语义信息在领域编译中的角色

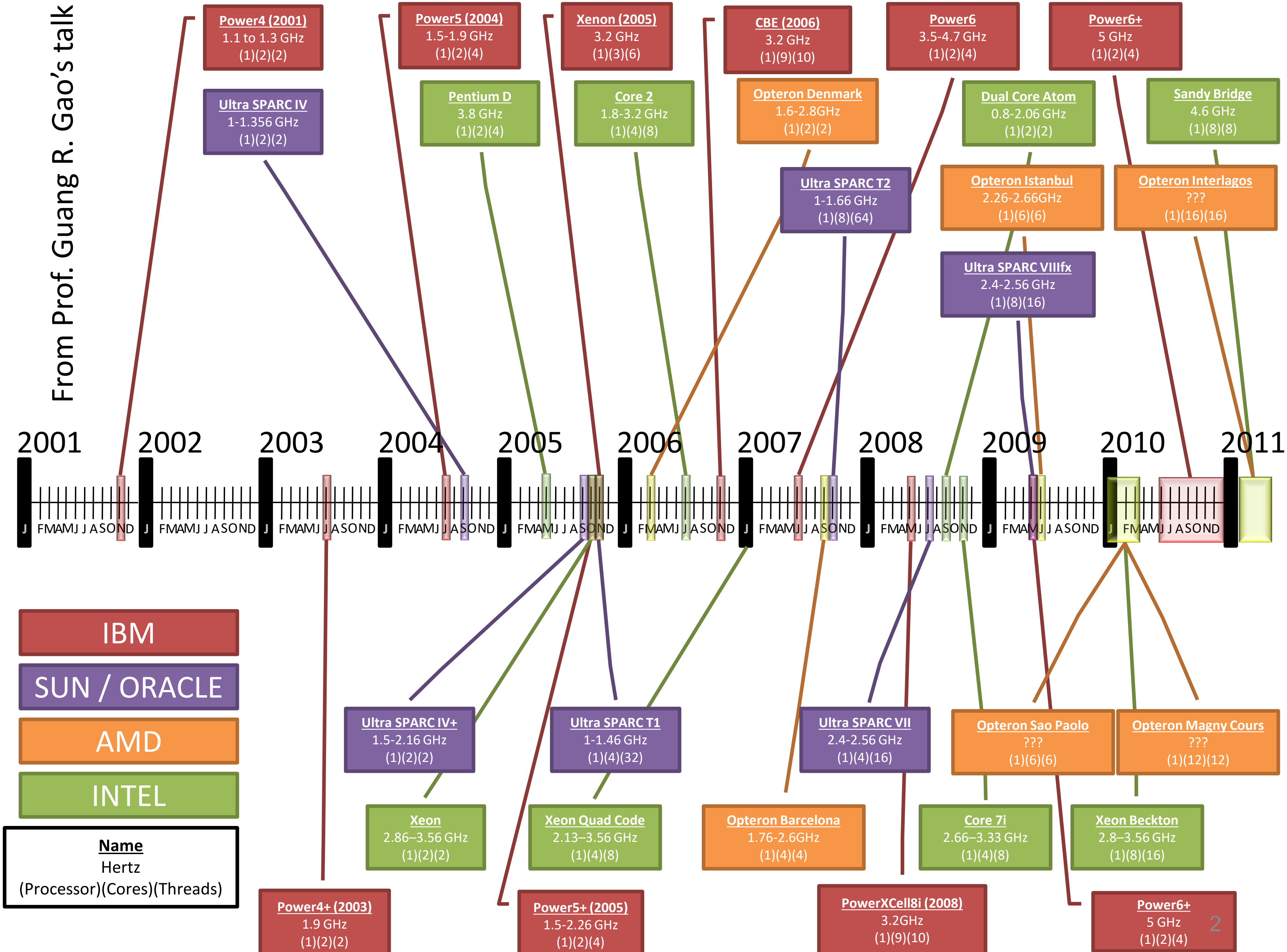
Using Semantic To Guide Compiler Optimizations

