Optimal Loop Parallelization

Alexander Aiken, Alexandru Nicolau
Cornell Univ.

2011-04-06  Presenter: Ingoo Heo
Contents

1. Introduction
2. Motivation
3. The Approach
4. Computing Patterns for Statements
5. Computing an Overall Pattern
6. Mapping Optimal Schedules to Processors
7. Experiments
8. Conclusion

SoC Optimizations and Restructuring
1. Introduction

- Parallelizing Compilers
  - Exploit the parallelism available in a given program
  - However, existing parallelization techniques do not handle loops in a satisfactory manner

- New technique is proposed
  - Bridge the gap between fine- and coarse-grain loop parallelization
  - Allow the exploitation of parallelism inside and across loop iteration
  - Proved that the transformation techniques is time-optimal
  - Provide a valuable guide in generating a legal schedule even when resources are constrained
    - NP-hard
2. Motivation

- Given a loop and its dependency graph, what is the best parallel schedule for the loop?
  - In multi processors, each iteration is executed without dependency violation.
  - Using reordering statements, performance could be improved.
    - Optimal is NP-hard
2. Motivation

for $i \leftarrow 1$ to $N$ do
    A: $A[i] \leftarrow f_1(B[i])$;
    B: $B[i] \leftarrow f_2(A[i], D[i - 1])$;
    C: $C[i] \leftarrow f_3(A[i], D[i - 1])$;
    D: $D[i] \leftarrow f_4(B[i], C[i])$;

(a) A sample loop.

(b) The dependency graph.

(c) An optimal Doacross schedule.

(d) An optimal schedule.

Figure 1: An example.
2. Motivation

- With several standard assumptions for simplicity
  - Statements execute in a single machine cycle
  - Loop-carried dependencies are from one iteration to the next
  - Only applicable for innermost loops
  - The loop body should contain no If-statements other than exit tests.

- The proposed Technique could be applied to general loop
  - By some other method
    - Unrolling
    - Transformation
3. The Approach

- Based on SW-Pipelining
  - Consider only true dependences

- Examine a partial execution history
  - For first i iterations, greedy schedule is performed
  - From the regularity in dependency through iterations, graph is given
  - By scheduling i iteration of the loop, the pattern of the loop is found.

- The loop pattern
  - Repeated behavior though iterations
  - Derived from examining a partial execution history
4. Computing Patterns for Statements

- **Definition 4.1**
  - A dependency chain is a sequence of statements $x^1_{h_1}, x^2_{h_2}, ..., x^k_{h_k}$ such that $(x^i, x^{i+1})$ is an edge in the dependency graph.

- **Definition 4.2**
  - Let $C = x^1_{h_1}, ..., x^k_{h_k}$ be a dependency chain
    - $C$ is a cycle if $x^1 = x^k$.
    - The length of $C$, written $|C|$, is $k$.
    - $C'$ is a subchain of $C$ if $C' = x^i, ..., x^j$ where $1 \leq i \leq j \leq k$.
    - The span of $C$ is the number of iterations $h_k - h_1$.
    - $C$ reaches statement $y$ if $x^i_{h_i}$ is scheduled at time $i$, $y$ is scheduled at time $k+1$, and $(x^k, y)$ is an edge in the dependency graph.
4. Computing Patterns for Statements

Definition 4.3
Let $C = x^1 \ldots x^k x^1$ be a cycle. Let $p$ be the number of loop-carried dependencies – including the dependency $(x^k, x^1)$ – in the cycle. The slope of $C$ is the ratio $k/p$.

- A bound on the rate that statements in the cycle can be executed.

When slope is $k/p$, $x_j^i, x_{j+p}^i$ must be scheduled at least $k$ steps.

$\text{dist}(x, y)$
- The number of steps separating $x$ and $y$ in a greedy schedule

$\text{Slope}(x) = \frac{k_x}{p_x}$
- The maximum slope of any cycle on which $x$ depends.
- 0/1 when $x$ is not dependent on any cycle.
4. Computing Patterns for Statements

- After scheduling $O(n^2)$ iterations, any subsequent occurrences of a statement $x$ are scheduled exactly $k_x$ steps after the occurrence of $x$ $p_x$ iterations before.

