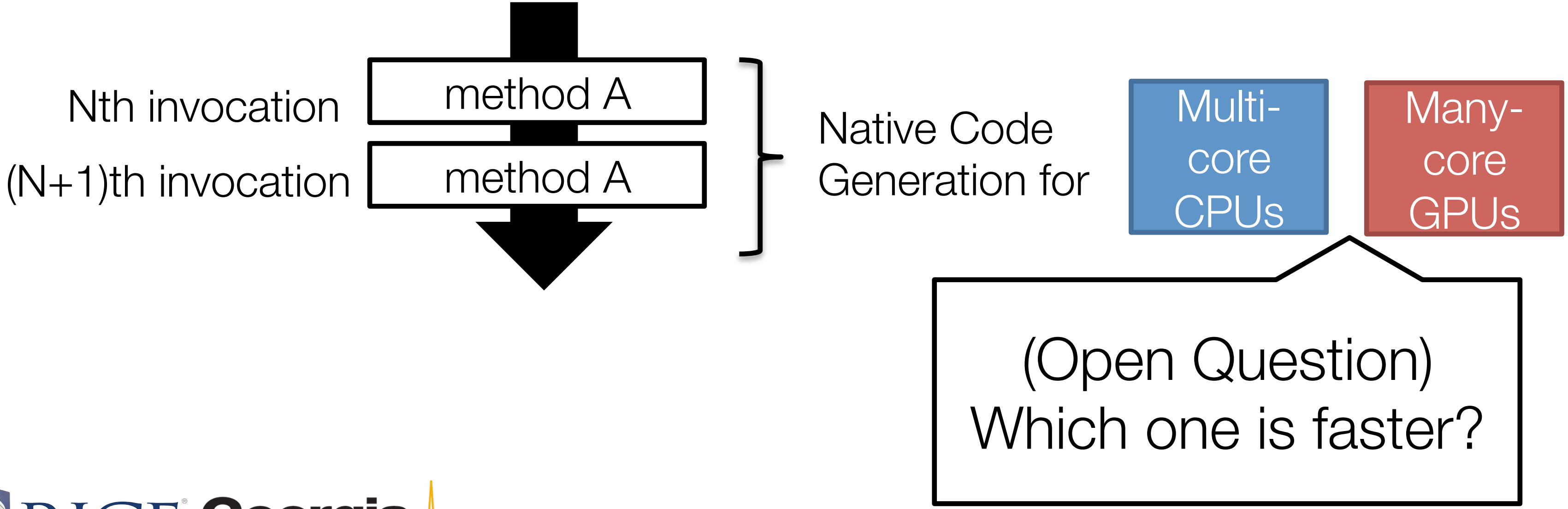
# Performance Evaluations



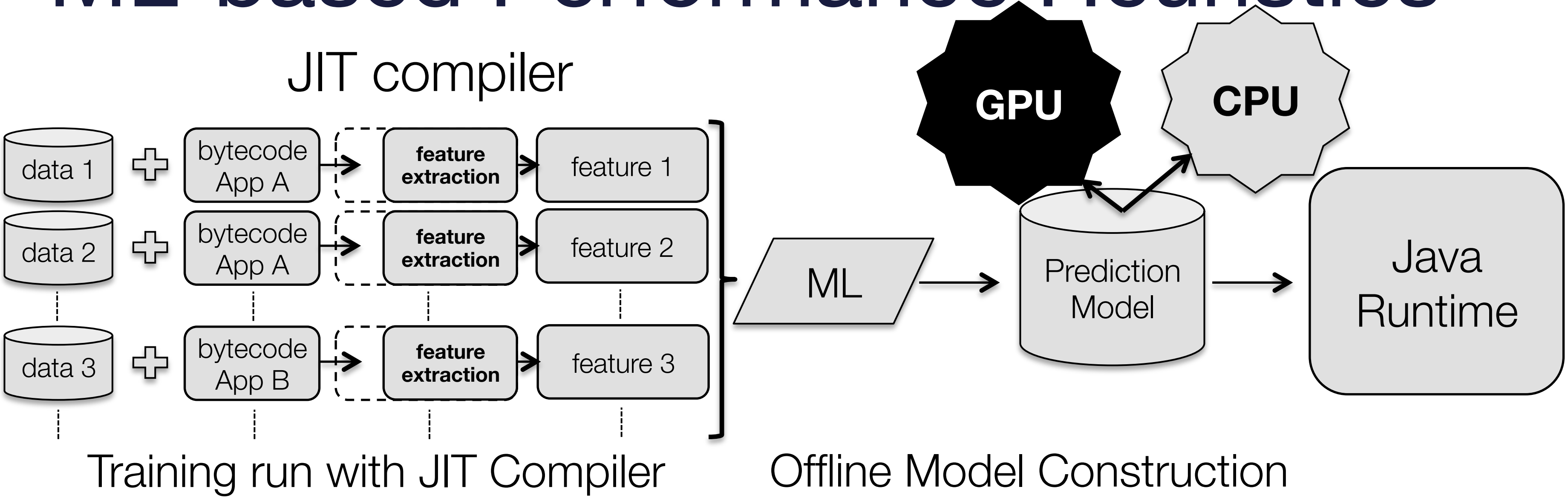

CPU: IBM POWER8 @ 3.69GHz , GPU: NVIDIA Tesla K40m

RICE Georgia Tech

# Runtime CPU/GPU Selection



# Our approach: ML-based Performance Heuristics



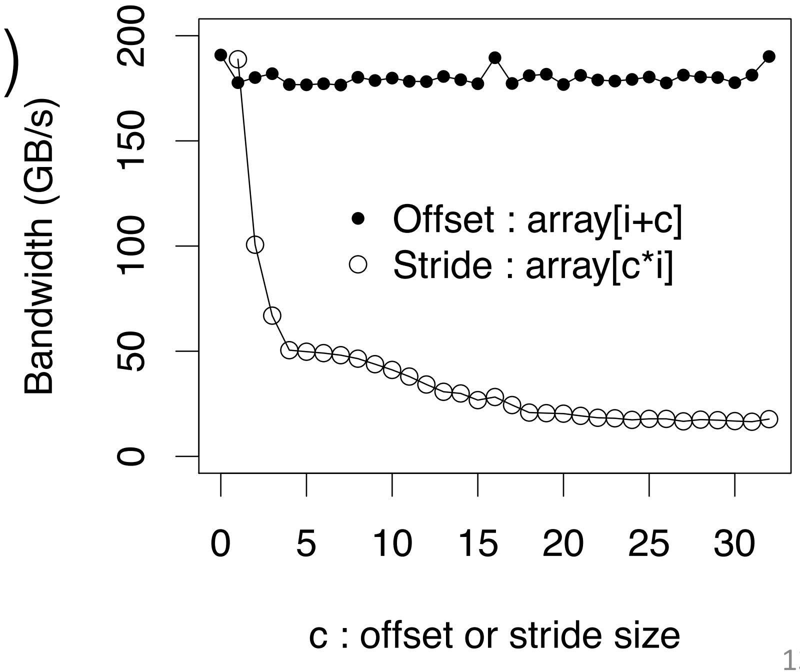

- A binary prediction model is constructed by supervised machine learning techniques

# Features of program

- Loop Range (Parallel Loop Size)
- The dynamic number of Instructions in IR
  - Memory Access
  - Arithmetic Operations
  - Math Methods
  - Branch Instructions
