LLVM Polly evaluation

cschen 陳鍾樞 2014/06/18
2015/03/31 update

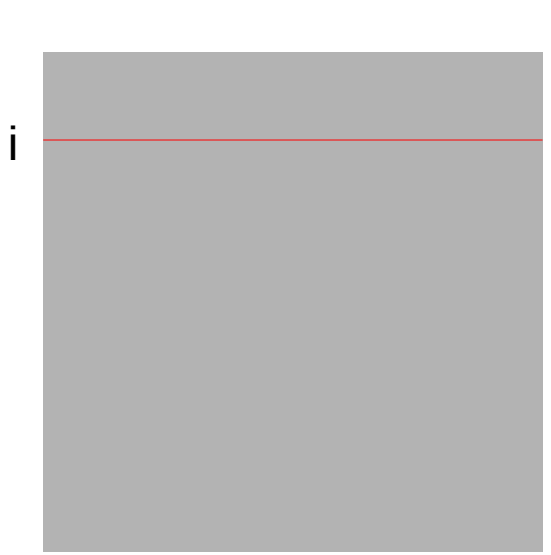
Outline

- Loop tiling
  - Concept
    - Reduce data cache miss (A*B) & correctness proof
  - Analysis by oprofiler (loop bounds effect)
  - Current limitation (polly is still improving)
- Polybench-c-3.2 (tiling v.s. No-tiling)
- Vectorize
- Muti-core
- Other optimization opportunity
- Reference

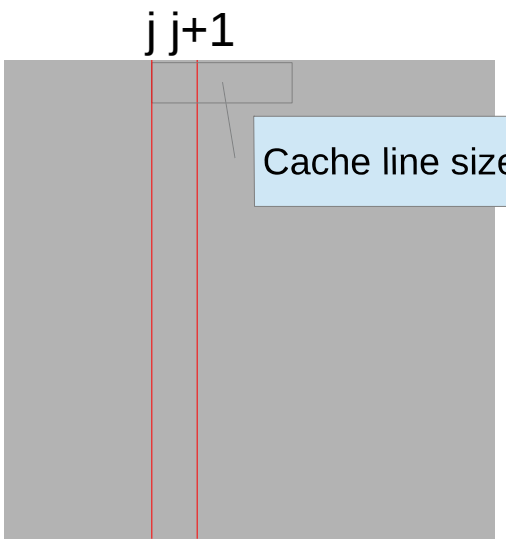
Data cache miss – regarding B

```
do i = 1,N
  do j = 1,N
    do k = 1,N
      C(i,j) = C(i,j)
          + A(i,k) * B(k,j)
    enddo
  enddo
enddo
```

Assume data cache line size is 4 words,
After B(-,j) is read, the B(-,j+1) is in cache too.
When N is large, the A(i,-)*B(-,j+1) will be cache miss.
B(i,j) B(i,j+1): spatial locality.



A



B

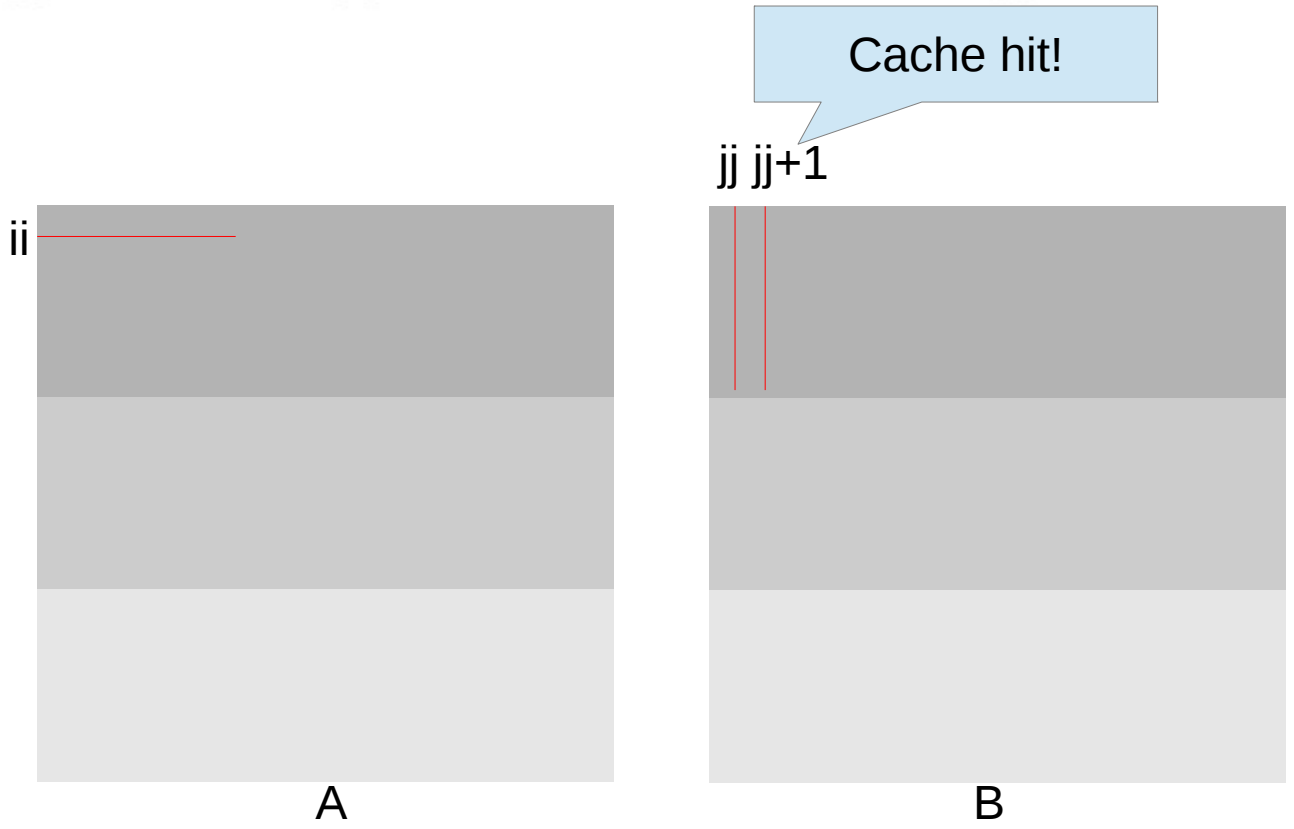
Tiling

```
do i = 1,N,T
  do ii = i,min(i+T-1,N)
    do j = 1,N,T
      do jj = j,min(j+T-1,N)
        do k = 1,N,T
          do kk = k,min(k+T-1,N)
            C(ii,jj) = C(ii,jj) + A(ii,kk) * B(kk,jj)
          enddo
        enddo
      enddo
    enddo
  enddo
enddo
```

Tile size

Tiling - Reduce data cache miss – Regarding B

```
do i = 1,N,T
  do j = 1,N,T
    do k = 1,N,T
      do ii = i,min(i+T-1,N)
        do jj = j,min(j+T-1,N)
          do kk = k,min(k+T-1,N)
            C(ii,jj) = C(ii,jj) + A(ii,kk) * B(kk,jj)
          enddo
        enddo
      enddo
    enddo
  enddo
enddo
```



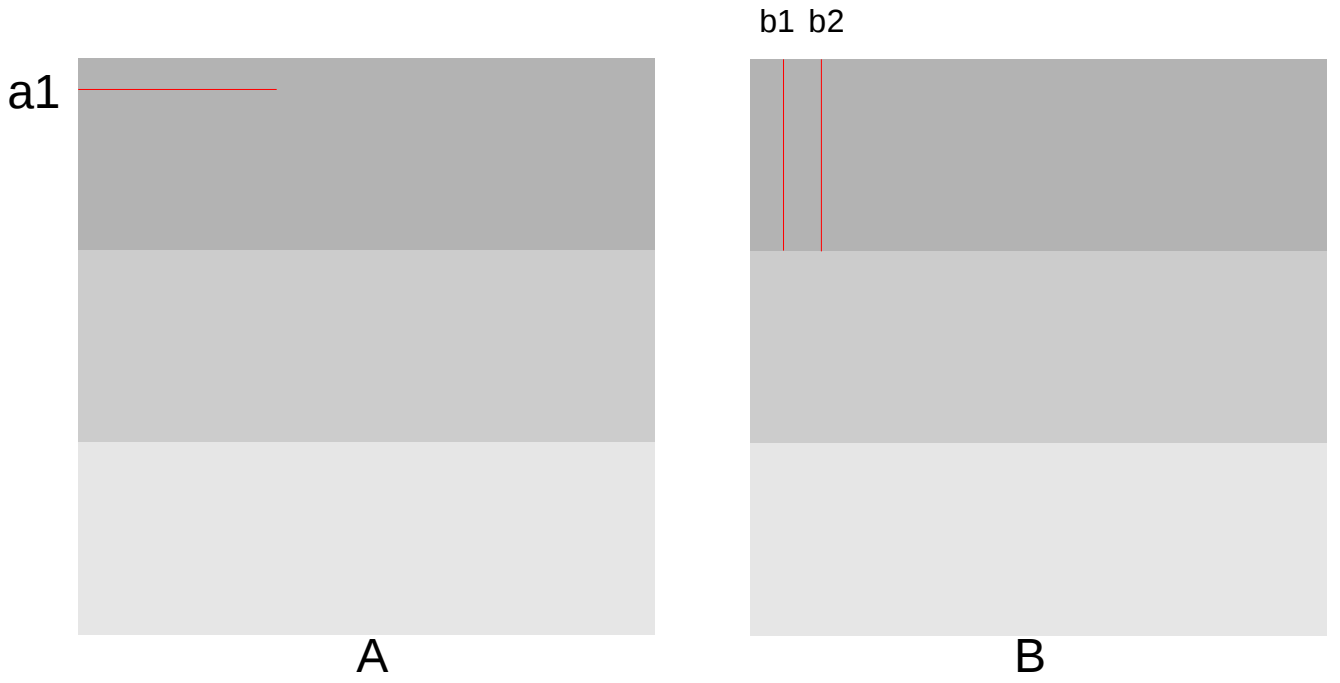
Tiling - Reduce data cache miss – Regarding A

```
do i = 1,N,T
  do j = 1,N,T
    do k = 1,N,T
      do ii = i,min(i+T-1,N)
        do jj = j,min(j+T-1,N)
          do kk = k,min(k+T-1,N)
            C(ii,jj) = C(ii,jj) + A(ii,kk) * B(kk,jj)
          enddo
        enddo
      enddo
    enddo
  enddo
enddo
```

