POET: Parameterized Optimizations for Empirical Tuning

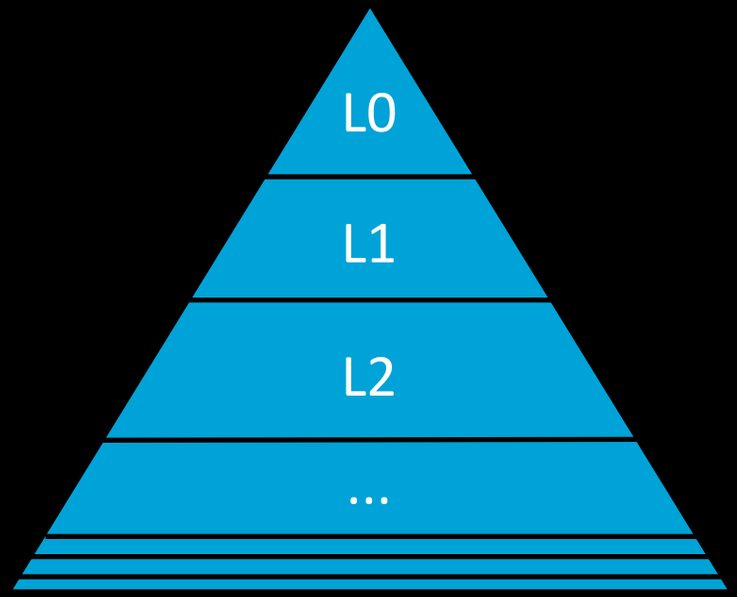
Qing Yi* Keith Seymour† Haihang You† Richard Vuduc‡ Dan Quinlan‡

* University of Texas at San Antonio, TX, 78249, USA

† University of Tennessee at Knoxville, TN, 37996, USA

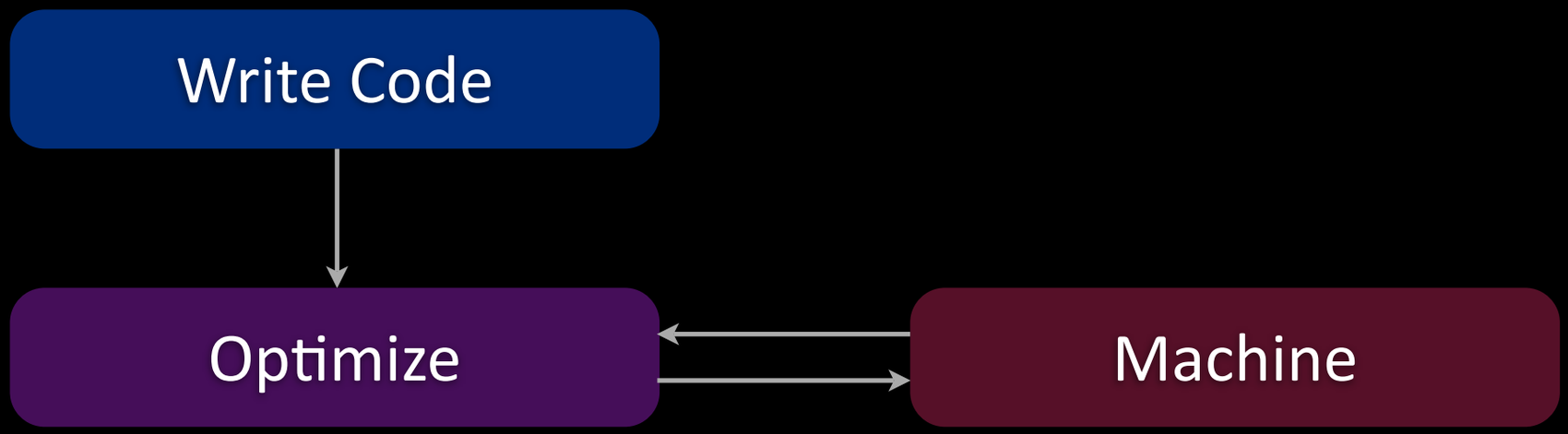
‡ Lawrence Livermore National Laboratory, CA, 94551, USA

Weibel Raphael, October 12th 2011



„http://media.bestofmicro.com/parallelprocessing-multicore,9-P-58669-13.jpg“

Empirical Tuning



PHiPAC

- BLAS Matrix-Matrix multiplication
- Code guidelines
- Empirical search for optimal loop parameters