中科院计算所

崔慧敏

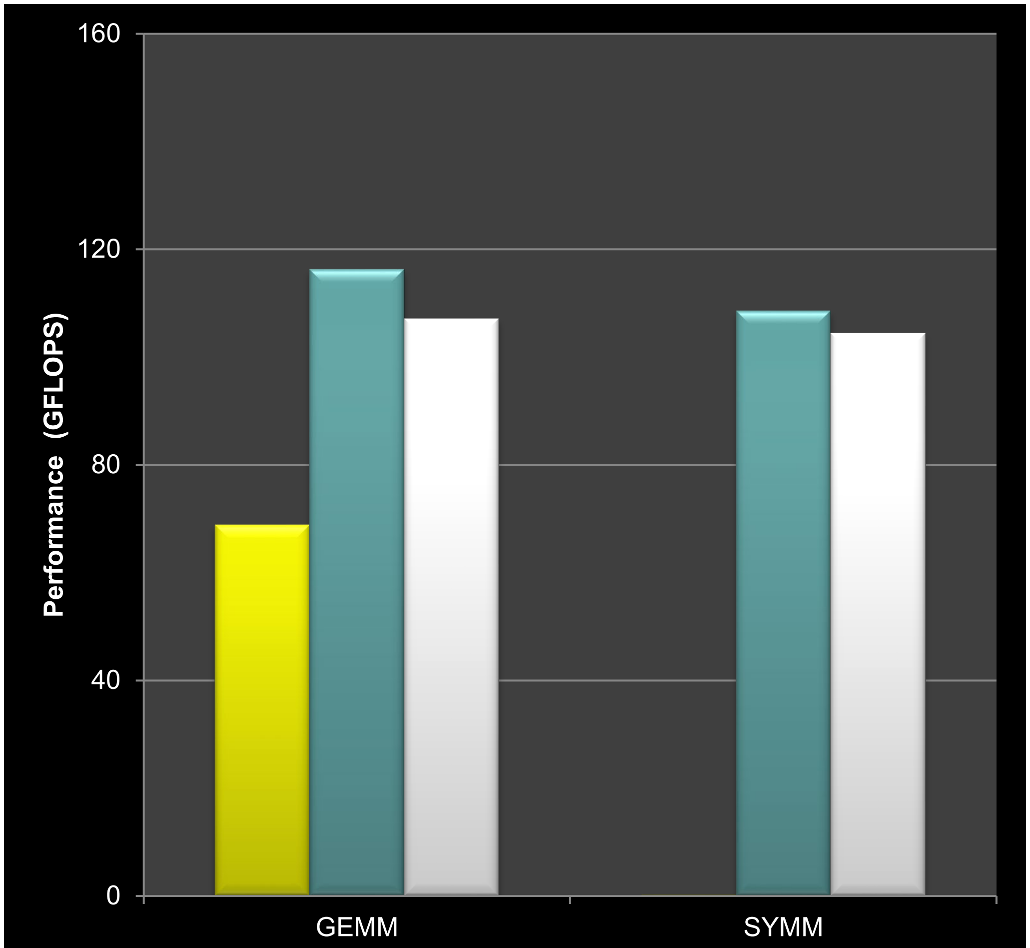
2020年10月

中国科学院计算技术研究所
INSTITUTE OF COMPUTING TECHNOLOGY, CHINESE ACADEMY OF SCIENCES



From Prof. Guang R. Gao's talk
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
J FMAMJ J ASOND
Power4 (2001)
1.1 to 1.3 GHz
(1)(2)(2)
Ultra SPARC IV
1-1.356 GHz
(1)(2)(2)
Power5 (2004)
1.5-1.9 GHz
(1)(2)(4)
Pentium D
3.8 GHz
(1)(2)(4)
Xenon (2005)
3.2 GHz
(1)(3)(6)
Core 2
1.8-3.2 GHz
(1)(4)(8)
CBE (2006)
3.2 GHz
(1)(9)(10)
Opteron Denmark
1.6-2.8GHz
(1)(2)(2)
Ultra SPARC T2
1-1.66 GHz
(1)(8)(64)
Power6
3.5-4.7 GHz
(1)(2)(4)
Dual Core Atom
0.8-2.06 GHz
(1)(2)(2)
Opteron Istanbul
2.26-2.66GHz
(1)(6)(6)
Ultra SPARC VIIIfx
2.4-2.56 GHz
(1)(8)(16)
Power6+
5 GHz
(1)(2)(4)
Sandy Bridge
4.6 GHz
(1)(8)(8)
Opteron Interlagos
???
(1)(16)(16)
IBM
SUN / ORACLE
AMD
INTEL
Name
Hertz
(Processor)(Cores)(Threads)
Ultra SPARC IV+
1.5-2.16 GHz
(1)(2)(2)
Xeon
2.86–3.56 GHz
(1)(2)(2)
Power4+ (2003)
1.9 GHz
(1)(2)(2)
Ultra SPARC T1
1-1.46 GHz
(1)(4)(32)
Xeon Quad Code
2.13–3.56 GHz
(1)(4)(8)
Power5+ (2005)
1.5-2.26 GHz
(1)(2)(4)
Opteron Barcelona
1.76-2.6GHz
(1)(4)(4)
Ultra SPARC VII
2.4-2.56 GHz
(1)(4)(16)
PowerXCell8i (2008)
3.2GHz
(1)(9)(10)
Opteron Sao Paolo
???
(1)(6)(6)
Core 7i
2.66–3.33 GHz
(1)(4)(8)
Opteron Magny Cours
???
(1)(12)(12)
Xeon Beckton
2.8–3.56 GHz
(1)(8)(16)
Power6+
5 GHz
(1)(2)(4)

Semantic Case 1: SYMM



SYMM Source Code:

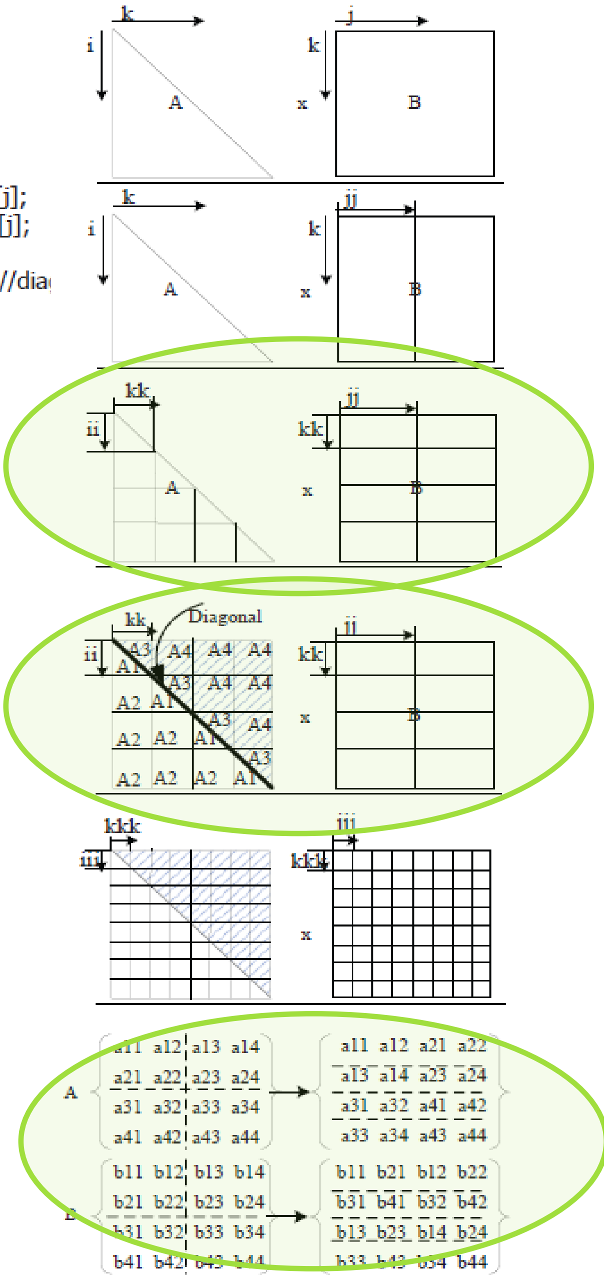
```
Li: for (i = 0; i < M; i++)
   Lj: for (j = 0; j < N; j++)
   {
     Lk: for (k = 0; k < i; k++)
         {
            C[i][j] += A[i][k] * B[k][j];
            C[k][j] += A[i][k] * B[i][j];
         }
     Ld: C[i][j] += A[i][i] * B[i][j]; //diagonal
   }
```

SYMM Source Code:
```
Li: for (i = 0; i < M; i++)
  Lj: for (j = 0; j < N; j++)
  {
    Lk: for (k = 0; k < i; k++)
       {
          C[i][j] += A[i][k] * B[k][j];
          C[k][j] += A[i][k] * B[i][j];
       }
    Ld: C[i][j] += A[i][i] * B[i][j]; //dia
  }
```

Triangular area

Symmetry

Matrix-mul



*第一步，线程划分：*对循环迭代j进行strip-mining，并对循环次序进行调整，原始的循环顺序变为 **j, i, jj, k**。并将最外层循环j分配到不同的线程。

*第二步，循环分块：*对循环迭代i和k进行循环分块（tiling），改善L2缓存的局部性。这时新的循环顺序为**j, i, k, ii, jj, kk**.

*第三步，循环分裂：*将循环体分裂为三个，分别对应real area (A1,…, A2,…), shadow area (A3,…, A4, …), 和对角线元素.

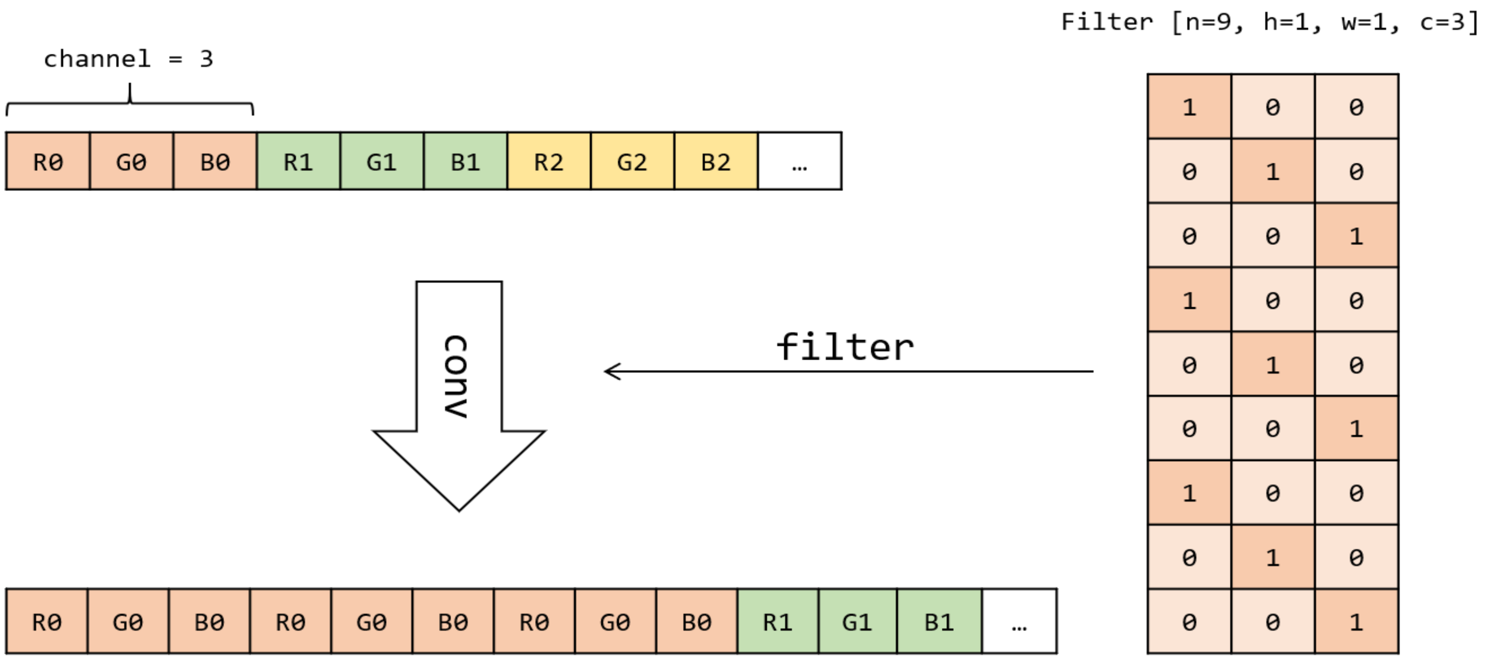
*第四步，循环剥离（peeling）：*将三角形区域从矩形区域中剥离出来，即将左图中的A1和A3分别从A2和A4中剥离。

