Chlorophyll: Synthesis-Aided Compiler for GreenArrays

Phitchaya Mangpo Phothilimthana
Ras Bodik

University of California, Berkeley

Future Low-Power Architectures

Many small cores → Fine-grained partitioning

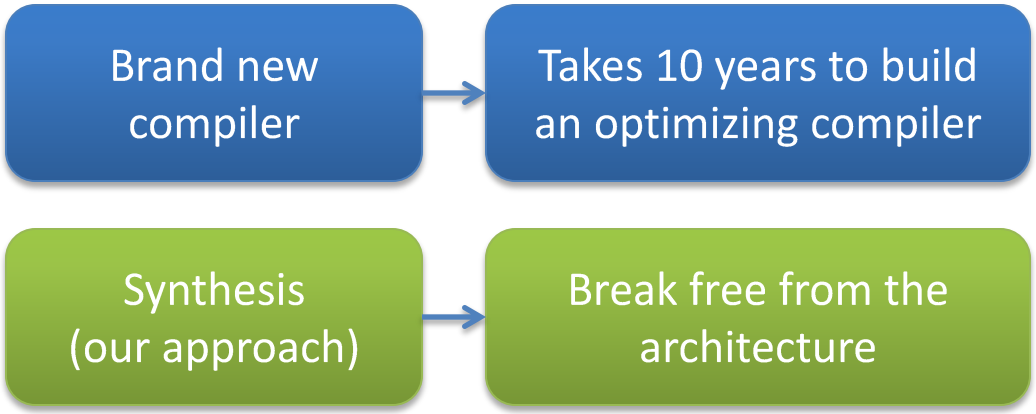
Simple interconnect → SW-controlled messages

New ISAs → New compiler optimizations

What we are working on

- New programming model for spatial architectures
- Synthesis-aided “compiler”

Compilers: State of the Art



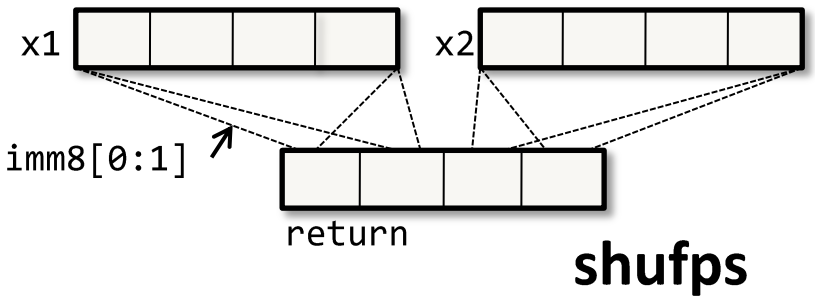
Synthesis, an alternative to compilation

- *Compiler*: transforms the source code
- *Synthesis*: searches for a correct, fast program

Program Synthesis (Example)

Specification:

```
int[16] transpose(int[16] M) {
  int[16] T = 0;
  for (int i = 0; i < 4; i++)
    for (int j = 0; j < 4; j++)
      T[4 * i + j] = M[4 * j + i];
  return T;
}
```

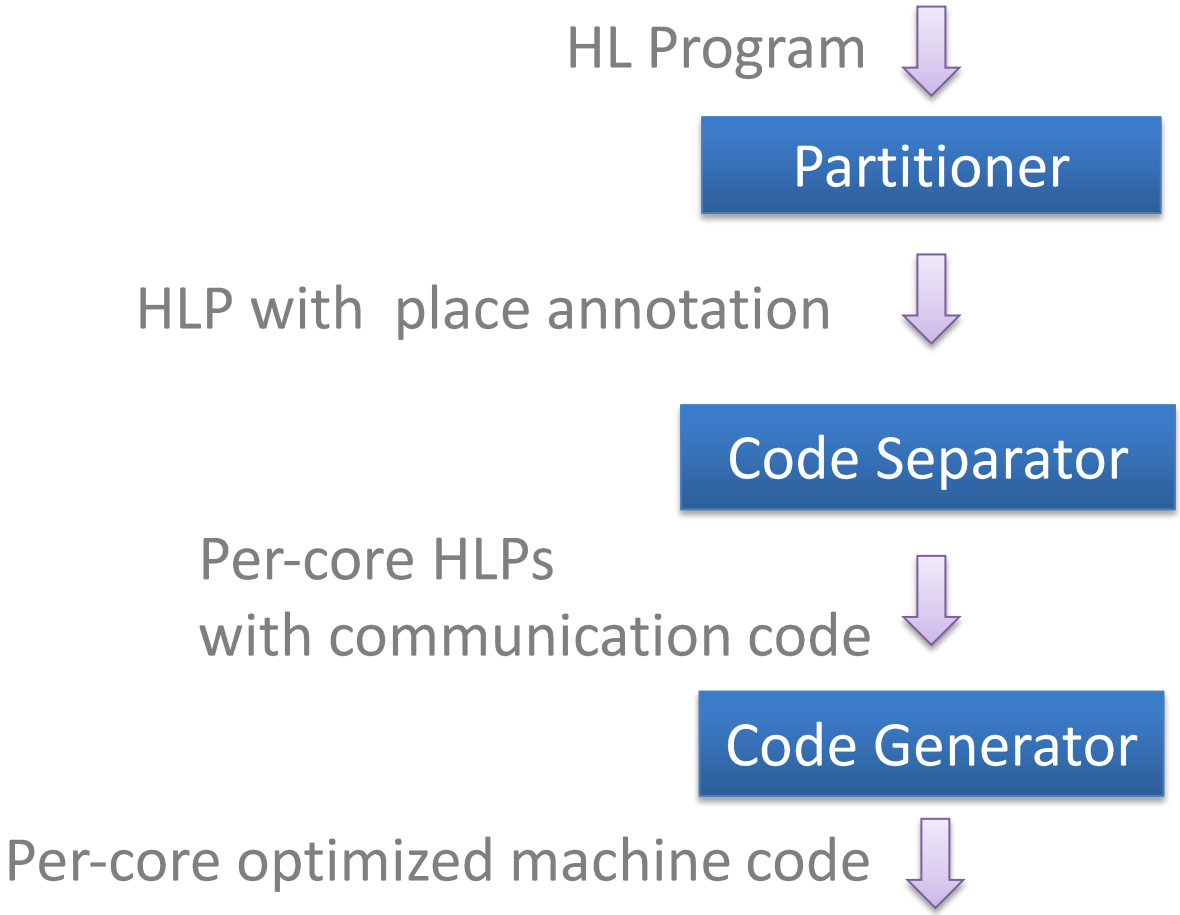


Sketch:

```
int[16] trans_sse(int[16] M) implements trans {
  int[16] S = 0, T = 0;
  repeat (??) S[??::4] = shufps(M[??::4], M[??::4], ??);
  repeat (??) T[??::4] = shufps(S[??::4], S[??::4], ??);
  return T;
}
```

