Magic Hexagon

Effiziente Programme WS23
Group 5

Overview

- Our Ideas
- Successful changes
- Final results
- Ineffective changes

💡 Our Ideas

✔️ What worked

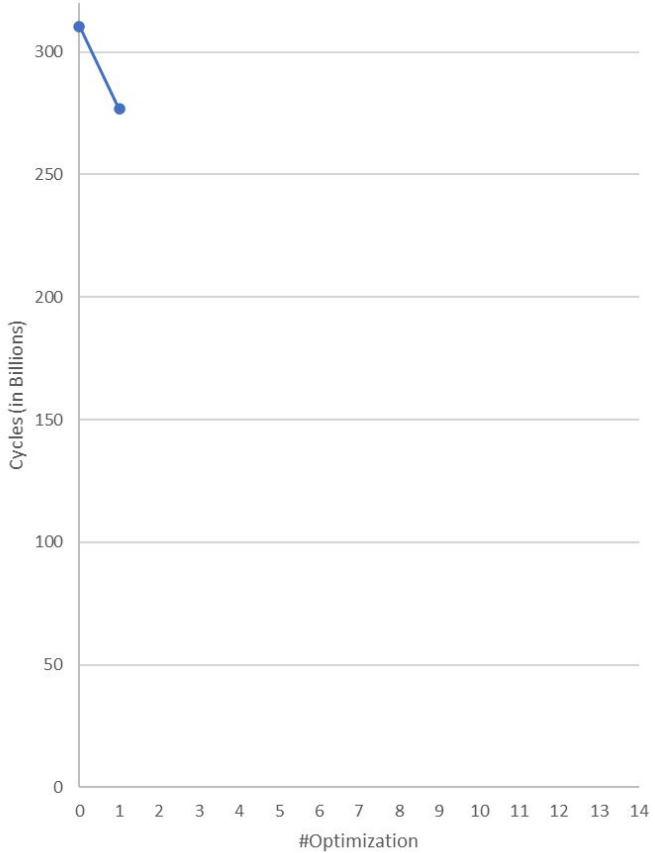
- Minimize work in `solve`
- Better variable selection in `labeling`
- Compiler flags
- Pack variables tighter

❌ What didn’t work

- Global Variables
- Manual Inlining
- Computed return codes
- Simple loop unrolling
- Simple Vectorization
- Iterative labeling

Do not restart when updating occupation array

- Occupation array entry updates do not affect each other
- Just remove the `goto`



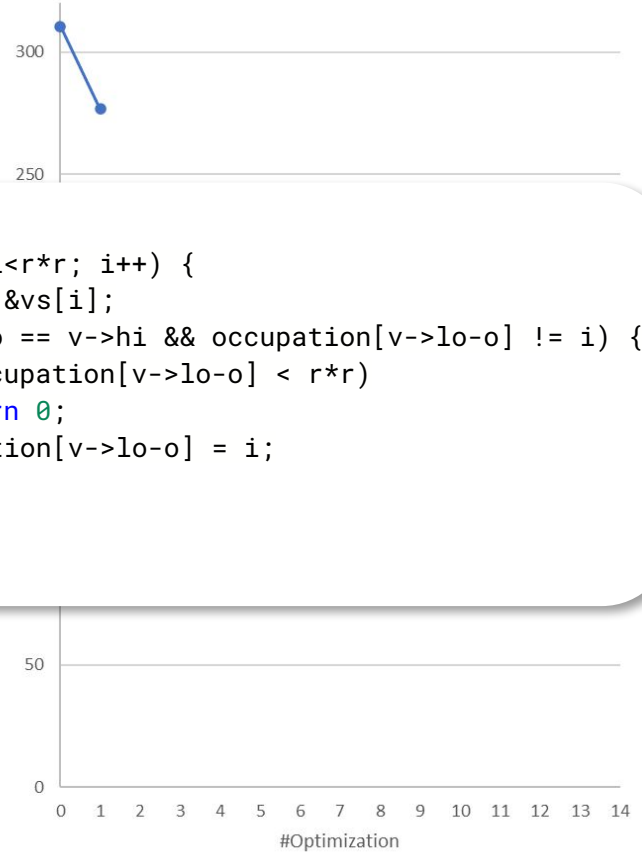
276,772,609,578 cycles (speedup **1.12**) (relative Δ10.82%)

Do not restart when updating occupation array

- Occupation array entry updates do not affect

```
restart:
  for (i=0; i<r*r; i++) {
    Var *v = &vs[i];
    if (v->lo == v->hi && occupation[v->lo-o] != i) {
      if (occupation[v->lo-o] < r*r)
        return 0;
      occupation[v->lo-o] = i;
      goto restart;
    }
  }
```

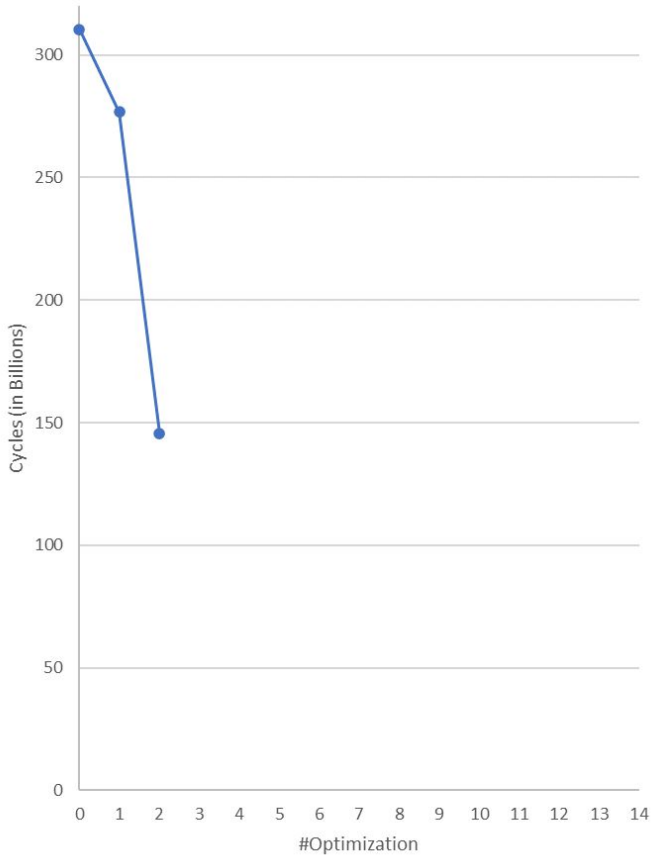
```
restart:
  for (i=0; i<r*r; i++) {
    Var *v = &vs[i];
    if (v->lo == v->hi && occupation[v->lo-o] != i) {
      if (occupation[v->lo-o] < r*r)
        return 0;
      occupation[v->lo-o] = i;

    }
  }
```



276,772,609,578 cycles (speedup **1.12**) (relative Δ10.82%)

Update occupation array in-place

- Check if variable boundary got fixed
- Update occupation array instead of rebuilding it
- Tighten boundaries multiple times before restarting alldifferent check



145,416,006,080 cycles (speedup **2.13**) (relative Δ47.46%)