*第五步：循环分块：*对剥离出的三角形区域A1和A3分别实施循环分块。对每个矩形区域（其循环顺序为ii, jj, kk）实施循环分块，这时新的循环顺序为**j, i, k, ii, jj, kk, iii, jjj, kkk**

*第六步：数据布局重组：*将源矩阵A和B进行数据布局重组，从行优先或者列优先转换为块优先，块内和块间的顺序则遵从原有的行/列优先顺序。如左图所示，这一变换是为了提高L1数据缓存的局部性。

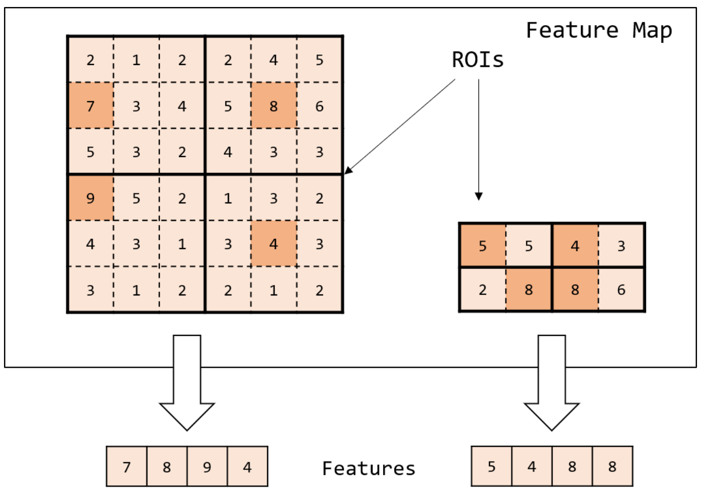
Semantic Case2 : resize

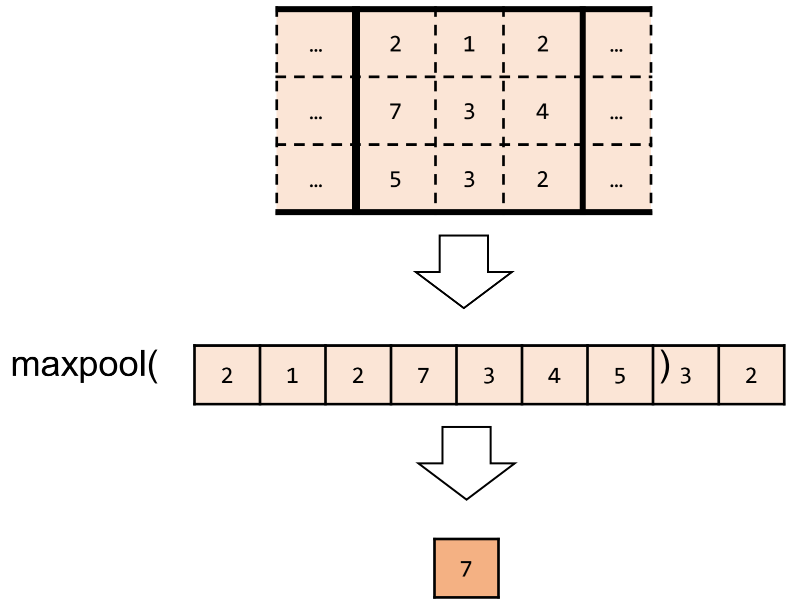
- **We can use conv. to compute resize**