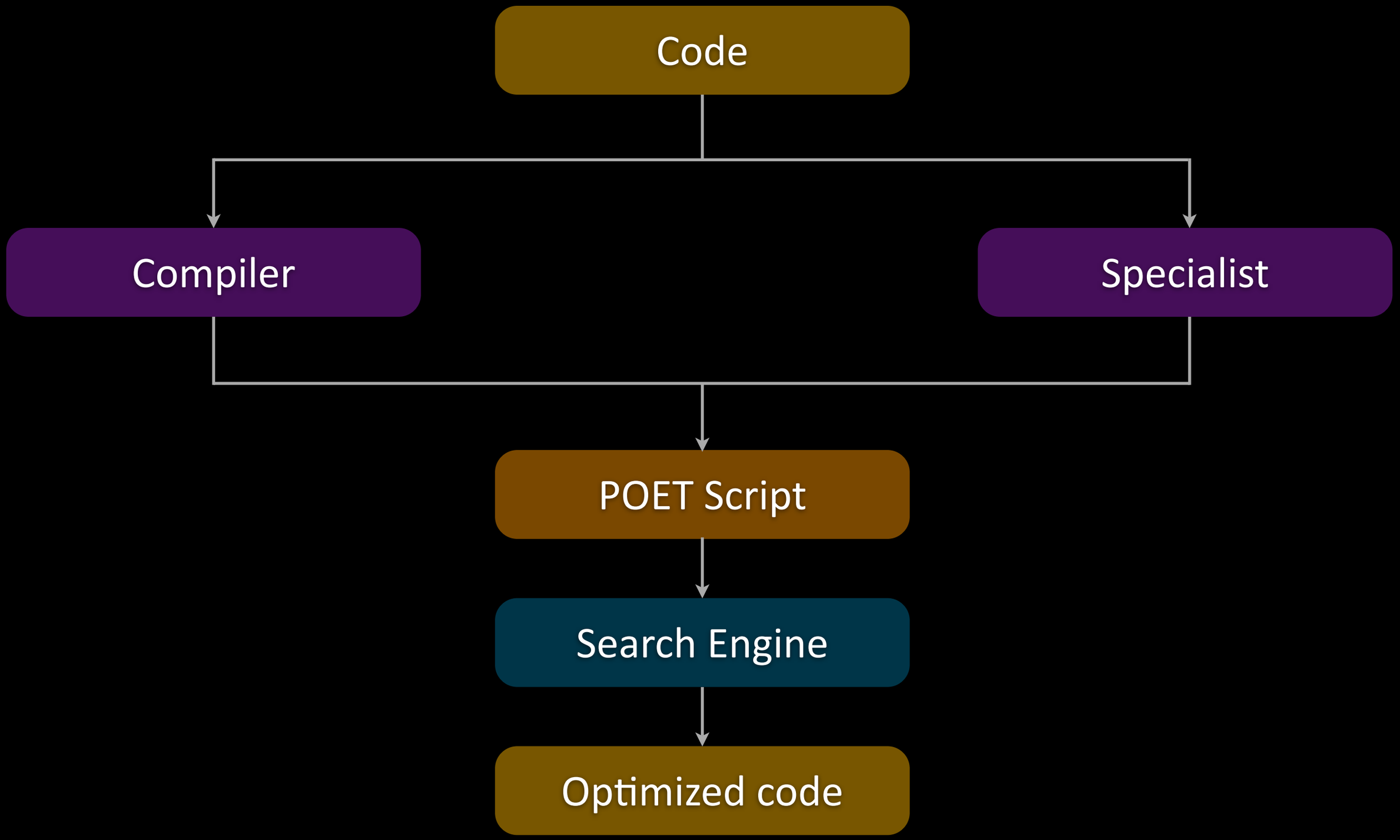
ATLAS

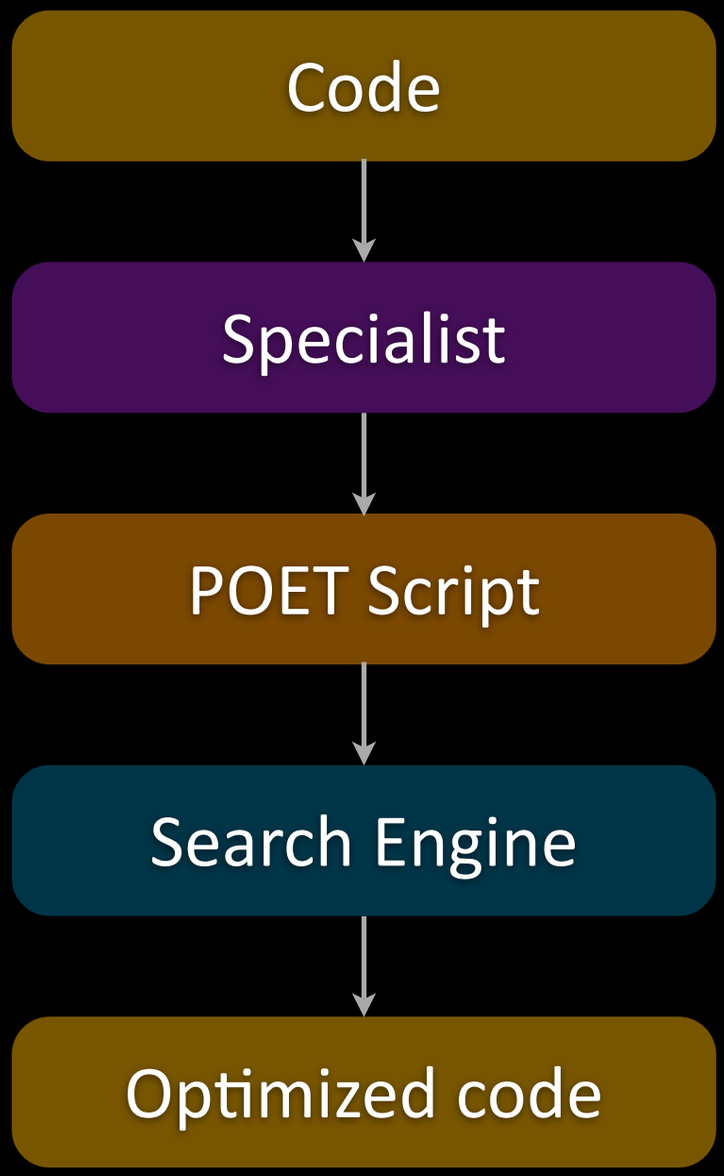
- For BLAS in general
- Automatic optimizations
- Emprical search of optimal code

