Script generated by TTT

Title: Seidl: Programmoptimierung (11.01.2012)

Date: Wed Jan 11 12:31:00 CET 2012

Duration: 89:32 min

Pages: 51

Warning:

- Our representation of numbers is not unique: 011101 should be accepted iff every word from $011101 \cdot 0^*$ is accepted!
- This property is preserved by union, intersection and complement :-)
- It is lost by projection !!!

$\Longrightarrow$ The automaton for projection must be enriched such that the property is re-established !!

$$\boxed{5x} + \boxed{17y} = 9$$

$$\rightarrow \boxed{x + y = 5}$$

$$\rightarrow \boxed{x + y = z}$$

$$x = 5$$

$$x \geq 20$$

$$x = 20 + 5$$

Observation:

The set of satisfying variable assignments is regular :-))

$\phi_1 \wedge \phi_2$	$\Longrightarrow$	$\mathcal{L}(\phi_1) \cap \mathcal{L}(\phi_2)$	(Intersection)
$\neg\phi$	$\Longrightarrow$	$\overline{\mathcal{L}(\phi)}$	(Complement)
$\exists\, x : \phi$	$\Longrightarrow$	$\pi_1(\mathcal{L}(\phi))$	(Projection)

$$\boxed{5x} + \boxed{17y} = 9$$

$$\rightarrow \boxed{x + y = 5}$$

$$\rightarrow \boxed{x + y = z}$$

$$x = 5$$

$$x \geq 20$$

$$x = 20 + 5$$

Projecting away the x-component:

213	t	1	0	1	0	1	0	1	1	0	0
42	z	0	1	0	1	0	1	0	0	0	0
89	y	1	0	0	1	1	0	1	0	0	0
17	x	1	0	0	0	1	0	0	0	0	0

Projecting away the x-component:

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0

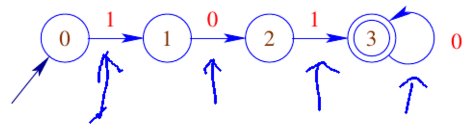
Warning:

- Our representation of numbers is not unique: 011101 should be accepted iff every word from $011101 \cdot 0^*$ is accepted!
- This property is preserved by union, intersection and complement :-)
- It is lost by projection !!!

$\Longrightarrow$ The automaton for projection must be enriched such that the property is re-established !!

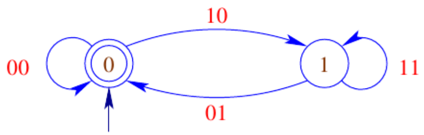
Automata for Basic Predicates:

$x = 5$



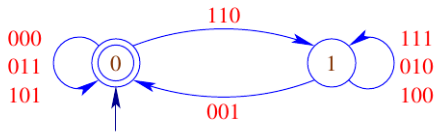
Automata for Basic Predicates:

$x+x=y$



Automata for Basic Predicates:

$x+y=z$



Results:

Ferrante, Rackoff,1973 : $\text{PSAT} \leq \text{DSPACE}(2^{2^{c \cdot n}})$

Results:

Ferrante, Rackoff,1973 : $\text{PSAT} \leq \text{DSPACE}(2^{2^{c \cdot n}})$

Fischer, Rabin,1974 : $\text{PSAT} \geq \text{NTIME}(2^{2^{c \cdot n}})$

Results:

Ferrante, Rackoff,1973 : PSAT $\leq$ DSPACE($2^{2^{c \cdot n}}$)

Fischer, Rabin,1974 : PSAT $\geq$ NTIME($2^{2^{c \cdot n}}$)

3.3 Improving the Memory Layout

Goal:

- Better utilization of caches

  $\Longrightarrow$ reduction of the number of cache misses
- Reduction of allocation/de-allocation costs

  $\Longrightarrow$ replacing heap allocation by stack allocation

  $\Longrightarrow$ support to free superfluous heap objects
- Reduction of access costs

  $\Longrightarrow$ short-circuiting indirection chains (Unboxing)

1. Cache Optimization:

Idea: local memory access

- Loading from memory fetches not just one byte but fills a complete cache line.
- Access to neighbored cells become cheaper.
- If all data of an inner loop fits into the cache, the iteration becomes maximally memory-efficient ...

Possible Solutions:

$\rightarrow$ Reorganize the data accesses !

$\rightarrow$ Reorganize the data !

Such optimizations can be made fully automatic only for arrays :-(

Example:

```
for (j = 1; j < n; j++)
    for (i = 1; i < m; i++)
        a[i][j] = a[i-1][j-1] + a[i][j];
```

⟹ At first, always iterate over the rows!

⟹ Exchange the ordering of the iterations:

for $(i = 1; i < m; i{+}{+})$

  for $(j = 1; j < n; j{+}{+})$

    $a[i][j] = a[i-1][j-1] + a[i][j];$

When is this permitted???