Semantic Case3: ROIPooling

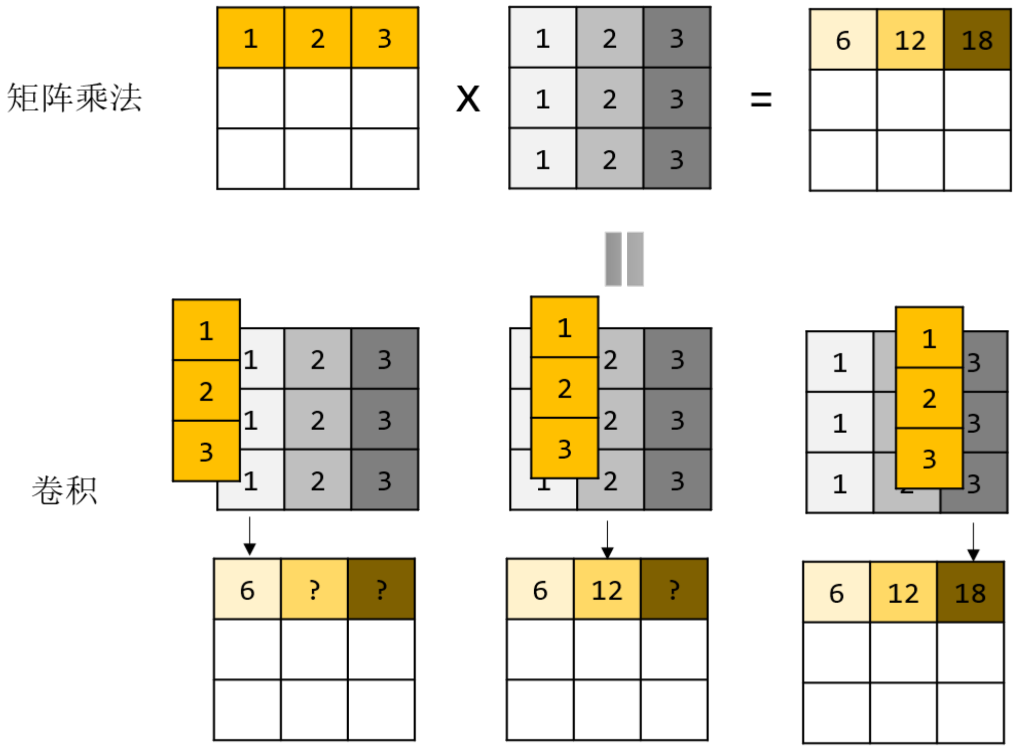
We can use Maxpooling to compute ROI pooling.