POET

„ ... parameterizing complex code transformations so that they can be empirically tuned.“

„POET: Parameterized Optimizations for Empirical Tuning (2007)“


Code
Compiler
Specialist
POET Script
Search Engine
Optimized code


Code
Specialist
POET Script
Search Engine
Optimized code

A

M

X

Alpha * B

L

=

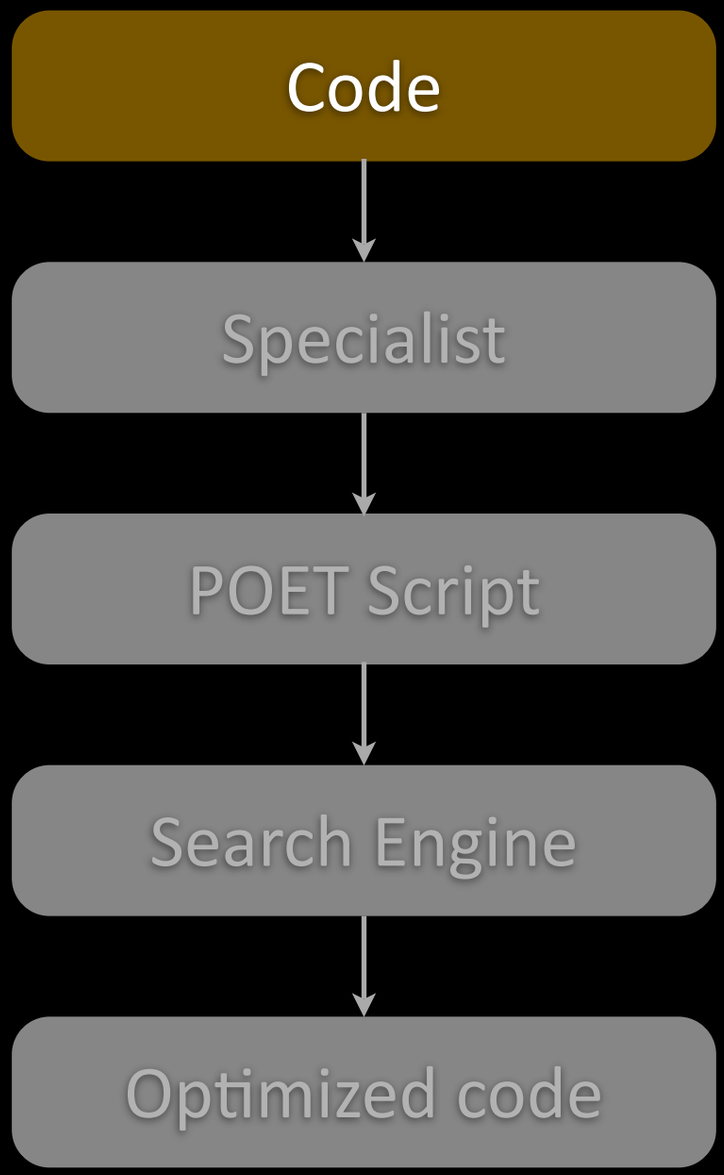
C

M

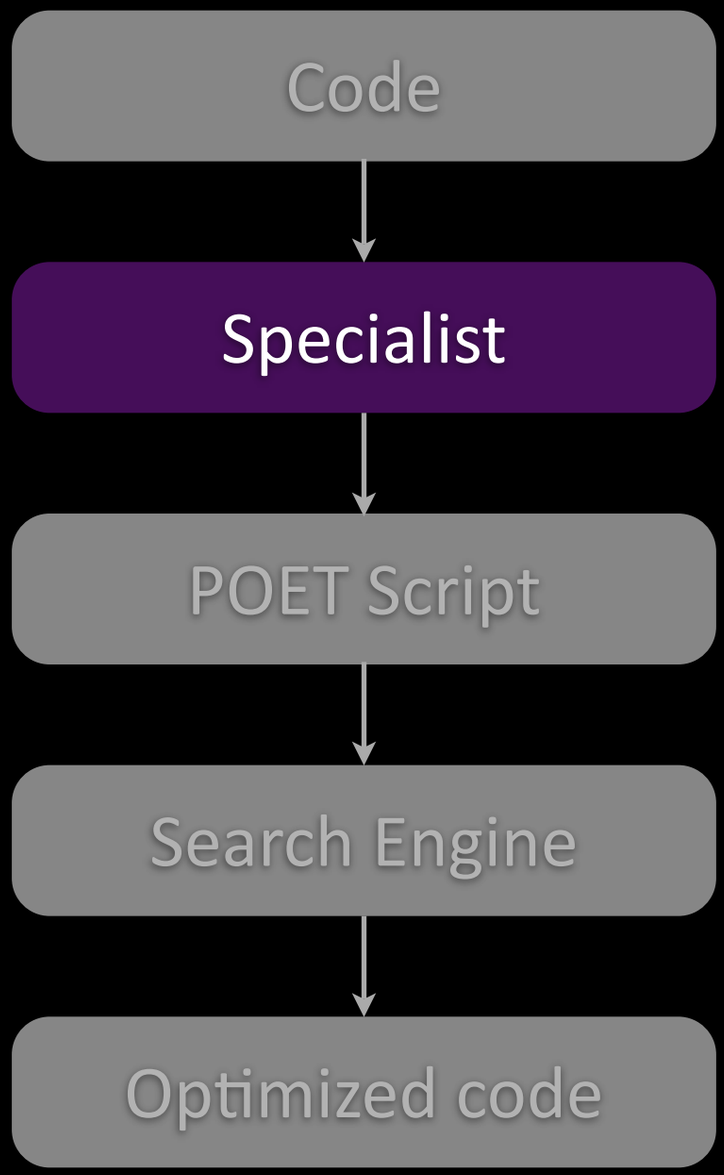
L

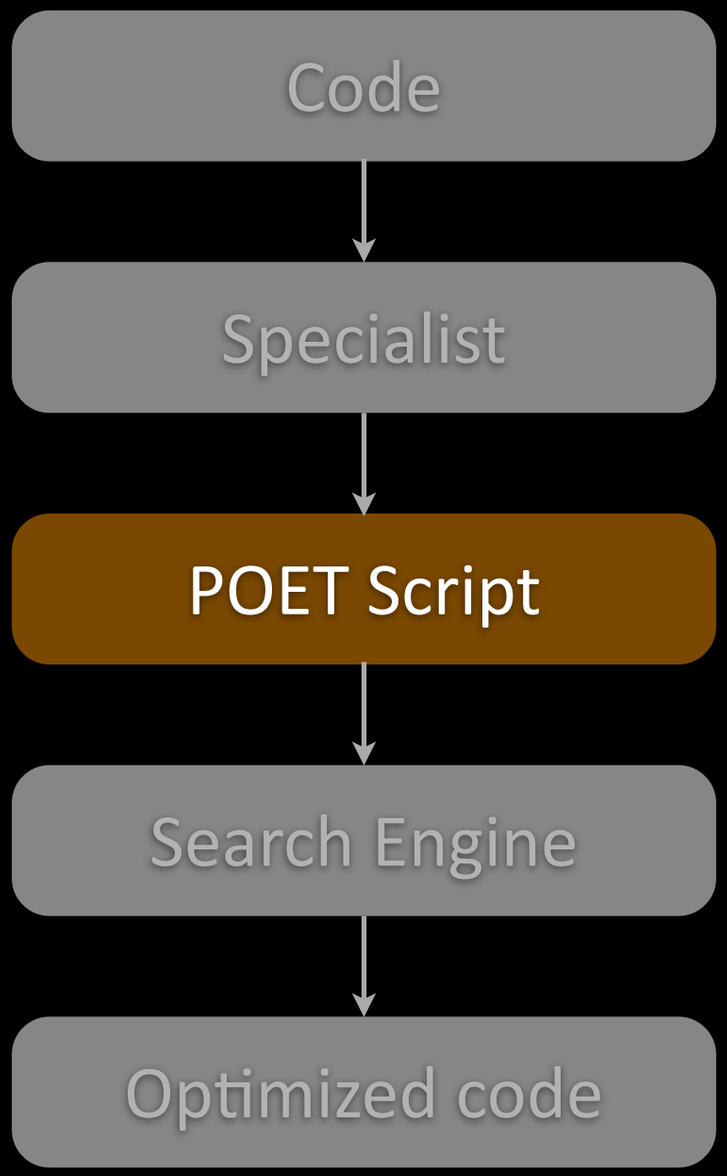
N

N

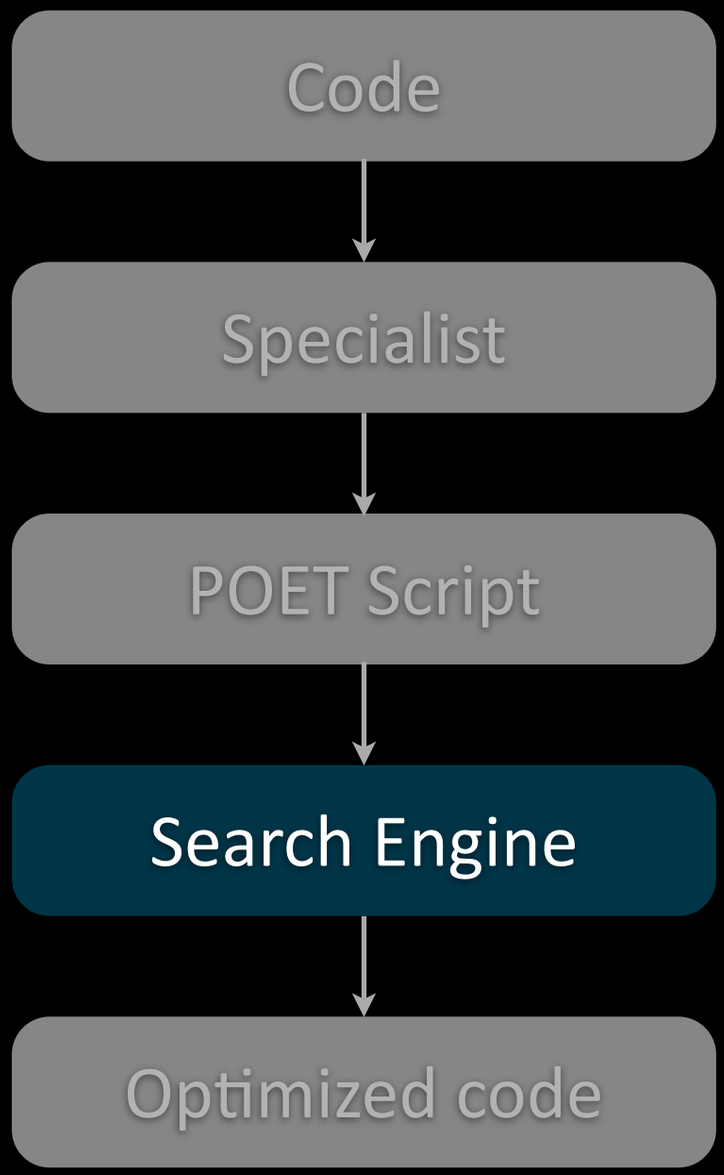

Code
Specialist
POET Script
Search Engine
Optimized code