**Lemma 4.4** Let $C$ be a chain with $|C| \geq (i + 1)n$ for some positive $i$. Then there are at least $i$ disjoint subchains of $C$ that are cycles.

**Proof:** Partition $C$ into subchains $C_1, C_2, \ldots, C_i$, where $|C_j| > n$. Each $C_j$ must contain a cycle, as there are only $n$ distinct statements. □

**Lemma 4.5** Given $p$ integers $a_1 \ldots a_p$, then there is a subset $S$ of the $a_i$ such that

$$\sum_{a_i \in S} a_i \equiv 0 \mod p$$

**Proof:** Let $s_i = (a_1 + \ldots + a_i) \mod p$. If all of the $s_i$ are distinct, then one must be zero, as there are only $p$ distinct numbers. If $s_i$ and $s_{i+j}$ are equal, then $0 \mod p \equiv s_{i+j} - s_i = a_i + \ldots + a_{i+j}$. □
4. Computing Patterns for Statements

**Theorem 4.6** Let $x$ be a statement with slope $k/p$, and let the loop body contain $n$ statements. In a greedy schedule, in any iteration $i$ greater than $2np + 3p$, $\text{dist}(x_{i-p}, x_i) = k$.

**Proof:** For brevity, we prove the theorem only for statements $x$ which are members of the cycle of maximum slope on which they depend. Assume for some $i > np + 2p$ that $\text{dist}(x_{i-p}, x_i) > k$. Let $C$ be a chain reaching $x_i$. There are two cases:

- **Span($C$) $\leq np + p$ iterations.** Let $C'$ be a chain reaching $x_{i-p}$. Because dependencies are regular, a chain of dependencies identical to $C$ reaches $x_{i-p}$. But $|C'| + k \leq |C|$, a contradiction.

- **Span($C$) $> np + p$ iterations.** By Lemma 4.4, there are at least $p$ disjoint subchains of $C$ that are cycles. By Lemma 4.5, there is a subset of these cycles $\{C_h\}$ such that $\sum_h |C_h| = jp$ for some $j > 0$. Deleting the cycles $\{C_h\}$ from $C$ produces a chain $C'$ which reaches $x_y$, where $y = i - jp$. By assumption, there is a chain of length $jk$ from $x_y$ to $x_i$. But $\sum_h |C_h| \leq jk$, or else some $C_h$ has slope greater than $k/p$, a contradiction. Therefore $|C'| + jk \geq |C|$, implying that $\text{dist}(x_y, x_i) = jk$. Since $\text{dist}(x_y, x_{i-p}) \geq (j-1)k$, $\text{dist}(x_{i-p}, x_i) \leq k$.
4. Computing Patterns for Statements

Corollary 4.7 After scheduling $O(n^2)$ iterations every statement is scheduled at the optimal rate. Furthermore, if $p_x \leq 1$ for all $x$, then $O(n)$ iterations are sufficient.

- Corollary 4.7 follows from Theorem 4.6 since $p<=n$ for any $x$.
- In many Cases, $p<=1$
  - For all statements, require $O(n^2)$ iterations
- If $p>1$
  - For all statements, require $O(n^3)$ iterations
5. Computing an Overall Pattern

