Register Pressure in Software-Pipelined Loop Nests

Fast Computation and Impact on Architecture Design

Alban Douillet   Guang R. Gao

{douillet,ggao}@capsl.udel.edu
Department of Electrical & Computer Engineering
University of Delaware

18^{th} International Workshop
on Languages and Compilers for Parallel Computing
Hawthorne, New York
October 20^{th}-22^{nd}, 2005

Introduction

- Scientific Applications
  - loop nests dominant
- Single-dimension Software Pipelining (SSP)
  - software pipelines most profitable loop in loop nest
  - high register pressure
  - register allocation is time-consuming
- Need for a fast method to evaluate register pressure
  - detect infeasible schedules before calling the register allocator
  - measure quality of register allocation solution
  - give estimate of register needs for future architecture designs

Outline

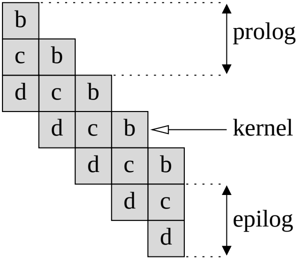
Outline

1. **Loop Nest Software-Pipelining**
2. Problem Statement
3. Definitions & Issues
4. Fast Register Pressure Computation
5. Experiments
6. Conclusion

Modulo Scheduling

- most popular SWP technique
- well studied and understood
- full array of loop optimizations
- single loop, parallel execution of iterations
- new iteration issued every T cyles (initiation interval)

```
FOR J=0,4
    b
    c
    d
END FOR
```



Modulo Scheduling

But...

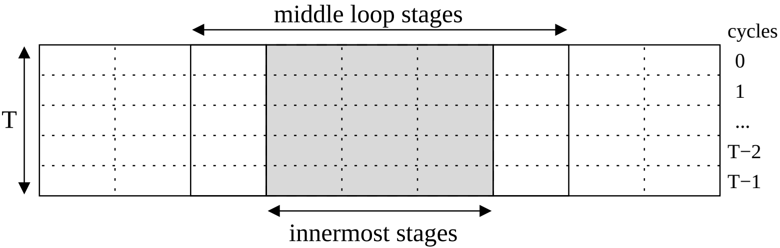
- limited to innermost loop
- loop transformations to bring ILP or data cache reuse potential to innermost loop not always possible

Single-Dimension Software-Pipelining (SSP)

- proposed by Rong et al. (CGO'04, PLDI'05)
- software pipelines the most profitable loop level in a loop nest
- equivalent to MS if innermost level selected
- can be seen as generalization of MS to loop nests
- proven performance boost vs. MS
- can take advantage of loop optimizations used for MS
- single-dimension b/c simplifies multi-dimensional DDG into a uni-dimensional DDG

SSP Kernel

- SSP generates a kernel similar to MS
- enclosed stages
- single initiation interval T
- L_1 is the outermost loop and L_n the innermost
- S_i: number of stages at level i

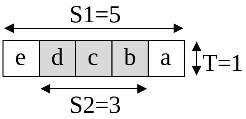


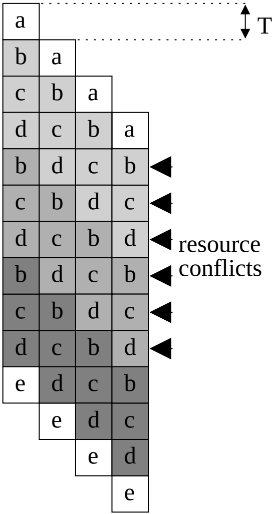
SSP Ideal Schedule

- Generated using kernel as a template
- new outermost iteration issued every T cycles
- outermost iterations executed in parallel
- inner iterations executed sequentially within one outermost iteration
- resource conflicts!