Update occupation array in-place

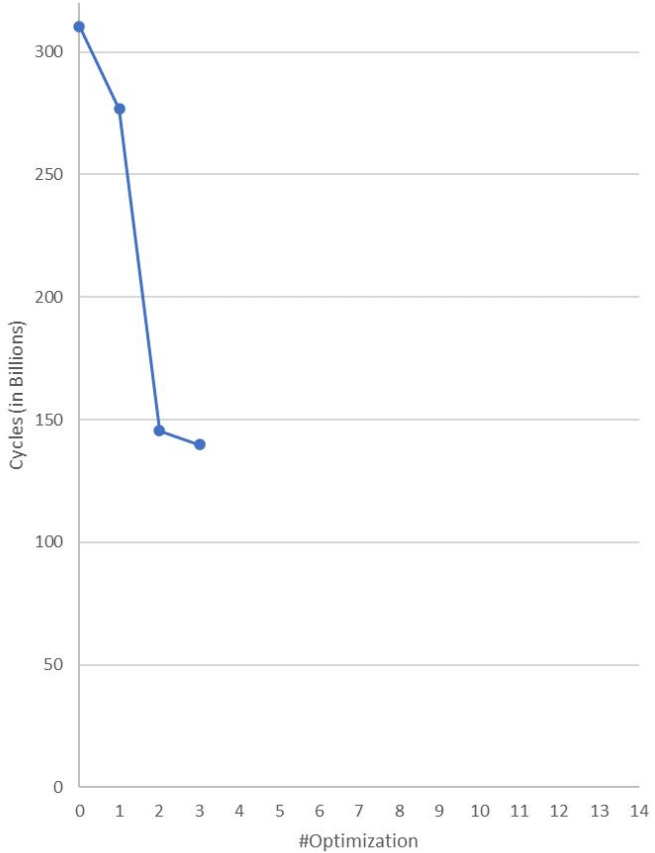
```
for (i=0; i<r*r; i++) {
  Var *v = &vs[i];
  if (v->lo < v->hi) {

    if (occupation[v->lo-o] < r*r) {
      v->lo++;
      goto restart;

    }
    if (occupation[v->hi-o] < r*r) {
      v->hi--;
      goto restart;

    }
  }
}
```

```
restart_propagate_alldifferent:
  for (i=0; i<r*r; i++) {
    Var *v = &vs[i];
    if (v->lo < v->hi) {
try_to_propagate_alldiff_again:
      if (occupation[v->lo-o] < r*r) {
        v->lo++;
        if( v->lo == v->hi ) {
          if( occupation[v->lo-o] < r*r ) {
            return 0;
          }
          occupation[v->lo-o]= i;
          goto restart_propagate_alldifferent;
        }
        goto try_to_propagate_alldiff_again;
      }
      if (occupation[v->hi-o] < r*r) {
        v->hi--;
        ...
        goto try_to_propagate_alldiff_again;
      }
    }
  }
```

145,416,006,080 cycles (speedup **2.13**) (relative Δ47.46%)

Make all functions static

- Use `static` keyword for better inlining
- No need for `__attribute__((always_inline))`
- Make function “private” for compilation unit



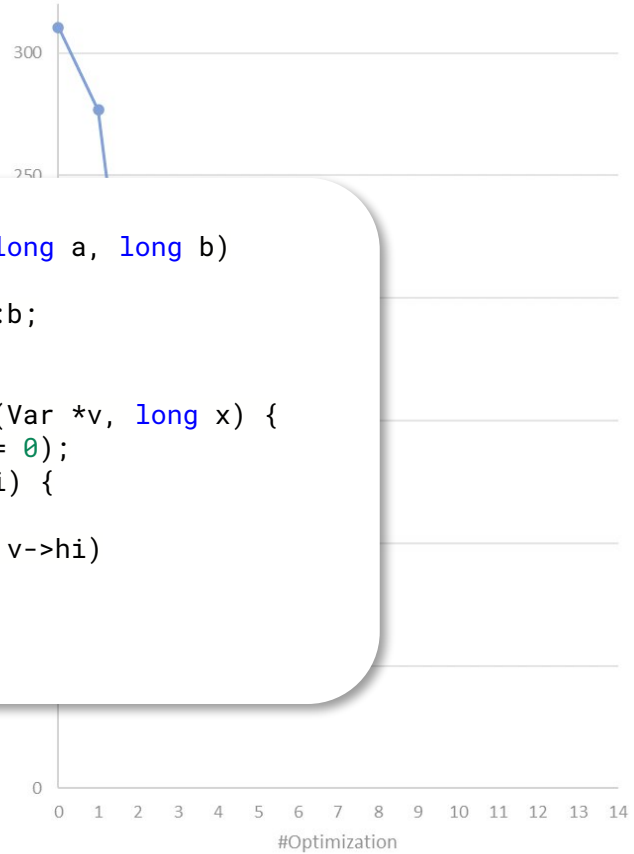
139,711,501,126 cycles (speedup **2.22**) (relative Δ3.92%)

Make all functions static

- Use `static` keyword for better inlining
- No nee
- Make fu

```
long max(long a, long b)
{
  return (a>b)?a:b;
}

int sethi(Var *v, long x) {
  assert(v->id >= 0);
  if (x < v->hi) {
    v->hi = x;
    if (v->lo <= v->hi)
      return 1;
...
```

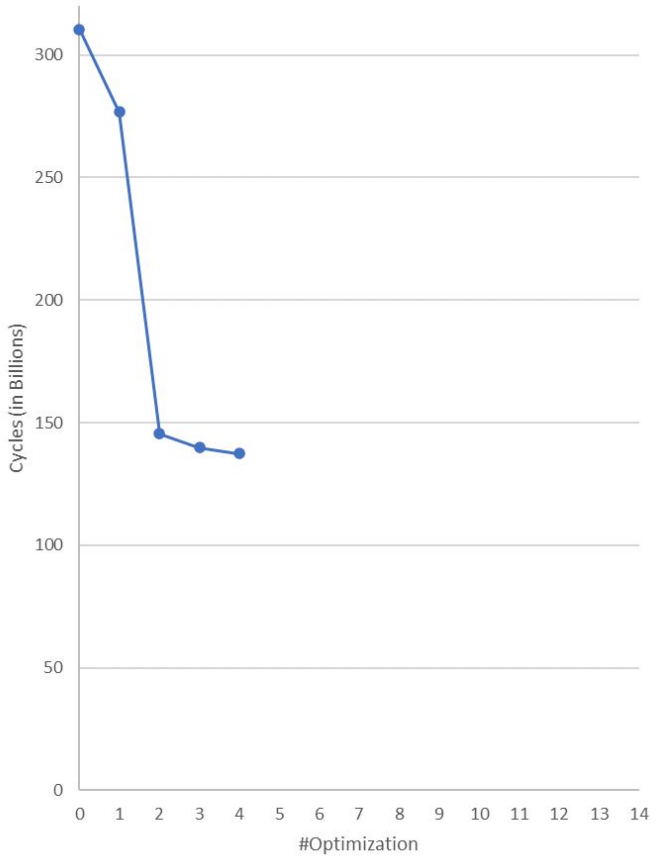
```
static long max(long a, long b)
{
  return (a>b)?a:b;
}

static int sethi(Var *v, long x) {
  assert(v->id >= 0);
    if (x < v->hi) {
      v->hi = x;
    if (v->lo <= v->hi)
      return 1;
...
```