  - Other Types of Instructions

# Features of program (Cont'd)

- The dynamic number of Array Accesses
  - Coalesced Access (a[i])
  - Offset Access (a[i+c])
  - Stride Access (a[c*i])
  - Other Access (a[b[i]])



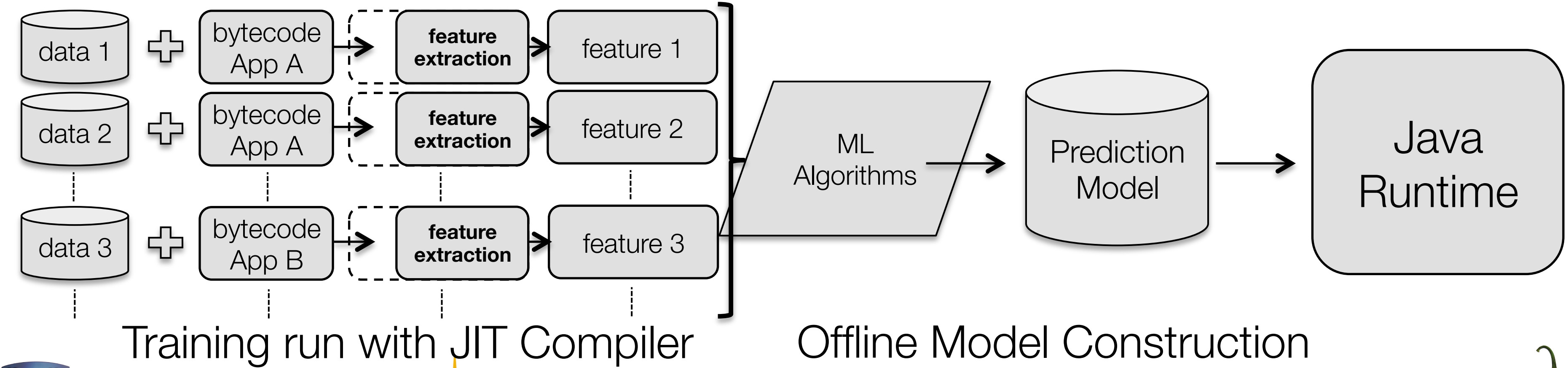
# An example of feature vector

```
IntStream.range(0, N)
            .parallel()
            .forEach(i -> { a[i] = b[i] + c[i];});
```

```
"features" : {
  "range": 256,
  "ILs" : {
    "Memory": 9, "Arithmetic": 7, "Math": 0,
    "Branch": 1, "Other": 1 },
  "Array Accesses" : {
    "Coalesced": 3, "Offset": 0, "Stride": 0, "Random": 0},
}
```