1. a1*b1
2. a1*b2 cache hit
3. a1*b3 hit

…

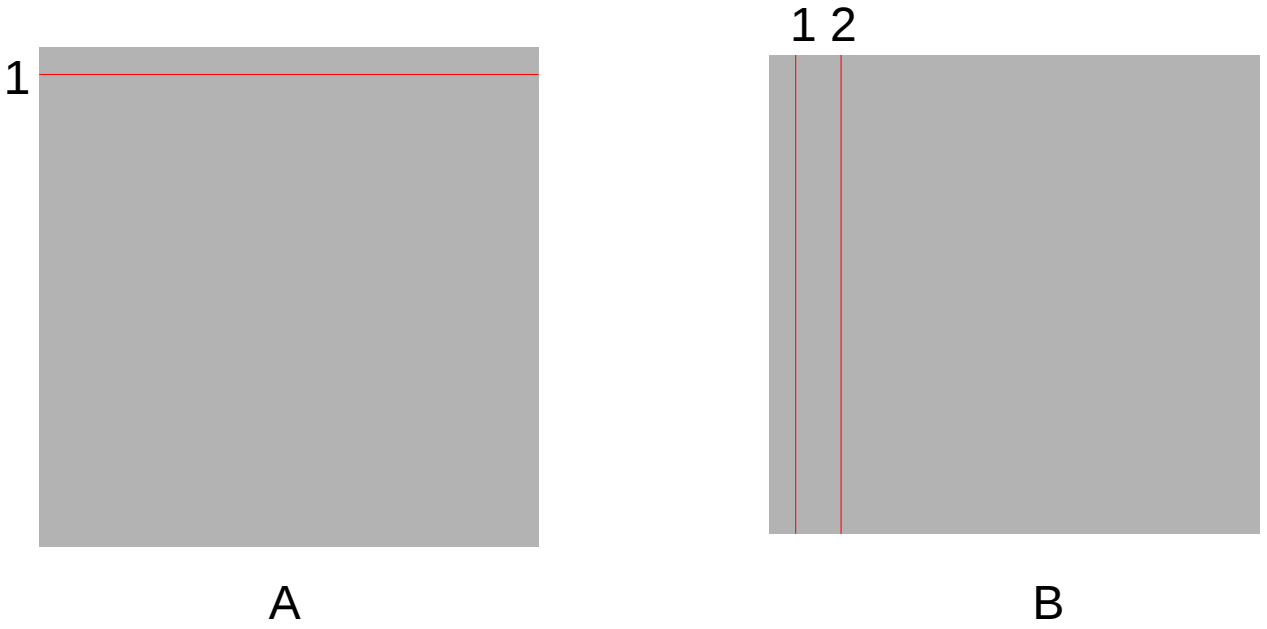
a1 hit T-1 times, miss 1 time



Data cache miss – Regarding A

```
do i = 1,N
  do j = 1,N
    do k = 1,N
      C(i,j) = C(i,j)
          + A(i,k) * B(k,j)
    enddo
  enddo
enddo
```

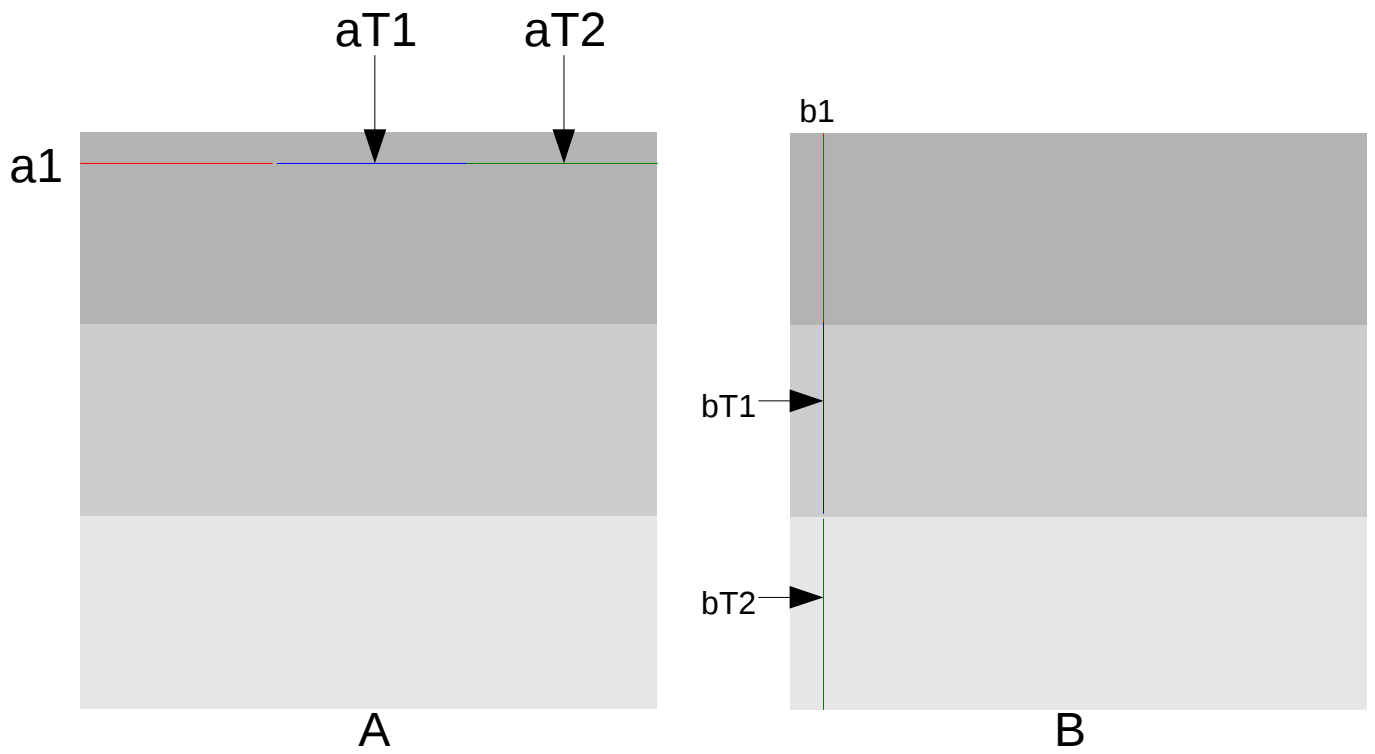
1. A[1,1]*B[1,1]
…
2. A[1,N/2+1]*B[N/2+1]
 (A[1,N/2+1] push A[1,1] out)
...
3. A[1,1]*B[2,1] (cache miss A[1,1])
…
A[1,1] never hit
Miss temporal locality



Tiling correctness – proof

```
do i = 1,N,T
  do j = 1,N,T
    do k = 1,N,T
      do ii = i,min(i+T-1,N)
        do jj = j,min(j+T-1,N)
          do kk = k,min(k+T-1,N)
            C(ii,jj) = C(ii,jj) + A(ii,kk) * B(kk,jj)
          enddo
        enddo
      enddo
    enddo
  enddo
enddo
```

1. a1*b1 (k=1)
2. aT1*bT1 (k=T+1)
3. aT2*bT2 (k=2T+1)



Tiling - note

- Tile size select:
  - The best value depends on:
    - cache size, cache way, loop bounds, program
    - Regard matmul.c. My X86-64 PC → 32; BG2 → 12
  - So, the size is chosen by programmer as follows
    - 1. Init T refer to cache setting & program
    - 2. Try different T and choose the best value
- Other name:
  - Strip mine and interchange – original paper – Wolfe
  - Unroll and jam - Callahan

Oprofile – analysis on X86-64

- sudo time ./ocount --events=l1d_pend_miss:pending ./matmul
  - l1d_pend_miss:

    0x01: (name=pending) L1D miss oustandings duration in cycles
- matmul.c:
  - As previous slide example code as well as the data type is float.
- Compiler option:
  - clang -O3 matmul.c (no tile)
  - pollycc -mllvm -polly -mllvm -polly-ignore-aliasing -O3 matmul.c (tile)
- My X86-64 Core i7:
  - L1: data cache size 32KB, i cache size 32KB
  - L2: 256KB
  - L3: 8MB

