Spill-Free Parallel Scheduling of Precedence Graphs

Balas Natarajan
Michael Schlansker
Hewlett Packard Labs
Palo Alto, CA

Motivation

Increased register pressure from

- Wider instruction words => more concurrent live ranges.
- Code motion across longer paths => longer live ranges.

Need effective techniques for integrating register allocation and ILP scheduling.

Given a precedence graph G.

Each vertex is an op.
Edges may be data flow or precedence edges.

Machine model M

- multiple units, can be pipelined, heterogenous.
- limited number of registers r.

Problem: Find the shortest schedule for G on M without spill.

Even simplified versions of problem are hard.

Min. length scheduling Minimizing schedule length with infinite registers is NP-hard.

Register Sufficiency Finding any spill-free schedule on bounded registers is NP-hard.

Good News

Min. Length Scheduling: Critical path scheduling is within a factor of 2 of optimal, and better in practice.

Register Sufficiency:

- Sethi-Ullman allocation is optimal for trees.
- Klein et al is within a log factor for general DAGS.

Integrated Scheduling:

- Prepass scheduling. Schedule before register allocation.
- Postpass scheduling. Schedule after register allocation.
- Alternating -- Goodman and Hsu.

Simplifying assumptions

- Binary graph; Indegree of each vertex is at most two.
- Single sink. Otherwise combine with zero latancy vertices and edges.

Results

Strategy

Convert an insensitive scheduling algorithm to a sensitive one using a divide-and-conquer paradigm.

Insensitive algorithm is your best scheduling algorithm that is insensitive to register pressure.

Let G_v be subgraph rooted at vertex v.

The weight of v is

W(v) = #sources in G_v +
(#edges in G_v - #vertices in G_v) + 1;

Prop. 1: Every schedule for v can be computed using at most $W(v)$ registers.

Prop. 2: If $W(v) \geq 2$, the there exists $u \in G_v$ such that

$1/4 W(v) \leq W(u) \leq 1/2 W(v)$.

Algorithm *sched*(a)

if insensitive algorithm requires
more than r registers for G **then**
find b such that $W(b) \sim W(a)/2$;
sched(b);
sched(a);
else construct insensitive schedule for a.

Claim

$$L_r \le 2^j L_\infty$$

where

- L_r is the length of the r register schedule.
- L_∞ is the length of the insensitive schedule.
- j is the smallest integer such that

$$(3/4)^j R \le r - jt$$

- $R = W(G)$
- t: thickness of G; deviation from a tree.

--

--

$$thickness = \max_{v} \left[\#edges\ from\ G_v\ to\ G - G_v \right]$$

Interpretation of Performance Bound

r_1: fewest number of registers at which spill free schedule exists.

r_2: fewest number of registers sufficient for insensitive schedule.

In the range $r_1 \leq r \leq r_2$,

$$L_r \sim (R/r)^{2.4} L_\infty .$$

A 10% decrease (increase) in the available registers

=>

a 24% increase (decrease) in the schedule length.

In the algorithm splits G, all of G_b is scheduled before any of $G - G_b$.

This could idle functional units.

Solution: Start computing $G - G_b$ as soon as possible.

Modified Algorithm

Algorithm *msched*(a)

if insensitive algorithm requires
    more than r registers for G_a **then**
    find b such that $W(b) \sim W(a)/2$;
    sched(b);
    sched(a);
else
    schedule next cycle of insensitive
    schedule for a.

Experiments

Basic blocks from the SPEC benchmarks using the IMPACT compiler. (Could be Hyperblocks or Superblocks.)

Machine Models studied: (not pipelined)

Model I	1 IPU	1 FPU	1 L/S
Model II 2 IPU	1 FPU	1 L/S	
Model III	2 IPU	2 FPU	2 L/S

Latencies

Iop	1
Fop	4
Load	2
Store	1

Insensitive scheduler: Critical path.

Summary

General divide-and-conquer paradigm for converting insensitive scheduler to sensitive one.

Extensions

- Branches with statistics.
- Spill

Run Time

Insensitive algorithm: $O(T(n+e))$, say;

Basic Algorithm: $O(T(n+e)log(n+e))$.

Modified Algorithm: $O(T^2(n+e)log(n+e))$.