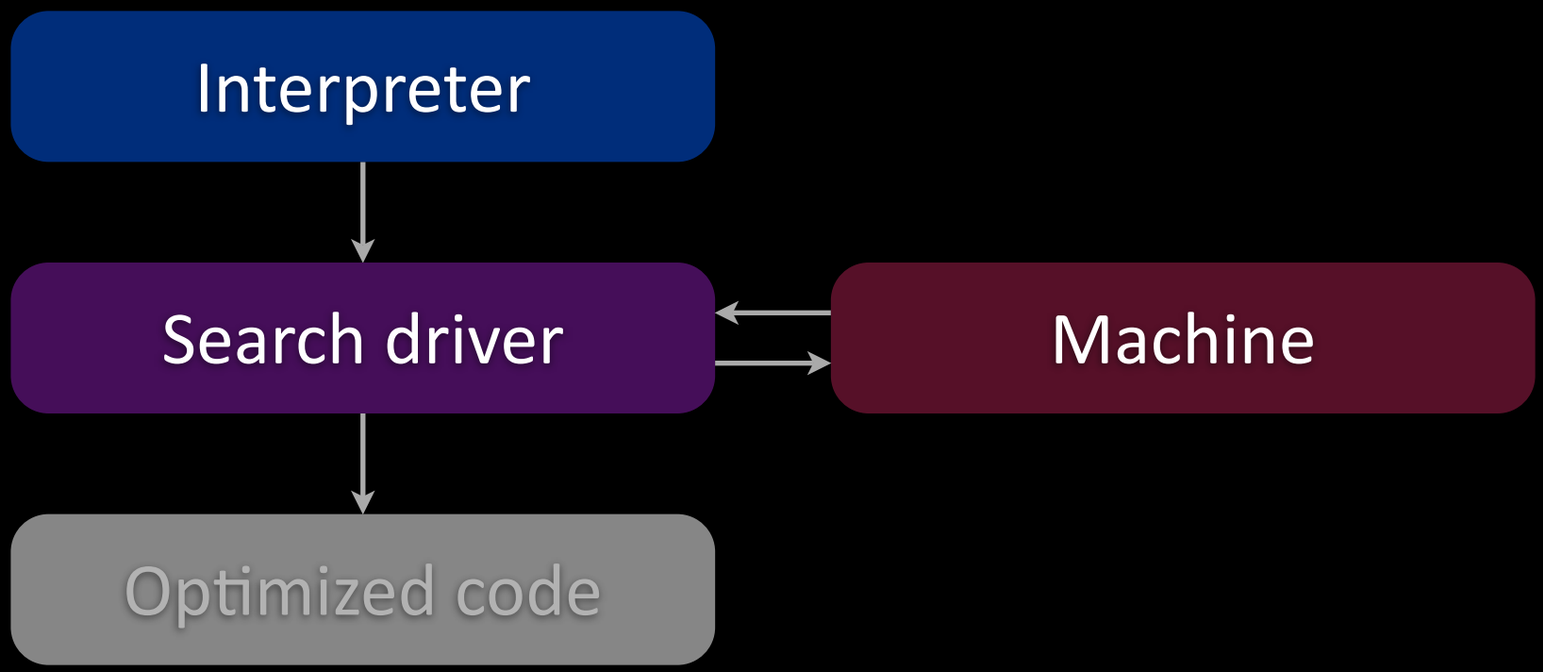
```
void dgemm(int m, int n, int l, double alpha, double *a,double *b, double *c)
{
   for (i=0; i < m; i += 1) {
      for (j=0; j < n; j += 1) {
         for (k=0; k < l; k += 1) {
            c[i+j*m]+= alpha*b[k+j*l] * a[i+m*k];
         }
      }
   }
}
```