⟹ At first, always iterate over the rows!

⟹ Exchange the ordering of the iterations:

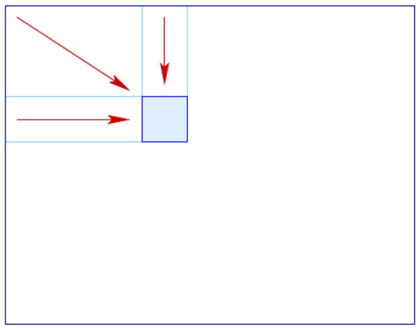
for $(i = 1; i < m; i{+}{+})$

  for $(j = 1; j < n; j{+}{+})$

    $a[i][j] = a[i-1][j-1] + a[i][j];$

When is this permitted???

Iteration Scheme: allowed dependencies:



⟹ At first, always iterate over the rows!

⟹ Exchange the ordering of the iterations:

for $(i = 1; i < m; i{+}{+})$

  for $(j = 1; j < n; j{+}{+})$

    $a[i][j] = a[i-1][j-1] + a[i][j];$

When is this permitted???

In our case, we must check that the following equation systems have no solution:

Write		Read
(i_1, j_1)	$=$	$(i_2 - 1, j_2 - 1)$
i_1	$\leq$	i_2
j_2	$\leq$	j_1
(i_1, j_1)	$=$	$(i_2 - 1, j_2 - 1)$
i_2	$\leq$	i_1
j_1	$\leq$	j_2

The first implies: $j_2 \leq j_2 - 1$ Hurra!

The second implies: $i_2 \leq i_2 - 1$ Hurra!

Example: Matrix-Matrix Multiplication

$$
\begin{array}{l}
\text{for } (i = 0; i < N; i{+}{+}) \\
\quad \text{for } (j = 0; j < M; j{+}{+}) \\
\quad\quad \text{for } (k = 0; k < K; k{+}{+}) \\
\quad\quad\quad c[i][j] = c[i][j] + a[i][k] \cdot b[k][j];
\end{array}
$$

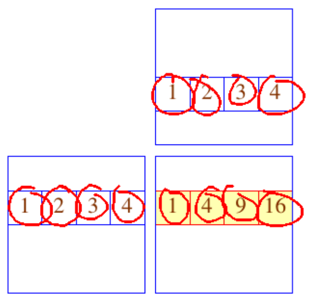
?

Over $b[][]$ the iteration is columnwise :-(

Exchange the two inner loops:

$$
\begin{array}{l}
\text{for } (i = 0; i < N; i{+}{+}) \\
\quad \text{for } (k = 0; k < K; k{+}{+}) \\
\quad\quad \text{for } (j = 0; j < M; j{+}{+}) \\
\quad\quad\quad c[i][j] = c[i][j] + a[i][k] \cdot b[k][j];
\end{array}
$$

Is this permitted ???



Exchange the two inner loops:

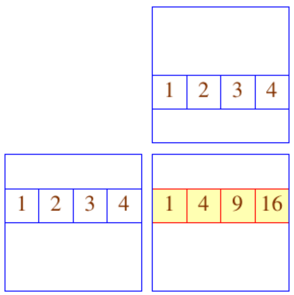
for $(i = 0; i < N; i{+}{+})$

  for $(k = 0; k < K; k{+}{+})$

    for $(j = 0; j < M; j{+}{+})$

      $c[i][j] = c[i][j] + a[i][k] \cdot b[k][j];$

Is this permitted ???



Discussion:

- Correctness follows as before :-)
- A similar idea can also be used for the implementation of multiplication for row compressed matrices :-))
- Sometimes, the program must be massaged such that the transformation becomes applicable :-(
- Matrix-matrix multiplication perhaps requires initialization of the result matrix first ...

758

Discussion:

- Correctness follows as before :-)
- A similar idea can also be used for the implementation of multiplication for row compressed matrices :-))
- Sometimes, the program must be massaged such that the transformation becomes applicable :-(
- Matrix-matrix multiplication perhaps requires initialization of the result matrix first ...

758

Exchange the two inner loops:

for $(i = 0; i < N; i{+}{+})$
  for $(k = 0; k < K; k{+}{+})$
    for $(j = 0; j < M; j{+}{+})$
      $c[i][j] = c[i][j] + a[i][k] \cdot b[k][j];$

Is this permitted ???