Synthesis time < 10 seconds.
Search space > 10^{70}

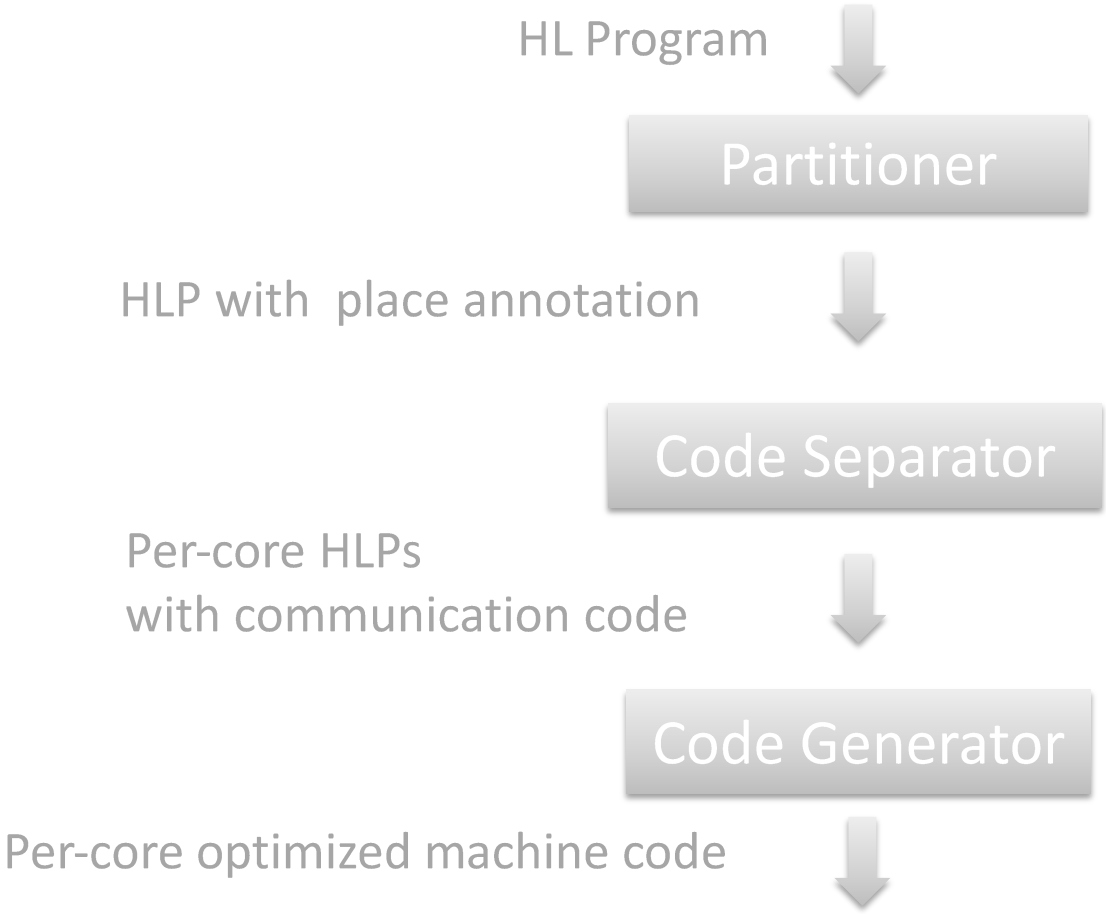
Our Plan



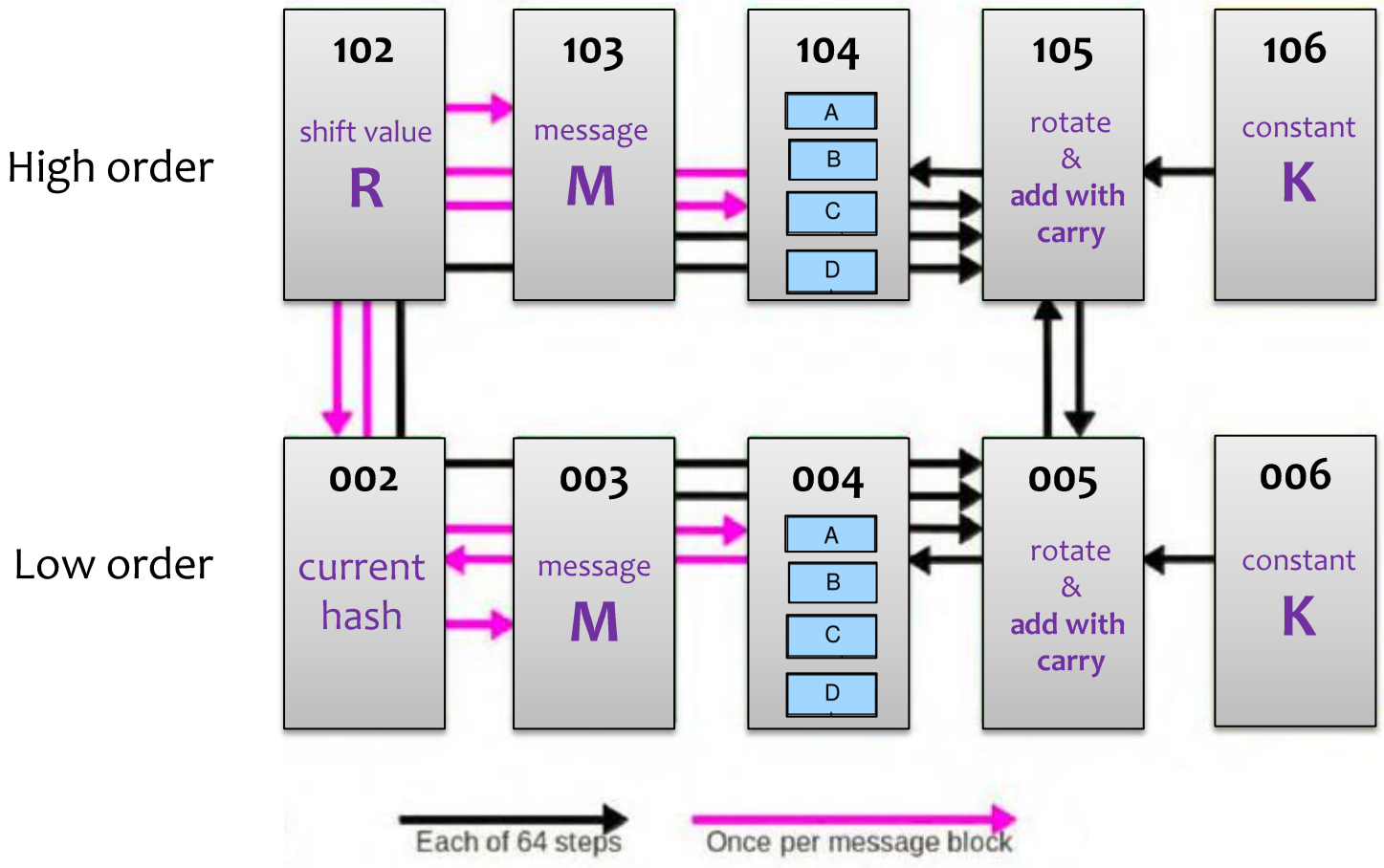
1. New *Programming Model*

2. New *Approach* ***Using*** *Synthesis*

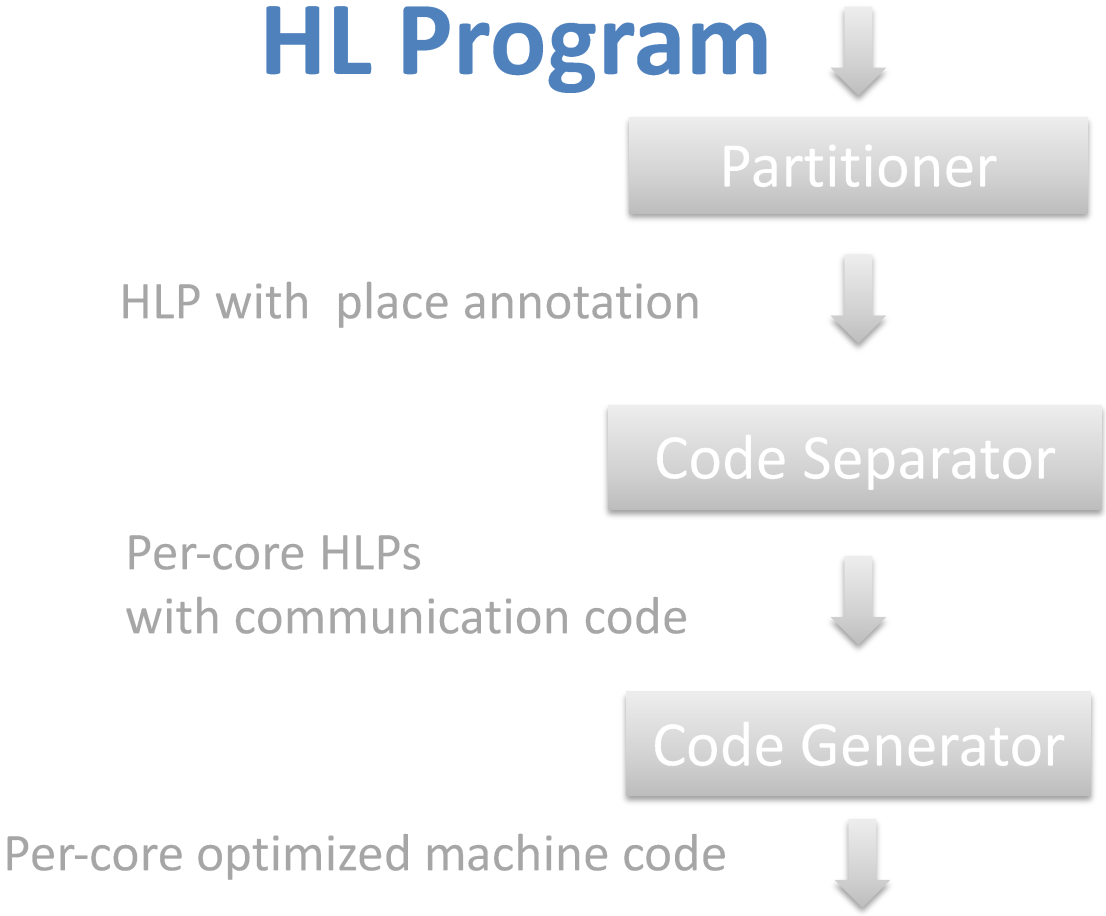
Our Plan


HL Program
Partitioner
HLP with place annotation
Code Separator
Per-core HLPs
with communication code
Code Generator
Per-core optimized machine code

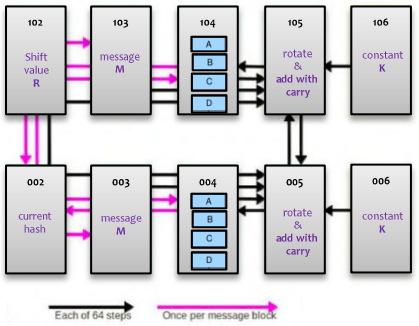
MD5 on GA144


High order
Low order
102
shift value
R
103
message
M
104
A
B
C
D
105
rotate
&
add with
carry
106
constant
K
002
current
hash
003
message
M
004
A
B
C
D
005
rotate
&
add with
carry
006
constant
K
Each of 64 steps
Once per message block