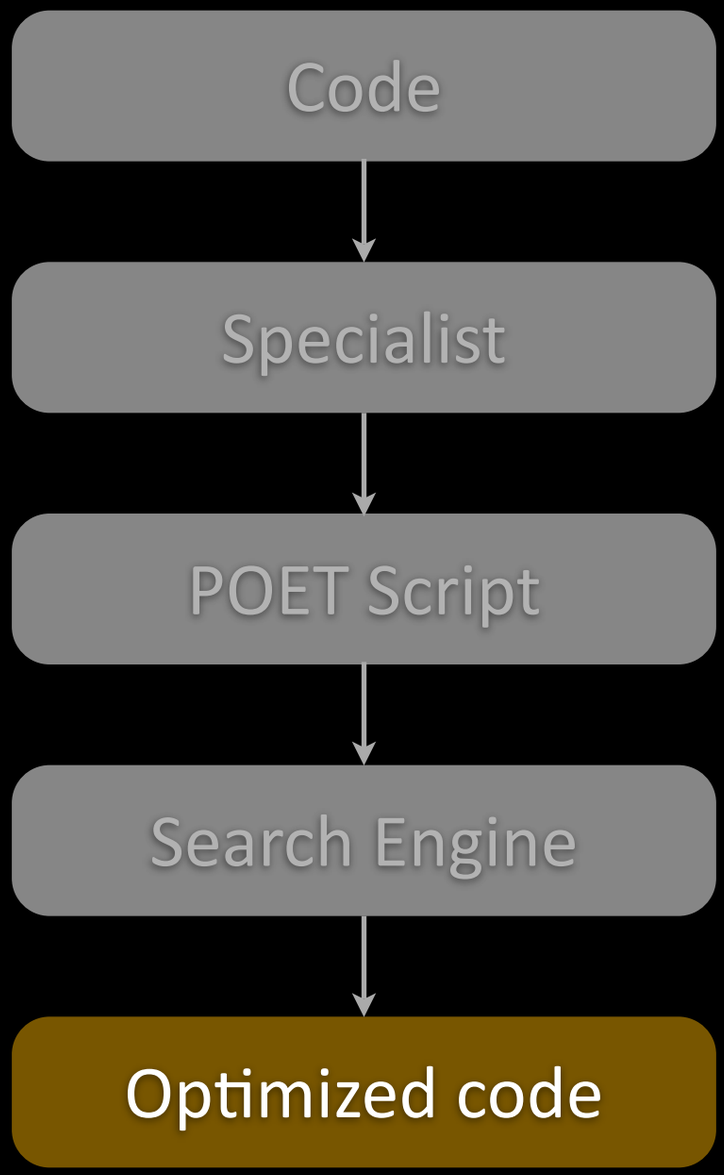

Code
Specialist
POET Script
Search Engine
Optimized code


Code
Specialist
POET Script
Search Engine
Optimized code

```
<code Loop pars=(i,start,stop,step) >
...
<xform Unroll pars=(input,uloop)...)> ... </xform>
...

<define loopJ Loop#("j",0,"n",1)>
...

<define mmStmt "c[i+j*m]+= alpha*b[k+j*l] * a[i+m*k];">
<define nest1 Nest#(loopK,mmStmt)>
...

<define mmHead "void dgemm(int m, int n, int l, double alpha, double *a,
        double *b, double *c)">
<define dgemm Function#(mmHead, "int i, j, k;", nest3)>
<define mm_unroll (Unroll#(dgemm,loopK)) >
...

<output mm_unroll.c (Unroll.ur=8; mm_unroll)>
...
```


Code
Specialist
POET Script
Search Engine
Optimized code


Interpreter
Search driver
Machine
Optimized code


Code
Specialist
POET Script
Search Engine
Optimized code

```
void dgemm(int m, int n, int l, double alpha, double *a, double *b,
double *c)
{
    int i, j, k;
    int kk;
    for (i=0; i < m; i += 1) {
        for (j=0; j < n; j += 1) {
            for (k=0; k < l-7; k += 1) {
                c[i+j*m]+= alpha*b[k+j*l] * a[i+m*k]; k+=1;
                c[i+j*m]+= alpha*b[k+j*l] * a[i+m*k]; k+=1;
                c[i+j*m]+= alpha*b[k+j*l] * a[i+m*k]; k+=1;
                c[i+j*m]+= alpha*b[k+j*l] * a[i+m*k]; k+=1;
                c[i+j*m]+= alpha*b[k+j*l] * a[i+m*k]; k+=1;
                c[i+j*m]+= alpha*b[k+j*l] * a[i+m*k]; k+=1;
                c[i+j*m]+= alpha*b[k+j*l] * a[i+m*k];
            } for ( ; k < l; k+=1) {
                c[i+j*m]+= alpha*b[k+j*l] * a[i+m*k];
            }
        }
    }
}
```

POET Script

<code>

```
<code Loop pars=(i,start,stop,step) >

  @init = (if start=="" then "" else (i "=" start ));
   test = (if stop=="" then "" else (i "<=" stop ));
   incr = (if step=="" then "" else (i "+=" step ));
  @for (@init@; @test@; @incr@)

</code>
```