Discussion:

- Correctness follows as before :-)
- A similar idea can also be used for the implementation of multiplication for row compressed matrices :-))
- Sometimes, the program must be massaged such that the transformation becomes applicable :-(
- Matrix-matrix multiplication perhaps requires initialization of the result matrix first ...

for $(i = 0; i < N; i{+}{+})$
  for $(j = 0; j < M; j{+}{+})$ {
    $c[i][j] = 0;$
    for $(k = 0; k < K; k{+}{+})$
      $c[i][j] = c[i][j] + a[i][k] \cdot b[k][j];$
  }

- Now, the two iterations can no longer be exchanged :-(
- The iteration over j, however, can be duplicated ...

for $(i = 0; i < N; i{+}{+})$ {
  for $(j = 0; j < M; j{+}{+})$ $c[i][j] = 0;$
  for $(j = 0; j < M; j{+}{+})$
    for $(k = 0; k < K; k{+}{+})$
      $c[i][j] = c[i][j] + a[i][k] \cdot b[k][j];$
}

Correctness:

$\Longrightarrow$ The read entries (here: no) may not be modified in the remaining body of the loop !!!

$\Longrightarrow$ The ordering of the write accesses to a memory cell may not be changed :-)

$$
\begin{array}{l}
\textsf{for}\ \ (i = 0; i < N; i{+}{+}) \\
\quad \textsf{for}\ \ (j = 0; j < M; j{+}{+})\ \ \{ \\
\qquad c[i][j] = 0; \\
\qquad \textsf{for}\ \ (k = 0; k < K; k{+}{+}) \\
\qquad\quad c[i][j] = c[i][j] + a[i][k] \cdot b[k][j]; \\
\quad \}
\end{array}
$$

- Now, the two iterations can no longer be exchanged :-(
- The iteration over j, however, can be duplicated ...

$$
\begin{array}{l}
\textsf{for}\ \ (i = 0; i < N; i{+}{+})\ \ \{ \\
\quad \textsf{for}\ \ (j = 0; j < M; j{+}{+})\ \ c[i][j] = 0; \\
\quad \textsf{for}\ \ (j = 0; j < M; j{+}{+}) \\
\qquad \textsf{for}\ \ (k = 0; k < K; k{+}{+}) \\
\qquad\quad c[i][j] = c[i][j] + a[i][k] \cdot b[k][j]; \\
\}
\end{array}
$$

Correctness:

$\Longrightarrow$ The read entries (here: no) may not be modified in the remaining body of the loop !!!

$\Longrightarrow$ The ordering of the write accesses to a memory cell may not be changed :-)

We obtain:

$$
\begin{array}{l}
\textsf{for}\ \ (i = 0; i < N; i{+}{+})\ \ \{ \\
\quad \textsf{for}\ \ (j = 0; j < M; j{+}{+})\ \ c[i][j] = 0; \\
\quad \textsf{for}\ \ (k = 0; k < K; k{+}{+}) \\
\qquad \textsf{for}\ \ (j = 0; j < M; j{+}{+}) \\
\qquad\quad c[i][j] = c[i][j] + a[i][k] \cdot b[k][j]; \\
\}
\end{array}
$$

Discussion:

- Instead of fusing several loops, we now have distributed the loops :-)
- Accordingly, conditionals may be moved out of the loop $\Longrightarrow$ if-distribution ...

761

We obtain:

for (i = 0; i < N; i++) {
for (j = 0; j < M; j++) c[i][j] = 0;
for (k = 0; k < K; k++)
for (j = 0; j < M; j++)
c[i][j] = c[i][j] + a[i][k] · b[k][j];
}

Discussion:

- Instead of fusing several loops, we now have distributed the loops :-)
- Accordingly, conditionals may be moved out of the loop ⟹ if-distribution ...

761

Warning:

Instead of using this transformation, the inner loop could also be optimized as follows:

$$
\begin{array}{l}
\text{for } (i = 0; i < N; i{+}{+}) \\
\quad \text{for } (j = 0; j < M; j{+}{+}) \ \{ \\
\qquad t = 0; \\
\qquad \text{for } (k = 0; k < K; k{+}{+}) \\
\qquad\quad t = t + a[i][k] \cdot b[k][j]; \\
\qquad c[i][j] = t; \\
\quad \}
\end{array}
$$

Idea:

If we find heavily used array elements $a[e_1] \ldots [e_r]$ whose index expressions stay constant within the inner loop, we could instead also provide auxiliary registers :-)

Warning:

The latter optimization prohibits the former and vice versa ...