139,711,501,126 cycles (speedup **2.22**) (relative Δ3.92%)

Inline and combine sum constraint function

- Inline the three function calls to `sum`
- Combine the three preparation loops
- Combine the three constraint check loops
- Use pointer arithmetic for each iteration pattern (line, column, diagonal)

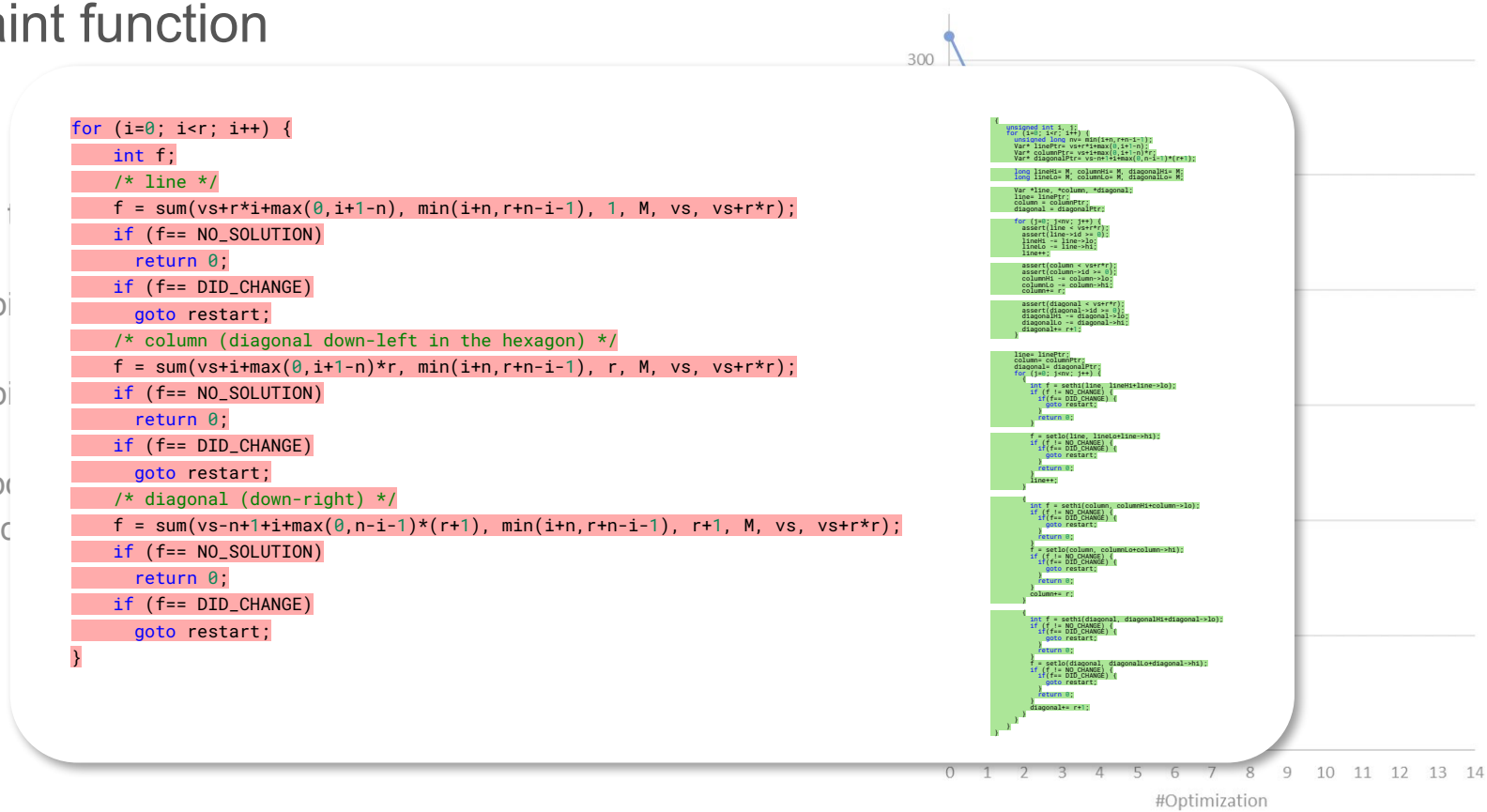


137,302,512,427 cycles (speedup **2.26**) (relative Δ1.72%)

Inline and combine sum constraint function

- Inline
- Combi
- Combi
- Use p
  (line, c

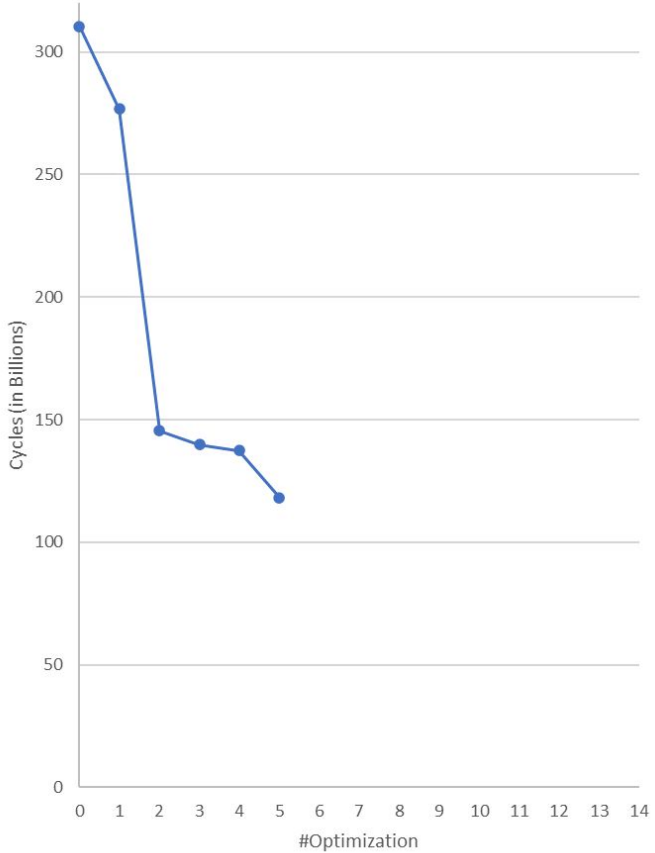
```
for (i=0; i<r; i++) {
    int f;
    /* line */
    f = sum(vs+r*i+max(0,i+1-n), min(i+n,r+n-i-1), 1, M, vs, vs+r*r);
    if (f== NO_SOLUTION)
      return 0;
    if (f== DID_CHANGE)
      goto restart;
    /* column (diagonal down-left in the hexagon) */
    f = sum(vs+i+max(0,i+1-n)*r, min(i+n,r+n-i-1), r, M, vs, vs+r*r);
    if (f== NO_SOLUTION)
      return 0;
    if (f== DID_CHANGE)
      goto restart;
    /* diagonal (down-right) */
    f = sum(vs-n+1+i+max(0,n-i-1)*(r+1), min(i+n,r+n-i-1), r+1, M, vs, vs+r*r);
    if (f== NO_SOLUTION)
      return 0;
    if (f== DID_CHANGE)
      goto restart;
}
```