<xform>

```
<xform Unroll pars=(input,uloop) tune=(ur=16)>

   if (!(input : Nest#(loop,body))) then (
      if (input : Function#(head,decl,body)) then
         Function#(head,decl,Unroll#(body,uloop))
      else if (input : Sequence#(s1,s2)) then
         Sequence#(Unroll#(s1,uloop),Unroll#(s2,uloop))
      else input
   )

   else if (loop != uloop) then
      Nest#(loop, Unroll#(body,uloop))
   else if (!ur || ur == 1) then input
   else (
      loop : Loop#(ivar,start,stop,step);
      dup = (body DUPLICATE#(_, ur-1,(ivar "+=1;" ENDL body)));
      Sequence#(Nest#(Loop#(ivar,start,(stop "-" ur-1),step),dup),
           Nest#(Loop#(ivar,"",stop,step),body))
   )

</xform>
```

<define>

```
<define loopJ Loop#("j",0,"n",1)>
<define loopI Loop#("i",0,"m",1)>
<define loopK Loop#("k",0,"l",1)>
<define mmStmt "c[i+j*m]+= alpha*b[k+j*l] * a[i+m*k];">
<define nest1 Nest#(loopK,mmStmt)>
<define nest2 Nest#(loopI,nest1)>
<define nest3 Nest#(loopJ,nest2)>
<define mmHead "void dgemm(int m, int n, int l, double alpha, double *a,
        double *b, double *c)">
<define dgemm Function#(mmHead, "int i, j, k;", nest3)>
<define mm_unroll (Unroll#(dgemm,loopK)) >
<define mm_block (Block#(dgemm,nest3,mmStmt))>
<define mm_permute (Permute#(dgemm,nest3,mmStmt))>
<define mm_block_unroll (Unroll#(mm_block, InnermostLoop#(mm_block)))>
```

<output>

```
<output mm.c dgemm>
<output mm_unroll.c (Unroll.ur=8; mm_unroll)>
<output mm_block.c (Block.bsize=(8 64 128); mm_block)>
<output mm_permute.c (Permute.order=(3 2 1); mm_permute)>
<output mm_block_unroll.c (Block.bsize=(16 16 8);Unroll.ur=8; mm_block_unroll)>
```

Formal Grammer

Sections

```
section :
    "<" "code" ID {codeAttr} ">" Exp "<" "/code" ">"
  |"<" "xform" ID {xformAttr} ">" Exp "<" "/xform" ">"
  |"<" "define" ID Exp ">"
  |"<" "output" FNAME Exp ">"
```

```
(1) poet : {section}
(2) section :
      "<" "code" ID {codeAttr} ">" Exp "<" "/code" ">"
(3)  |"<" "xform" ID {xformAttr} ">" Exp "<" "/xform" ">"
(4)  |"<" "define" ID Exp ">"
(5)  |"<" "output" FNAME Exp ">"
(6) codeAttr : "pars" "=" "(" ID {"," ID} ")"
(7) xformAttr : "pars" "=" "(" ID {"," ID} ")"
(8)  | "tune" "=" "("ID"="INT{","INT} {";"ID"="INT{","INT}}")"
(9) Exp : Op
     | Op ";" Exp | Op "," Exp | Op Exp
(10)Op : "if" Op "then" Op "else" Op (11) | ID {"." ID} "=" Op
(12) |ID":"Op
(13) | ID {"." ID} "#" Unit | ["car","cdr",INT] "#" Op
(14) | REPLACE "#" "(" Op "," Op "," Op ")"
(15) | PERMUTE "#" "(" Op "," Op ")"
(16) | DUPLICATE "#" "(" Op "," Op "," Op ")"
(17) | Op ["+","-","*"] Op | "-" Op
(18) | Op ["<","<=",">",">=","==","!="] Op (19)|"!"Op|Op"&&"Op |Op"||"Op
(20) | Unit
(21)Unit : ID {"." ID} | INT | SOURCE | "(" Exp ")"

ID          identifiers, e.g. Loop, Nest;
FNAME       file names, e.g., mm.c, ../dgemm.c;
INT         integer literals, e.g., 1, 0, 16, 64;
SOURCE      strings of code fragments;
```

