Object Oriented Machine Learning with a Multicore Real-Time Java Processor

JTRES '10

Rasmus Ulslev Pedersen
Dept. of Informatics, CBS

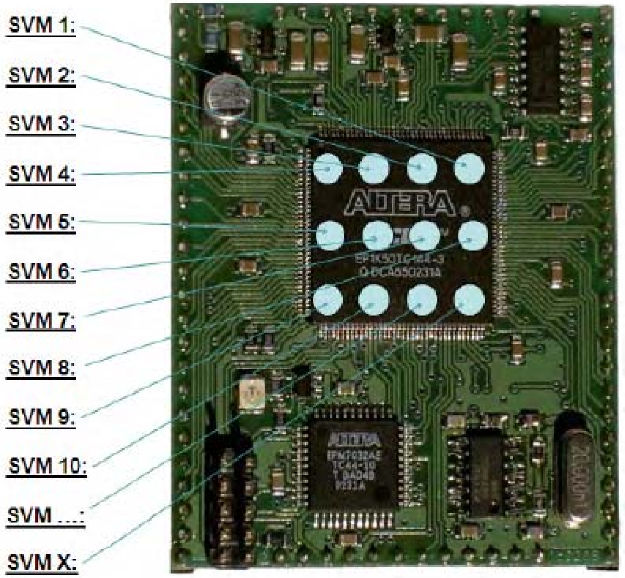
Martin Schoeberl
DTU Informatics
Department of Informatics and Mathematical Modeling
Technical University of Denmark

Overview of talk

- Motivation
- Real time machine learning
- Multicore machine learning
- Implementation platform
- Experiments
- Conclusion

Motivation

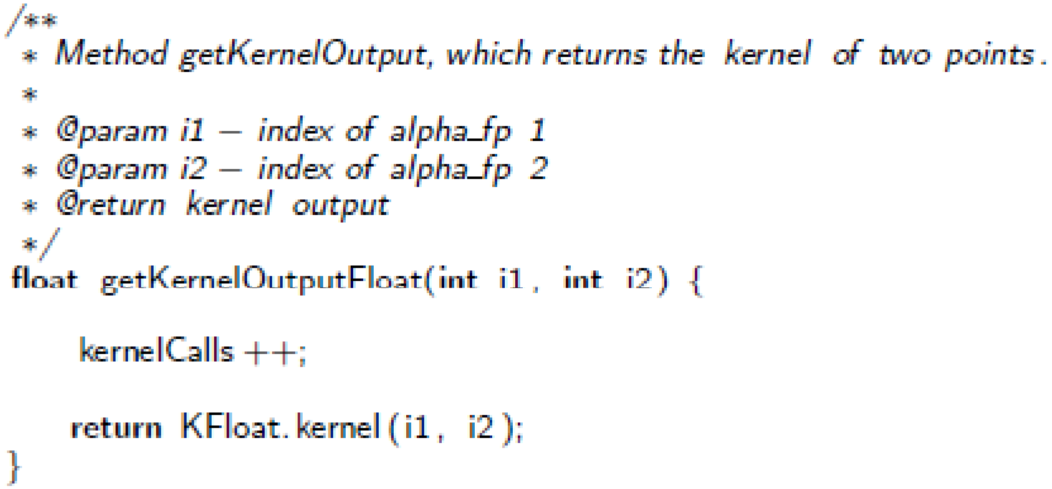
- Machine learning is important in a number of domains
- Java is widely used – also in machine learning systems
- Previous experience with JOP (multicore)



Support Vector Machines

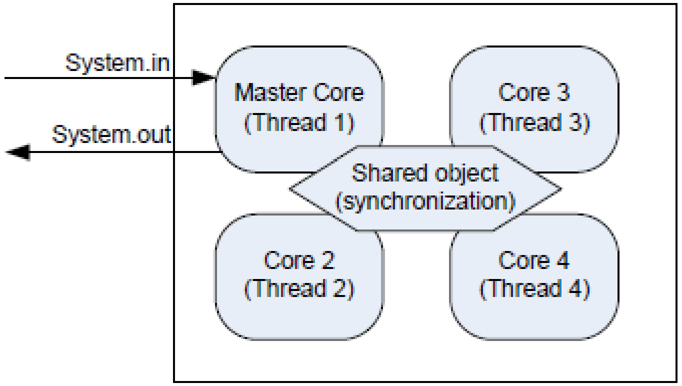
$$f(\boldsymbol{x}, \boldsymbol{\alpha}, b) = \{\pm 1\} = sgn\left(\sum_{i=1}^{l} \alpha_i y_i k(\boldsymbol{x}_i, \boldsymbol{x}) + b\right)$$

$$maximize\ W(\boldsymbol{\alpha}) = \sum_{i=1}^{l} \alpha_i \vdash \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} y_i y_j \alpha_i \alpha_j k(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