137,302,512,427 cycles (speedup **2.26**) (relative Δ1.72%)

Update occupation array in-place in sum-constraint

- Check if variable boundary got fixed in sum-constraint checker
- Update occupation array instead of rebuilding it



118,185,261,745 cycles (speedup **2.63**) (relative Δ13.92%)

Update occupation array in-place in sum-

- Check
  sum-c
- Updat

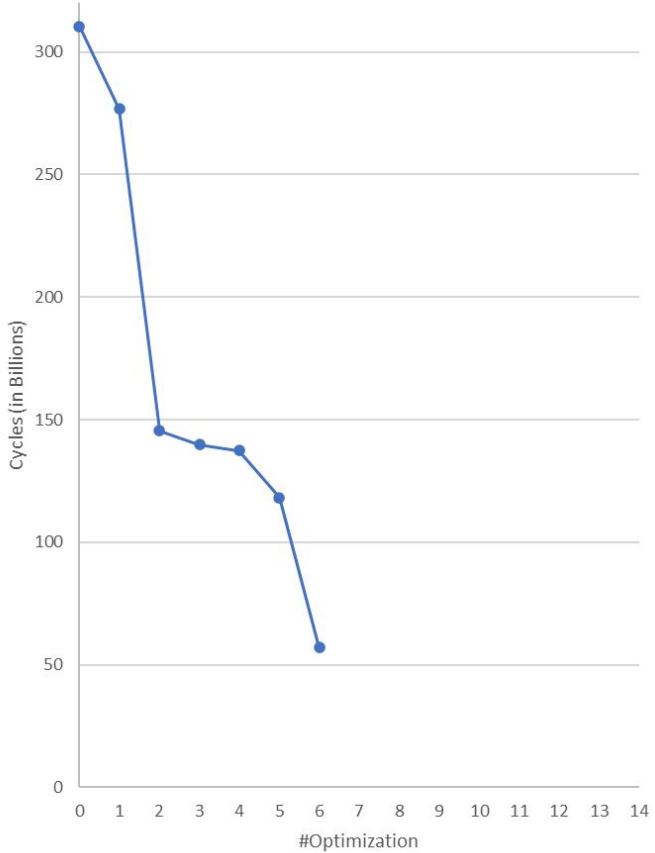
```
for (j=0; j<nv; j++) {
  {
    int f = sethi(line, lineHi+line->lo);
    if (f != NO_CHANGE) {
      if(f== DID_CHANGE) {
        goto restart;
      }
      return 0;
    }

    f = setlo(line, lineLo+line->hi);
    if (f != NO_CHANGE) {
      if(f== DID_CHANGE) {
        goto restart;
      }
      return 0;
    }
    line++;
  }
  ...
```

```
for (j=0; j<nv; j++) {
{
   int f = sethi(line, lineHi+line->lo);
   if (f != NO_CHANGE) {
     if(f== DID_CHANGE) {
       if( line->hi == line->lo ) {
         if( occupation[line->lo-o] < r*r ) {
           return 0;
         }
         occupation[line->lo-o]= line- vs;
       }

       goto restart_propagate_alldifferent;
     }
     return 0;
   }

   f = setlo(line, lineLo+line->hi);
   if (f != NO_CHANGE) {
     if(f== DID_CHANGE) {
       if( line->hi == line->lo ) {
         if( occupation[line->lo-o] < r*r ) {
           return 0;
         }
         occupation[line->lo-o]= line- vs;
       }

       goto restart_propagate_alldifferent;
     }
     return 0;
   }
  line++;
}
...
```

118,185,261,745 cycles (speedup **2.03**) (relative Δ15.92%)

Replace most restart jumps with a do-while-loop + flag

- Continue remaining restraint checks before restarting
- Loop condition checks if at least one variable was modified
- Differentiate between two restart types
  - Full restart (rebuild occupation array)
  - Constraint restart (start with `alldifferent`)



56,917,950,209 cycles (speedup **5.45**) (relative Δ51.84%)

Replace most restart jumps with a

```
restart:

for (i=0; i<r*r; i++) {
  Var *v = &vs[i];
  if (v->lo == v->hi && occupation[v->lo-o] != i) {
    if (occupation[v->lo-o] < r*r)
      return 0;
    occupation[v->lo-o] = i;
  }
}
```

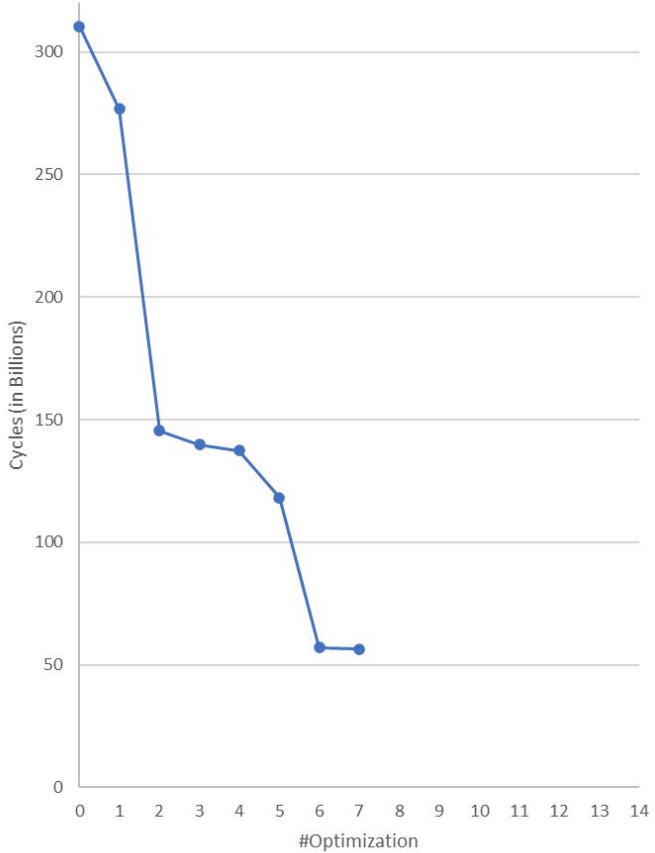
```
    unsigned int change= 0b10;
    do {
      if( change & 0b10 ) {
        for (i=0; i<r*r; i++) {
          Var *v = &vs[i];
          if(v->lo == v->hi && occupation[v->lo-o] != i) {
            if (occupation[v->lo-o] < r*r)
              return 0;
            occupation[v->lo-o] = i;
          }
        }
      }
      change= 0;
```

```
int f = lessthan(&vs[corners[0]],&vs[corners[i]]);
if (f== NO_SOLUTION)
  return 0;
if (f== DID_CHANGE)
  goto restart;
```

```
int f = lessthan(&vs[corners[0]],&vs[corners[i]]);
if (f== NO_SOLUTION)
  return 0;
if (f== DID_CHANGE)
  change |= 0b10;
```