Oprofile – analysis on X86-64

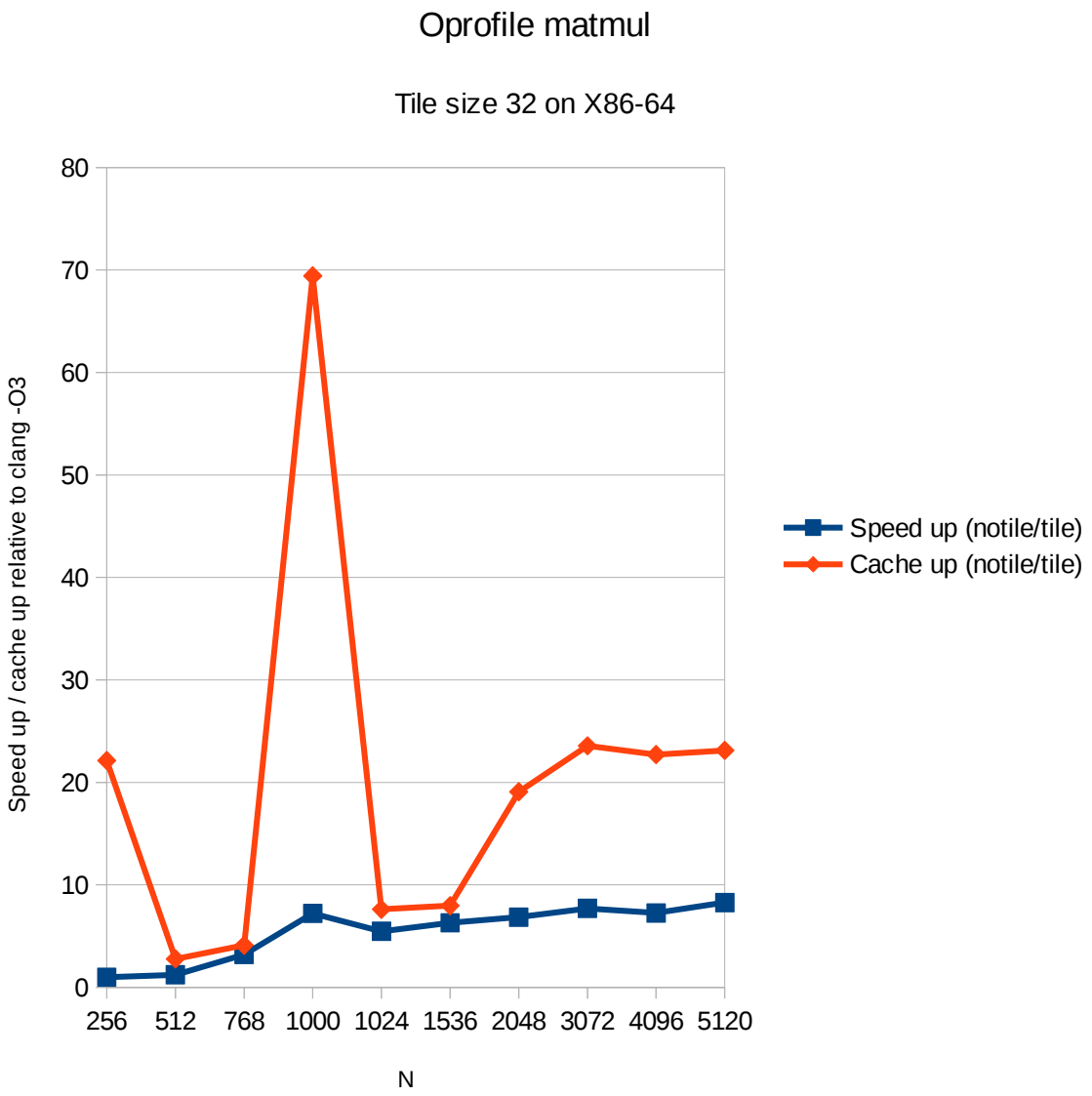
- N=256
  - (0.018 Sec/0.018=1, 0.332 Giga cycles/0.015=22)
- N=512
  - (0.16/0.13=1.2, 3.9/1.4=2.7)
- N=768
  - (1.35/0.42=3.2, 11.1/2.7=4.1)
- N=1000
  - (5.92/0.82=7.2, 13.19/0.19=69.4)
- N=1024
  - (6.62/1.21=5.4, 103.6/13.6=7.6)
- N=1536
  - (22.7/3.6=6.3, 277.5/34.8=7.9)
- N=2048
  - (65.8/9.6=6.8, 1792.4/93.9=19.0
- N=3072
  - (239.9/33.0=7.7, 6714.1/375.5=23.5)
- N=4096
  - (644.9/84.2=7.2, 18740.3/821.2=22.7) (661.5/83.9=7.8, 19326.8/793.2=24.3)
- N=5120
  - (1235.6/149.3=8.2, 33463.1/1447.0=23.1) (1198.8/150.0=7.9, 33551.3/1469.6=22.8)

Cache up 22 but the total data cache miss amount is low as 0.332G. The Instruction cycles take most of time since matrix multiplication is O(N*N*N).

When I use int instead of float result (0.23/0.16=1.4, 0.286/0.012=23.8)

Note:
1. Both the instruction cycles and data cache miss is variable.
2. When N is small the data cache miss got 10 times variance if run the same program repeatedly, but when N is 3072 the variance is in 10%.

Oprofile – analysis on X86-64

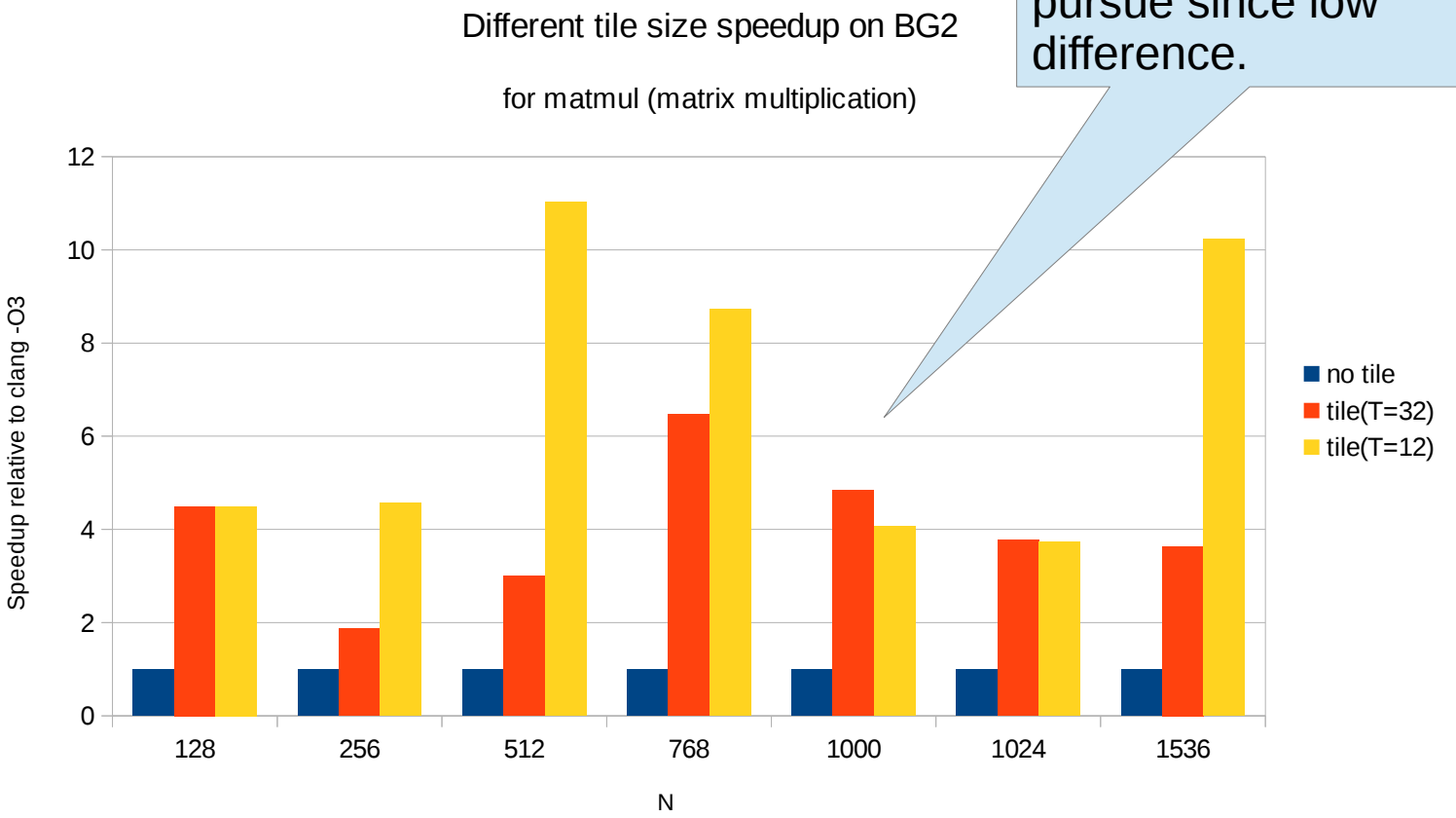


A particular combination of loop bounds and tile size can be very effective for Reducing collision misses
…
20.4.3 of ref 1.

Conclusion:
1. N up => (no tile / tile) data cache miss up significantly.
2. N 1024 → 5120, data cache miss 7.6 → 23, speed up 5.4 → 8.2 (because instruction cycles up too).

Matmul.c - tiling size 32 v.s. 12

For N=1000, best tile size is 32 but not deserve to pursue since low difference.