Programming Model

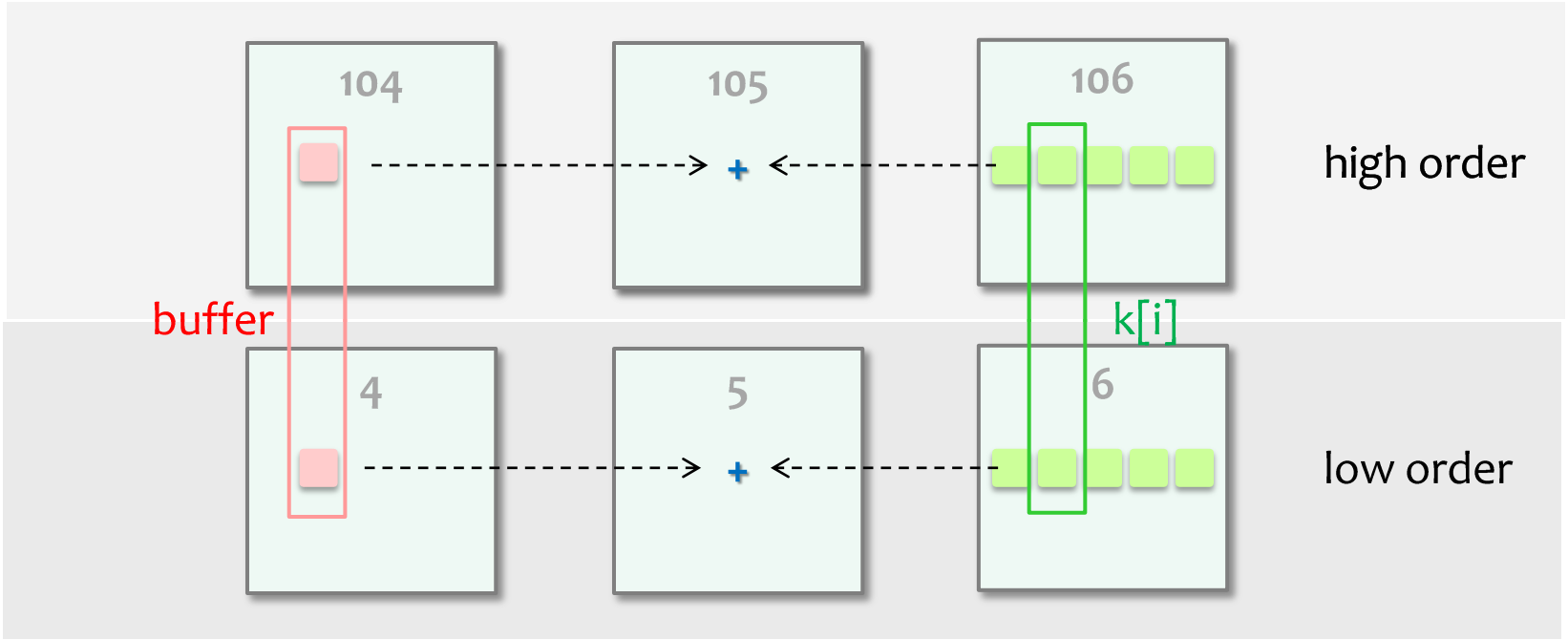


Spatial programming model

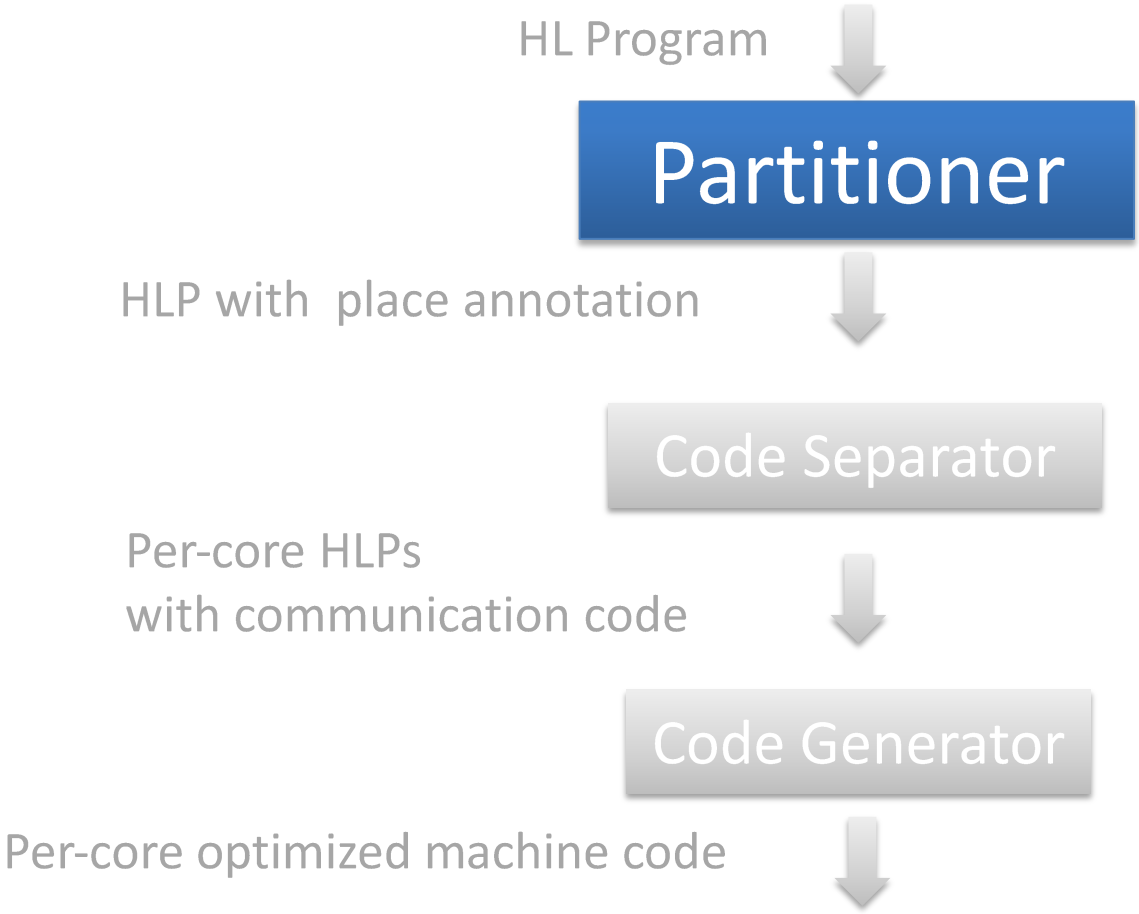



```
typedef pair<int,int> myInt;

vector<myInt>@{[0:64]=(106,6)} k[64];

myInt@(105,5) sumrotate(myInt@(104,4) buffer, ...) {
  myInt@here sum = buffer +@here k[i] + message[g];
  ...
}
```

buffer is at (104,4)
\+ is at (105,5)
k[i] is at (106,6)

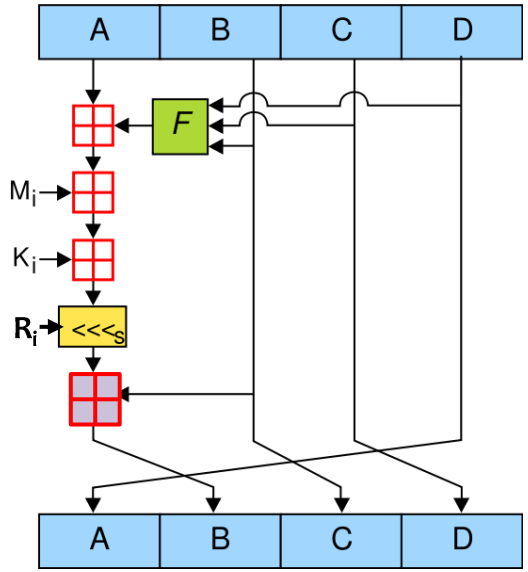


Partitioning Synthesizer

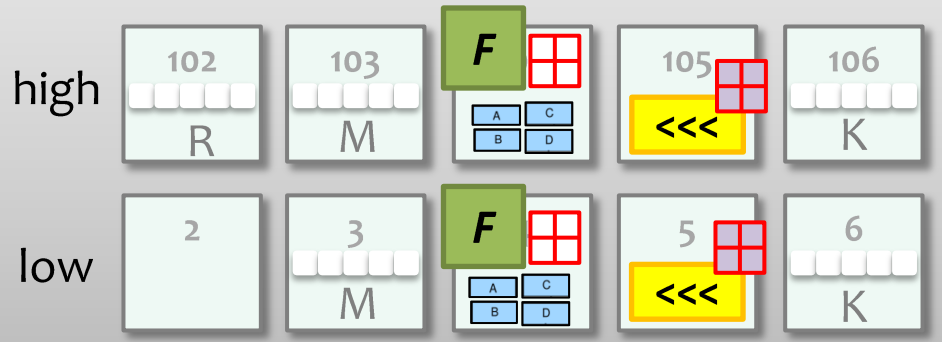


Optimal Partitions from Our Synthesizer

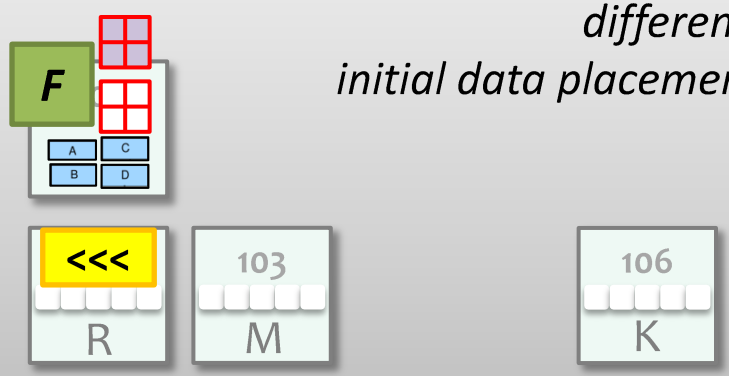
- Benchmark: simplified MD5 (one iteration)
- Partitions are automatically generated.