SSP Ideal Schedule: Example

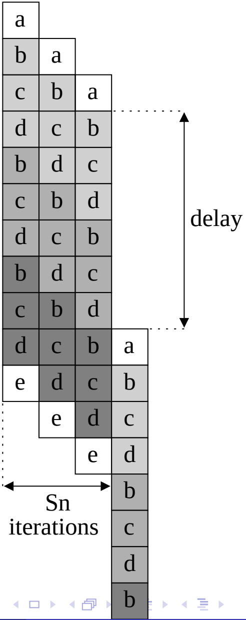
```
FOR I=1,4
    a
    FOR J=1,3
        b
        c
        d
    END FOR
    e
END FOR
```





SSP Final Schedule

- delays some outermost iterations to avoid resource conflicts
- outermost iterations executed in groups of S_n
- resource conflict-free schedule



SSP Loop Patterns

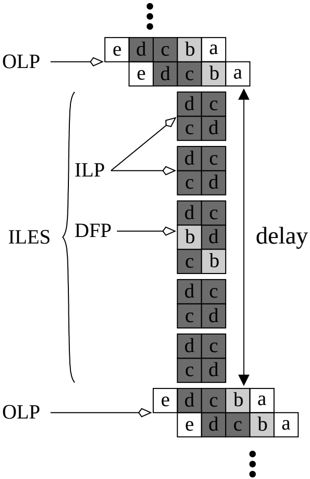
Patterns:

- Outermost Loop Pattern
- Inner Loop Execution Segment
  - Innermost Loop Pattern
  - Draining & Filling Pattern

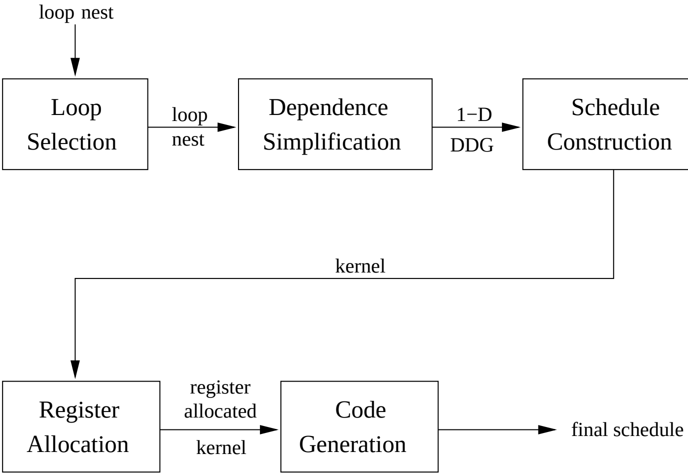
Composition:

- OLP: all S kernel stages
- ILES: cyclic combination of S_n consecutive stages

Kernel



SSP Implementation



Outline

Motivation

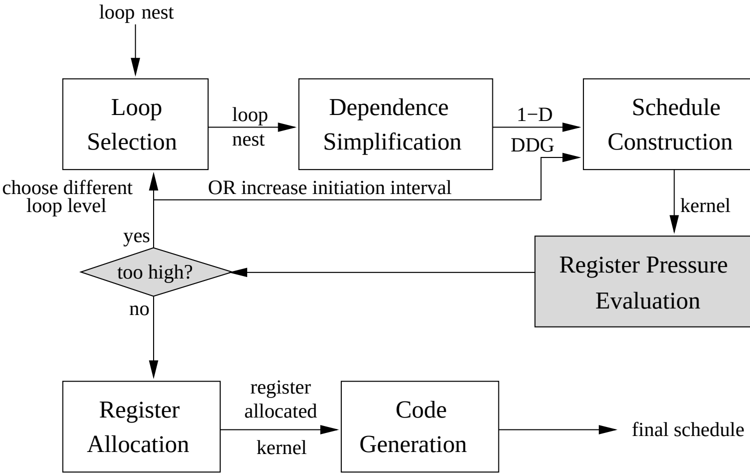
- need to determine feasibility of schedules
  - register allocation is time-consuming
  - unfeasible schedules b/c of high register pressure not uncommon
- need to evaluate quality of register allocator
  - how far from optimal solution?
- need to evaluate actual register needs for architectural designs
  - are register files big enough?