\[
\text{for } i \leftarrow 1 \text{ to } N \text{ do }
\]
\[
\begin{align*}
A: & \ A1[i] \leftarrow B[i]; \\
B: & \ A2[i] \leftarrow \quad A8[i - 1]; \\
C: & \ A3[i] \leftarrow \quad A5[i - 1]; \\
D: & \ A4[i] \leftarrow A3[i] \quad +A7[i - 1]; \\
E: & \ A5[i] \leftarrow A2[i]; \\
F: & \ A6[i] \leftarrow A1[i] \quad +A13[i - 1]; \\
G: & \ A7[i] \leftarrow A4[i]; \\
H: & \ A8[i] \leftarrow A4[i] + A5[i] \quad +A17[i - 1]; \\
I: & \ A9[i] \leftarrow A1[i]; \\
J: & \ A10[i] \leftarrow A9[i] \quad +A15[i - 1]; \\
K: & \ A11[i] \leftarrow A9[i]; \\
L: & \ A12[i] \leftarrow A9[i]; \\
M: & \ A13[i] \leftarrow A12[i]; \\
N: & \ A14[i] \leftarrow A13[i]; \\
P: & \ A15[i] \leftarrow A14[i]; \\
Q: & \ A16[i] \leftarrow A14[i]; \\
R: & \ A17[i] \leftarrow A14[i]; \\
\end{align*}
\]

Figure 2: A sample loop.
5. Computing an Overall Pattern

(a) The dependency graph.  
(b) The code after five phases.

Figure 3: Greedy scheduling.

SoC Optimizations and Restructuring
5. Computing an Overall Pattern

- Two drawbacks to the simple greedy scheduling algorithm
  - To be assured that the pattern has been detected, it is necessary to run for $O(n^3)$ time
  - In many cases, the pattern is observed earlier.
  - The information greedy scheduling provides is not immediately useful for generation practical code
    - It is inefficient because it requires too many processors.

- Needs for some modifications
  - To detect a pattern for the entire loop body ASAP.
  - Using less processors
5. Computing an Overall Pattern

- Modified Scheduling
  - Reschedule statements not on the critical path so that they have the same slope as statements on the critical path.
  - In Figure 3b, The statements with slope 0/1 in iterations could be delayed without affecting the length of the schedule because they are not on critical path
    - Eliminating the gap.
5. Computing an Overall Pattern

**Definition 5.1** A *region* of a scheduled iteration is an interval of time steps $A = t_1..t_k$ such that some statement from the iteration is scheduled at every $t_j$.

**Definition 5.2** Let $A_1, \ldots, A_j$ be the maximal regions of an iteration, where $A_i = t_i..t'_i$ and $t'_i < t_{i+1}$ for all $i$. Then $\text{gap}(A_i, A_{i+1}) = t_{i+1} - t'_i$. 

SoC Optimizations and Restructuring
5. Computing an Overall Pattern

<table>
<thead>
<tr>
<th>iteration</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>ABC</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>DEFI</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>3</td>
<td>GHJKL</td>
<td>CKL</td>
<td>KL</td>
<td>KL</td>
<td>KL</td>
</tr>
<tr>
<td>4</td>
<td>M</td>
<td>BDM</td>
<td>M</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>5</td>
<td>N</td>
<td>EFGN</td>
<td>FN</td>
<td>FN</td>
<td>FN</td>
</tr>
<tr>
<td>6</td>
<td>PQR</td>
<td>PQR</td>
<td>CPQR</td>
<td>PQR</td>
<td>PQR</td>
</tr>
<tr>
<td>7</td>
<td>HJ</td>
<td>DJ</td>
<td>I</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>BG</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>H</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b) The code after five phases.

Region

Gap

SoC Optimizations and Restructuring
5. Computing an Overall Pattern

Theorem 5.3 Consider $i + c$ scheduled iterations, where iteration $i$ and $i + c$ are alike, with $j$ maximal regions. Assume there is no unbroken chain of dependent statements from a statement in $A_k$ of iteration $i$ ($k < j$) to a statement in $A_j$ of iteration $i + c$. Then the inter-region gaps in iteration $i + c$ can be reduced to the size of the corresponding gaps in iteration $i$ and for any additional (greedily) scheduled iterations the resulting schedule is optimal.