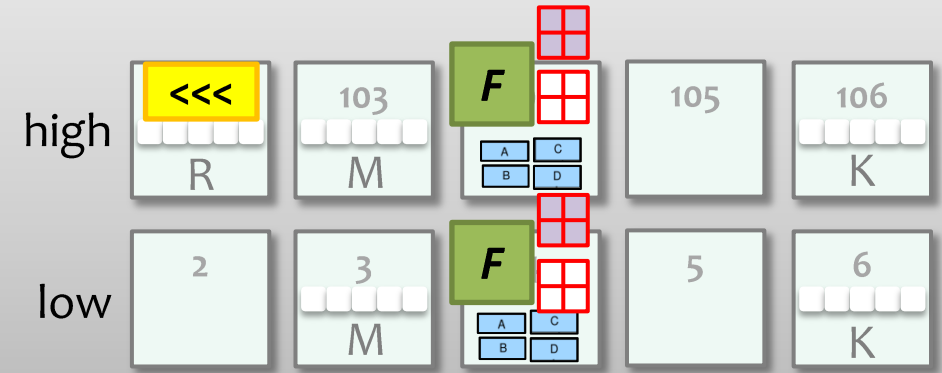
256-byte mem per core
initial data placement specified



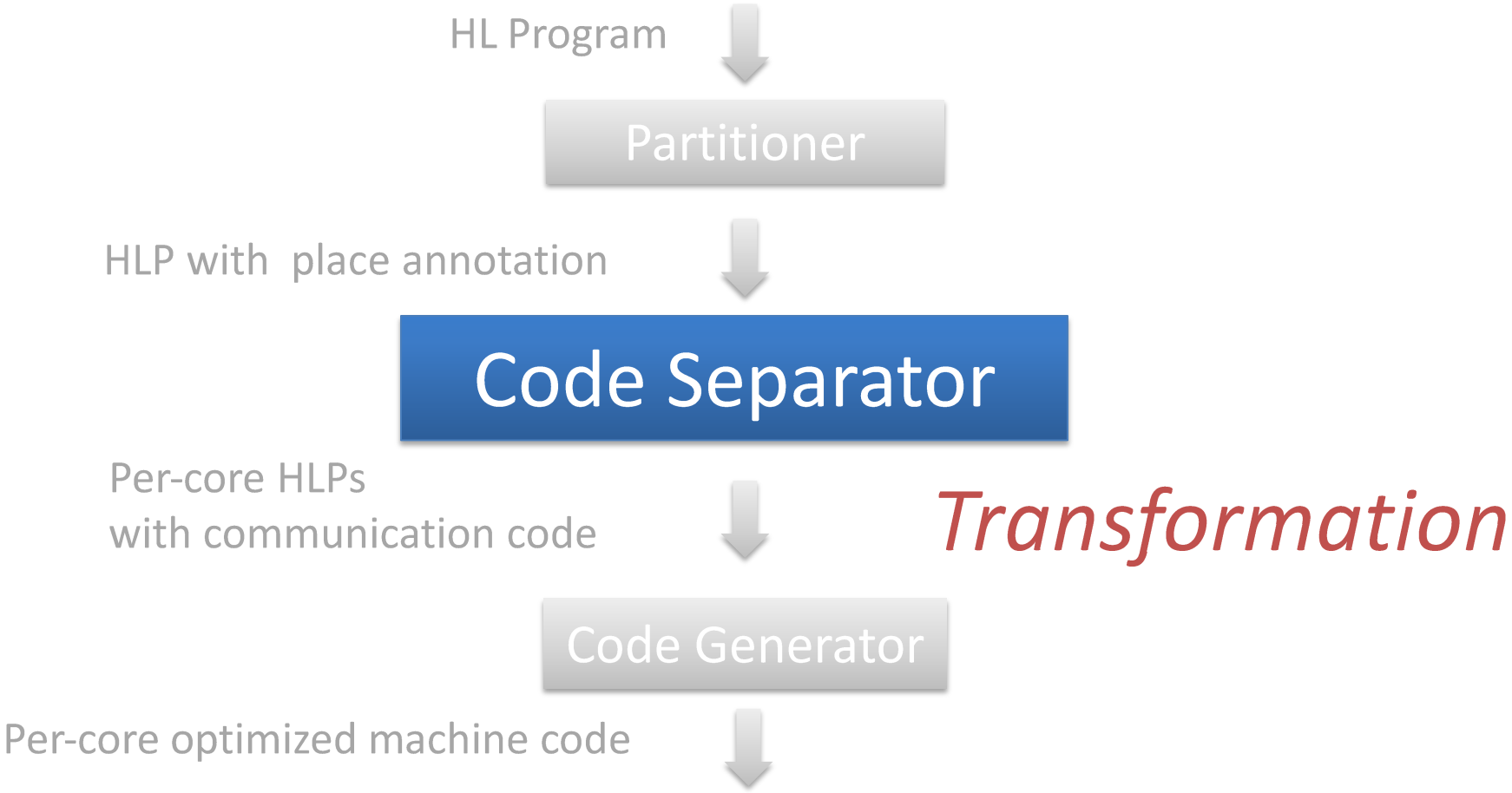
512-byte mem per core
different
initial data placement



512-byte mem per core
same initial data placement



Code Separator & Communication Code



Matrix Multiplication: HLP

```
// C = A x B

int[] A[36], B[36], C[36];

for(i from 0 to 6) {
   for(j from 0 to 6) {
      int sum = 0;
      for(k from 0 to 6) {
         sum = sum + A[6*i+k] * B[6*k+j];
      }
      C[6*i+j] = sum;
   }
}
```

Matrix Multiplication: HLP with Partition Annotation

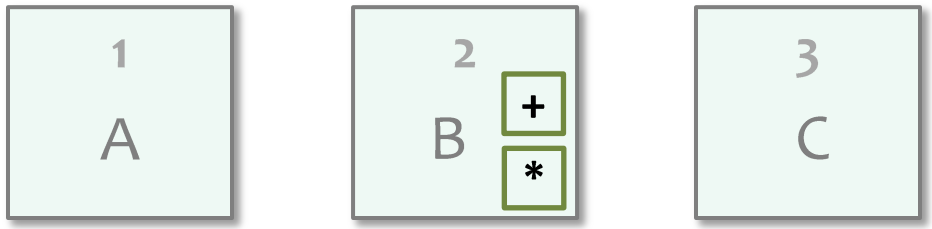
```
// C = A x B

int[]@1 A[36];
int[]@2 B[36];
int[]@3 C[36];

for(i from 0 to 6) {
   for(j from 0 to 6) {
      int@2 sum = 0;
      for(k from 0 to 6) {
         sum = sum +@2 A[6 *@1 i+ @1 k] * B[6 *@2 k +@2 j];
      }
      C[6 *@3 i +@3 j] = sum;
   }
}
```

Matrix Multiplication: Per-core HLPs

```
int A[36];

for(int i = 0; i < 6; ++i) {
   for(int j = 0; j < 6; ++j) {
      for(int k = 0; k < 6; ++k) {
         write("right",A[6*i+k]);
      }
   }
}
```

```
int B[36];

for(int i = 0; i < 6; ++i) {
   for(int j = 0; j < 6; ++j) {
      int sum = 0;
      for(int k = 0; k < 6; ++k) {
         sum = sum + read("left") * B[6*k+j];
      }
      write("right",sum);
   }
}
```

```
int C[36];

for(int i = 0; i < 6; ++i) {
   for(int j = 0; j < 6; ++j) {
      C[6*i+j] = read("left");
   }
}
```

