Concurrent Programming: Project Proposal

S Pradeep Kumar, Mahesh Gupta

2013-04-09 Tue

Problem Statement

Automatically extract parallelism from a sequential program for running on a multi-core processor

Area: Languages and Compilers

What are we going to do?

- Definitely: Given a program in Continuation-Passing Style (CPS) form, we will convert it into parallel CPS (pCPS), which will use the Futures synchronization construct to achieve speedup.
- Maybe: Given a program in Java (called MicroJava) we will convert it to an intermediate form called NanoJava which is in CPS.
- Possibly: Use some heuristics to avoid creating Futures for some parts of the code and treat loops specially so that we can extract more parallelization.

Why is it interesting?

- By automatically converting CPS code into pCPS, we can get faster programs without manually parallelizing the code.
- It is also interesting because we will use Futures to extract parallelism. We will start computations as usual but delay the actual extraction of the result as long as possible. This way we hope to have multiple computations running in parallel.

Language / Libraries

- We tentatively plan to use Java to implement the program which translates Sequential code to CPS code and then to pCPS code.
- Libraries:
  1. JavaCC for parsing the code
  2. Java Tree Builder (JTB) for building Visitors for the parse trees
- Algorithms:
  - We will use some standard algorithm (after surveying the literature) for pushing the use of a variable as far down as possible in a block of code. Note that this would mean transitively pushing other statements down as well to get the optimal code.

Testing and Experimentation strategy

- Experimenting with the speed of a naive pCPS translation and comparing against the sequential program
- Experimenting with the heuristics for inserting Futures to get more speedup

  e.g., using a Thread pool, delaying usage of the Future's result to maximize parallelism, etc.
- Experimenting with preserving loops to generate efficient code
- Experimenting with Tail-call optimization

Benchmarks

Matrix Multiplication, Signal convolution, Mergesort, Threshold program, Shuffle program

Timeline and Division of labour

- Definitely
  - ***2013-04-16 Tue*** Finish naively translating 4 programs manually to pCPS form and test speedup (if any)

    This is without thread pools, i.e., simply using Futures for parallelization
    - 2 programs [Pradeep]
    - 2 programs [Mahesh]
  - ***2013-04-20 Sat*** [Pradeep] Use Thread pools to reduce overhead of creating threads
  - ***2013-04-20 Sat*** [Mahesh] Manually push down the code using the return value of Futures and test speedup
  - ***2013-04-22 Mon*** [Pradeep] Manually preserve loops in CPS and optimize them
  - ***2013-04-22 Mon*** [Mahesh] Try finding a VM with tail-call optimization and test speedup
  - ***2013-04-26 Fri*** [Pradeep] Translate CPS to pCPS (which will have a Thread pool)
  - ***2013-04-26 Fri*** [Mahesh] Write algorithm to automatically push down the usage of Future's result
  - ***2013-05-03 Fri*** [Together] Final presentation
- Maybe
  - ***2013-05-04 Sat*** [Pradeep] Microjava to CPS
  - ***2013-05-04 Sat*** [Mahesh] Combining some Futures together
- Possibly
  - ***2013-05-06 Mon*** [Pradeep] Automatically preserve loops
  - ***2013-05-06 Mon*** [Mahesh] Avoid creating Futures for some parts of the code
- Definitely
  - ***2013-05-08 Wed*** [Together] Final report