Refactor repeating alldifferent improvements into loop

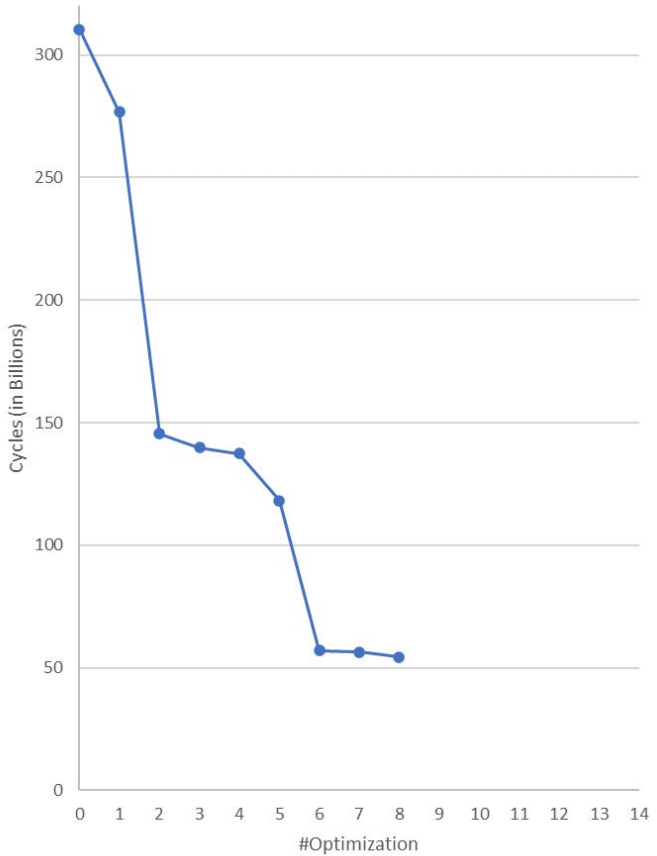
- Replace `goto` with a do-while loop to implement restarting



56,370,145,311 cycles (speedup **5.51**) (relative Δ0.96%)

Replace alldifferent constraint restart goto with flagged loop

- Check all variables before restarting even if a value gets fixated
- Implemented using a while loop + flag
  - Similar to the larger outer loop



54,395,824,405 cycles (speedup **5.71**) (relative Δ3.5%)

Replace alldifferent constraint restart g

- Check
  value
- Impler
  -

```
restart:
    for (i=0; i<r*r; i++) {
      Var *v = &vs[i];
      if (v->lo < v->hi) {
try_to_propagate_alldiff_again:
        if (occupation[v->lo-o] < r*r) {
          v->lo++;
          if( v->lo == v->hi ) {
            if( occupation[v->lo-o] < r*r ) {
              return 0;
            }
            occupation[v->lo-o]= i;
            goto restart;

          }
          goto try_to_propagate_alldiff_again;
        }

        if (occupation[v->hi-o] < r*r) {
          ...
        }


      }
    }
```

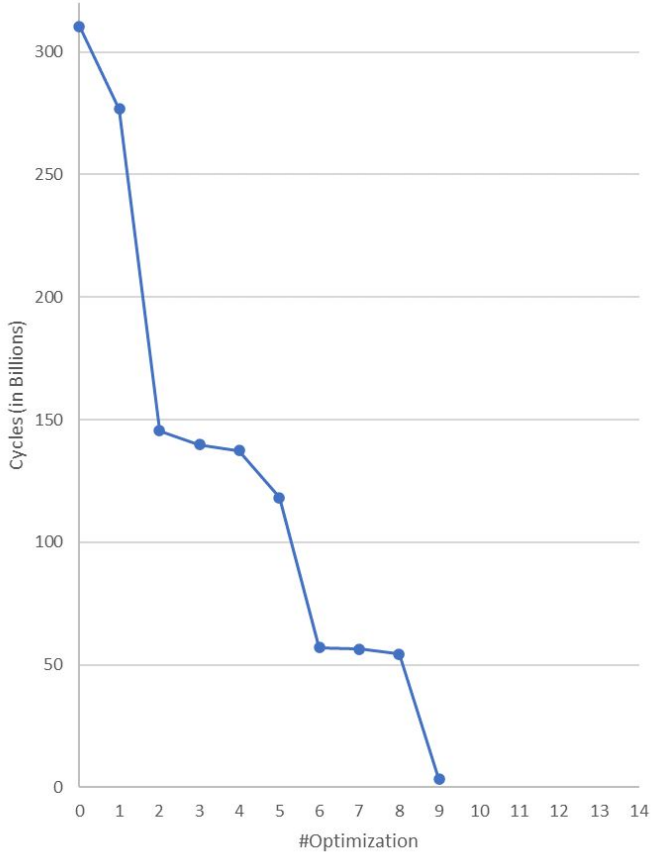
```
    unsigned int alldiffRestart;
    do {

      alldiffRestart= 0;
      for (i=0; i<r*r; i++) {
        Var *v = &vs[i];
        if (v->lo < v->hi) {
          do {
            if (occupation[v->lo-o] < r*r) {
              v->lo++;
              if( v->lo == v->hi ) {
                if( occupation[v->lo-o] < r*r ) {
                  return 0;
                }
                occupation[v->lo-o]= i;
                alldiffRestart= 1;
                break;
              }
              continue;
            }

            if (occupation[v->hi-o] < r*r) {
              ...
            }

            break;
          } while( 1 );
        }
      }

    } while( alldiffRestart );
```

10 11 12 13 14

54,395,824,405 cycles (speedup 3.71) (relative Δ3.5%)

Heuristicly select next variable to set while labeling

- Use “maximum lower bound” heuristic
- Global array to track remaining variables available for selection
- Use unstable take in O(1) time



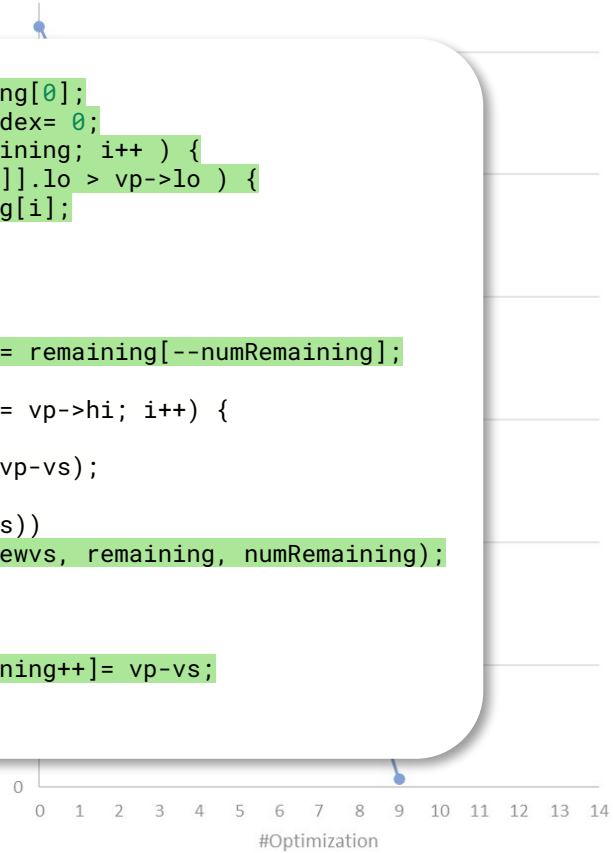
3,364,493,696 cycles (speedup **92.24**) (relative Δ93.81%)