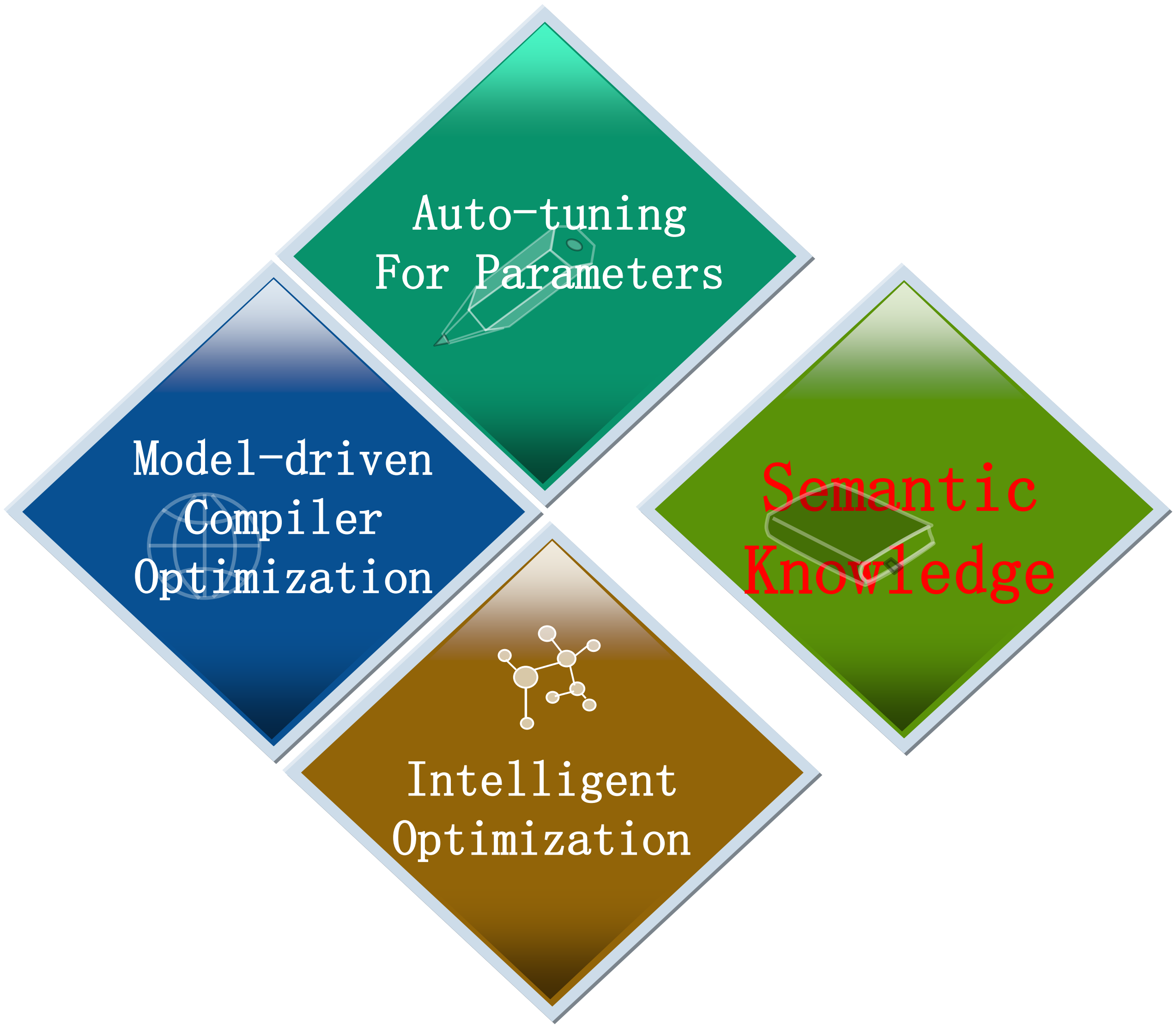


Semantic Case4: GEMM

We can use Conv to compute GEMM.



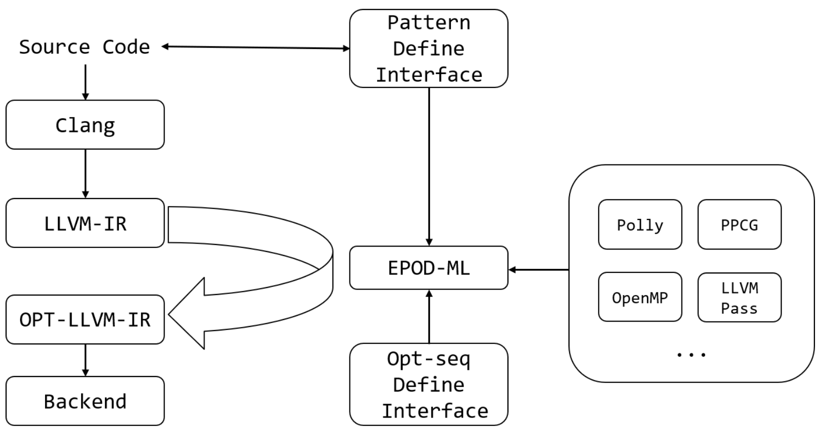
How to get Peak Performance?

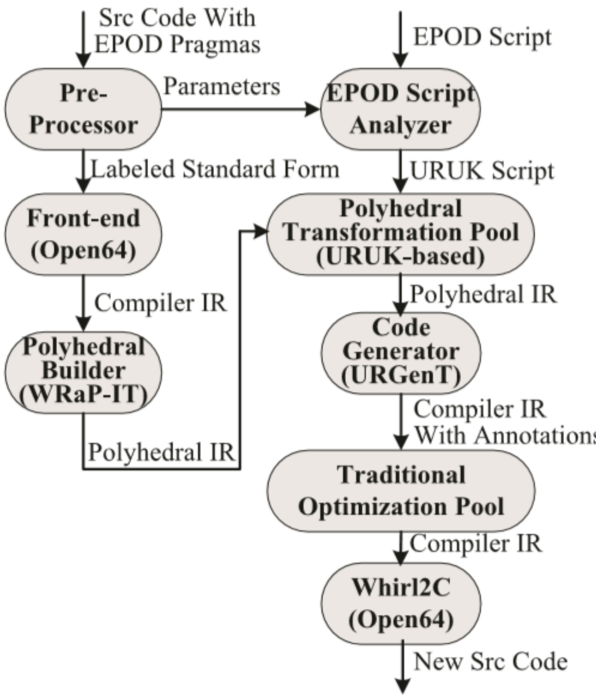


Key Challenges

- How to connect into the workflow of compilers?
- How to specify an optimization sequence?
- How to apply specific opt to specific code region?

Extendable Pattern for Compilers





UDOL – User-defined Optimization Language

```
stmts {
 stmt_1 {
   domain { i: (0, 63), j:(0, 63)},
   position {i, j, 0, 0},
   mem_acc {
     store_0 { mem {C}, domain {i:(0, 63), j:(0, 63)}}
   },
   dep { waw { stmt_1, stmt_2 } },
   expr { st_0(const(0)) }
 },
 stmt_2 {
   domain { i: (0, 63), j:(0, 63), k(0, 63)},
   positon {i, j, 1, k},
   mem_acc {
     load_0  { mem {C}, domain {i:(0, 63), j:(0, 63)}},
     load_1  { mem {A}, domain {i:(0, 63), k:(0, 63)}},
     load_2  { mem {B}, domain {k:(0, 63), j:(0, 63)}},
     store_0 { mem {C}, domain {i:(0, 63), j:(0, 63)}}
   },
   dep { waw { stmt_1, stmt_2 } },
   expr { st_0(add(ld(0), mul(ld(1), ld(2))))}
 }
}
```

1. Define a semantic pattern

- Similar with Polyhedral SCoP
- Each statement includes:
  - Loop domain
  - Position
  - Memory Reference
  - Dependency
  - Expression

