Constraint Programming in Compiler Optimization: Lessons Learned

Peter van Beek
University of Waterloo
Acknowledgements

• Joint work with:
  Omer Beg
  Alejandro López-Ortiz
  Abid Malik
  Jim McInnes
  Wayne Oldford
  Claude-Guy Quimper
  John Tromp
  Kent Wilken
  Huayue Wu

• Funding:
  NSERC
  IBM Canada
Application-driven research

• Idea:
  • pick an application—a real-world problem—where, if you solve it, there would be a significant impact

• Along the way, if all goes well, you will also:
  • identify and fill gaps in theory
  • identify and solve interesting sub-problems whose solutions will have general applicability
Optimization problems in compilers

• Instruction selection
• Instruction scheduling
  • basic-block scheduling
  • super-block scheduling
  • loop scheduling: tiling, unrolling, fusion
• Memory hierarchy optimizations
• Register allocation
Optimization problems in compilers

- Instruction selection
- Instruction scheduling
  - basic-block scheduling
  - super-block scheduling
- Loop scheduling: tiling, unrolling, fusion
- Memory hierarchy optimizations
- Register allocation
“At the outset, note that basic-block scheduling is an NP-hard problem, even with a very simple formulation of the problem, so we must seek an effective heuristic, rather than an exact approach.”

Outline

• Introduction
  • computer architecture
  • superblock scheduling
• Constraint programming approach
  • temporal scheduler
  • spatial and temporal scheduler
• Experiments
  • experimental setup
  • experimental results
• Lessons learned
Computer architecture:
Performing instructions in parallel

- Multiple-issue
  - multiple functional units; e.g., ALUs, FPUs, load/store units, branch units
  - multiple instructions can be issued (begin execution) each clock cycle
  - *issue width*: max number of instructions that can be issued each clock cycle
  - on most architectures issue width less than number of functional units
Computer architecture: Performing instructions in parallel

- Pipelining
  - overlap execution of instructions on a single functional unit
  - latency of an instruction
    number of cycles before result is available
  - execution time of an instruction
    number of cycles before next instruction can be issued on same functional unit
  - serializing instruction
    instruction that requires exclusive use of entire processor in cycle in which it is issued

Analogy: vehicle assembly line
Superblock instruction scheduling

- Instruction scheduling
  - assignment of a clock cycle to each instruction
  - needed to take advantage of complex features of architecture
  - sometimes necessary for correctness (VLIW)

- Basic block
  - straight-line sequence of code with single entry, single exit

- Superblock
  - collection of basic blocks with a unique entrance but multiple exits

- Given a target architecture, find schedule with minimum expected completion time
**Example superblock**

*dependency DAG*

- **nodes**
  - one for each instruction
  - labeled with execution time
  - nodes F and G are branch instructions, labeled with probability the exit is taken

- **arcs**
  - represent precedence
  - labeled with latencies
Example superblock

optimal cost schedule for 2-issue processor

<table>
<thead>
<tr>
<th>cycle</th>
<th>ALU</th>
<th>FPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>E</td>
</tr>
<tr>
<td>9</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>G</td>
<td></td>
</tr>
</tbody>
</table>
Computer architecture:
General purpose architectures
Computer architecture: Clustered architectures
Computer architecture: Clustered architectures

• Current: digital signal processing
  • multimedia, audio processing, image processing
  • wireless, ADSL modems, …

• Future trend: general purpose multi-core processors
  • large numbers of cores
  • fast inter-processor communication
Spatial and temporal scheduling

<table>
<thead>
<tr>
<th>cycle</th>
<th>c0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>D</td>
</tr>
<tr>
<td>5</td>
<td>E</td>
</tr>
<tr>
<td>6</td>
<td>F</td>
</tr>
<tr>
<td>7</td>
<td>G</td>
</tr>
<tr>
<td>8</td>
<td>H</td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

cost = 9.8

<table>
<thead>
<tr>
<th>cycle</th>
<th>c0</th>
<th>c1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>D</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>G</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>H</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

cost = 7.6
Spatial and temporal scheduling

cost = 7.6
Approaches

• Superblock instruction scheduling is NP-complete

• Heuristic approaches in all commercial and open-source research compilers
  • greedy list scheduling algorithm coupled with a priority heuristic

• Here: Optimal approach
  • useful when longer compile times are tolerable
  • e.g., compiling for software libraries, digital signal processing, embedded applications, final production build
Outline

