Classifying Computation Using Machine-Level Features

David Sheffield, Michael Anderson, Kurt Keutzer

Motivation

• We suspect that the problems commonly solved using computers can be classified as a small number of distinct computations
  • e.g. matrix multiply, sorting, convolution, etc.

• We are interested in finding the distinguishing characteristics of these computations

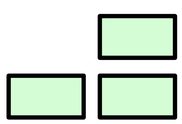
• Automatically detecting these patterns in software would allow us to suggest optimizations or libraries to the programmer. (An *intelligent* profiler)

• We could also predict how an arbitrary program would perform on a number of different architectures based on its composition of computations

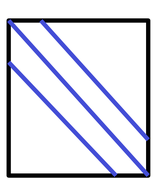
Computational Motifs

• 13 Computational Patterns were identified by a group of researchers from UC Berkeley and LLNL in 2006 [Asanovic et al].

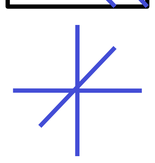
• We try to classify:
  • **Dense Linear Algebra**



  • **Sparse Linear Algebra**



  • **Structured Grid**



  • **Sort**

Machine-Level Features

• **Feature vector:**
  • Indirect Loads
  • Indirect Stores
  • Loads
  • Stores
  • Floating-point add/sub
  • Floating-point multiplies
  • Floating-point divides
  • Integer instructions

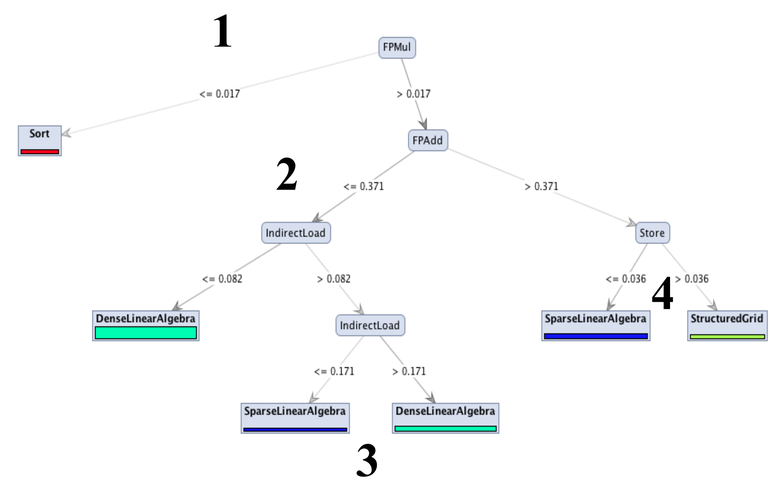
• **Assembly Code**

```
load r1, mem[r2]
mulsd xmm0, xmm1
addsd xmm2, dmm0
load r3, mem[r1]
. . .
```

• Arrow indicates an indirect load, common in sparse codes

• **Feature Collection**
  • Used Intel Software Development Emulator (SDE)
  • Picked the top “basic blocks” of assembly instructions
  • Extracted features by parsing and simulating the assembly
  • Compiled 42 training examples

Decision Tree Classifier



1. Sort does not use floating-point so very few FP Multiplies is a good indicator

2. Dense Linear Algebra tends to load directly and sequentially from memory

3. This is overfitting slightly. A few outliers in the Dense Linear Algebra training set (Givens Rotations) had many indirect loads

4. This reflects the fact that sparse matrix-vector multiply tends to accumulate results in registers, while structured grid computations write continuously to memory

Next Steps

• We plan on modifying GCC or LLVM to gather feature vectors
  •The compiler to insert counters in order to profile interesting events (such as indirect loads)
    •Similar in principle to traditional profilers such as gprof
•Consider adding data access patterns, including data structure shape
•Train classifier with known examples such as SPEC2006
  • LLNL has a computational pattern benchmark suite too.
•Use RPM package manager to rebuild complete Linux userland with pattern profiling code
  •Detect computational patterns in the “wild”