Oprofile – on X86-64 – clang(O3/polly) vs gcc (O0/O3)

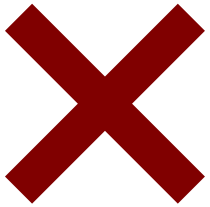
Clang -O3/-O3+polly

- N=256
  - (0.018 Sec/0.018, 0.332 Giga cycles/0.015)
- N=512
  - (0.16/0.13, 3.9/1.4)
- N=768
  - (1.35/0.42, 11.1/2.7)
- N=1024
  - (6.62/1.21, 103.6/13.6)
- N=1536
  - (22.7/3.6, 277.5/34.8)
- N=2048
  - (65.8/9.6, 1792.4/93.9)
- N=3072
  - (239.9/33.0, 6714.1/375.5)

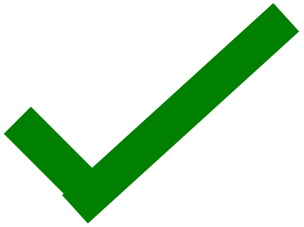
Gcc -O0/O3

- N=256
  - (0.078 Sec/0.011, 0.285 Giga cycles/0.088)
- N=512
  - (0.56/0.05, 2.6/0.9)
- N=768
  - (2.50/0.35, 10.1/2.8)
- N=1024
  - (15.05/1.72, 85.9/24.5)
- N=1536
  - (58.8/6.1, 385.8/155.0)
- N=2048
  - (142.2/18.8, 1357.5/539.0)
- N=3072
  - (609.6/62.5, 5127.0/1850.1)

Current limitation



```
static int m = 1000;
static int n = 1000;
int i, j, j1, j2;
for (j = 0; j < m; j++)
{
  mean[j] = 0.0;
  for (i = 0; i < n; i++)
    mean[j] += data[i][j];
  mean[j] /= float_n;
}
```



```
#define M 1000
#define N 1000
for (j = 0; j < M; j++)
{
  mean[j] = 0.0;
  for (i = 0; i < N; i++)
    mean[j] += data[i][j];
  mean[j] /= float_n;
}
```

Current limitation

```
// polybench.h
/* Scalar loop bounds in SCoPs. By default, use parametric loop bounds. */
# ifdef POLYBENCH_USE_SCALAR_LB
#  define POLYBENCH_LOOP_BOUND(x,y) x
# else
/* default: */
#  define POLYBENCH_LOOP_BOUND(x,y) y
# endif

// covariance.h
#  ifdef STANDARD_DATASET /* Default if unspecified. */
#   define N 1000
#   define M 1000
#  endif

# define _PB_N POLYBENCH_LOOP_BOUND(N,n)
# define _PB_M POLYBENCH_LOOP_BOUND(M,m)
```

Current limitation

```
// covariance.c
static
void kernel_covariance(int mm, int nn,
                       DATA_TYPE float_n,
                       DATA_TYPE POLYBENCH_2D(data,M,N,m,n),
                       DATA_TYPE POLYBENCH_2D(symmat,M,M,m,m),
                       DATA_TYPE POLYBENCH_1D(mean,M,m))
{
  static int m = 1000;
  static int n = 1000;
  int i, j, j1, j2;
  ...
  /* Calculate the m * m covariance matrix. */
  for (j1 = 0; j1 < _PB_M; j1++)
    for (j2 = j1; j2 < _PB_M; j2++)
      {
        symmat[j1][j2] = 0.0;
        for (i = 0; i < _PB_N; i++)
          symmat[j1][j2] += data[i][j1] * data[i][j2];
        symmat[j2][j1] = symmat[j1][j2];
      }
  ...
```

Current limitation

- Clang option and -D need:
  - Polly-ignore-aliasing, -DPOLYBENCH_USE_SCALAR_LB
  - Author (Tobias) mail as follows,

-polly-ignore-aliasing:

We still miss run-time alias checks, so you need to declare that no aliasing will happen

-DPOLYBENCH_USE_SCALAR_LB

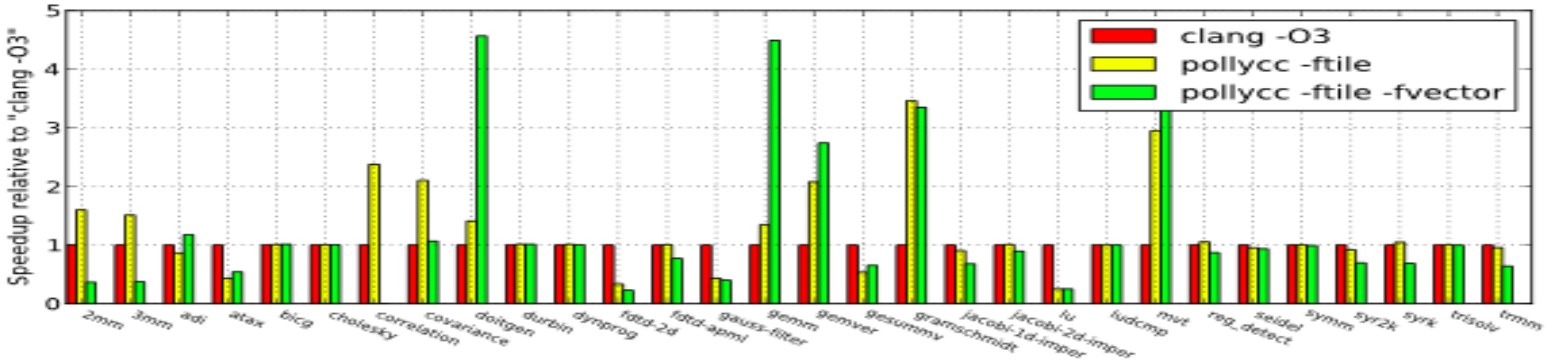
Otherwise, polybench will generate a mix of scalar and parametric loop bounds. We can work with that, but the generated code still needs some tuning.