# Offline Prediction Model Construction

- Obtained **291** samples by running 11 applications with different data sets
  - Additional 41 samples by running additional 3 applications
- Choice is either **GPU** or **160 worker threads on CPU**




RICE Georgia Tech

# Platform

- CPU
  - IBM POWER8 @ 3.69GHz
    - ✓20 cores
    - ✓8 SMT threads per core = up to 160 threads
    - ✓256 GB of RAM
- GPU
  - NVIDIA Tesla K40m
    - ✓12GB of Global Memory

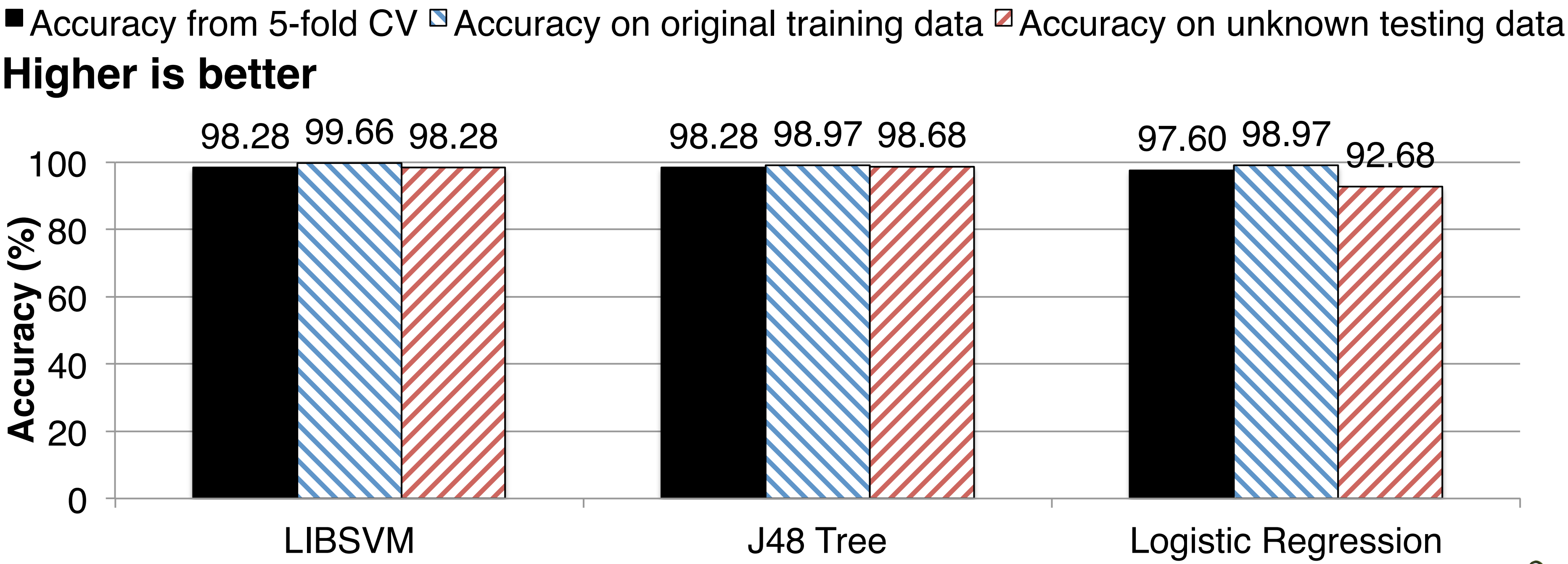
# Applications

| Application | Source | Field | Max Size | Data Type |
|---|---|---|---|---|
| **BlackScholes** | | Finance | 4,194,304 | double |
| **Crypt** | JGF | Cryptography | Size C (N=50M) | byte |
| **SpMM** | JGF | Numerical Computing | Size C (N=500K) | double |
| **MRIQ** | Parboil | Medical | Large (64^3) | float |
| **Gemm** | Polybench | Numerical Computing | 2K x 2K | int |
| **Gesummv** | Polybench | | 2K x 2K | int |
| **Doitgen** | Polybench | | 256x256x256 | int |
| **Jacobi-1D** | Polybench | | N=4M, T=1 | int |
| **Matrix Multiplication** | | | 2K x 2K | double |
| **Matrix Transpose** | | | 2K x 2K | double |
| **VecAdd** | | | 4M | double |

# Explored ML Algorithms

- ❑Support Vector Machines (LIBSVM)
- ❑Decision Trees (Weka 3)
- ❑Logistic Regression (Weka 3)
- ❑Multilayer Perceptron (Weka 3)
- ❑k Nearest Neighbors (Weka 3)
- ❑Decision  Stumps (Weka 3)
- ❑Naïve Bayes (Weka 3)