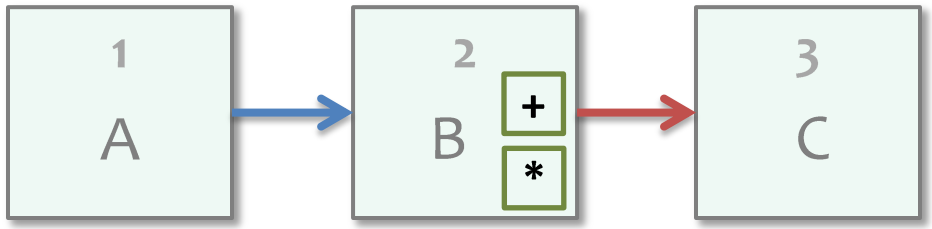


Matrix Multiplication: Per-core HLPs

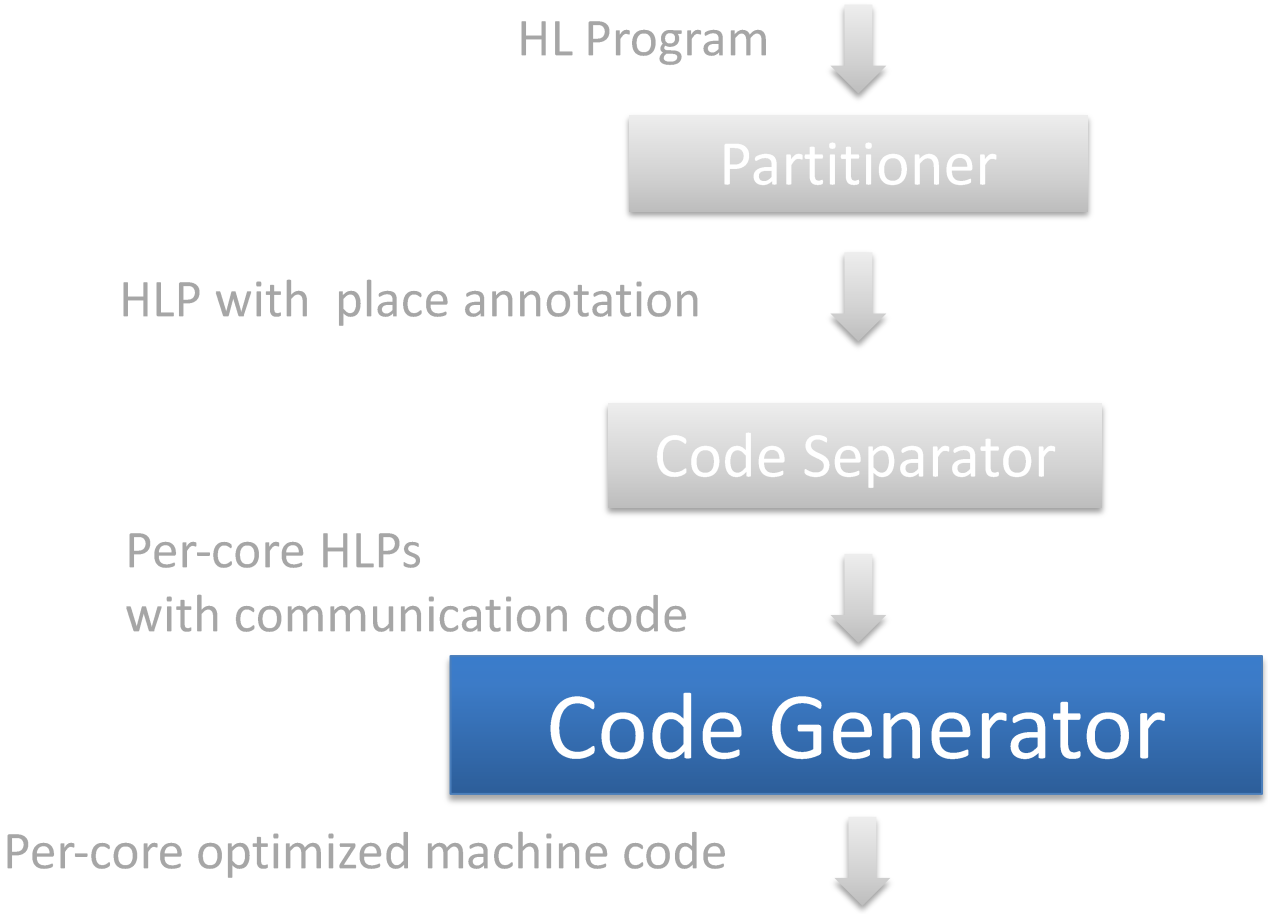
```
int A[36];

for(int i = 0; i < 6; ++i) {
   for(int j = 0; j < 6; ++j) {
      for(int k = 0; k < 6; ++k) {
         write("right",A[6*i+k]);
      }
   }
}
```

```
int B[36];

for(int i = 0; i < 6; ++i) {
   for(int j = 0; j < 6; ++j) {
      int sum = 0;
      for(int k = 0; k < 6; ++k) {
         sum = sum + read("left") * B[6*k+j];
      }
      write("right",sum);
   }
}
```

```
int C[36];

for(int i = 0; i < 6; ++i) {
   for(int j = 0; j < 6; ++j) {
      C[6*i+j] = read("left");
   }
}
```



Code Generation



Code Generation via Superoptimization

Input (specification):

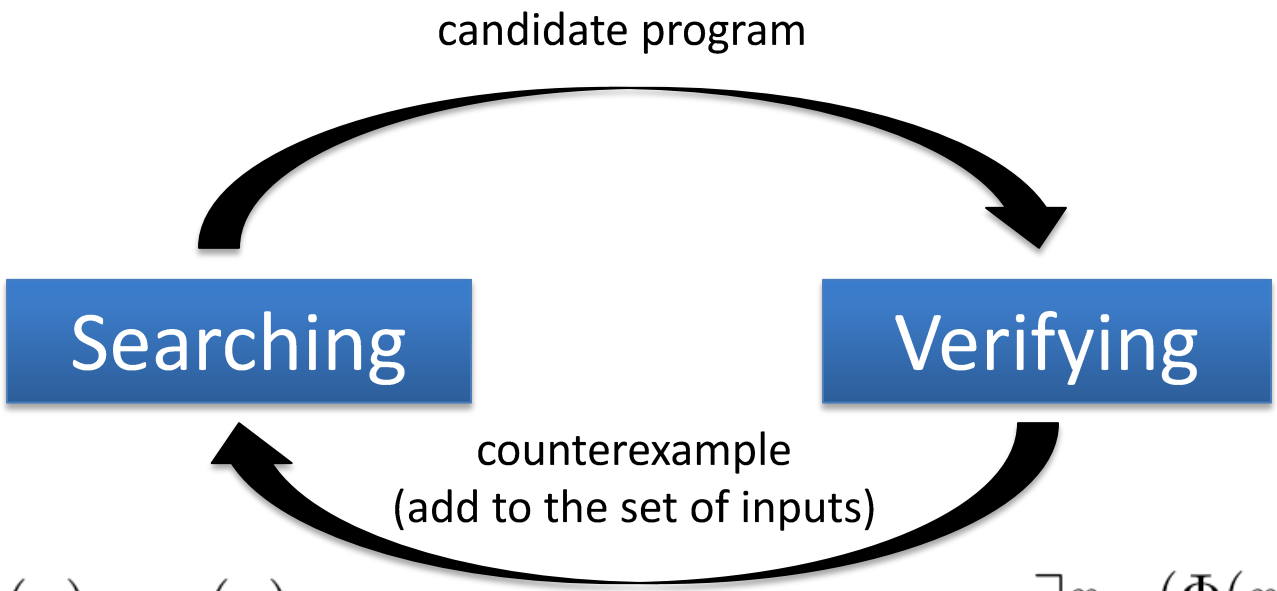
naïve generated code

(Synthesizer use the specification program to generate input-output pairs.)