Cheers,

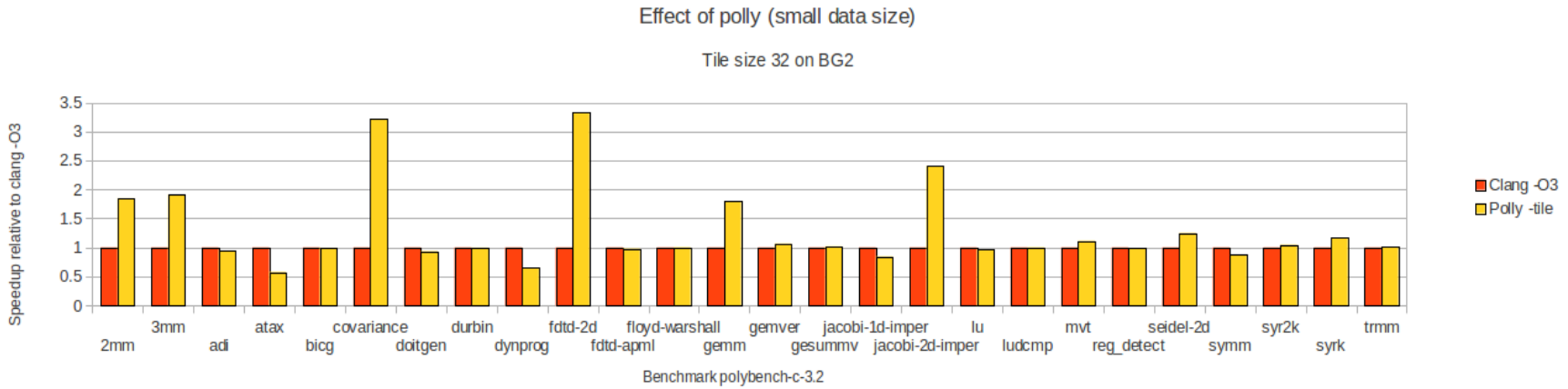
Tobias

Polybench-c-3.2 - tiling v.s. No-tiling

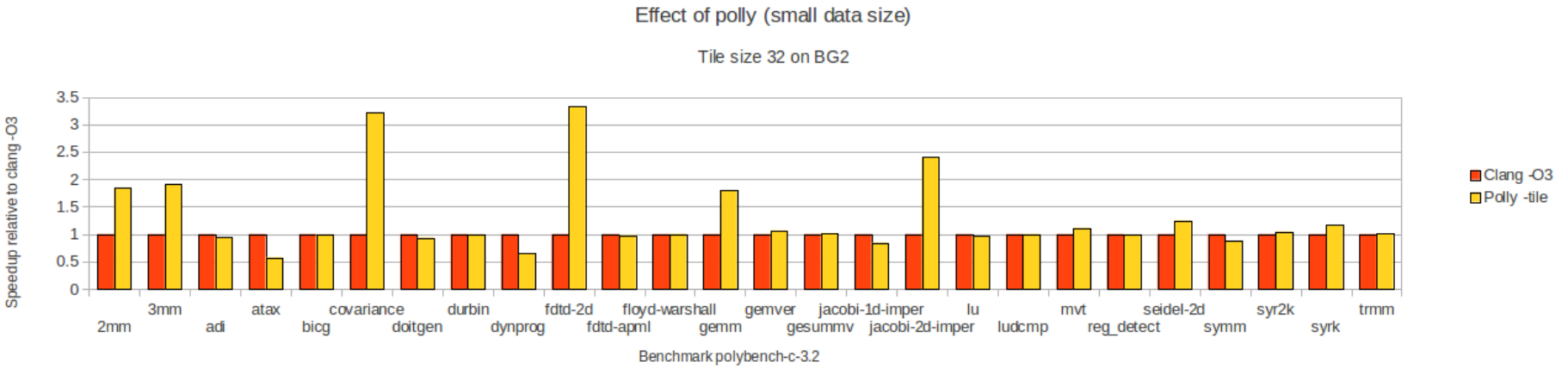
- Small data size

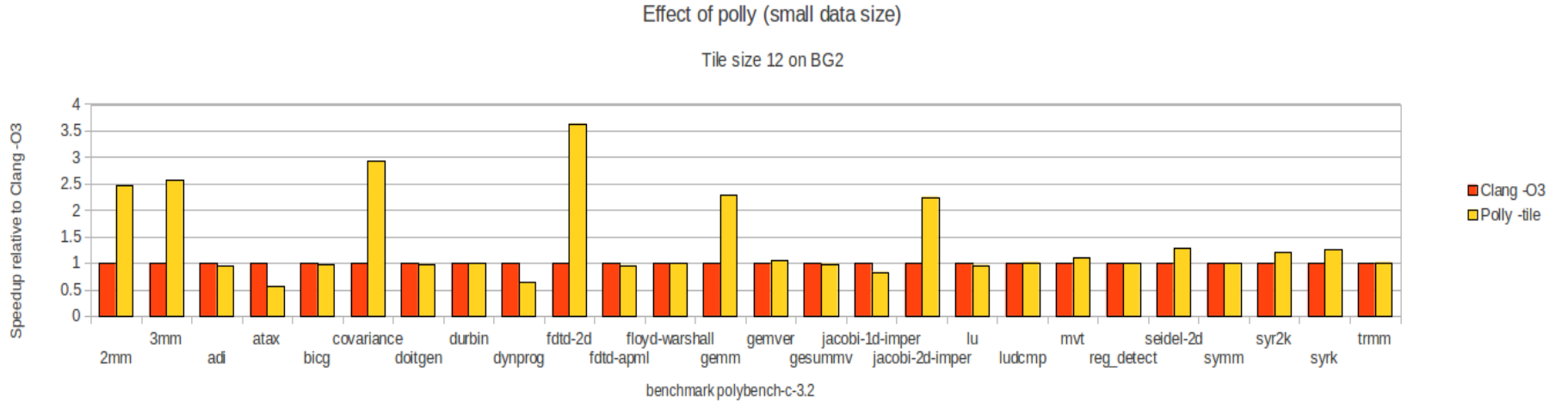


- Result: My build and run on BG2



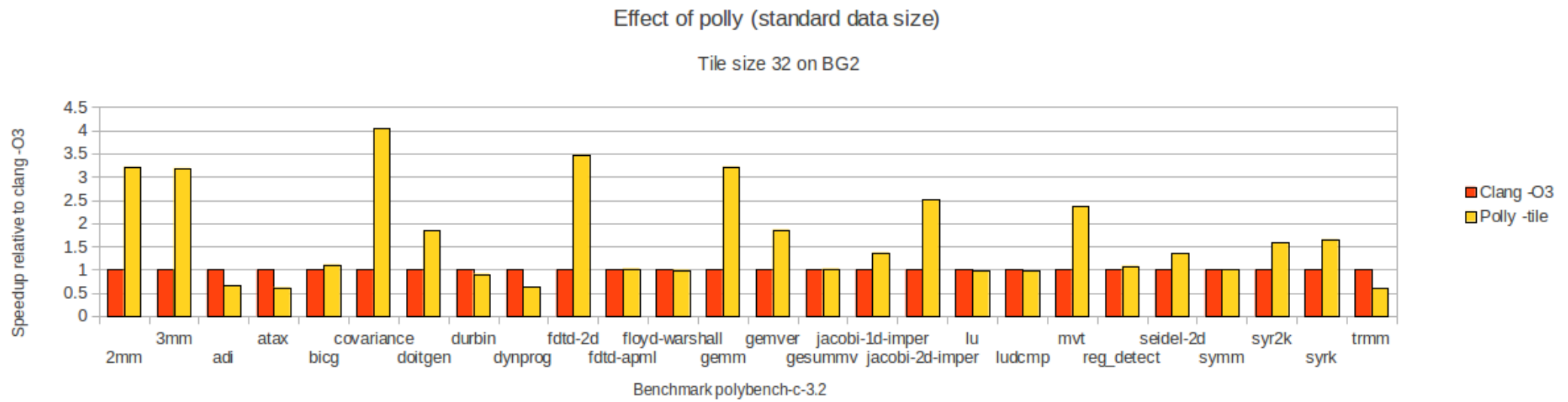
Polybench-c-3.2 – small data size - tiling size 32 v.s. 12

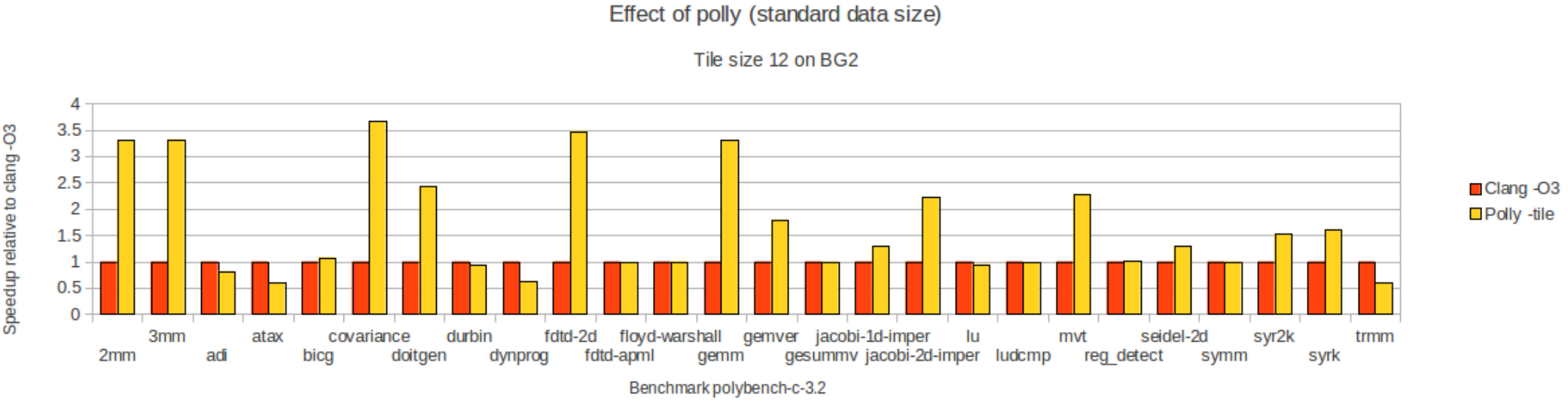




For different benchmark, the N is different.

Polybench-c-3.2 – standard data size - tiling size 32 v.s. 12





Loop optimization – data locality & parallel

- Data locality & parallel optiomization are same thing in loop optimization

```
for (i = 1; i <= 100; i++)
    for (j = 1; j <= 100; j++) {
        X[i,j] = X[i,j] + Y[i-1,j];  /* (s1) */
        Y[i,j] = Y[i,j] + X[i,j-1];  /* (s2) */
    }
```

Figure 11.26: A loop nest exhibiting long chains of dependent operations

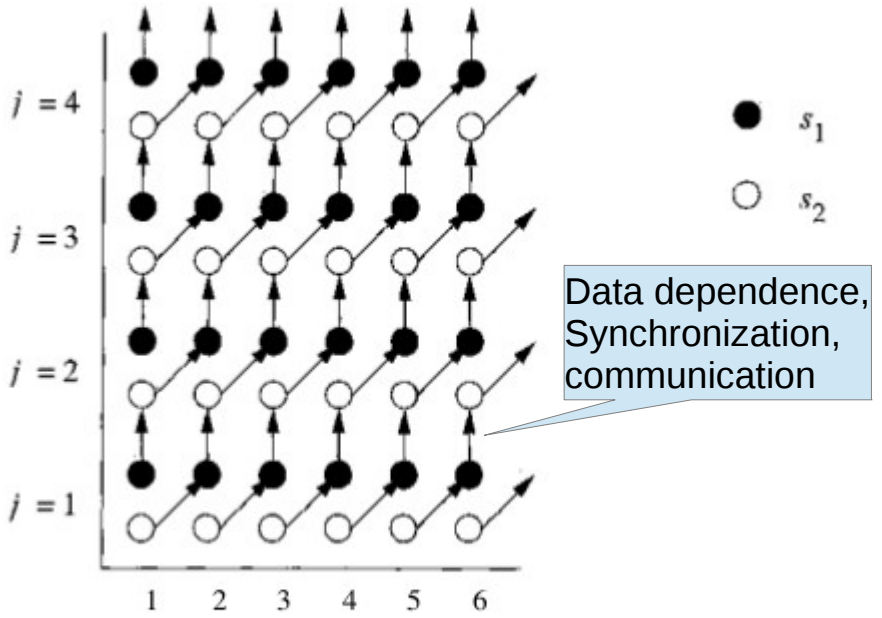


Figure 11.27: Dependences of the code in Example 11.41

Loop optimization – data locality & parallel

- Create M[2][100][100] can formulate this example
- By Polyhedra (convex region with polynomial form) is more effective to formulate dependence relationship
- Polyhedra model is a systematic way for loop optimization in compiler
- Leave the concept of Polyhedra introduction next time

Vectorize

- Polly version 3.4 has no effect in vector instruction but the previous slide has it
  - Clang has vectorizer now (as Tobias's following email)

Hi chencs,

the numbers you are seeing are from 2011 and should probably be updated. If you want to reproduce them, you need to use the corresponding LLVM/Polly versions.

If you are interested in the Polly version from today, the numbers probably do not apply any more. LLVM meanwhile got a vectorizer which should have improved the LLVM vector baseline. The benefits of tiling should still be as visible as before.