Problem Statement

Given a loop nest and an SSP schedule for it, evaluate the register pressure MaxLive of the final schedule.

- only rotating registers
- *MaxLive* = maximum number of live variables at any given cycle in the final schedule
- *MaxLive* definition similar to the one for MS.

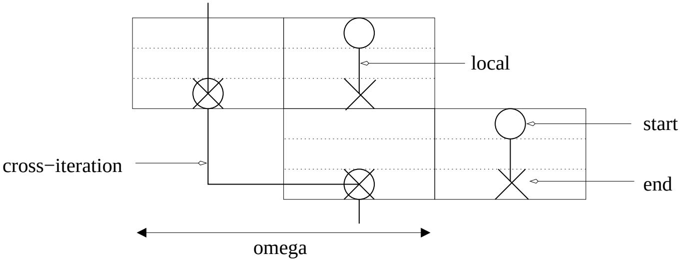
Updated SSP Implementation



Outline

Definitions

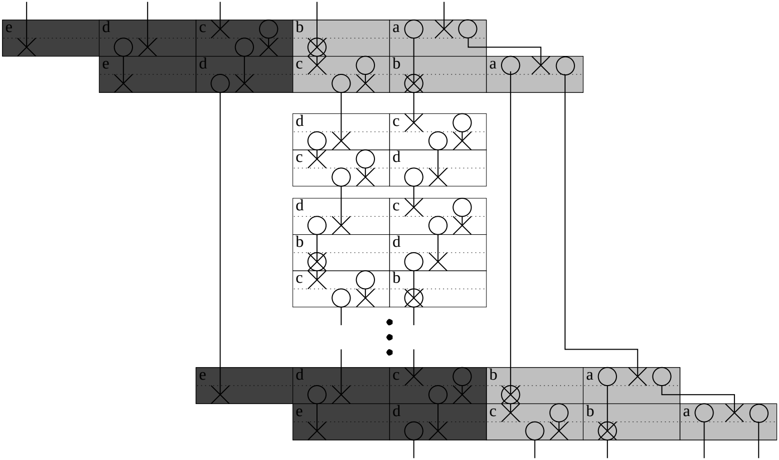
- scalar lifetime:
  - start: definition cycle of the value
  - end: kill cycle of the value
  - omega: number of outermost iterations spanned
- classification
  - global: constant values, ignored
  - input & output: prolog and epilog, ignored
  - local: within same outermost iteration
  - cross-iteration: between outermost iterations



Issues

- more complex lifetime patterns than MS
  - non-constant initiation rate
  - stretched lifetimes
- same stage may have different lifetimes patterns
  - a stage is not always followed by the same stages
  - difference between *first* and *last* instance of the same stage

Lifetimes Example



Outline

Method Overview

Method Keys:

- separate OLP from ILES instances of stages
- separate *first* from *last* instances of stages
- separate local from cross-iteration lifetimes

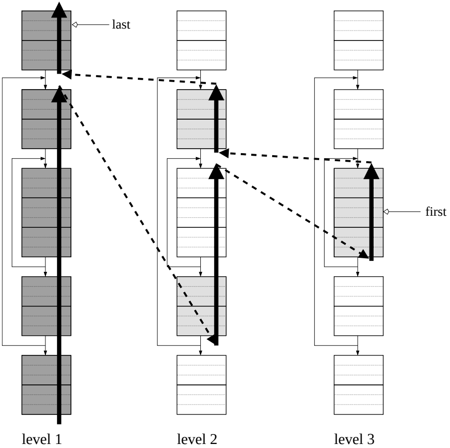
Steps:

- count number of local lifetimes in *first* instance of stages
- count number of local lifetimes in *last* instance of stages
- count number of cross-iteration lifetimes in each stage
- list all possible combinations of stages in schedule
- add number of lifetimes for each combination in OLP and ILES
- *MaxLive* is the highest value