Heuristicly select next variable to set while labeling

- Us
- Us
  be
- Us

```
if (vp->id < 0)
  return labeling(n,d,vs,index+1);



for (i = vp->lo; i <= vp->hi; i++) {
  Var newvs[r*r];
  Var* newvp=newvs+index;
  ...
  if (solve(n,d,newvs))
    labeling(n,d,newvs,index+1);
  else
    leafs++;


}
```

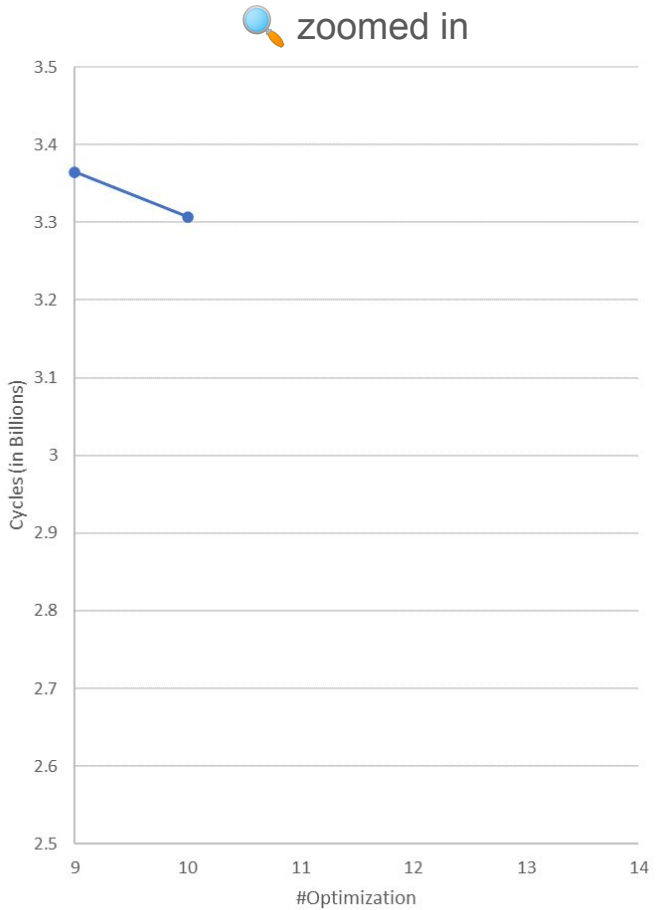
```
    Var *vp= vs+ remaining[0];
    unsigned long takeIndex= 0;
    for( i= 1; i<numRemaining; i++ ) {
      if( vs[remaining[i]].lo > vp->lo ) {
        vp= vs+ remaining[i];
        takeIndex = i;
      }
    }

    remaining[takeIndex]= remaining[--numRemaining];

    for (i = vp->lo; i <= vp->hi; i++) {
      Var newvs[r*r];
      Var* newvp=newvs+(vp-vs);
      ...
      if (solve(n,d,newvs))
        labeling(n, d, newvs, remaining, numRemaining);
      else
        leafs++;

      remaining[numRemaining++]= vp-vs;
    }
```



3,364,493,696 cycles (speedup **92.24**) (relative Δ93.81%)

Kickstart the heuristic by first picking a corner

- Allow preselection of variables before using the labeling heuristic
- Create a list of pre-ordered variables that are taken before considering the heuristic



3,307,299,769 cycles (speedup **93.84**) (relative Δ1.7%)

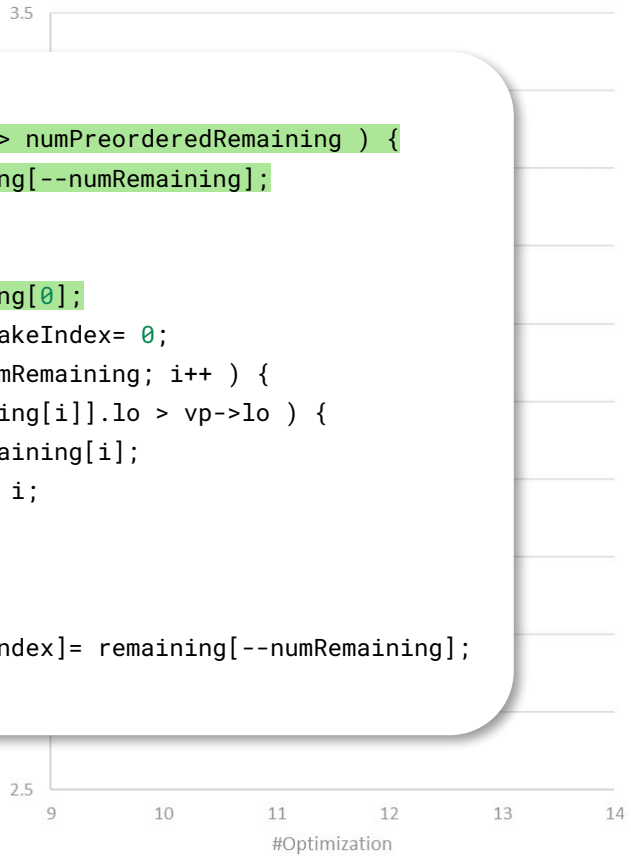
Kickstart the heuristic by first picking a corner

- A
  l
- (
  t

```
Var *vp= vs+ remaining[0];
unsigned long takeIndex= 0;
for( i= 1; i<numRemaining; i++ ) {
  if( vs[remaining[i]].lo > vp->lo ) {
    vp= vs+ remaining[i];
    takeIndex = i;
  }
}

remaining[takeIndex]= remaining[--numRemaining];
```

```
Var *vp;
if( numRemaining > numPreorderedRemaining ) {
  vp= vs+ remaining[--numRemaining];

} else {
  vp= vs+ remaining[0];
  unsigned long takeIndex= 0;
  for( i= 1; i<numRemaining; i++ ) {
    if( vs[remaining[i]].lo > vp->lo ) {
      vp= vs+ remaining[i];
      takeIndex = i;
    }
  }

  remaining[takeIndex]= remaining[--numRemaining];
}
```