# Top 3 ML Algorithms Full Set of Features (10 Features)



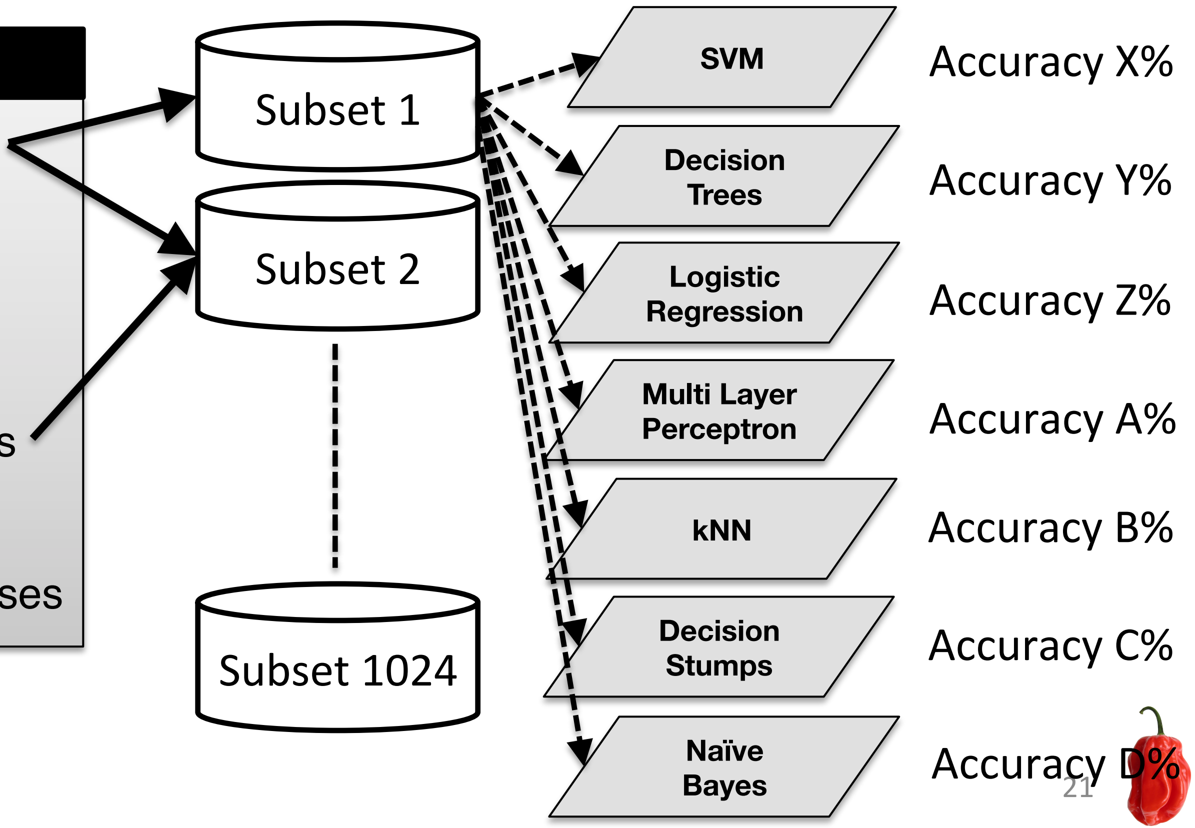
# Further Explorations and Analyses by Feature Subsetting

- Research Questions
  - Are the 10 features the best combination?
    - ✓Accuracy
    - ✓Runtime prediction overheads
  - Which features are more important?

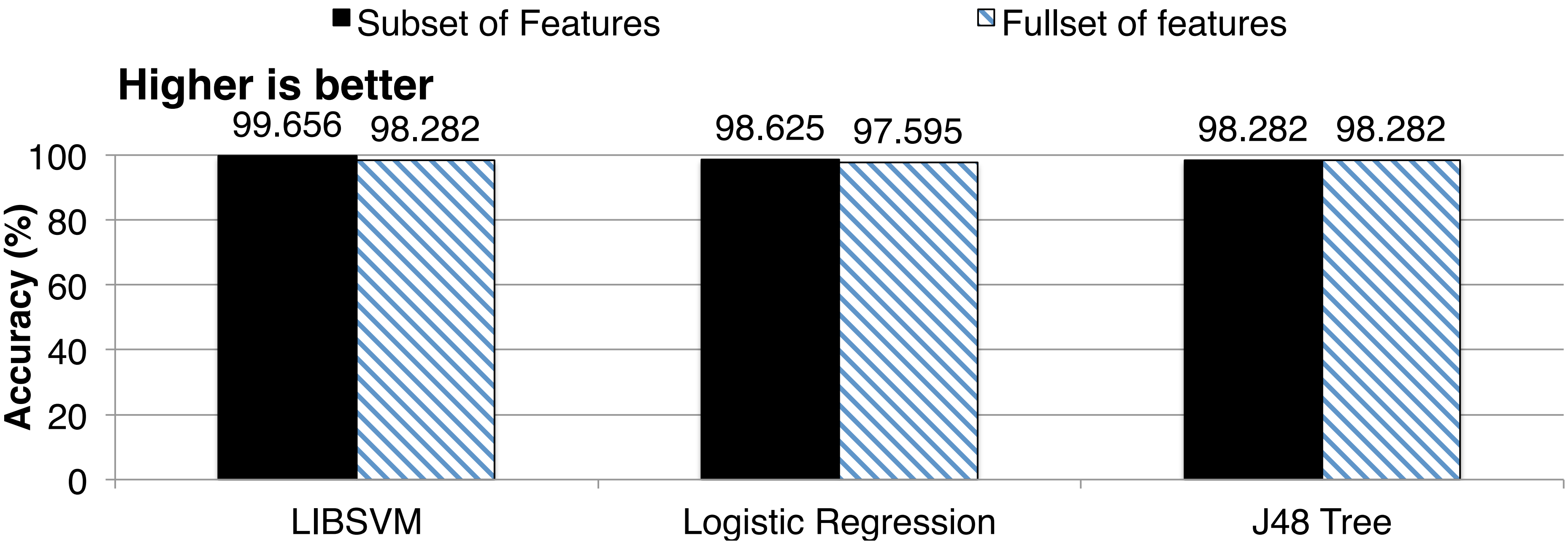
# Feature Sub-Setting for further analyses: 10 Algorithms x 1024 Subsets