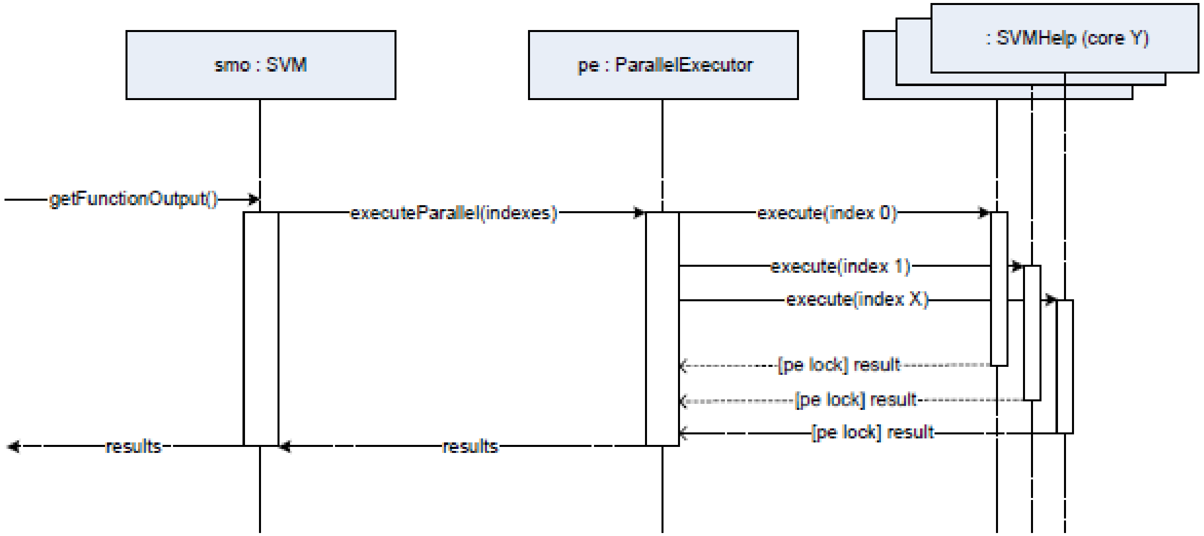


```
/**
 * Method getKernelOutput, which returns the kernel of two points.
 *
 * @param i1 - index of alpha_fp 1
 * @param i2 - index of alpha_fp 2
 * @return kernel output
 */
float getKernelOutputFloat(int i1, int i2) {

    kernelCalls++;

    return KFloat.kernel(i1, i2);
}
```

Implementation

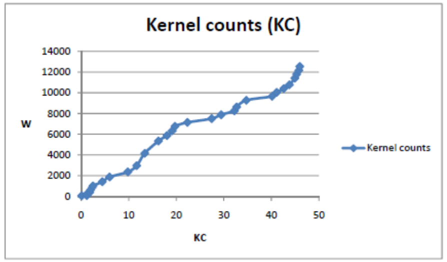


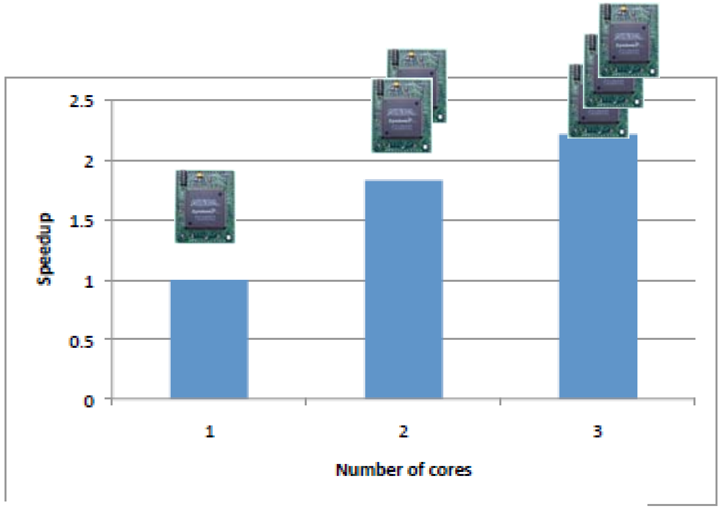
```
private static class Test implements Execute {
    final static int N = 100;
    static int a[] = new int[N];

  // the work method for one iteration
    public void execute(int nr) {
        a[nr] = nr;
    }

    public static void result () {
        for (int i=0; i<N; ++i) {
            System.out.println (a[i]);
        }
    }
}
```