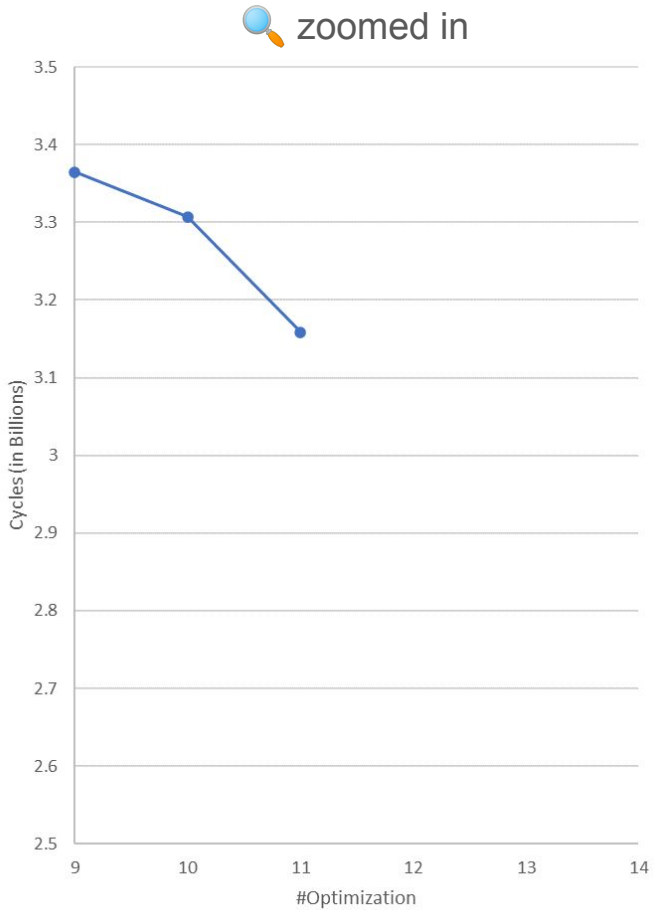
🔍 zoomed in



3,307,299,769 cycles (speedup **93.84**) (relative Δ1.7%)

Try to bisect the variable range while labeling

- Try bisect variable range if range > 3
- Allows earlier detection of bad branches



3,158,364,801 cycles (speedup **98.26**) (relative Δ4.5%)

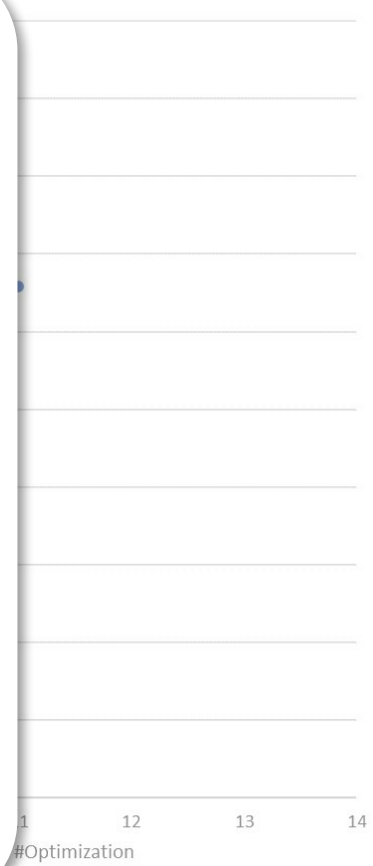
Try to bisect the variable range while labeling

- Try bisect variab
- Allows earlier de

```
for (i = vp->lo; i <= vp->hi; i++) {
  Var newvs[r*r];
  Var* newvp=newvs+(vp-vs);
  memmove(newvs,vs,r*r*sizeof(Var));
  newvp->lo = i;
  newvp->hi = i;

  if (solve(n,d,newvs))
    labeling( ... );
  else
    leafs++;
}
remaining[numRemaining++]= vp-vs;
```

```
Var newvs[r*r];
long hi= vp->hi, lo= vp->lo;
long range= hi - lo;
if( range < 4 ) {
  for (i = lo; i <= hi; i++) {

    Var* newvp=newvs+(vp-vs);
    memmove(newvs,vs,r*r*sizeof(Var));
    newvp->lo = i;
    newvp->hi = i;

    if (solve(n,d,newvs))
      labeling( ... );
    else
      leafs++;
  }
  remaining[numRemaining++]= vp-vs;
  return;
}

long middle= lo+ range / 2;
remaining[numRemaining++]= vp-vs;

Var* newvp=newvs+(vp-vs);
memmove(newvs,vs,r*r*sizeof(Var));
newvp->lo = lo;
newvp->hi = middle;

if (solve(n,d,newvs))
  labeling( ... );
else
  leafs++;

memmove(newvs,vs,r*r*sizeof(Var));
newvp->lo= middle+1;
newvp->hi= hi;
if (solve(n,d,newvs))
  labeling( ... );
else
  leafs++;
```