Cheers,

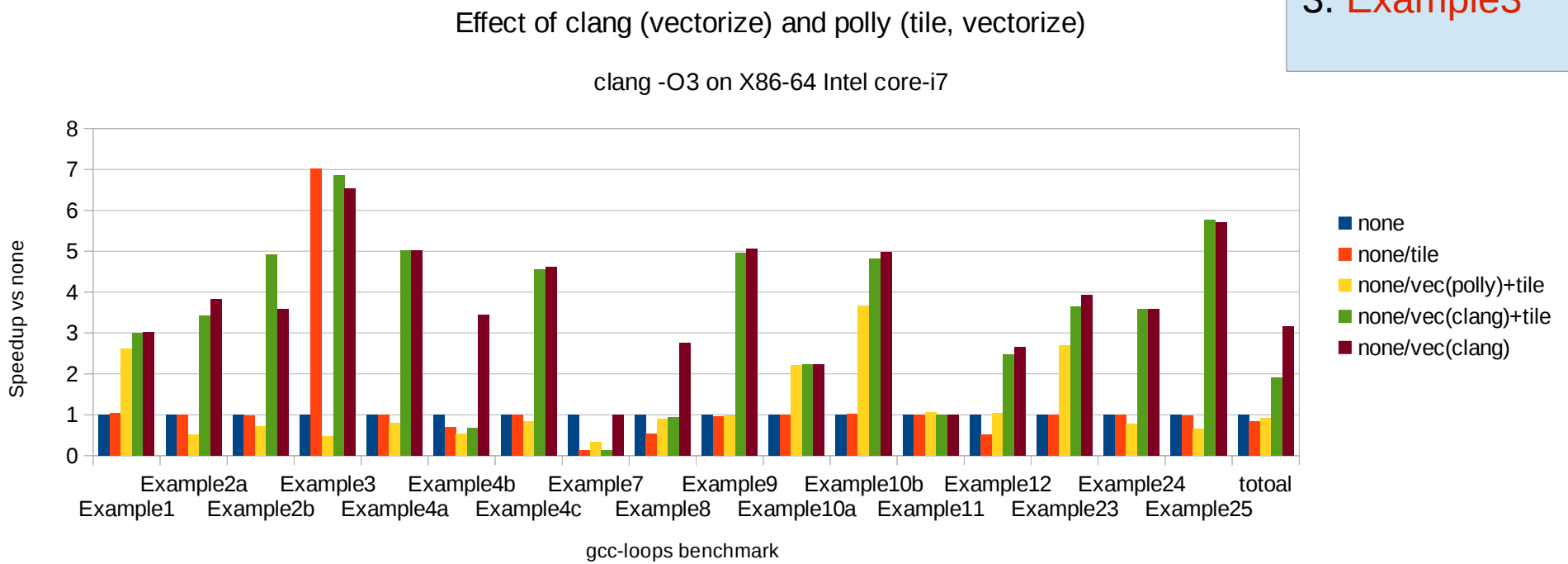
Tobias

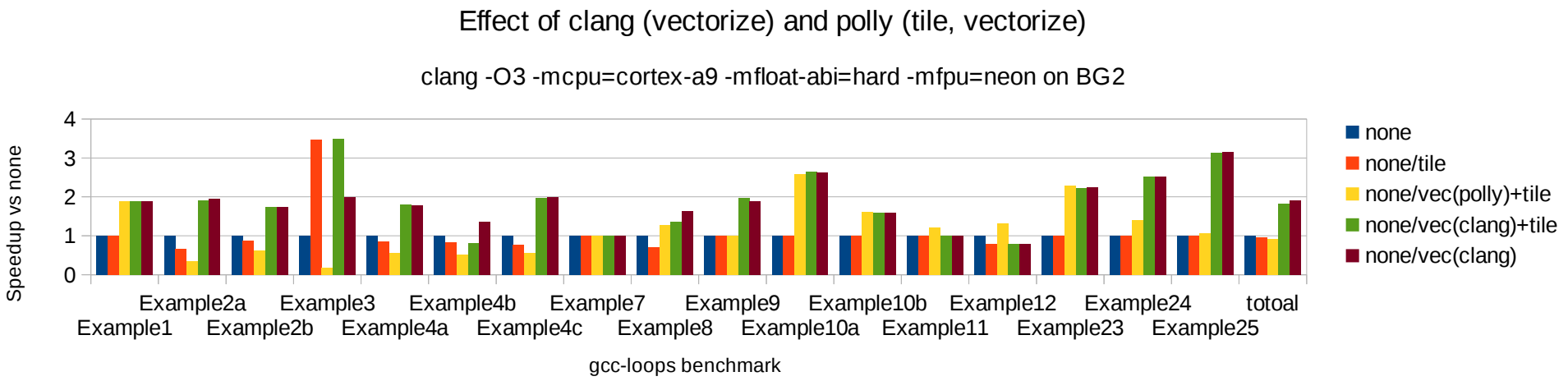
Vectorize

- Clang -O3: has vectorize
- Polly has vectorize but clang is better
- The order of options does not matter:
  - Clang -O3 -mllvm -polly -mllvm -polly-vectorizer=polly
  - Clang -mllvm -polly -mllvm -polly-vectorizer=polly -O3

Gcc-loops benchmark

1. Example 7
2. Example 12
3. Example3





Example7 – X86-64 - Why polly make it worse

```
__attribute__((noinline))
void  example7 (int x) {
   int i;

   /* feature: support for read accesses with an
unknown misalignment  */
   for (i=0; i<N; i++){
      a[i] = b[i+x];
   }
}
```

None or vec(clang)

```
define void @_Z8example7i(i32 %x) #2 {
  %1 = sext i32 %x to i64
  %scevgep = getelementptr [2048 x i32]* @b, i64
0, i64 %1
  %scevgep2 = bitcast i32* %scevgep to i8*
  call void @llvm.memcpy.p0i8.p0i8.i64(i8* bitcast
([2048 x i32]* @a to i8*), i8* %scevgep2, i64 4096,
i32 4, i1 false)
  ret void
}
```

X86-64 ex7 is better while BG2 is tie with vectorize. Why?
Can arm's memcpy be improved?

vec(clang)+tile

```
define void @_Z8example7i(i32 %x) #2 {
.split:
  %0 = zext i32 %x to i64
  br label %polly.loop_header

polly.loop_exit:                                  ; preds =
%polly.loop_header
  ret void

polly.loop_header:                                ; preds =
%polly.loop_header, %.split
  %polly.indvar = phi i64 [ 0, %.split ], [ %polly.indvar_next,
%polly.loop_header ]
  %p_ = add i64 %polly.indvar, %0
  %p_scevgep = getelementptr [2048 x i32]* @a, i64 0, i64
%polly.indvar
  %sext = shl i64 %p_, 32
  %p_14 = ashr exact i64 %sext, 32
  %p_18 = getelementptr inbounds [2048 x i32]* @b, i64 0,
i64 %p_14
  %vector_ptr = bitcast i32* %p_18 to <4 x i32>*
  %_p_vec_full = load <4 x i32>* %vector_ptr, align 8
  %vector_ptr22 = bitcast i32* %p_scevgep to <4 x i32>*
  store <4 x i32> %_p_vec_full, <4 x i32>* %vector_ptr22,
align 16
  %polly.indvar_next = add nsw i64 %polly.indvar, 4
  %polly.loop_cond = icmp slt i64 %polly.indvar, 1020
  br i1 %polly.loop_cond, label %polly.loop_header, label
%polly.loop_exit
}
```

Example12 - X86-64

vec(clang)

```
define void @_Z9example12v() #2 {
vector.ph:
  br label %vector.body

vector.body:                                      ; preds = %vector.body,
%vector.ph
  %index = phi i64 [ 0, %vector.ph ], [ %index.next, %vector.body ]
  %0 = getelementptr inbounds [2048 x i32]* @a, i64 0, i64
%index
  %1 = trunc i64 %index to i32
  %broadcast.splatinsert3 = insertelement <4 x i32> undef, i32
%1, i32 0
  %broadcast.splat4 = shufflevector <4 x i32>
%broadcast.splatinsert3, <4 x i32> undef, <4 x i32> zeroinitializer
  %induction5 = add <4 x i32> %broadcast.splat4, <i32 0, i32 1,
i32 2, i32 3>
  %induction6 = add <4 x i32> %broadcast.splat4, <i32 4, i32 5,
i32 6, i32 7>
  %2 = bitcast i32* %0 to <4 x i32>*
  store <4 x i32> %induction5, <4 x i32>* %2, align 16
  %.sum7 = or i64 %index, 4
  %3 = getelementptr [2048 x i32]* @a, i64 0, i64 %.sum7
  %4 = bitcast i32* %3 to <4 x i32>*
  store <4 x i32> %induction6, <4 x i32>* %4, align 16
  %index.next = add i64 %index, 8
  %5 = icmp eq i64 %index.next, 1024
  br i1 %5, label %middle.block, label %vector.body, !llvm.loop !26

middle.block:                                     ; preds = %vector.body
  ret void
}
```

```
__attribute__((noinline))
void example12() {
  for (int i = 0; i < N; i++) {
    a[i] = i;
  }
}
```

none

```
define void @_Z9example12v() #2 {
  br label %1

; <label>:1                                       ; preds
= %1, %0
  %indvars.iv = phi i64 [ 0, %0 ],
[ %indvars.iv.next, %1 ]
  %2 = getelementptr inbounds [2048 x i32]*
@a, i64 0, i64 %indvars.iv
  %3 = trunc i64 %indvars.iv to i32
  store i32 %3, i32* %2, align 4, !tbaa !1
  %indvars.iv.next = add nuw nsw i64
%indvars.iv, 1
  %exitcond = icmp eq i64 %indvars.iv.next,
1024
  br i1 %exitcond, label %4, label %1

; <label>:4                                       ; preds
= %1
  ret void
}
```

Example12 - arm

```
__attribute__((noinline))
void example12() {
  for (int i = 0; i < N; i++) {
    a[i] = i;
  }
}
```

vec(clang)

```
00008ef0 <_Z9example12v>:
    8ef0:   e30c1080   movw    r1, #49280 ; 0xc080
    8ef4:   e3a00000   mov   r0, #0
    8ef8:   e3401001   movt  r1, #1
    8efc:   e28f202c   add   r2, pc, #44 ; 0x2c
    8f00:   eea20b90   vdup.32    q9, r0
    8f04:   f4620aef   vld1.64    {d16-d17}, [r2 :128]
    8f08:   e2800004   add   r0, r0, #4
    8f0c:   f26208e0   vadd.i32   q8, q9, q8
    8f10:   e3500b01   cmp   r0, #1024   ; 0x400
    8f14:   f4410aef   vst1.64    {d16-d17}, [r1 :128]
    8f18:   e2811010   add   r1, r1, #16
    8f1c:   1afffff6   bne   8efc <_Z9example12v+0xc>
    8f20:   e12fff1e   bx    lr
    8f24:   e320f000   nop   {0}
    8f28:   e320f000   nop   {0}
    8f2c:   e320f000   nop   {0}
    8f30:   00000000   .word 0x00000000
    8f34:   00000001   .word 0x00000001
    8f38:   00000002   .word 0x00000002
    8f3c:   00000003   .word 0x00000003
```

none

```
00008c14 <_Z9example12v>:
    8c14:   e30c1080   movw    r1, #49280 ; 0xc080
    8c18:   e3a00000   mov   r0, #0
    8c1c:   e3401001   movt  r1, #1
    8c20:   e7810100   str   r0, [r1, r0, lsl #2]
    8c24:   e2800001   add   r0, r0, #1
    8c28:   e3500b01   cmp   r0, #1024   ; 0x400
    8c2c:   1afffffb   bne   8c20 <_Z9example12v+0xc>
    8c30:   e12fff1e   bx    lr
```