Strategy:

searching for a sequence of instructions whose behavior is the same as the specification program

Counter Example Guided Inductive Synthesis (CEGIS)



$\exists p.\forall x.\Phi(x) = p(x)$

Given a set of inputs to the program, find a program whose outputs are the same as the spec's

$\exists x.\neg(\Phi(x) = p(x))$

Given a candidate program, find an input to the program such that the output from the candidate program is different from the spec's

Superoptimization: Example

```
int leftrotate(int x, int y, int r) {
   if(r  > 16) { int swap = x;  x = y;  y = swap;  r = r - 16; }
   return ((y >> (16 - r)) | (x << r)) & 65535;
}
```

Per-core high level program

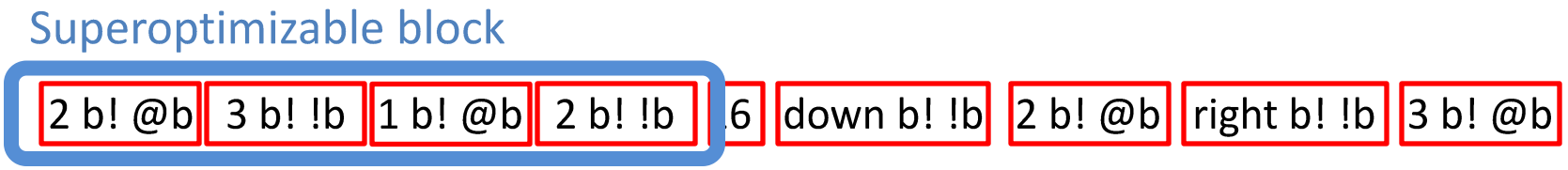
Naïve generated code

```
: 1rep 16 0 b! @b - 1 . + . + ;
: 1if
    if
  2 b! @b  3 b! !b  1 b! @b  2 b! !b
  3 b! @b 1 b! !b 0 b! @b 16
  - 1 . + . + 0 b! !b ; ] then ;
: leftrotate 0 a! !+ !+ !+
 1rep 1if 1 b! @b 1rep
 .. if -1 . +
  for 2/ unext dup
 then drop 2 b! @b 0 b! @b
 .. if -1 . +
  for 2* unext dup
 then drop over - and . + 65535 and ;
```

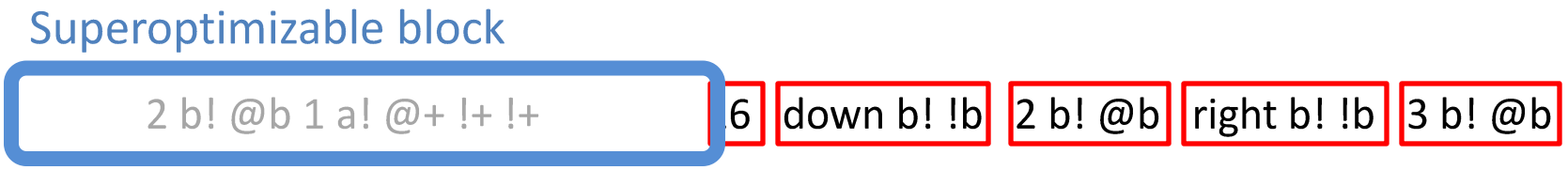
Superoptimizable block

Superoptimizable units

Superoptimization: Example

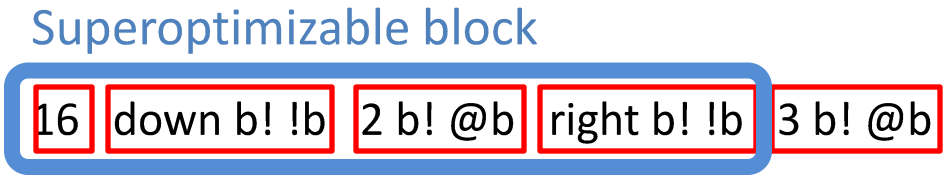


Superoptimization: Example



Superoptimization: Example

2 b! @b 1 a! @+ !+ !+



No better implementation found!

Superoptimization: Example

2 b! @b 1 a! @+ !+ !+

Superoptimizable block

16 down b! !b | 2 b! @b | right b! !b | 3 b! @b

Superoptimization: Example

Superoptimizable block

```
2 b! @b 1 a! @+ !+ !+
16 down b! 2 a! !b @+ right b! !b @
```

Superoptimization: Example

```
int leftrotate(int x, int y, int r) {
   if(r  > 16) { int swap = x;  x = y;  y = swap;  r = r - 16; }
   return ((y >> (16 - r)) | (x << r)) & 65535;
}
```