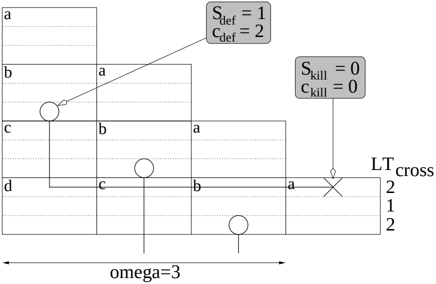
Local Lifetimes

- traditional liveness analysis
- computes for both *first* and *last* instances of stage s
  - each cycle c in stage between 0 and $T - 1$
  - live-out set of stage $(c = T)$
- stage of level i visited i times

$$LT_{local}(s, c, first/last)$$



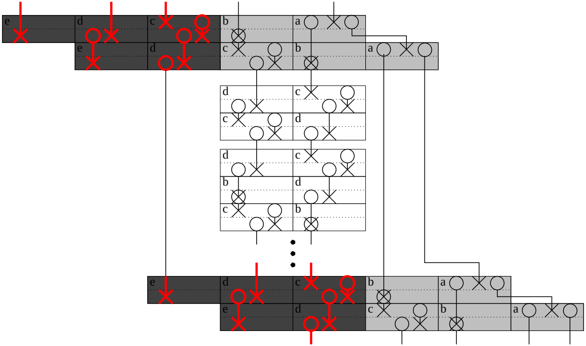
- need stage and cycle of definition and kill
- direct formula
  - for each cycle c in OLP between 0 and $T-1$
  - not stage-specific



$$LT_{cross}(c) = \sum_{v \in civs} ((S_{kill}(v) - S_{def}(v) + 1) + \delta_{def}(c,v) + \delta_{kill}(c,v))$$

$$\text{where } \begin{cases} \delta_{def}(c,v) = -1 \text{ if } c < c_{def}(v), \text{ 0 otherwise} \\ \delta_{kill}(c,v) = -1 \text{ if } c > c_{kill}(v), \text{ 0 otherwise} \end{cases}$$

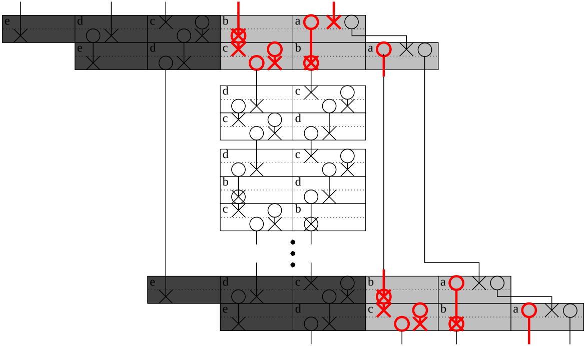
OLP: *last* stages local lifetimes



Combination of *first* and *last* stages not always the same

$$\sum_{s=l_n-i}^{l_1} LT_{local}(s, c, last)$$

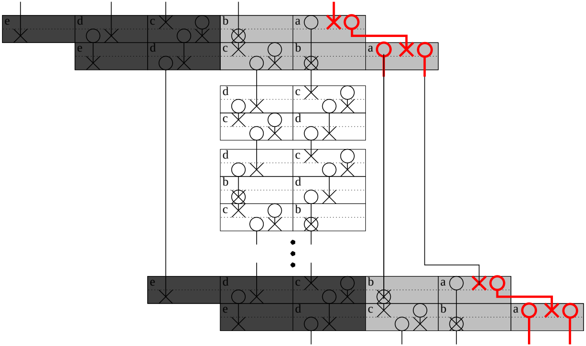
OLP: *first* stages local lifetimes



Combination of *first* and *last* stages not always the same

$$\sum_{s=f_1}^{l_n-1-i} LT_{local}(s, c, first)$$

OLP: cross-iteration lifetimes



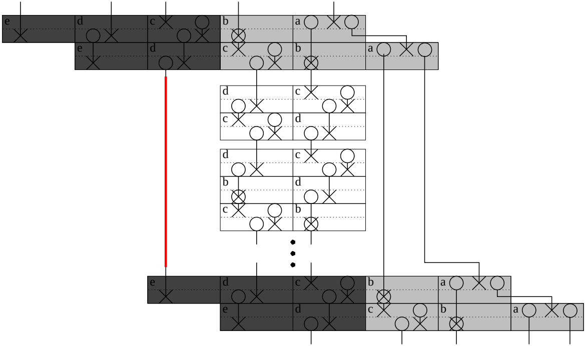
Number of cross-iteration lifetimes identical between instances of OLP