Example3 - X86-64

```
typedef int aint __attribute__ ((__aligned__(16)));
__attribute__((noinline))
void example3 (int n, aint * __restrict__ p, aint * __restrict q)
{

  /* feature: support for (aligned) pointer accesses.  */
  while (n--){
    *p++ = *q++;
  }
}
```

none

```
define void @_Z8example3iPiS_(i32 %n, i32* noalias nocapture
%p, i32* noalias nocapture readonly %q) #2 {
  %1 = icmp eq i32 %n, 0
  br i1 %1, label %._crit_edge, label %.lr.ph

.lr.ph:                                           ; preds = %0, %.lr.ph
  %.05 = phi i32* [ %3, %.lr.ph ], [ %q, %0 ]
  %.014 = phi i32* [ %5, %.lr.ph ], [ %p, %0 ]
  %.023 = phi i32 [ %2, %.lr.ph ], [ %n, %0 ]
  %2 = add nsw i32 %.023, -1
  %3 = getelementptr inbounds i32* %.05, i64 1
  %4 = load i32* %.05, align 16, !tbaa !1
  %5 = getelementptr inbounds i32* %.014, i64 1
  store i32 %4, i32* %.014, align 16, !tbaa !1
  %6 = icmp eq i32 %2, 0
  br i1 %6, label %._crit_edge, label %.lr.ph

._crit_edge:                                      ; preds = %.lr.ph, %0
  ret void
}
```

vec(clang)

```
define void @_Z8example3iPiS_(i32 %n, i32* noalias nocapture %p,
i32* noalias nocapture readonly %q) #2 {
...
vector.body:                                      ; preds = %.lr.ph.preheader,
%vector.body
  %index = phi i64 [ %index.next, %vector.body ], [ 0,
%.lr.ph.preheader ]
  %next.gep = getelementptr i32* %q, i64 %index
  %next.gep33 = getelementptr i32* %p, i64 %index
 ...
```

tile or vec(clang)+tile

```
define void @_Z8example3iPiS_(i32 %n, i32*
noalias nocapture %p, i32* noalias nocapture
readonly %q) #2 {
...
polly.loop_header13.preheader:                    ; preds
= %polly.cond9
  %6 = shl nuw nsw i64 %0, 2
  call void @llvm.memcpy.p0i8.p0i8.i64(i8* %p29,
i8* %q30, i64 %6, i32 4, i1 false)
  br label %polly.merge
}
```

Vector – my setting need to check!!!

- Use llvm arm polly build they are same as follows,

```
cschen@cschen-BM6835-BM6635-
BP6335:~/test/lbd/docs/BackendTutorial/note/gcc-loops$
~/test/polly/llvm_arm_build/bin/clang++ -Xclang -load -Xclang
~/test/polly/llvm_arm_build/lib/LLVMPolly.so -O3 -fno-vectorize -S -emit-llvm
gcc-loops.cpp -o gcc-loops-none.arm.ll
cschen@cschen-BM6835-BM6635-
BP6335:~/test/lbd/docs/BackendTutorial/note/gcc-loops$
~/test/polly/llvm_arm_build/bin/clang++ -Xclang -load -Xclang
~/test/polly/llvm_arm_build/lib/LLVMPolly.so -O3 -S -emit-llvm gcc-loops.cpp -o
gcc-loops-clangvec.arm.ll
cschen@cschen-BM6835-BM6635-
BP6335:~/test/lbd/docs/BackendTutorial/note/gcc-loops$ diff gcc-loops-
clangvec.arm.ll gcc-loops-none.arm.ll
```

Multi-core

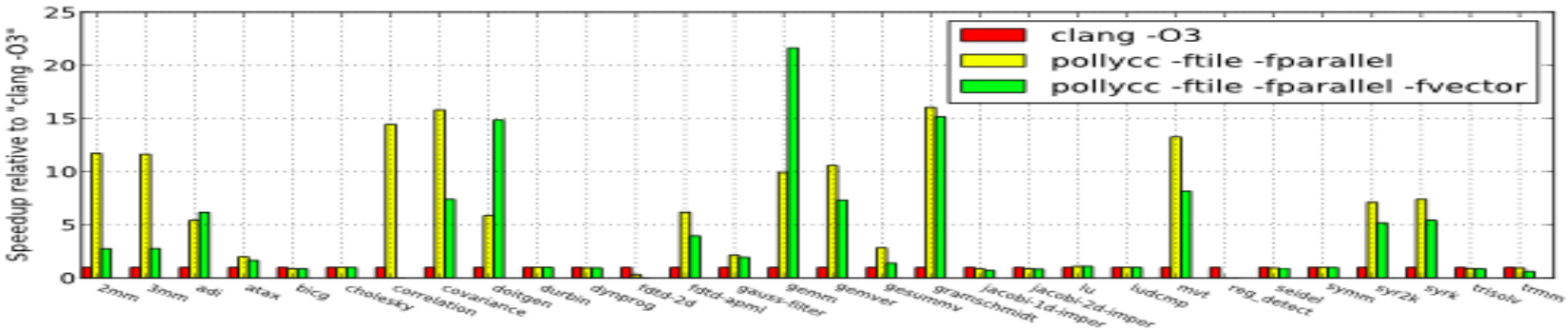
- Multi-thread for multi-core → OpenMP

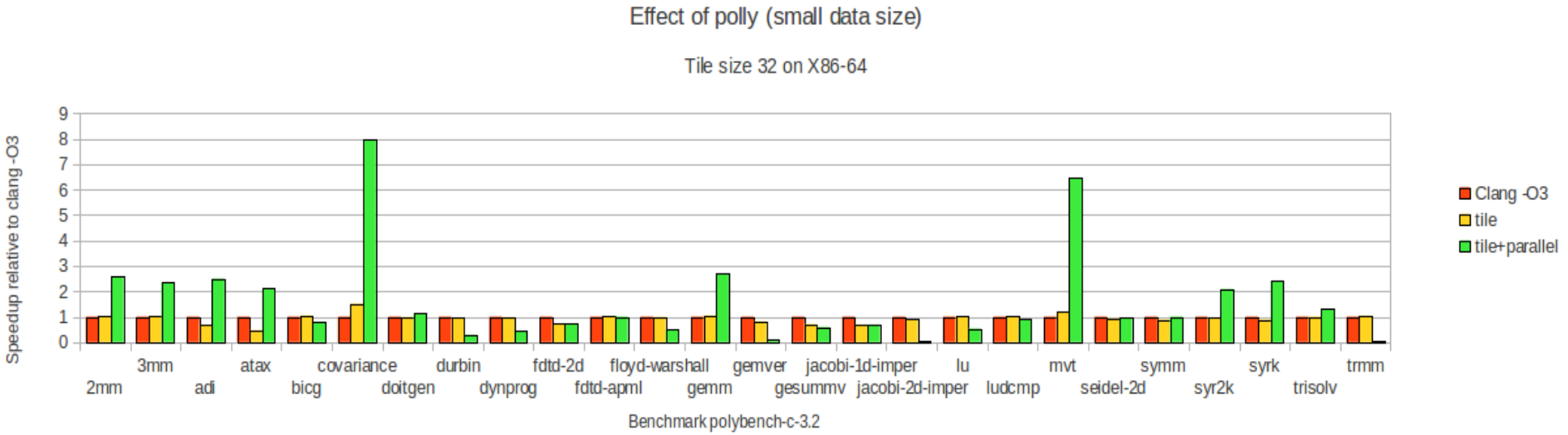
  Hi Tobias,

  How about the multi-thread take multi-core advantage through OpenMP? Is it clang inside now or polly has this feature.

  That one is polly specific.