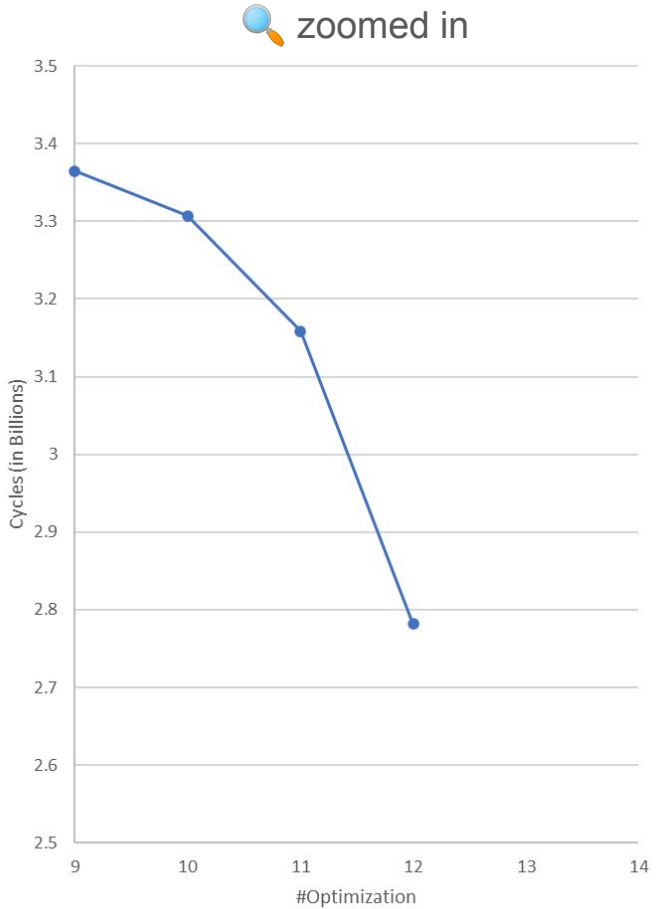
zoomed in



3,158,364,801 cycles (speedup **98.**

Disable assertions

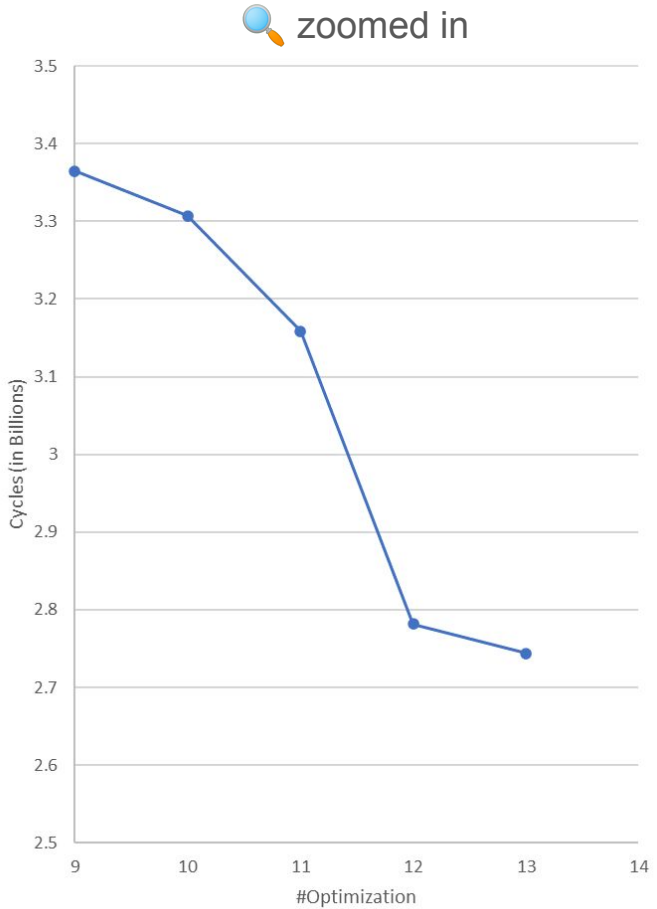
- Use the DNDEBUG compiler flag
- Removes assertion checks using preprocessor



2,781,654,645 cycles (speedup **111.57**) (relative Δ11.93%)

Remove the id field from variables for non debug builds

- Id field is only used by debug assertions
- Pack the variable struct tighter
  - Less stack usage
  - Less memory to copy
  - Better cache usage



2,743,846,568 cycles (speedup **113.1**) (relative Δ1.36%)

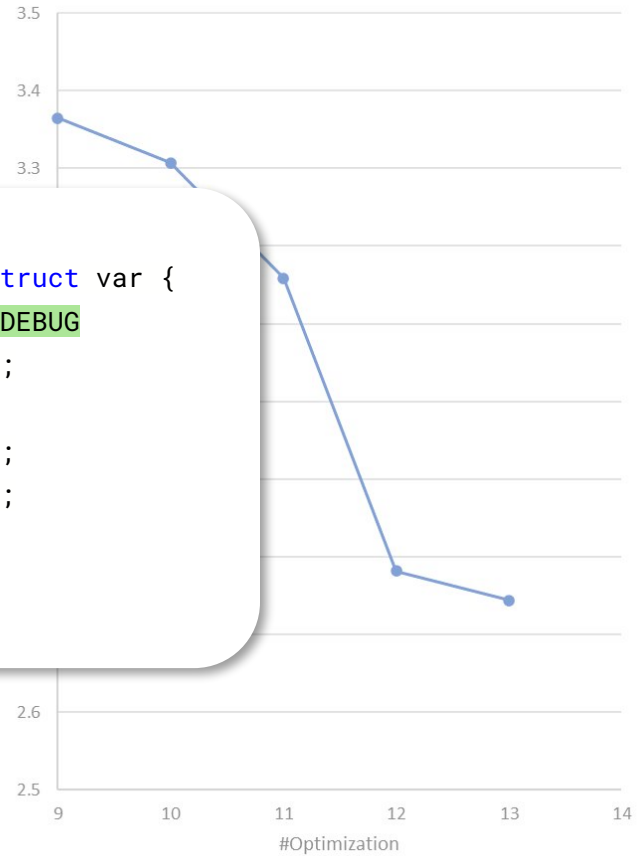
Remove the id field from variables for non debug builds

- Id field is only used by debug assertions
- Pack the variable
  - Less stack
  - Less memo
  - Better cach

```
typedef struct var {

  long id;

  long lo;
  long hi;
} Var;
```

```
typedef struct var {
#ifndef NDEBUG
  long id;
#endif
  long lo;
  long hi;
} Var;
```

🔍 zoomed in



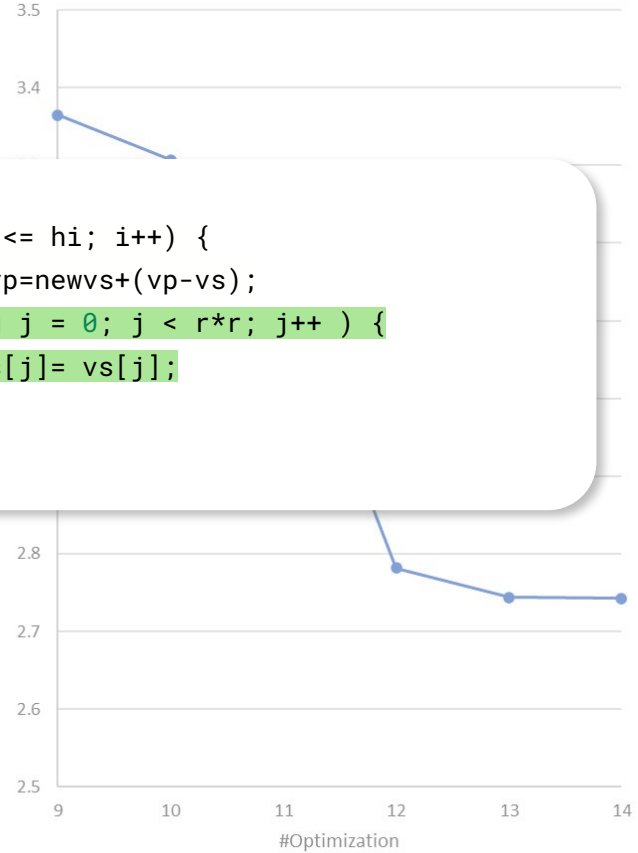
2,743,846,568 cycles (speedup **113.1**) (relative Δ1.36%)

Replace memove with simple loops

```
for (i = lo; i <= hi; i++) {
      Var* newvp=newvs+(vp-vs);
      memmove(newvs,vs,r*r*sizeof(Var));
```

```
for (i = lo; i <= hi; i++) {
      Var* newvp=newvs+(vp-vs);
      for( long j = 0; j < r*r; j++ ) {
          newvs[j]= vs[j];
      }
```

🔍 zoomed in



2,742,664,915 cycles (speedup **113.15**) (relative Δ0.04%)

📈 Exercise Statistics

Version	**Cycles** [Billions]	**Instructions** [Billions]	**Cycle Speedup***	**Time**** [s]	**Leaves**	**Solutions**
Original	310.34	897.83	-	66.32	15,809,528	40
Final	2.74	7.47	113.15	0.59	384,599	40
Final (-1 pre-select)	52.49	142.76	80.42	11.19	7,362,483	530
Final (-2 pre-select)	977.21	2,613.93	-	208.73	137,607,460	12,641

*Compared to running same configuration on original
**5 runs median on submission server

🗑️ Some Ineffective Changes

- Manual inlining of `lessthan` in two places
- Precompute `r`, `H`, `M` and make them global (including `n` & `d`)
- Iterative labeling
  - State machine with custom stack
  - Naive flattening with a FIFO queue (no allocations, still slower)
- Loop unrolling
  - Occupation array init-loop
  - Sum constraint init-loop
- Compute `sethi`/`setlo` return codes with bit fiddling

Thank you for your attention