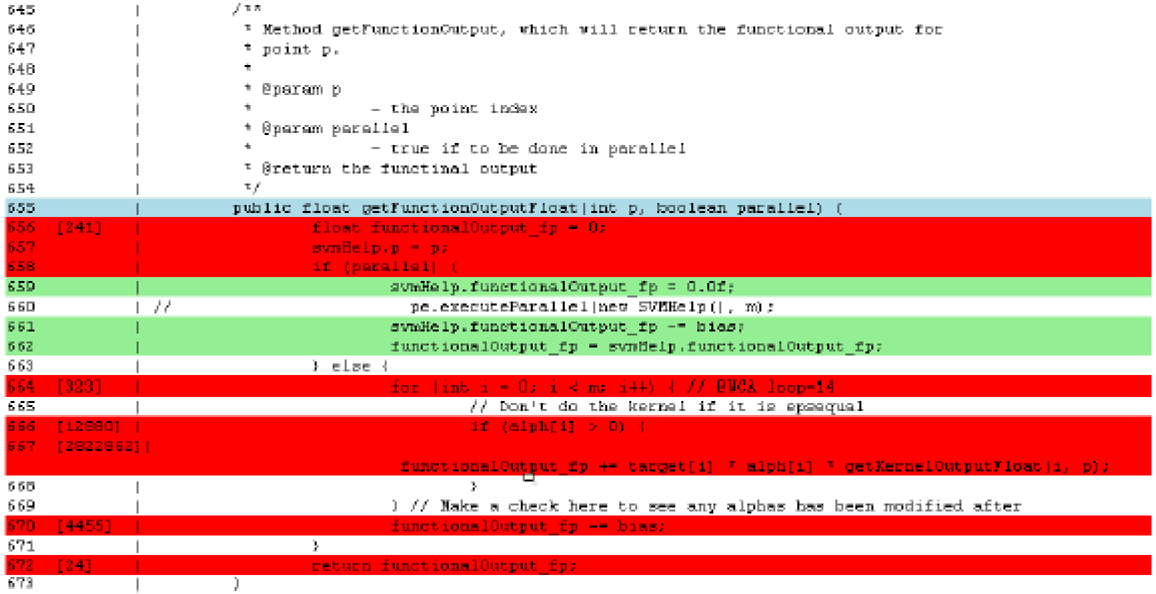


Discussion

- Hard-real time SVM







```
645             |          /**
646             |           * Method getFunctionOutput, which will return the functional output for
647             |           * point p.
648             |           *
649             |           * @param p
650             |           *             - the point index
651             |           * @param parallel
652             |           *             - true if to be done in parallel
653             |           * @return the functional output
654             |           */
655             |          public float getFunctionOutputFloat(int p, boolean parallel) {
656   [241]     |                  float functionalOutput_fp = 0;
657             |                  svmHelp.p = p;
658             |                  if (parallel) {
659             |                          svmHelp.functionalOutput_fp = 0.0f;
660             | //                         pe.executeParallel(new SVMHelp(), m);
661             |                          svmHelp.functionalOutput_fp -= bias;
662             |                          functionalOutput_fp = svmHelp.functionalOutput_fp;
663             |                  } else {
664   [323]     |                          for (int i = 0; i < m; i++) { // @WCA loop=14
665             |                                  // Don't do the kernel if it is epsequal
666   [12880]   |                                  if (alph[i] > 0) {
667   [2822862] |
                                                    functionalOutput_fp += target[i] * alph[i] * getKernelOutputFloat(i, p);
668             |                                  }
669             |                          } // Make a check here to see any alphas has been modified after
670   [4455]    |                          functionalOutput_fp -= bias;
671             |                  }
672   [24]      |                  return functionalOutput_fp;
673             |          }
```

Conclusion

- We achieved linear scalability for two cores
- Presented a popular machine learning algorithm
- Conclusion that objected oriented intelligent algorithms are prime for further investigation