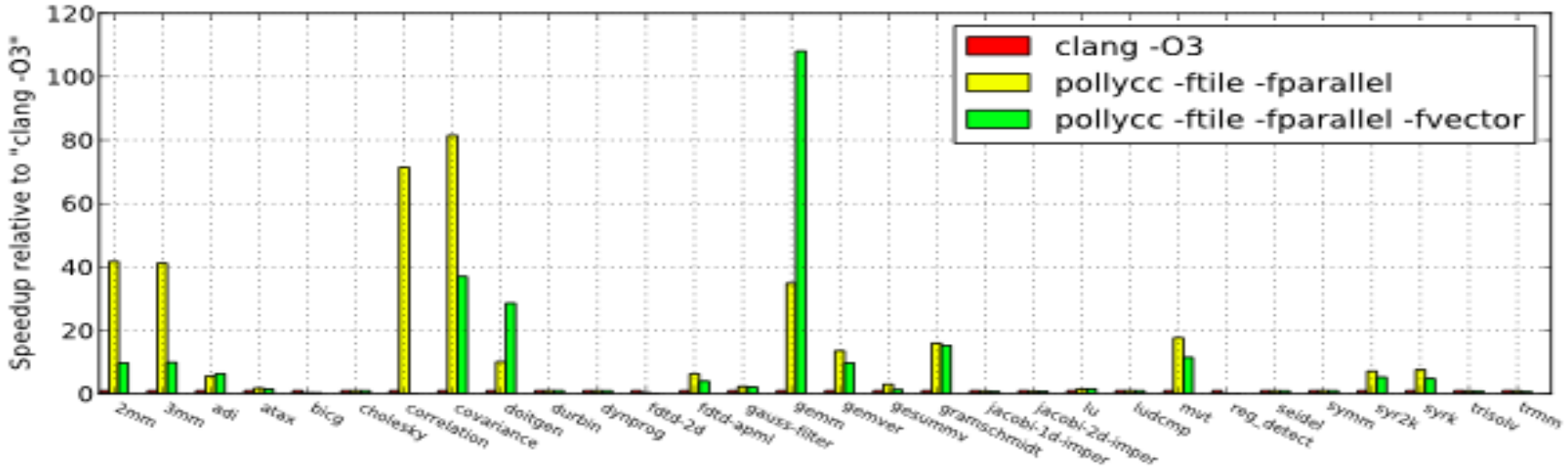
Multi-core - Small data size



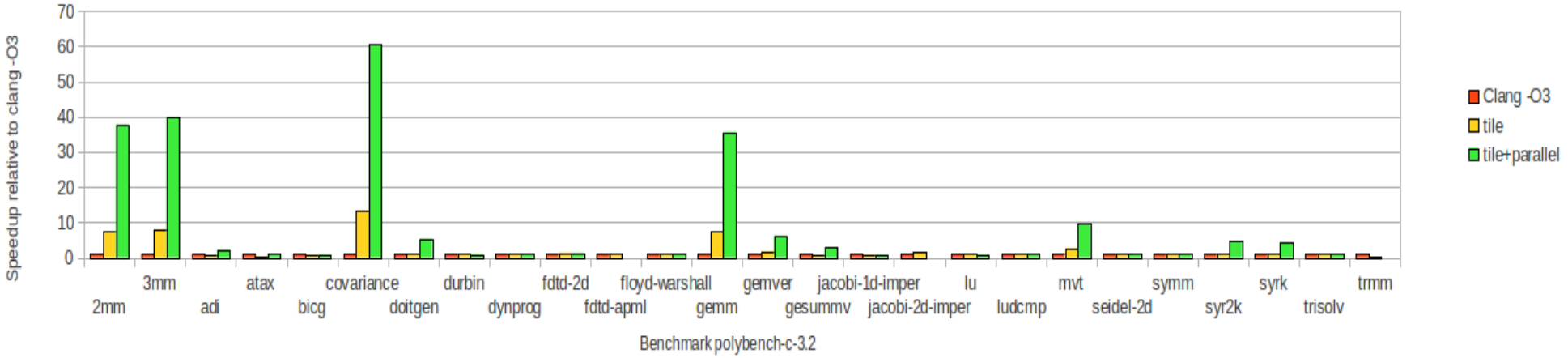


Segmentation fault for:
fdtd-apml,
jacobi-2d-imper,
trmm

Multi-core - Large data size



Tile size 32 on X86-64



Multi-core - Large data size

2mm.c

```
define i32 @main(i32 %argc, i8** nocapture readonly %argv) #0 {
.split:
  %omp.userContext.i = alloca { [2000 x double]*, double }, align 8
  %omp.userContext35.i = alloca { [2000 x double]* }, align 8
  %omp.userContext37.i = alloca { [2000 x double]*, double, [2000 x double]*, [2000 x double]* }, align 8
  %omp.userContext39.i = alloca { [2000 x double]*, [2000 x double]*, [2000 x double]* }, align 8
  ...
  call fastcc void @init_array(double* %alpha, double* %beta, [2000 x double]* %5, [2000 x double]* %6,
[2000 x double]* %7, [2000 x double]* %8)
  ...
  %16 = getelementptr inbounds { [2000 x double]*, double }* %omp.userContext.i, i64 0, i32 0
  store [2000 x double]* %8, [2000 x double]** %16, align 8
  %17 = getelementptr inbounds { [2000 x double]*, double }* %omp.userContext.i, i64 0, i32 1
  store double %10, double* %17, align 8
  call void @GOMP_parallel_loop_runtime_start(void (i8*)* @kernel_2mm.omp_subfn, i8* %12, i32 0, i64 0,
i64 2000, i64 32) #3
  call void @kernel_2mm.omp_subfn(i8* %12) #3
  call void @GOMP_parallel_end() #3
  %18 = getelementptr inbounds { [2000 x double]* }* %omp.userContext35.i, i64 0, i32 0
  store [2000 x double]* %11, [2000 x double]** %18, align 8
  call void @GOMP_parallel_loop_runtime_start(void (i8*)* @kernel_2mm.omp_subfn6, i8* %13, i32 0, i64
0, i64 2000, i64 32) #3
  call void @kernel_2mm.omp_subfn6(i8* %13) #3
  call void @GOMP_parallel_end() #3
  …
}
```

Multi-core - Large data size

- The segmentation fault can be stopped by command ulimit as below

```
cschen@cschen-BM6835-BM6635-BP6335:~/test/polybench-c-3.2/polybench$
ulimit -s
8192
cschen@cschen-BM6835-BM6635-BP6335:~/test/polybench-c-3.2/polybench$
ulimit -s unlimited
cschen@cschen-BM6835-BM6635-BP6335:~/test/polybench-c-3.2/polybench$
ulimit -s
unlimited
cschen@cschen-BM6835-BM6635-BP6335:~/test/polybench-c-3.2/polybench$
./jacobi-2d-imper.tile.parallel
4.319366
cschen@cschen-BM6835-BM6635-BP6335:~/test/polybench-c-3.2/polybench$
./trmm.tile.parallel
102.427773
```

Reboot if not work.

Multi-core - Large data size

Hi cschen,

I can reproduce these failures. It seems when we try to parallelize the initialization routine there is a problem. If you limit Polly to the kernel with -mllvm -polly-only-func=kernel_fdtd_apml, everything works nicely.

I currently did not investigate further. Would you be interested to do so? Johannes also recently worked with OpenMP, so he might have an idea what is going on as well.

I will post 2 codegen bugs/fixes later (related to OpenMP) and I will look into these segfaults, but this may take till next week.

Best regards,

Johannes

Multi-core - note

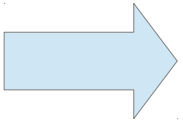
- 2mm.c has no #pragma omp
  - But polly translate 2mm.c into @GOMP_parallel_loop_runtime_start in IR
- Still has limits
  - Segmentation fault in fdtd-apml, jacobi-1d-imper, trmm
  - Polly team member Johannes is fixing it
- Fail to open polly parallel feature for ARM in build? Fixed, by put on /usr/libgomp.so.1 on BG2
  - Currently, I can run it on X86-64 only and has segmentation fault limitation
  - Put in /usr/lib/libgomp.so.1 then work in BG2
  - Segmentation fault: gesummv, durbin: 20140618 check out
  - No result: fdtd-apml

Other optimization opportunity - fusion

```
for(i=0; i<N; i++)  {
    for(j=0; j<N; j++)  {
        C[i][j] = 0;
        for(k=0; k<N; k++)
            C[i][j] = C[i][j] + A[i][k] * B[k][j];
        D[i][j] = 0;
        for(k=0; k<N; k++)
            D[i][j] = D[i][j] + A[i][k] * 2.0;
    }
}
```

```
for(i=0; i<N; i++)  {
    for(j=0; j<N; j++)  {
        C[i][j] = 0;
        for(k=0; k<N; k++)
            C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
}

for(i=0; i<N; i++)  {
    for(j=0; j<N; j++)  {
        D[i][j] = 0;
        for(k=0; k<N; k++)
            D[i][j] = D[i][j] + A[i][k] * 2.0;
    }
}
```



```
for(i=0; i<N; i++)  {
    for(j=0; j<N; j++)  {
        C[i][j] = 0;
        D[i][j] = 0;
        for(k=0; k<N; k++) {
            C[i][j] = C[i][j] + A[i][k] * B[k][j];
            D[i][j] = D[i][j] + A[i][k] * 2.0;
        }
    }
}
```

Future work

- Can memcpy be improved on arm?
- Gcc tile size tune on marvell platform for matmul.c & some other program
  - Matmul.c: best T=32 (next slide)
  - Much time/effort for many program depend on cache+loop bound if allow different T for different program
- Polytope model & llvm polly enable
- Llvm polly
  - Work from N to→ n
  - Segmentation fault in polly parallel

Gcc tile size for matmul.c on BG2

- ~/marvell/work/assigned-job/run_released_benchmark/armv7-marvell-linux-gnueabi-hard-4.6.4_x86_64_rc2_20140325/bin/arm-marvell-linux-gnueabi-gcc -O3 -mcpu=cortex-a9 -mfloat-abi=hard -mfpu=neon -floop-strip-mine --param loop-block-tile-size=51 matmul.c -o matmul.51

time ./matmul.tiledefault (51)
real 0m 17.97s
user0m 17.97s
sys 0m 0.00s

time ./matmul.24
real 0m 18.31s
user0m 18.29s
sys 0m 0.01s

time ./matmul.12
real 0m 18.15s
user0m 18.14s
sys 0m 0.01s

time ./matmul.32
real 0m 11.54s
user0m 11.53s
sys 0m 0.01s

time ./matmul.16
real 0m 14.83s
user0m 14.81s
sys 0m 0.01s

time ./matmul.48
real 0m 18.03s
user0m 18.00s
sys 0m 0.03s

Run Marvell benchmark

- Run with tile option only (with omp)
  - No improvement and 1 benchmark cannot build.
  - Results in dir polly-benchmark/

Reference

- Advanced compiler design implementation
  - 20.4.2 Loop Transformations
  - 20.4.3 Locality and Tiling
- Book, chapter 11 of compilers principles, techniques, & tools
- http://en.wikipedia.org/wiki/Loop_tiling
- http://en.wikipedia.org/wiki/Polytope_model
- http://pluto-compiler.sourceforge.net/
  - http://drona.csa.iisc.ernet.in/~uday/publications/uday-cc08.pdf
- http://polly.llvm.org