**10 Features**

1. Loop Range (Parallel Loop Size)
2. # of Memory Accesses
3. # of Arithmetic Operations
4. # of Math Methods
5. # of Branch Instructions
6. # of Other Types of Instructions
7. # of Coalesced Memory Accesses
8. # of Offset Memory Accesses
9. # of Stride Memory Accesses
10. # of Other Types of Array Accesses

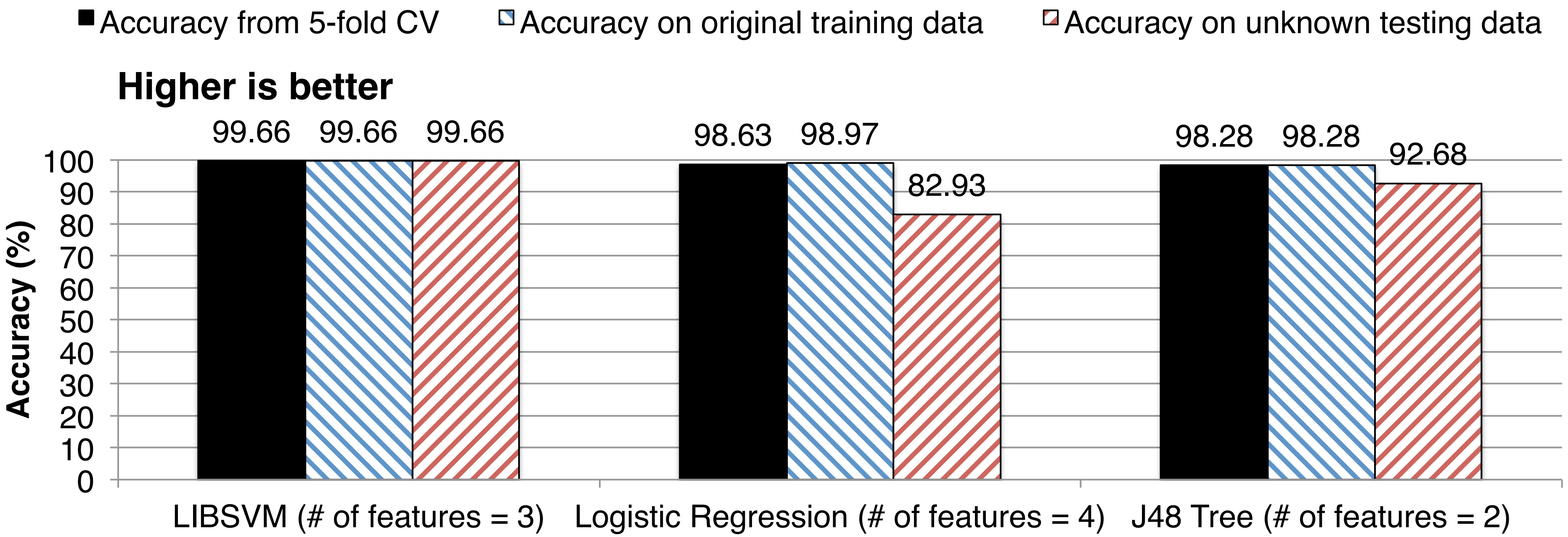


# The impact of Subsetting (Fullset vs. the best subset)




Feature subsetting can yield better accuracies

# The impact of Subsetting (Fullset vs. the best subset)




Subsets with only a few features show great accuracies

# An example of Subsetting analyses

- With LIBSVM, 99 subsets achieved the highest accuracy

| Feature | # of Models with this feature | Percentage of Models with this feature |
|---|---|---|
| **range** | **99** | **100.0%** |
| **stride** | **96** | **97.0%** |
| **arithmetic** | **65** | **65.7%** |
| Other 2 | 56 | 56.6% |
| memory | 56 | 56.6% |
| offset | 55 | 55.6% |
| branch | 54 | 54.5% |
| math | 46 | 46.5% |
| Other 1 | 43 | 43.4% |