Proof: We outline the complete proof. After shrinking the gaps iteration $i + c$ is identical to iteration $i$. Then (greedily) scheduled iterations $i + c$ to $i + 2c$ in the new schedule are identical to iterations $i$ to $i + c$ in the original schedule. We claim there is no shorter schedule for $i + 2c$ iterations. For iterations $i$ to $i + c$, the critical chain of dependencies is between the regions $A_j$ of iteration $i$ and $A_j$ of iteration $i + c$. By symmetry, the critical chain of iterations $i + c$ to $i + 2c$ is between the regions $A_j$ of iteration $i + c$ and $A_j$ of iteration $i + 2c$. This implies no statement in $A_j$ of iteration $i + 2c$ is delayed by shrinking the gaps in iteration $i + c$. Applying this argument inductively shows that any larger greedy schedule is also time optimal. □
5. Computing an Overall Pattern

- If \( p \leq 1 \) for all statements \( x \), then checking consecutive iterations \( (c=1) \) is sufficient
  - \( O(n^2) \) algorithm

- Else if \( p > 1 \)
  - Require \( O(n^3) \)
  - However, it is unusual
6. Mapping Optimal Schedules to Processors

- Mapping is specified to processor architectures
  - In VLIW or SIMD
    - The final program graph can run directly
  - Synchronous Multi Processor
    - Vertically sliced with one statement from each node assigned to a processor
  - Asynchronous Multi Processor
    - Heavily dependent on topology and communication cost
6. Mapping Optimal Schedules to Processors

![Diagram of program graph and schedule]

(a) The final program graph.

(b) Schedule for a synchronous machine.

Figure 5: An optimal schedule.
### 7. Experiments

<table>
<thead>
<tr>
<th>Loop</th>
<th>Original Code</th>
<th>Limited Processors</th>
<th>Ideal Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mflops</td>
<td>1 proc Mflops</td>
<td>2 procs Mflops</td>
</tr>
<tr>
<td>LL1</td>
<td>9</td>
<td>31-50</td>
<td>57-100</td>
</tr>
<tr>
<td>LL2</td>
<td>8</td>
<td>20-35</td>
<td>40-60</td>
</tr>
<tr>
<td>LL3</td>
<td>7</td>
<td>16-20</td>
<td>20-23</td>
</tr>
<tr>
<td>LL4</td>
<td>6</td>
<td>16</td>
<td>20</td>
</tr>
<tr>
<td>LL5</td>
<td>6</td>
<td>12-15</td>
<td>15-16</td>
</tr>
<tr>
<td>LL6</td>
<td>8</td>
<td>6-16</td>
<td>6-20</td>
</tr>
<tr>
<td>LL7</td>
<td>20</td>
<td>36-51</td>
<td>71-99</td>
</tr>
<tr>
<td>LL8</td>
<td>11</td>
<td>40-55</td>
<td>80-110</td>
</tr>
<tr>
<td>LL9</td>
<td>17</td>
<td>35-49</td>
<td>68-97</td>
</tr>
<tr>
<td>LL10</td>
<td>10</td>
<td>18-25</td>
<td>36-48</td>
</tr>
<tr>
<td>LL11</td>
<td>4</td>
<td>4-9</td>
<td>4-11</td>
</tr>
<tr>
<td>LL12</td>
<td>4</td>
<td>13-20</td>
<td>27-40</td>
</tr>
<tr>
<td>LL13</td>
<td>4</td>
<td>11-12</td>
<td>22-24</td>
</tr>
<tr>
<td>LL14 (avg)</td>
<td>4</td>
<td>14-18</td>
<td>25-31</td>
</tr>
<tr>
<td>Average</td>
<td>8</td>
<td>19-28</td>
<td>35-50</td>
</tr>
<tr>
<td>Harmonic Mean</td>
<td>7</td>
<td>13-20</td>
<td>18-33</td>
</tr>
</tbody>
</table>

#### Table 1: Performance ranges for 14 Livermore Loops.
7. Experiments

- Some features improve speed up
  - Indirect addressing (auto increments) HW support
  - Compiler optimization techniques
    - Ex. Redundant load remove

- Pre-loop, Post-loop is not considered.
8. Conclusion

- Optimal loop parallelization algorithm is suggested.
  - With some simplicity
  - Proved by mathematical method
  - Show substantial speedup

- If constraints are added
  - Such as limited number of processors, asynchronous machines
    - NP hard
  - Proposed algorithm would be useful as a step to generating good parallel code