- Introduction
  - computer architecture
  - superblock scheduling
- Constraint programming approach
  - temporal scheduler
  - spatial and temporal scheduler
- Experiments
  - experimental setup
  - experimental results
- Lessons learned
Temporal scheduler: Basic constraint model

variables
A, B, C, D, E, F, G

domains
{1, ..., m}

constraints
B \geq A + 1, \ C \geq A + 1, \\
D \geq B + 5, \ldots, \ G \geq F

\text{gcc}(A, B, C, F, G, nALU)
\text{gcc}(D, E, nFPU)
\text{gcc}(A, \ldots, G, issuewidth)

cost function
40 \times F + 60 \times G
Temporal scheduler
Basic constraint model (con’t)

non-fully pipelined instructions

• introduce auxiliary variables
  \[ P_{B,1} \]
  \[ P_{B,2} \]

• introduce additional constraints
  \[ B + 1 = P_{B,1} \]
  \[ B + 2 = P_{B,2} \]
  gcc(A, B, P_{B,1}, P_{B,2}, C, F, G, nALU)

serializing instructions

• similar technique
Temporal scheduler: Improving the model

- Add constraints to increase constraint propagation (e.g., Smith 2006)
  - implied constraints: do not change set of solutions
  - dominance constraints: preserve an optimal solution

- Here:
  - many constraints added to constraint model in extensive preprocessing stage that occurs once
  - extensive preprocessing effort pays off as model is solved many times
Temporal scheduler: Improving the solver

- From optimization to satisfaction
  - find bounds on cost function
  - enumerate solutions to cost function (knapsack constraint; Trick 2001)
  - step through in increasing order of cost
- Improved bounds consistency algorithm for gcc constraints
- Use portfolio to improve performance (Gomes et al. 1997)
  - increasing levels of constraint propagation
- Impact-based variable ordering (Refalo 2004)
- Structure-based decomposition technique (Freuder 1994)
Spatial and temporal scheduler: Basic constraint model

variables
- cycle of issue: $x_A, x_B, \ldots, x_H$
- cluster: $y_A, y_B, \ldots, y_H$

domains
- $\text{dom}(x) = \{1, \ldots, m\}$
- $\text{dom}(y) = \{0, \ldots, k-1\}$

communication constraints
- $y_A \neq y_C \rightarrow x_C \geq x_A + 1 + \text{cost}$
- $y_A = y_C \rightarrow x_C \geq x_A + 1$
- ...

cost function
- $80 \times x_H + 20 \times x_G$
Spatial and temporal scheduler: Search tree of basic model

\begin{align*}
\mathbf{y} &= (0, 0, 0, 2) \\
\mathbf{A} &= \mathbf{y} \\
\mathbf{B} &= \mathbf{y} \\
\mathbf{C} &= \mathbf{y} \\
\mathbf{D} &= \mathbf{y}
\end{align*}
Spatial and temporal scheduler: Improving the model

- Symmetry breaking
  - add auxiliary variables: $z_{AC}$, $z_{BC}$, …
  - $\text{dom}(z) = \{‘=’, ‘≠’\}$
  - instead of backtracking on the $y$’s backtrack on the edges with $z$’s
  - preserves at least one optimal solution
Spatial and temporal scheduler:
Search tree of improved model

A → 2 → C → 1 → D

\[
\begin{align*}
& z_{AC} = (\neq) \\
& z_{BC} = (\neq) \quad (\neq) \\
& z_{CD} = (\neq) \quad (\neq) \quad (\neq) \\
\end{align*}
\]

- determine \( y \), find temporal schedule for \( y = (0,0,0,0) \)
- same as \( y = (1,1,1,1) \) etc.

\[
\begin{align*}
& z_{BC} = (\neq) \quad (\neq) \\
& z_{CD} = (\neq) \quad (\neq) \\
\end{align*}
\]

- determine \( y \), find temporal schedule for \( y = (0,1,1,0) \)
- same as \( y = (2,3,3,2), y = (0,2,2,3) \) etc.
Spatial and temporal scheduler:
Improving the **solver**

- Preprocess DAG to find instructions which must be on same cluster
  - preserve an optimal solution
- Variable ordering
  - assign $z$ variables first, in breadth-first order of DAG
  - determine assignment for corresponding $y$ variables
  - determine cost of temporal schedule for these assignments
Outline

• Introduction
  • computer architecture
  • superblock scheduling
• Constraint programming approach
  • temporal scheduler
  • spatial and temporal scheduler
• Experiments
  • experimental setup
  • experimental results
• Lessons learned
Experimental setup: Instances

• All 154,651 superblocks from SPEC 2000 integer and floating pt. benchmarks
  • standard benchmark suite
  • consists of software packages chosen to be representative of types of programming languages and applications
  • superblocks generated by IBM’s Tobey compiler when compiling the software packages
  • compilations done using Tobey’s highest level of optimization
Experimental setup: Target architectures