Warning:

Instead of using this transformation, the inner loop could also be optimized as follows:

$$
\begin{array}{l}
\text{for } (i = 0; i < N; i{+}{+}) \\
\quad \text{for } (j = 0; j < M; j{+}{+}) \ \{ \\
\qquad t = 0; \\
\qquad \text{for } (k = 0; k < K; k{+}{+}) \\
\qquad\quad t = t + a[i][k] \cdot b[k][j]; \\
\qquad c[i][j] = t; \\
\quad \}
\end{array}
$$

$$
\begin{array}{l}
\text{for } (i = 0; i < N; i{+}{+}) \ \{ \\
\quad \text{for } (j = 0; j < M; j{+}{+}) \ \ c[i][j] = 0; \\
\quad \text{for } (j = 0; j < M; j{+}{+}) \\
\qquad \text{for } (k = 0; k < K; k{+}{+}) \\
\qquad\quad c[i][j] = c[i][j] + a[i][k] \cdot b[k][j]; \\
\}
\end{array}
$$

Correctness:

$\Longrightarrow$ The read entries (here: no) may not be modified in the remaining body of the loop !!!

$\Longrightarrow$ The ordering of the write accesses to a memory cell may not be changed :-)

Warning:

Instead of using this transformation, the inner loop could also be optimized as follows:

$$
\begin{array}{l}
\text{for } (i = 0; i < N; i{+}{+}) \\
\quad \text{for } (j = 0; j < M; j{+}{+}) \; \{ \\
\quad\quad t = 0; \\
\quad\quad \text{for } (k = 0; k < K; k{+}{+}) \\
\quad\quad\quad t = t + a[i][k] \; \boxed{b[k][j];} \\
\quad\quad c[i][j] = t; \\
\quad \}
\end{array}
$$

Idea:

If we find heavily used array elements $a[e_1] \ldots [e_r]$ whose index expressions stay constant within the inner loop, we could instead also provide auxiliary registers :-)

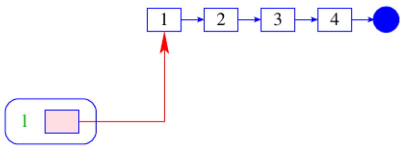
Warning:

The latter optimization prohibits the former and vice versa ...

Discussion:

- so far, the optimizations are concerned with iterations over arrays.
- Cache-aware organization of other data-structures is possible, but in general not fully automatic ...

Example: Stacks

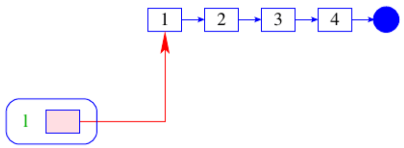


764

Discussion:

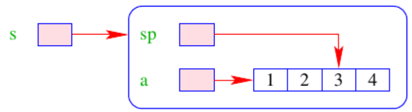
- so far, the optimizations are concerned with iterations over arrays.
- Cache-aware organization of other data-structures is possible, but in general not fully automatic ...

Example: Stacks



764

Alternative:



Advantage:

+ The implementation is also simple :-)
+ The operations push / pop still require constant time :-)
+ The data are consequtively allocated; stack oscillations are typically small

$\Longrightarrow$ better Cache behavior !!!

Discussion:

→ The same idea also works for queues :-)

→ Other data-structures are attempted to organize blockwise.

Problem: how can accesses be organized such that they refer mostly to the same block ???

$\Longrightarrow$ Algorithms for external data

2. Stack Allocation instead of Heap Allocation

Problem:

- Programming languages such as Java allocate all data-structures in the heap — even if they are only used within the current method :-(
- If no reference to these data survives the call, we want to allocate these on the stack :-)

$\Longrightarrow$ Escape Analysis

Idea:

Determine points-to information.

Determine if a created object is possibly reachable from the out side ...

Example: Our Pointer Language

$$x = \mathsf{new}();$$
$$y = \mathsf{new}();$$
$$x[A] = y;$$
$$z = y;$$
$$\mathsf{ret} = z;$$

... could be a possible method body ;-)

Accessible from the outside world are memory blocks which:

- are assigned to a global variable such as ret; or
- are reachable from global variables.

... in the Example:

$$x = \text{new}();$$
$$y = \text{new}();$$
$$x[A] = y;$$
$$z = y;$$
$$\text{ret} = \boxed{z};$$

Accessible from the outside world are memory blocks which:

- are assigned to a global variable such as ret; or
- are reachable from global variables.

... in the Example:

$$x = \text{new}();$$
$$y = \boxed{\text{new}()};$$
$$x[A] = y;$$
$$z = \boxed{y};$$
$$\text{ret} = \boxed{z};$$

We conclude:

- The objects which have been allocated by the first new() may never escape.
- They can be allocated on the stack :-)

Warning:

This is only meaningful if only few such objects are allocated during a method call :-(

If a local new() occurs within a loop, we still may allocate the objects in the heap ;-)

Extension: Procedures

- We require an interprocedural points-to analysis :-)
- We know the whole program, we can, e.g., merge the control-flow graphs of all procedures into one and compute the points-to information for this.
- Warning: If we always use the same global variables $y_1, y_2, \ldots$ for (the simulation of) parameter passing, the computed information is necessarily imprecise :-(
- If the whole program is not known, we must assume that each reference which is known to a procedure escapes :-((