$$LT_{cross}(c)$$

OLP count

$$LT_{olp}(c) = LT_{cross}(c) + \max_{i \in [1, S_n]} \left(\sum_{s=l_n-i}^{l_1} LT_{local}(s, c, last) + \sum_{s=f_1}^{l_n-1-i} LT_{local}(s, c, first) \right)$$

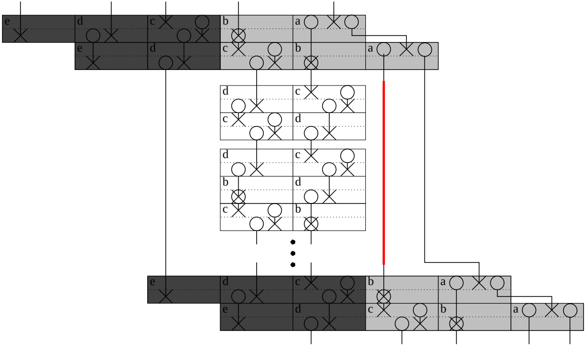
ILES: *last* stretched local lifetimes



Live-out of *last* stages

$$\sum_{s=l_n}^{l_1} LT_{local}(s, T, last)$$

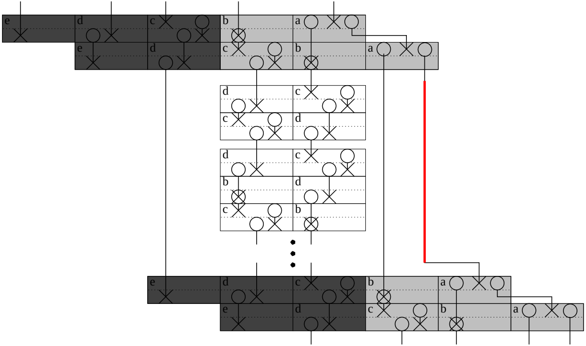
ILES: *first* stretched local lifetimes



Live-out of *first* stages

$$\sum_{s=f_1}^{f_n-2} LT_{local}(s, T, first)$$

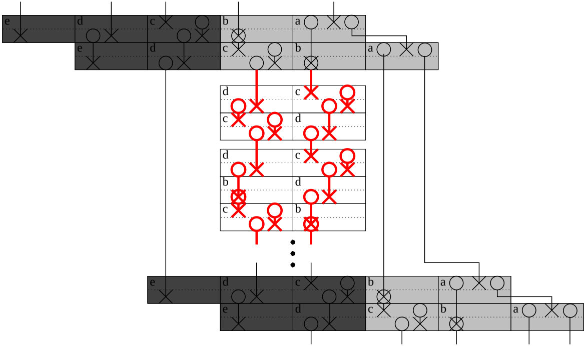
ILES: stretched cross-iteration lifetimes



live-out cross-iteration lifetimes from OLP

$$LT_{cross}(T)$$

ILES: local lifetimes



S_n consecutive stages (cyclic)

$$\max_{l \in [2,n]} \left(\max_{i_0 \in [0, S_l - 1]} \left(\sum_{i=0}^{S_n - 1} LT_{local}(f_l + (i_0 + i) \% S_l, c, first) \right) \right)$$

