An Approach for Direct Dataflow Execution on Contemporary Multicore Systems

Dumitrel Loghin, B.M. Tudor, Y.M. Teo

Department of Computer Science
National University of Singapore

3rd Workshop on Data-Flow Execution Models for Extreme Scale Computing, Edinburgh, 8 Sep 2013
Outline

• Introduction
• Related Work
• Approach
  – Executable Dataflow Graph
  – Runtime Dataflow Engine
• Evaluation
  – Effect of problem size and speedup
  – Effect of dataflow task granularity
  – Cost of synchronization
• Conclusions
Introduction

• Multicore are becoming ubiquitous

• Traditional imperative programming models
  – Explicit and coarse-grain parallelism

• Dataflow programming model
  – Implicit and fine-grain parallelism

• Dataflow machines are not widely available
Research Question

Can fine-grain dataflow programs be efficiently executed on current multicore systems?
Objective

Design, implement and evaluate a system for direct execution of dataflow programs on multicore systems
Related Work

• Dataflow architectures and simulators
  – Manchester Dataflow Machine (1985)
  – MIT Tagged-token Dataflow Architecture (1990)
  – Manchester Multi-ring Dataflow Simulation (1985)

• Dataflow execution on non-dataflow machines
  – Programming models and tools
    • Data-driven Multithreading (DDM) (2006)
    • Function-level dataflow execution (2011)
Approach

Two steps:

1. Executable Dataflow Graph (EDFG) generation

2. Execution of EDFG on Runtime Dataflow Engine
Executable Dataflow Graph

• Fine-grain DF graph to static EDFG
  – Node fusion
    • Fuse loop bodies
    • Fuse functions
  – Node optimization
    • Remove redundant housekeeping nodes
    • Rearrange nodes (e.g. Selector condition)
define main
  type IntArray = array[integer];
  function main(n: integer; A, B: IntArray
     returns IntArray)
    for i in 1, n
      returns array of A[i] + B[i]
  end for
end function
Node Optimization

if Condition then …
else …
end if
Runtime Dataflow Engine

1. Initial data flows to Static EDFG.
2. Tag Generator receives node<instr,op,dest> from Static EDFG and generates node<tag,instr,op,dest>.
3. Dynamic Dataflow Graph receives node<tag,instr,op,dest> and generates dataflow tasks.
4. Result tokens flow from Tag Generator to Worker 1, Worker 2, ..., Worker m, which are connected to the Task Pool.
Evaluation

• Programs
  – $\text{MM}(n)$ – matrix multiplication on square matrices of size $n$
  – $\text{PR}(n)$ – prime number counting in range 1, $n$

• Granularities
  – $\Phi_1$ - no fusion
  – $\Phi_2$ - fused inner-most loop body or function
  – $\Phi_3$ - fused all, except the outer-most loop
  – $\Phi_4$ - fused outer-most loop until number tasks equals number of cores
  – $\Phi_5$ - fused into one dataflow task

• System
  – 48-core AMD Opteron, 64GB memory (NUMA)
Effect of Problem Size using $\Phi_3$

O1: Low overhead of dataflow task management on medium-size granularity
Effect of Granularity

$\Phi_1$ - fine-grain, ran out of memory
$\Phi_5$ – sequential, close to sisalc

O2: Medium granularity achieves the best performance.
O3: Medium-large granularity suffers from lack of work.
Cost of Synchronization – MM(n=2000)

- VNI – von Neumann instructions (perf)
- S - # of synchronization in OS kernel (strace)

O4: Higher cost of synchronization for fine-grain dataflow tasks.
Conclusions

• An approach for direct execution of dataflow programs on multicore systems
  – EDFG with node fusion optimizations with different task granularities
  – RDE for multithreaded direct EDFG execution

• Preliminary Evaluation
  – Medium size granularities achieve good performance
  – Medium-large size granularity suffers from work shortage
  – Fine-grain dataflow tasks execution suffer from high synchronization overhead
Future Work

• Adaptive application execution
  – Generation of EDGF with malleable tasks
  – RDE with dynamic task granularity selection and execution

• Heterogeneous dataflow execution
  – CPU-GPGPU
  – von Neumann-dataflow

• Use other languages as starting point
Q&A

Thank you!

dumitrel [at] comp.nus.edu.sg