# Runtime Prediction Overheads

| | LIBSVM | Logistic Regression | J48 Decision Trees |
|---|---|---|---|
| Full Set (10 features) | 2.278 usec | 0.158 usec | 0.020 usec |
| Subset | 2.107 usec (3 Features) | 0.106 usec (4 Features) | 0.020 usec (2 Features) |

- Subsetting can reduce prediction overheads
- However, this part is very small compared to the kernel execution part
  - Based on our analysis, kernels usually take a few milliseconds

RICE Georgia Tech

# Lessons Learned

- ML-based CPU/GPU Selection is a promising way to choose faster devices
- *LIBSVM, Logistic Regression, and J48 Decision Tree* are machine learning techniques that produce models with best accuracies
- *Range, coalesced, other types of array accesses, and arithmetic instructions* are particularly important features

RICE Georgia Tech

# Lessons Learned (Cont'd)

- While *LIBSVM* (Non-linear classification) shows excellent accuracy in prediction, runtime prediction overheads are relatively larger compared to other algorithms
  - However, those overheads are negligible in general
- *J48 Decision Tree* shows comparable accuracy to *LIBSVM*. Also, the output of the *J48 Decision Tree* is more human-readable and fine-tunable

# Conclusions

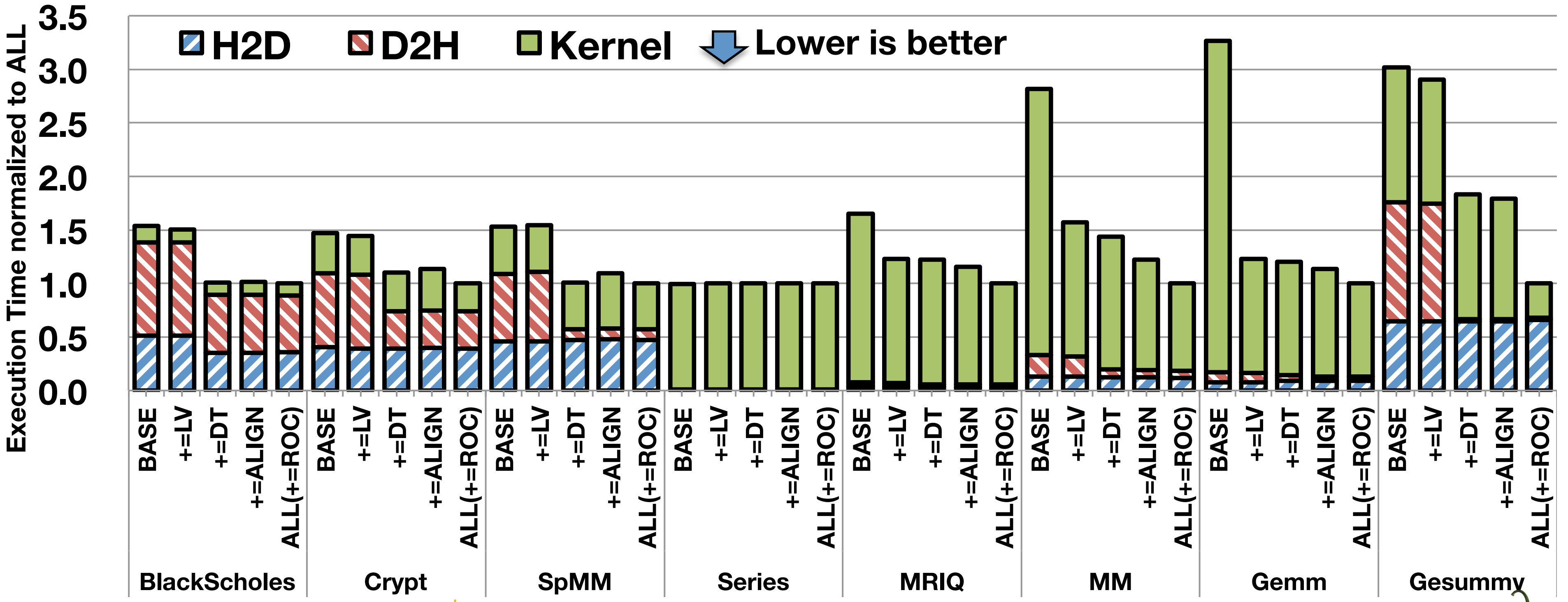
- Conclusions
  - ML-based CPU/GPU Selection is a promising way to choose faster devices
- Future Work
  - Evaluate End-to-end performance improvements
  - Explore the possibility of applying our technique to OpenMP, OpenACC, and OpenCL
  - Perform experiments on recent versions of GPUs