Realistic architectures:
- not fully pipelined
- issue width not equal to number of functional units
- serializing instructions

<table>
<thead>
<tr>
<th>architecture</th>
<th>issue width</th>
<th>simple int. units</th>
<th>complex int. units</th>
<th>memory units</th>
<th>branch units</th>
<th>floating pt. units</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-issue</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-issue</td>
<td>2</td>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4-issue</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6-issue</td>
<td>6</td>
<td>2</td>
<td></td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>
Experimental results: Temporal scheduler

Total time (hh:mm:ss) to schedule all superblocks and percentage solved to optimality, for various time limits for solving each instance

<table>
<thead>
<tr>
<th>architecture</th>
<th>1 sec.</th>
<th></th>
<th>10 sec.</th>
<th></th>
<th>1 min.</th>
<th></th>
<th>10 min.</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time</td>
<td>%</td>
<td>time</td>
<td>%</td>
<td>time</td>
<td>%</td>
<td>time</td>
<td>%</td>
</tr>
<tr>
<td>2-issue</td>
<td>3:57:13</td>
<td>91.83</td>
<td>30:53:83</td>
<td>93.90</td>
<td>108:50:01</td>
<td>97.18</td>
<td>665:31:00</td>
<td>97.70</td>
</tr>
<tr>
<td>4-issue</td>
<td>2:17:44</td>
<td>95.47</td>
<td>17:09:48</td>
<td>96.60</td>
<td>61:29:31</td>
<td>98.43</td>
<td>343:04:46</td>
<td>98.87</td>
</tr>
<tr>
<td>6-issue</td>
<td>3:04:18</td>
<td>93.59</td>
<td>25:03:44</td>
<td>94.76</td>
<td>87:04:34</td>
<td>97.78</td>
<td>511:19:14</td>
<td>98.29</td>
</tr>
</tbody>
</table>
Spatial and temporal scheduler: Some related work

- Bottom Up Greedy (BUG) [Ellis. MIT Press ‘86]
  - greedy heuristic algorithm
  - localized clustering decisions

- Hierarchical Partitioning (RHOP) [Chu et al. PLDI ‘03]
  - coarsening and refinement heuristic
  - weights of nodes and edges updated as algorithm progresses
Experimental results:
Spatial and temporal scheduler

4-cluster-2-issue-2-cyl

Average Speedup

Benchmarks

ammp applu apsi art bzip2 crafty eon equake facerec fma3d galael gcc gzip lucas mcf mesa mgrid parser perlbench slixtrack swim twolf vortex vpr wupwise AVERAGE
Experimental results:
Spatial and temporal scheduler

Average Speedup

Architecture Configuration (#Clusters – IssueWidth)

applu-2-cyl

rhop-ls  rhop-opt  cp
Outline

• Introduction
  • computer architecture
  • superblock scheduling
• Constraint programming approach
  • temporal scheduler
  • spatial and temporal scheduler
• Experiments
  • experimental setup
  • experimental results
• Lessons learned
Lessons learned (I)

• Pick problem carefully
  • is a new solution needed?
  • what is the likelihood of success?
• Existing heuristics may not leave any room for improvement
Instruction Selection

DAG:

TILES:

OUTPUT:

OR
Instruction Selection

• Given
  • an expression DAG \( G \)
  • a set of tiles representing machine instructions

• Find a mapping of tiles to nodes in \( G \) of minimal cost (size) that covers \( G \)

• Complexity:
  • polynomial for trees
  • NP-hard for DAGs
Experimental evaluation

![Bar chart showing code size (KB) for different benchmarks. The chart includes bars for burg (B), dp (D), and cp (C). The x-axis represents benchmarks, and the y-axis represents code size in kilobytes.](image)
Lessons learned (II)

• Be prepared for adversity
  • significant overhead
    • learning domain of application
    • significant implementation
    • significant engineering
  • different research cultures
    • researchers are tribal
    • different standards of reviewing (number & contentiousness)
    • different standards of evaluation, formalization, assumptions
Lessons learned (III)

• Rewards
  • can be attractive to students
  • can lead to identifying and solving interesting sub-problems whose solutions have general applicability
    • bounds consistency for alldifferent and gcc global constraints
    • restarts and portfolios
    • machine learning of heuristics
Optimization problems in compilers

- Instruction selection
- Instruction scheduling
  - basic-block scheduling
  - super-block scheduling
- Loop scheduling: tiling, unrolling, fusion
- Memory hierarchy optimizations
- Register allocation
Selected publications

• Applications

• Global constraints

• Portfolios and restarts
  H. Wu and P. van Beek. On portfolios for backtracking search in the presence of deadlines. *ICTAI-2007*.

• Heuristics and machine learning
Next project:
Smart water infrastructure