UDOL – User-defined Optimization Language

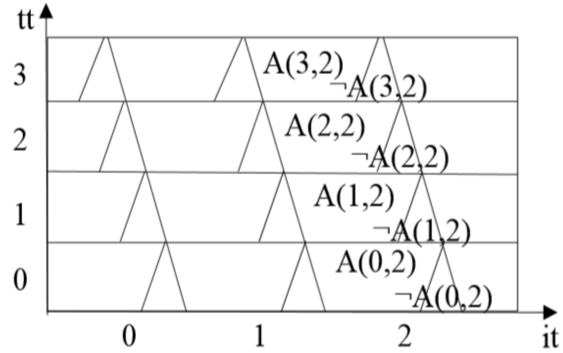
```
def rep_conv(loop) {
  s = loop.get_stmt();
  d = s.get_domain(0);
  col = d.range();
  row = d.next().range();
  dst = s.get_store();
  src = s.get_load(d);
  filter = s.get_load();
  s.replace_with("llvm_conv(dst, src, filter,
              col, 1, row, 1, 1, 1, 1, 1);");
}

def matmul {
  I = 16, J = 16
  ll = polly.loop_tiling($$, [0, 1], [I, J])
  l 0, l_1 = polly.loop_fission(ll, 3);
  llvm.auto_vec(l_0);
  llvm.rep_conv(l_1);
}
```

2. Define an optimization seq.

- Polly for loop transformations.
  - Easy for extension
  - e.g. split tiling/ overlap tiling/ diamond tiling
- Backend opt. on LLVM/Open64.
  - Vectorization
  - On-chip memory usage

UDOL – User-defined Optimization Language

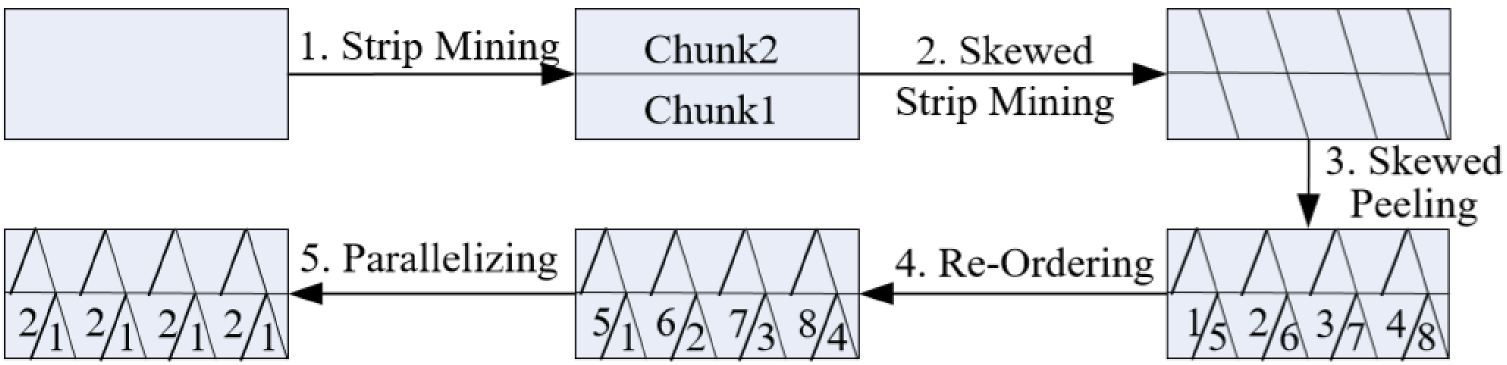
```
Original Loop:
   for (t=1; t < T; t++) {
      for (i = 1; i < N; i++)
         B[i] = (A[i-1]+A[i]+A[i+1])/3;
      for (i = 1; i < N; i++)
         A[i] = B[i];
}

Single Statement Form:
for (t=1; t < T; t++){
   for (i = 1; i < N; i++)
      A[t, i] = (A[t-1,i-1]+A[t-1,i]+A[t-1,i+1])/3;
}
```



(a) Tile Shape

```
for (tt=0; tt<t_tiles; tt++) {
  forall(it=0; it<i_tiles, it++)
          compute triangle ¬A(tt,it);
  barrier;
  forall (it=0; it<i_tiles, it++)
          compute trapezoid A(tt,it);
  barrier;
}
```