# Further Readings

- Code Generation and Optimizations
  - **"Compiling and Optimizing Java 8 Programs for GPU Execution."** Kazuaki Ishizaki, Akihiro Hayashi, Gita Koblents, Vivek Sarkar. 24th International Conference on Parallel Architectures and Compilation Techniques (PACT), October 2015.
- Performance Heuristics (SVM-based)
  - **"Machine-Learning-based Performance Heuristics for Runtime CPU/GPU Selection."** Akihiro Hayashi, Kazuaki Ishizaki, Gita Koblents, Vivek Sarkar. 12th International Conference on the Principles and Practice of Programming on the Java Platform: virtual machines, languages, and tools (PPPJ), September 2015.

# Backup Slides

# Performance Breakdown



# Precisions and Recalls with cross-validation

**Higher is better**

| | Precision CPU | Recall CPU | Precision GPU | Recall GPU |
|---|---|---|---|---|
| Range | 79.0% | 100% | 0% | 0% |
| +=nIRs | 97.8% | 99.1% | 96.5% | 91.8% |
| +=dIRs | 98.7% | 100% | 100% | 95.0% |
| +=Arrays | 98.7% | 100% | 100% | 95.0% |
| ALL | 96.7% | 100% | 100% | 86.9% |

*All prediction models except Range rarely make a bad decision*

# Java Features on GPUs

- Exceptions
  - Explicit exception checking on GPUs
    - ✓ `ArrayIndexOutOfBoundsException`
    - ✓ `NullPointerException`
    - ✓ `ArithmeticException` (Division by zero only)
  - Further Optimizations
    - ✓ **Loop Versioning** for redundant exception checking elimination on GPUs based on [Artigas+, ICS'00]
- Virtual Method Calls
  - Direct De-virtualization or Guarded De-virtualization [Ishizaki+, OOPSLA'00]

# Loop Versioning Example

```
IntStream.range(0, N)
.parallel()
.forEach(
    i -> {
        Array[i] = i / N;
    }
);
```

May cause OutOfBoundsException

May cause NullPointerException

May cause ArithmeticException

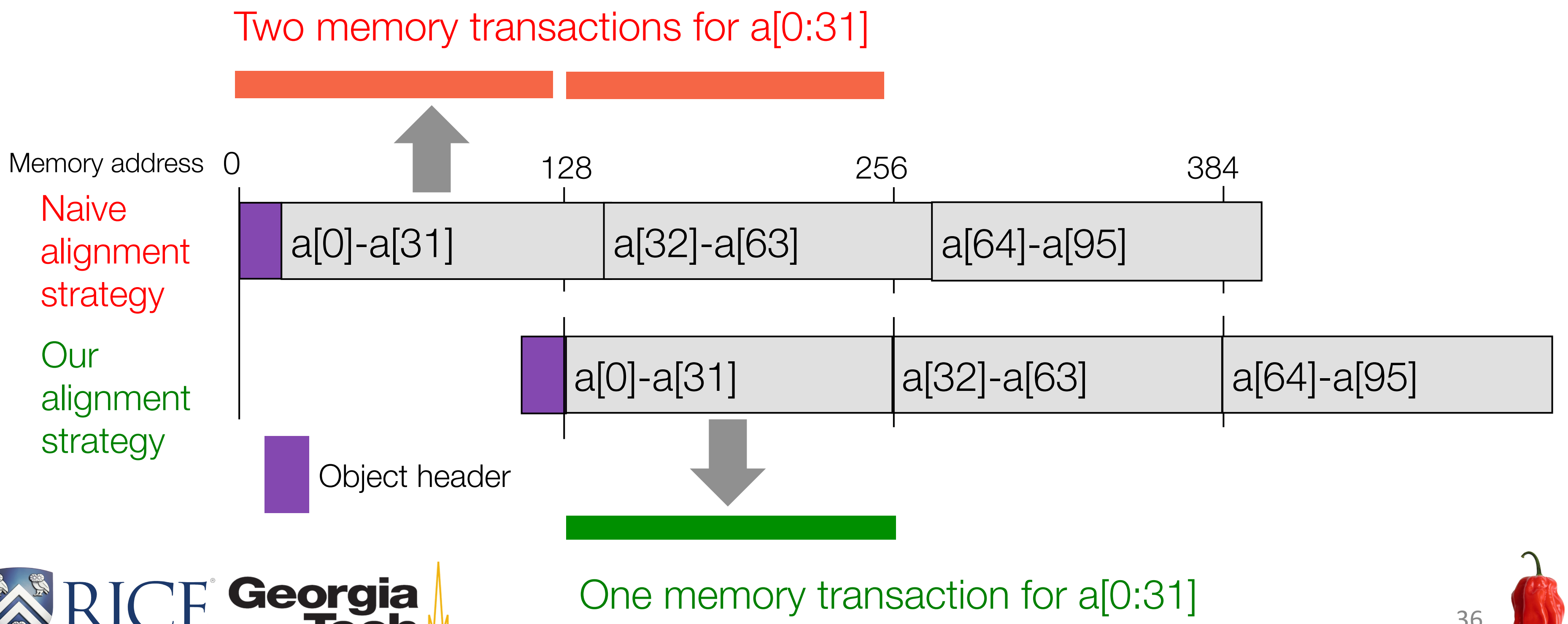
```
// Speculative Checking on CPU
if (Array != null
    && 0 <= Array.length
    && N <= Array.length
    && N != 0) {
    // Safe GPU Execution
} else {
```

