Semi-Static Dataflow

Bruno R. Preiss
Department of Electrical Engineering
University of Waterloo
Waterloo, Ontario, Canada, N2L 3G1

V. Carl Hamacher
Departments of Electrical Engineering and Computer Science
University of Toronto
Toronto, Ontario, Canada, M5S 1A4

This research was supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada under Grants A5192 and OGP0036635.
Outline

- Background and Motivation
  - Static and Dynamic Dataflow
  - Semi-Static Dataflow
- Contexts
- Communication Primitives
- Dynamic Dataflow Graph Splicing
  - Function Invocation Paradigm
  - Sequential Iteration Paradigm
  - Parallel Iteration Paradigm
- Simulation Results
- Modified Derivation of Amdahl’s Law
Static Dataflow Architectures

- graph nodes allocated statically
- single activation of nodes
- impure code
- activity templates:
  - opcode + operand slots + destination pointers
- untagged tokens
- finite arc capacity

Dynamic Dataflow Architectures

- nodes instantiated at run time
- simultaneous, multiple activations
- pure code
- activity templates:
  - opcode + destination pointers
- tagged tokens
- unbounded arc capacity

Semi-Static Dataflow
Semi-Static Dataflow

- activity templates like static:
  - opcode + operand slots + destination pointers
- pure code like dynamic
- untagged tokens like static
- simultaneous, multiple activations like dynamic
Contexts

- small ↔ medium granularity “process”
- evaluates *acyclic dataflow graph*
- components:
  - static dataflow graph
  - data token space
  - communication channel pair:
    - “input” channel
    - “output” channel
- *dynamic context creation used for:*
  - function invocation
  - iteration
  - conditional execution
Communication Primitives

- channels — unidirectional communication path between two contexts
- established:
  - dynamically — created as contexts are created
  - statically — declared by the programmer
- channel capacity (width) = 1 (scalar) token
- unbuffered, synchronous communication (‘‘rendezvous’’)
- blocking ‘‘send’’ and ‘‘receive’’ primitives

Semi-Static Dataflow
Context Generation Primitives

- **“rfork”** *(recursive fork)*
  - two inputs, three outputs
  - effect:
    - new context + two new channels (input & output)

- **“ifork”** *(iterative fork)*
  - three inputs, two outputs
  - effect:
    - new context + one new channel (input)
    - inherit parent context’s output channel

Function Invocation

• dynamic dataflow graph splicing:
  I  context generation phase
  II parameter passing phase
  III concurrent computation phase
  IV result passing phase
Function Invocation — Example

proc f (var y, value x) =
    y := x + 1:
var result:
f (result, 5)
Sequential Iteration

- accomplished via dynamic dataflow graph splicing
- uses 3 graphs:
  - main program graph
  - loop body graph
  - loop terminator graph
- sample program:

```plaintext
var sum, result:
seq
  sum := 0
  seq i = [1 for 10]
    sum := sum + i
  result := sum
```
Sequential Iteration — Dataflow Graphs

Semi-Static Dataflow
Parallel Iteration

- provides dynamic process creation
- accomplished using dynamic dataflow graph splicing
- uses 3 graphs:
  - main program graph
  - loop body graph
  - loop terminator graph
- example program:

```plaintext
par i = [1 for 10]
    Process (i)
```
Parallel Iteration — Dataflow Graphs

Semi-Static Dataflow

Copyright (c) 1988 by Bruno R. Preiss, P.Eng. All rights reserved.
Modified Derivation of Amdahl’s Law

- **Workload:**
  - **user:** $X^U$, time $T_n^U$
  - **kernel:** $X^K$, time $T_n^K$
- **Total time:** $T_n = T_n^U + T_n^K$
- **Throughput:** $X^U/T_n$
- **Throughput ratio:** $R_n = \frac{X^U/T_n}{X^U/T_1} = T_1/T_n$
- **User workload:** $T_n^U = S^U + P^U/n$
- **Kernel workload:**
  - **assumption:** independent kernel calls: $T_n^K = Y/n$
  - **assumption:** some of workload is proportional to the number of processes on a processor: $Y = P^K/n + S^K$
**Modified Amdahl’s Law (continued)**

- \( R_n = \frac{1}{1 + (1/n - 1)f + (1/n^2 - 1)g} \)

- \( f = \frac{S^K + P^U}{P^K + S^K + P^U + S^U}, \quad g = \frac{P^K}{P^K + S^K + P^U + S^U} \)

- \( \lim_{n \to \infty} R_n = \frac{1}{1 - (f + g)}, \quad \frac{dR_n}{dn} \bigg|_{n=1} = f + 2g \)

- **N.B.:** \( S^U < P^K \implies f + 2g > 1 \) superlinearity

\[
R_n
\]

\[
f = 0.5, \quad g = 0.45
\]

- neglecting kernel overhead \( \implies f = \frac{P^U}{P^U + S^U} \) and \( g = 0 \)

- Amdahl’s law: \( R_n = \frac{1}{1 + (1/n - 1)f} \)
Compiler

- “Occam” language
- partitioning rules: separate DFG for each
  - procedure + outcome of conditional + loop body

Architectural Model

- shared-bus multiprocessor
- processing elements optimized for acyclic graph execution “Queue machines”
- message-passing via message caches

Benchmark Programs

<table>
<thead>
<tr>
<th>program</th>
<th>S</th>
<th>G</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix Multiplication</td>
<td>15</td>
<td>8</td>
<td>236</td>
</tr>
<tr>
<td>Fast Fourier Transform</td>
<td>63</td>
<td>17</td>
<td>587</td>
</tr>
<tr>
<td>Cholesky Decomposition</td>
<td>74</td>
<td>22</td>
<td>428</td>
</tr>
<tr>
<td>Congruence Transformation</td>
<td>66</td>
<td>18</td>
<td>434</td>
</tr>
</tbody>
</table>

Semi-Static Dataflow
Simulation Results

Legend:
- ×× Matrix Multiplication
- ○○ Fast Fourier Transform
- — Cholesky Decomposition
- •• Congruence Transformation

Semi-Static Dataflow
Summary

- semi-static dataflow:
  - programs partitioned into collection of acyclic dataflow graphs
  - graphs evaluated by low-level processes (contexts)
  - contexts created dynamically
- paradigms for:
  - function invocation
  - sequential iteration
  - parallel iteration
- Occam compiler
- simulation results show promise
- results justified using a modified derivation of Amdahl’s law