Evaluation rules

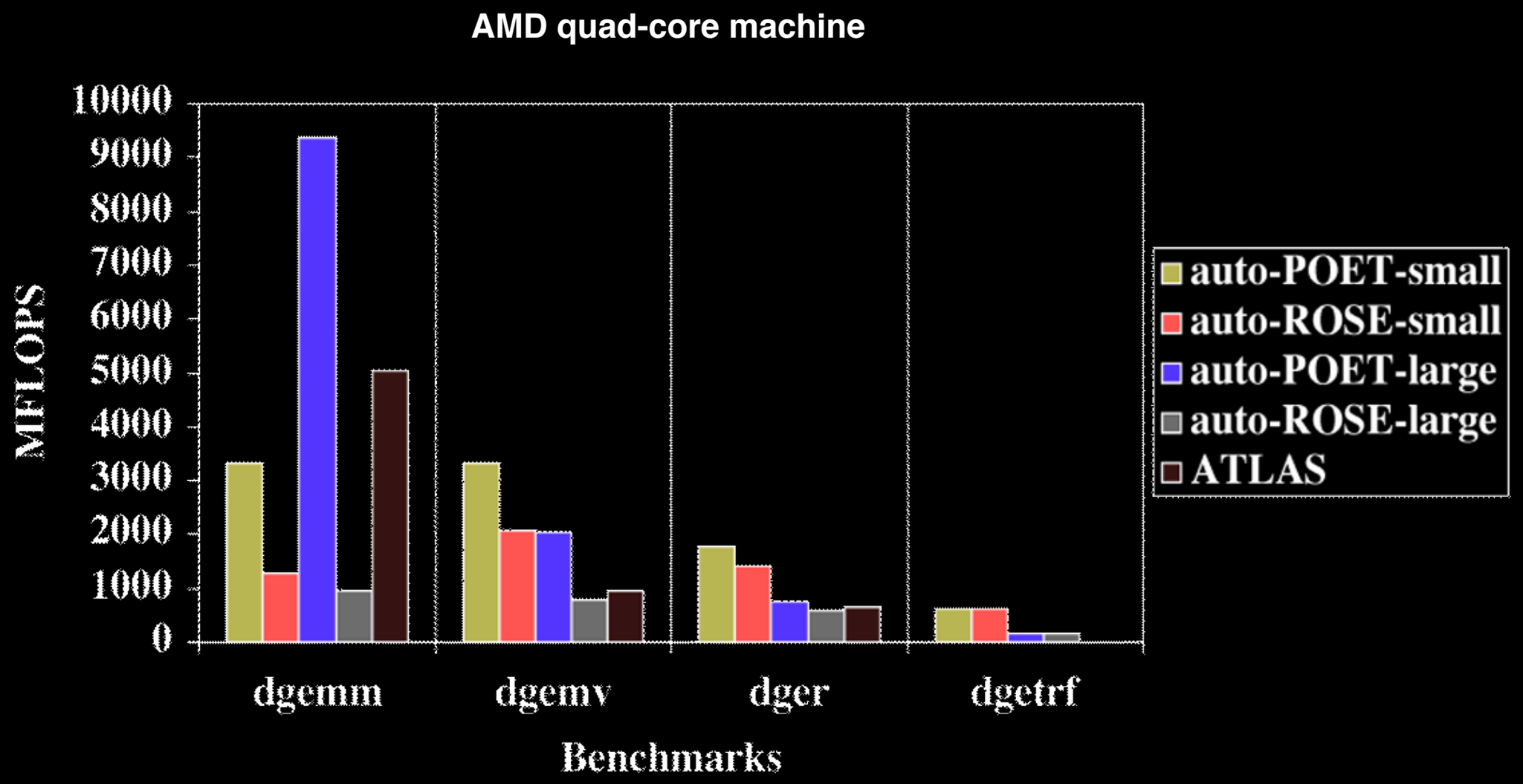
DUPLICATE#

```
e := dup(lv,t,e1) for(i=0;i<t;++i)
                  {set value(lv,i); e.val = list(eval(e1), e.val)}
```

```
<xform Unroll pars=(input,uloop) tune=(ur=16)>
    ...
    dup = (body DUPLICATE#(_, ur-1,(ivar "+=1;" ENDL body)));
    ...
</xform>
```

```
(1)e:=i                    e.val = i.val
(2) | s                    e.val = s.val
(3) | e1 e2                e.val = list(eval(e1), eval(e2))
(4) | e1 ,e2 ,...,em       e.val = tuple(eval(e1),eval(e2), ..., eval(em))
(5) | car # e1             t = eval(e1); e.val = is list(t)?f irst(t) : t
(6) | cdr # e1             t = eval(e1); e.val = is list(t)?rest(t) : “”
(7) | i#e1                 t = eval(e1); e.val = tuple elem(t,i.val)
(8) | cv#e1                if(!replaceCode) e.val = cv#eval(e1)
                           else {set pars(cv, eval(e1 )); e.val = eval(find_code(cv))
(9) | e1 : e2              e.val = match_AST(e1,e2)
(10)| dv                   e.val = find_code(dv)
(11)| lv                   e.val = find_value(lv)
(12)| lv = e1              set_value(lv, eval(e1)); e.val = “”
(13)| xv # e1              set pars(xv, eval(e1)); e.val = eval(find_code(xv))
(14)| repl(e1 ,e2 ,e3 )    e.val = Replace(eval(e1), eval(e2), eval(e3))
(15)| perm(e1,e2)          e.val = Permute(eval(e1), eval(e2))
(16)| dup(lv,t,e1)         for(i=0;i<t;++i)
                           {set_value(lv,i); e.val = list(eval(e1), e.val)}
(17)| e1 ;e2               eval(e1): e.val = eval(e2 )
(18)| if(e1 ,e2 ,e3 )      e.val = bool(eval(e1))? eval(e2) : eval(e3)
(19)| e1 opi e2            e.val = eval(e1)opi e2 .val
(20)| ! e1                 e.val = ! bool(e1.val)
(21)| e1 opb e2            e.val = bool(e1.val) opb bool(e2.val)

i: integer constants;
s:string constants;
lv: local variables;
cv: global handles of code sections;
dv: global handles of define sections;
xv: global handles of xform sections;
opi: integer binary operations (+,-,*,<,>,==,<=,>=,!=);
opb: boolean binary operations (&&, ||).
```

Results



„Automated Programmable Control and Parameterization of Compiler Optimizations (2011)“ by Qing Yi

Conclusion

- Supports any code transformation
- Supports all programming languages
- Supports Multi-Core CPU Architecture
- Does only optimize on the source code level

Further information

- http://bigbend.cs.utsa.edu/poet/index.php