ILES count

$$
\begin{aligned}
LT_{iles}(c) &= LT_{cross}(T) \\
&+ \sum_{s=l_n}^{l_1} LT_{local}(s, T, last) \\
&+ \sum_{s=f_1}^{f_n-2} LT_{local}(s, T, first) \\
&+ \max_{l \in [2,n]} \left(\max_{i_0 \in [0, S_l - 1]} \left(\sum_{i=0}^{S_n-1} LT_{local}(f_l + (i_0 + i)\% S_l, c, first) \right) \right)
\end{aligned}
$$

MaxLive

$$
\begin{aligned}
FatCover_{olp} &= \max_{\forall c \in [0, T-1]} \left(LT_{olp}(c) \right) \\
FatCover_{iles} &= \max_{\forall c \in [0, T-1]} \left(LT_{iles}(c) \right) \\
MaxLive &= \max\left(FatCover_{iles}, FatCover_{olp} \right)
\end{aligned}
$$

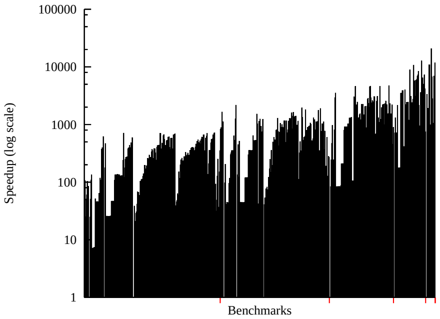
Outline

Experimental Framework

- ORC2.1 compiler
- 1.4Ghz Itanium workstation, 1GB RAM
- Livermore, SPEC2000 FP, NPB 2.2 benchmarks
- 127 loop nests
- 328 test cases

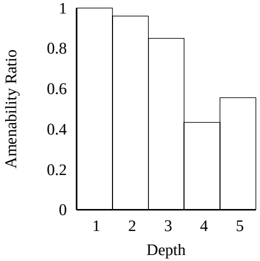
Running Time

- in the order of 1/1000 sec
- quadratic running time
- 3 orders of magnitude faster than register allocator
- speedup increases as loop gets deeper
- $\Rightarrow$ fast enough to use in SSP framework



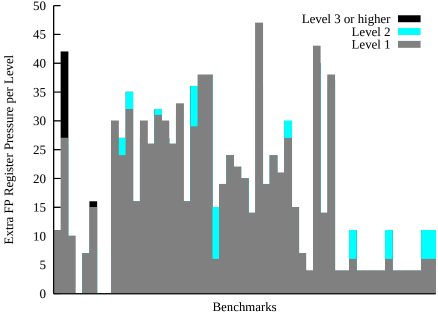
MaxLive

- Average: INT=42 FP=15
- register pressure too high for 43% of loop nests of depth 4



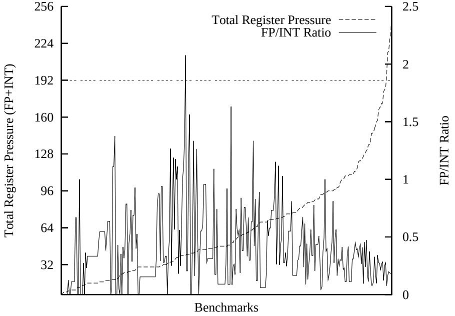
FP/INT Comparison

- FP register pressure stable as loop gets deeper
  - pressure never exceeds 64 registers
- INT register pressure increases as loop gets deeper
  - loop overheads
  - array indexes
  - longer live ranges



Register File Size

- max FP register file size: 64
- ideal INT/FP ratio: 2
- 77% and 67% of loop nests of depth 4 and 5 would become feasible



Outline

Conclusion

- SSP
  - software-pipelines loop nests at the most profitable level
  - register pressure is however very high
- need for fast method to evaluate register pressure
  - detect infeasible schedules early
  - measure quality of register allocator
  - give an estimate of the actual register needs for future architecture designs
- proposed solution
  - deals with issues specific to loop nest SWP and SSP
  - is very fast and can be used before register allocation
- future work
  - incremental solution to be integrated to the scheduler
  - different architectures: clustered VLIW,...