On Optimizing The Longest Common Subsequence Problem
by
Loop Unrolling Along Wavefronts

Authors: Johann Steinbrecher, Weijia Shang
Affiliation: Computer Engineering, Santa Clara University, CA, USA
- Problem definition: The Longest Common Subsequence problem
- Basic ideas and concepts
- Modeling the total execution time
- Sufficient and necessary conditions for optimality
- Benchmarking the technique
The longest common subsequence algorithm:

Inputs:

x = “MZJAWXU”
y = “XMJYAUZ”

Output:
4 because the longest common subsequence is “MJAU”

Applications:

- DNA matching in bio-engineering: human DNA organized in 23 chromosomes each holding 50 x 10^6 to 220 x 10^6 base pairs
- File comparison, UNIX diff to generate patches, plagiarism
The longest common subsequence algorithm:

```c
for (i=0, M) {
    for (j=0, N) {
        if (i == 0 || j == 0)
            c[i][j] = 0;
        else if (x[i] == y[j])
            c[i][j] = c[i-1][j-1] + 1;
        else
            c[i][j] = max(c[i][j-1], c[i-1][j]);
    }
}
```

The LCS problem is a dynamic programming algorithm with \( O(N^2) \) time complexity.

Human DNA is organized in 23 chromosomes, each holding 50 x 10^6 to 220 x 10^6 base pairs, leading to a significant amount of computation.
The longest common subsequence algorithm:

\[
\begin{array}{l}
\text{for } (i=0, M) \{ \\
\quad \text{for } (j=0, N) \{ \\
\quad\quad \text{if } (i == 0 \text{ || } j == 0) \\
\quad\quad \quad c[i][j] = 0; \\
\quad\quad \text{else if } (x[i] == y[j]) \\
\quad\quad \quad c[i][j] = c[i-1][j-1] + 1; \\
\quad\quad \text{else} \\
\quad\quad \quad c[i][j] = \max(c[i][j-1], c[i-1][j]); \\
\quad \} \\
\} \\
\rightarrow \text{Data dependence}
\end{array}
\]
The longest common subsequence algorithm:

Dependence matrix: \[ D = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \]

Wavefronts: Hyperplanes in the index set

Linear schedule \( \pi = (1, 1) \)

→ All instructions on one wavefront are independent of each other

Such wavefronts may be generated using CLooG, a loop transformation and code generation tool based on the polyhedral model.
Modern computer architectures

- Instruction level parallelism (ILP) single core
  - Superscalar architectures
  - VLIW architectures

- Thread level parallelism (TLP) multicore
  - SIMD architectures (GPGPU)
  - MIMD architectures (MIC, Desktops)
The longest common subsequence algorithm:

for (i=0, M) {
    for (j=0, j<N, j+=2) {
        if (i == 0 || j == 0)
            c[i][j] = 0;
        else if (x[i] == y[j])
            c[i][j] = c[i-1][j-1] + 1;
        else
            c[i][j] = max(c[i][j-1], c[i-1][j]);
        if (i == 0 || j+1 == 0)
            c[i][j+1] = 0;
        else if (x[i] == y[j+1])
            c[i][j+1] = c[i-1][j] + 1;
        else
            c[i][j+1] = max(c[i][j], c[i-1][j+1]);
    }
}

\[
D = \begin{bmatrix}
0 & 1 & 1 \\
1 & 0 & 1 \\
\end{bmatrix}
\]
\[
\pi = [1 \quad 1]
\]
for (w=0; w<P; w++) {
    for (v=0; v<wave[w]; v+=2) {
        i = min(M,w) - v;
        j= max(0,w-M) + v;

        if ((i == 0) || (j == 0))
            c[i][j] = 0;
        else if (x[i] == y[j])
            c[i][j] = c[i-1][j-1]+1;
        else
            c[i][j] = max(c[i][j-1], c[i-1][j]);

        k= i-1;
        l= j+1;
        if(k>=0 && l<N ) {
            if ((k == 0) || (l == 0))
                c[k][l] = 0;
            else if (x[k] == y[l])
                c[k][l] = c[k-1][l-1]+1;
            else
                c[k][l] = max(c[k][l-1], c[k-1][l])
        }
    }
}

w: wavefront index (w=i+j)
wave[x]: number of nodes at wavefront w=x
P: number of wavefronts
Assembly instructions:

I1: load x[i], R1
I2: load y[j], R2
I3: cmp R1, R2
I4: load c[i][j-1], R3
I5: load c[i-1][j], R4
I6: max R5, R3, R4
I7: store c[i][j], R5
I8: load x[k], R6
I9: load y[l], R7
I10: cmp R6, R7
I11: load c[k][l-1], R8
I12: load c[k-1][l], R9
I13: max R10, R8, R9
I14: store c[k][l], R10

Loop unrolling along wavefronts:

Traditional unrolling:
Architecture model:

\[ p \] \quad \text{number of pipelines} \\
\[ p_{\text{stage}} \] \quad \text{number of pipeline stages} \\
\[ n_{\text{regfile}} \] \quad \text{register file size} \\
\[ n_{\text{str}} \] \quad \text{number of stalls to resolve a hazard}
\[ T_{superiter} = \max[T_{cp}, T_2] \]

\[ T_{cp} = p_{stage} + lcp \cdot (n_{str} + 1) \]
\[ T_{\text{superiter}} = \max[T_{cp}, T_2] \]

\[ T_2 = \left[ \sum_{i=1}^{lcp} Ic_{i(u)} \right] + p_{\text{stage}} + n_{\text{str}} \]
\[ T_{\text{superiter}} = \max[T_{cp}, T_2] \]

\[ T_{cp} = p_{stage} + lcp \cdot (n_{str} + 1) \]

\[ T_2 = \left[ \sum_{i=1}^{lcp} Ic_{i(u)} \right] + p_{stage} + n_{str} \]

\[ T_{\text{total}} = \frac{|J|}{u} \cdot (T_{\text{superiter}} + 2 \cdot (n_{str} + 1)) \]

closing index increment and branch instruction of each super iteration
Conditions for optimality:

Necessary condition 1:
Consider the case where the instructions at stage $i$ will not be executed until all the instructions in stage $i - 1$ are executed. If all the $p$ pipelines are saturated then the following is true:

$$\min\{Ic_1, Ic_2, \ldots, Ic_{lcp}\} \geq p \cdot (n_{str} + 1)$$

If instructions at stage $i$ are allowed to be executed before all the instructions in stage $i - 1$ are executed as long as the dependencies are respected, the necessary condition will be less restrictive.

Necessary condition 2:
Consider the set of all instructions IS. If there exists a partition $\{IS_1, \ldots, IS_{lcp}\}$ such that:

$$IS_i \cap IS_j = \emptyset, \ i \neq j$$
$$\{IS_1 \cup IS_2 \cup \ldots \cup IS_{lcp}\} = IS$$

and all instructions in $IS_i, \ i = 1, \ldots, lcp$ are independent, then if the $p$ pipelines are saturated, the following is true:

$$\min\{|IS_1|, |IS_2|, \ldots, |IS_{lcp}|\} \geq p \cdot (n_{str} + 1)$$

Sufficient condition:
If there are at least $(n_{str} + 1) \cdot p$ classes in the DDG with the same number of dependence stages, then all pipelines will be saturated.
Optimal unrolling factor $u_{opt}$ for loop unrolling along wavefronts:

\[
\begin{align*}
I_{c_1} &= 2u \\
I_{c_2} &= u \\
I_{c_3} &= 2u \\
I_{c_4} &= u \\
I_{c_5} &= u
\end{align*}
\]

\[
\min[I_{c_1}, I_{c_2}, I_{c_3}, I_{c_4}, I_{c_5}] \geq p \cdot (n_{str} + 1)
\]

\[
u \geq 3 \cdot (2 + 1)
\]

\[
u \geq 9
\]
Execution time for loop unrolling along wavefronts:

\[
t_{\text{cp}} = l_{\text{cp}} \cdot (n_{\text{str}} + 1) + p_{\text{stage}} = 5 \cdot 3 + 5 = 20
\]

\[
t_{\text{lower}} = \left[ \frac{\sum_{i=1}^{l_{\text{cp}}} (Ic_i(u))}{p} \right] + p_{\text{stages}} + n_{\text{str}} = 21 + 5 + 2 = 28
\]

\[
t = \max[t_{\text{cp}}, t_{\text{lower}}] = \max[20, 28] = 28
\]

\[
t_{\text{total}} = \frac{|J|}{u} \cdot (t_{\text{superiter}} + 2 \cdot (n_{\text{str}} + 1))
\]

\[
t_{\text{total}} = \frac{M \cdot N}{9} \cdot (28 + 6) = M \cdot N \cdot 3.77
\]

speed-up = \frac{t_{\text{total,old}}}{t_{\text{total}}} = \frac{5}{3.77} = 1.326
Experimental results:

Intel® Core™ i5-2500 machine (3.3 GHz)
- Sandy Bridge micro-architecture (out-of-order execution of instructions)
- 6 pipelines where 3 for arithmetic integer and floating point operations; 3 perform memory accesses
- GCC 4.6.1 with all compiler optimizations enabled
- 25000 characters input size

speedup of 1.475
Our Contribution:

- Using feasible linear schedules, wavefronts, we find a way to generate supernodes that do not hold any dependencies.
- As our method is based on an abstract algorithm model, this technique is applicable to all uniform dependence algorithms.
- We contribute two necessary and one sufficient condition for the optimal unrolling factor.
- We express the total execution time as a function considering algorithm parameters and architecture parameters.
- We speedup the longest common subsequence problem on an Intel Sandy Bridge architecture by 1.475 over compiler optimized traditional methods.
Future work

- The optimal unrolling factor may not exist if the critical path length increases with the unrolling factor
- While we have been trying to exploit instruction level parallelism we ignored the impact of the register file size and data locality
- The performance of this technique may be further improved by applying software pipelining
Thank you for your attention

Q & A