RICE Georgia Tech

# Optimizing GPU Execution

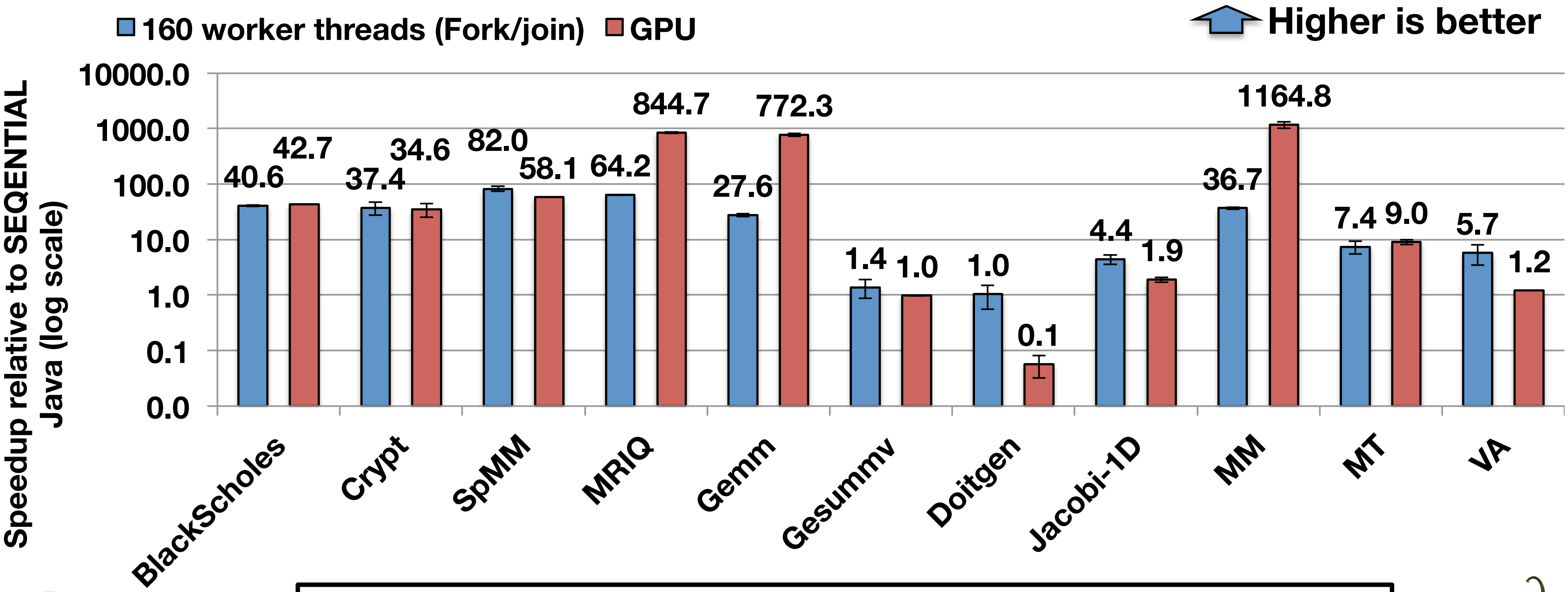
- Kernel optimization
  - Optimizing alignment of Java arrays on GPUs
  - Utilizing “*Read-Only Cache*” in recent GPUs
- Data transfer optimizations
  - Redundant data transfer elimination
  - Partial transfer if an array subscript is affine

# The Alignment Optimization Example




RICE Georgia Tech

# Performance Evaluations




CPU: IBM POWER8 @ 3.69GHz , GPU: NVIDIA Tesla K40m

# How to evaluate prediction models: 5-fold cross validation

- *Overfitting* problem:
  - Prediction model may be tailored to the eleven applications if training data = testing data
- To avoid the *overfitting* problem:
  - Calculate the accuracy of the prediction model using 5-fold cross validation

**TRAINING DATA (291 samples)**

**Subset 1 | Subset 2 | Subset 3 | Subset 4 | Subset 5**

**Used for TESTING**
**Accuracy : X%**

**Build a prediction model trained on Subset 2-5**

**Subset 1 | Subset 2 | Subset 3 | Subset 4 | Subset 5**

**Used for TESTING**
**Accuracy : Y%**

**Build a prediction model trained on Subset 1, 3-5**