(b) Execution Order

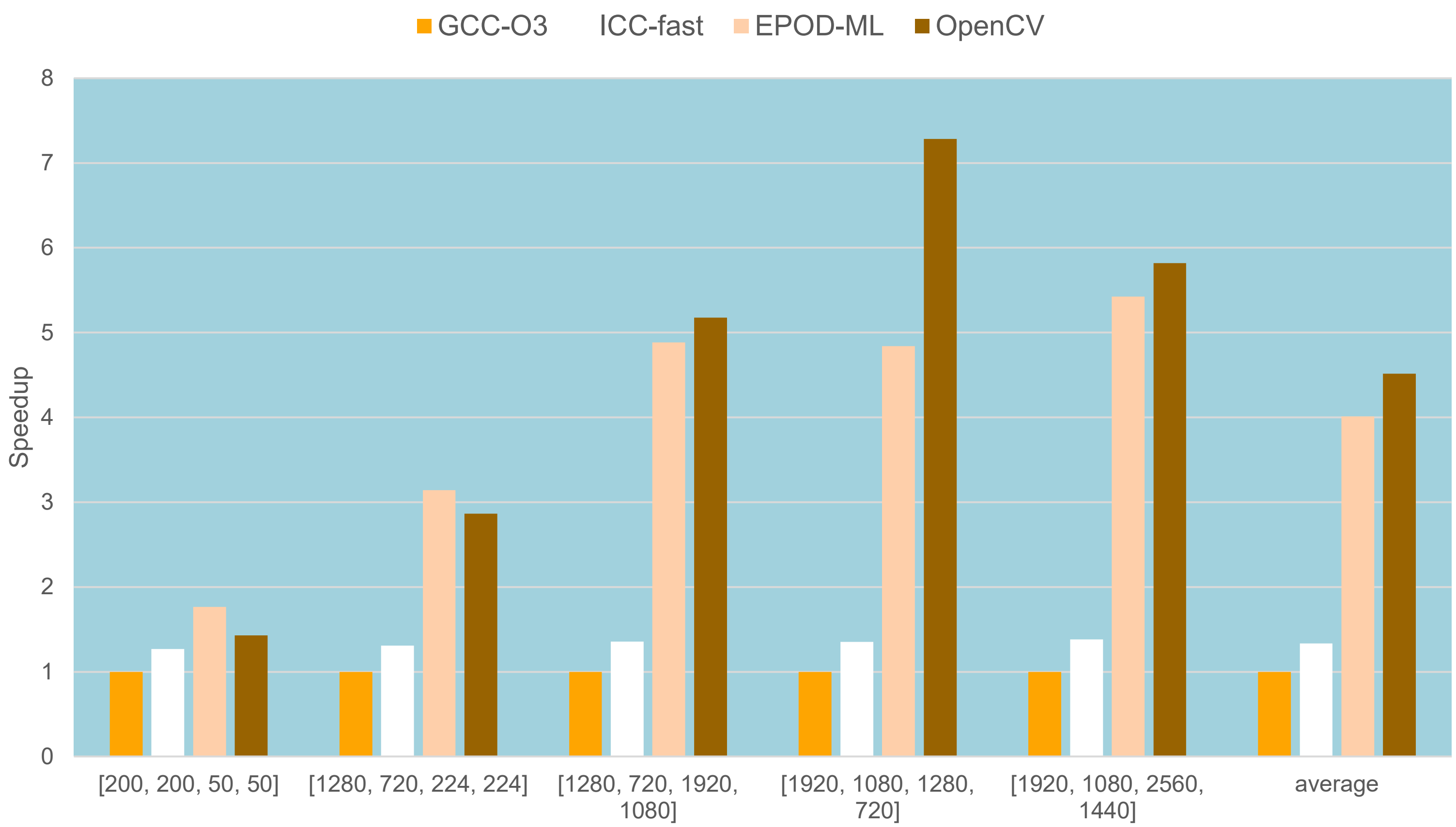


UDOL – User-defined Optimization Language

3. Specify a semantic region

```
#pragma EPOD-ML jacobi
for (t=0; t<T; t++) {
  for (i=0; i<N; i++) {
    for (j=0;j<N;j++) {
      if (i!=j) {
        sum += A[i][j]*xl[i];
      }
    }
    x[i] = (b[i]-sum)/A[i][i];
  }
  for (int i=0; i<N; i++) {
    xl[i] = x[i];
  }
}
#pragma EPOD-ML end
```

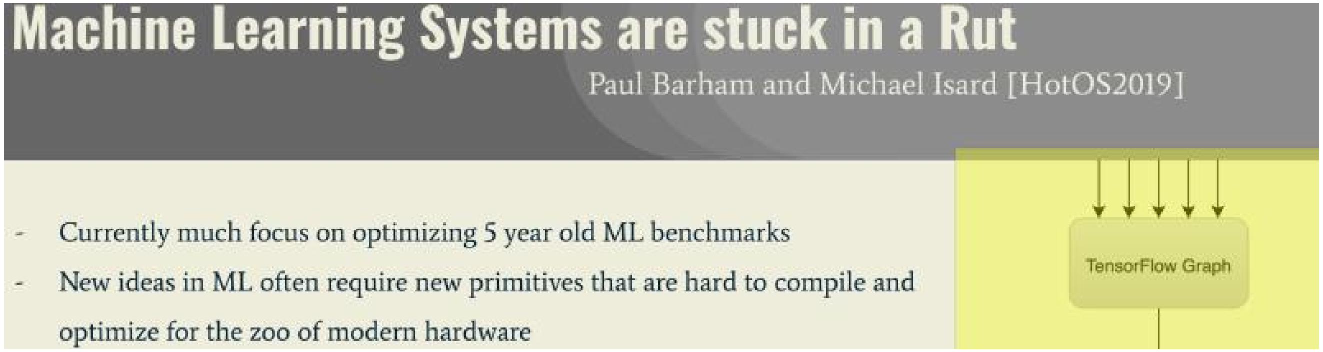
Evaluation



Resize加速比-CPU

人工智能编程语言与编译器

- 问题1：人工智能需要通用的编程语言吗？



- 问题2：芯片敏捷化 vs. 编译器敏捷化

- 问题3：编译器敏捷化 vs. 峰值性能

Thanks

cuihm@ict.ac.cn