Per-core high level program

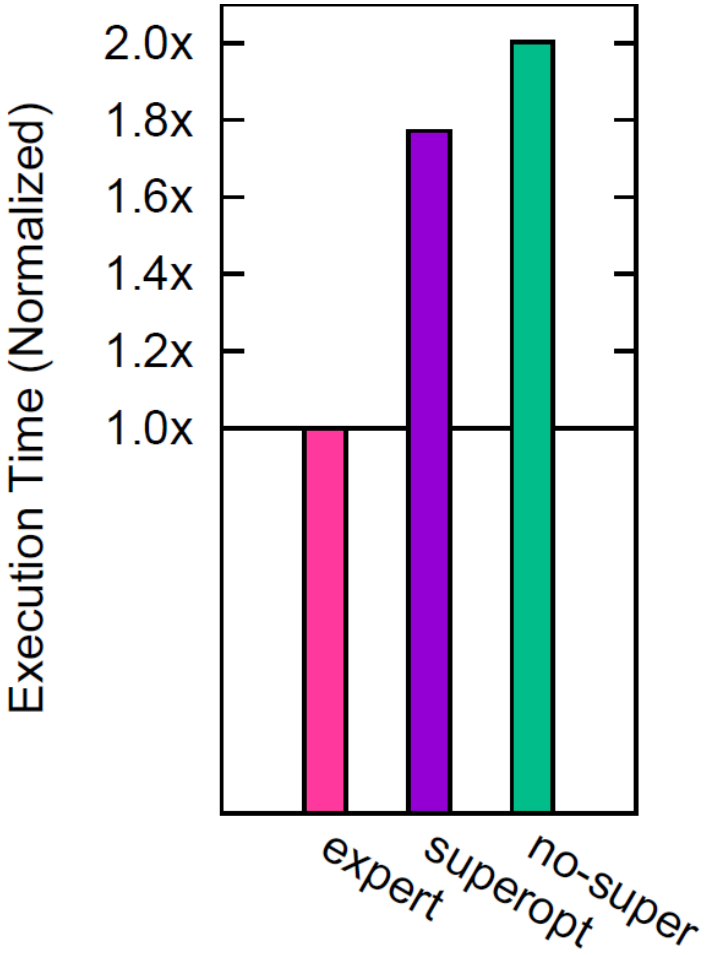
Naïve generated code

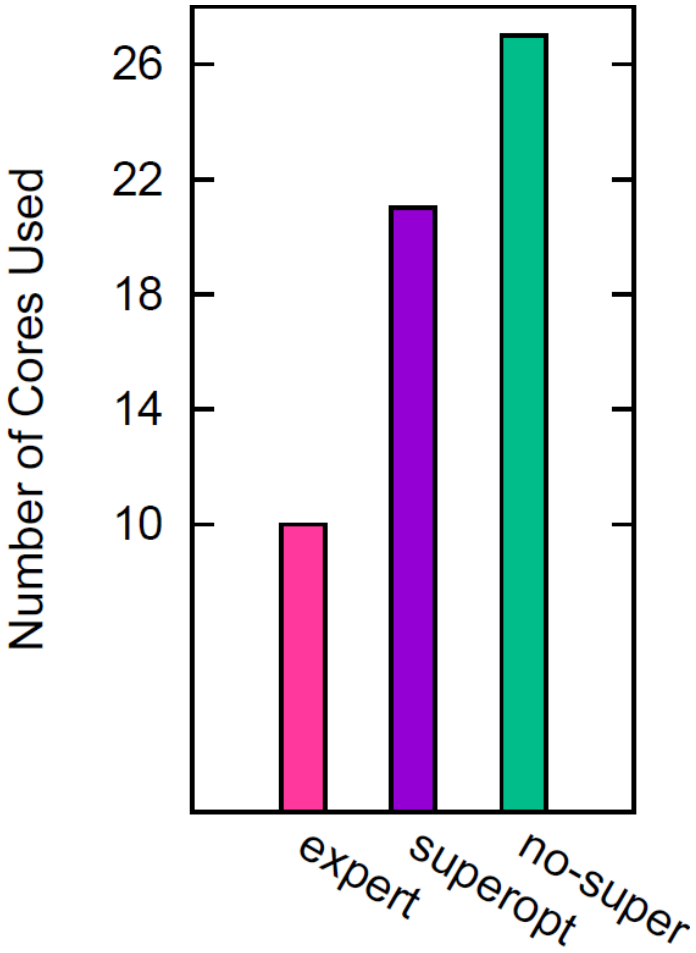
```
: 1rep 16 0 b! @b - 1 . + . + ;
: 1if
  .. -if
    2 b! @b 3 b! !b 1 b! @b 2 b! !b
    3 b! @b 1 b! !b 0 b! @b 16
    - 1 . + . + 0 b! !b ; ] then ;
: leftrotate 0 a! !+ !+ !+
  1rep 1if 1 b! @b 1rep
  .. if -1 . +
    for 2/ unext dup
  then drop 2 b! @b 0 b! @b
  .. if -1 . +
    for 2* unext dup
  then drop over - and . + 65535 and ;
```

Superoptimized code

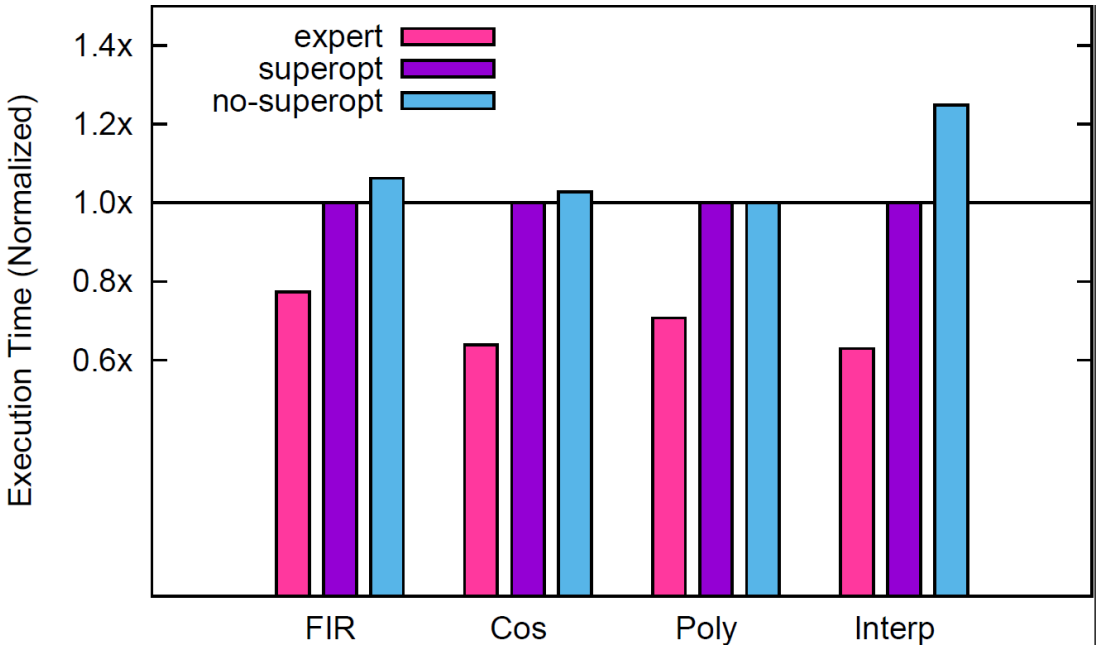
```
: 1rep dup dup or a! @ 262127 . + - ;
: 1if
  .. -if
    2 b! @b 1 a! @+ !+ !+
    dup dup or a! @+ 3 b! @b !+ 16
    - 1 . + dup dup or b! . + !b ; ] then ;
: leftrotate dup dup or a! !+ !+ !+
  1rep 1if 1 b! @b 1rep
  .. if -1 . +
    for 2/ unext dup
  then dup or b! 2 a! @ @b
  .. if -1 . +
    for 2* unext dup
  then drop over - and . + 65535 and ;
```

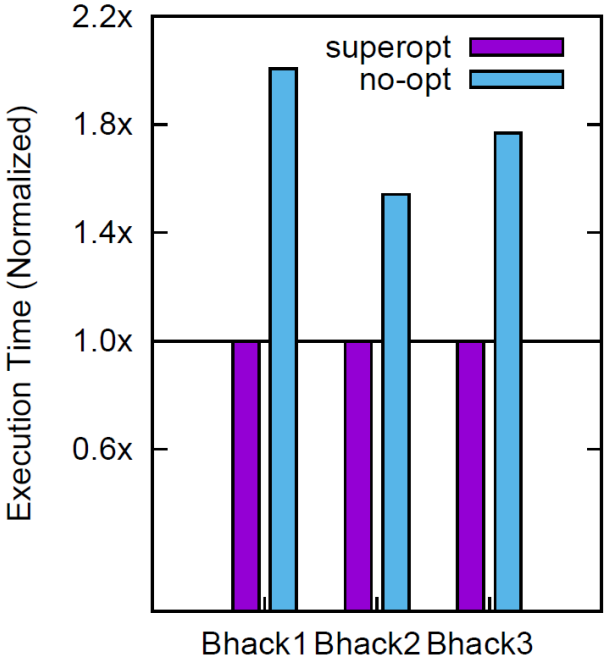
MD5


Execution Time (Normalized)
2.0x
1.8x
1.6x
1.4x
1.2x
1.0x
expert
superopt
no-super

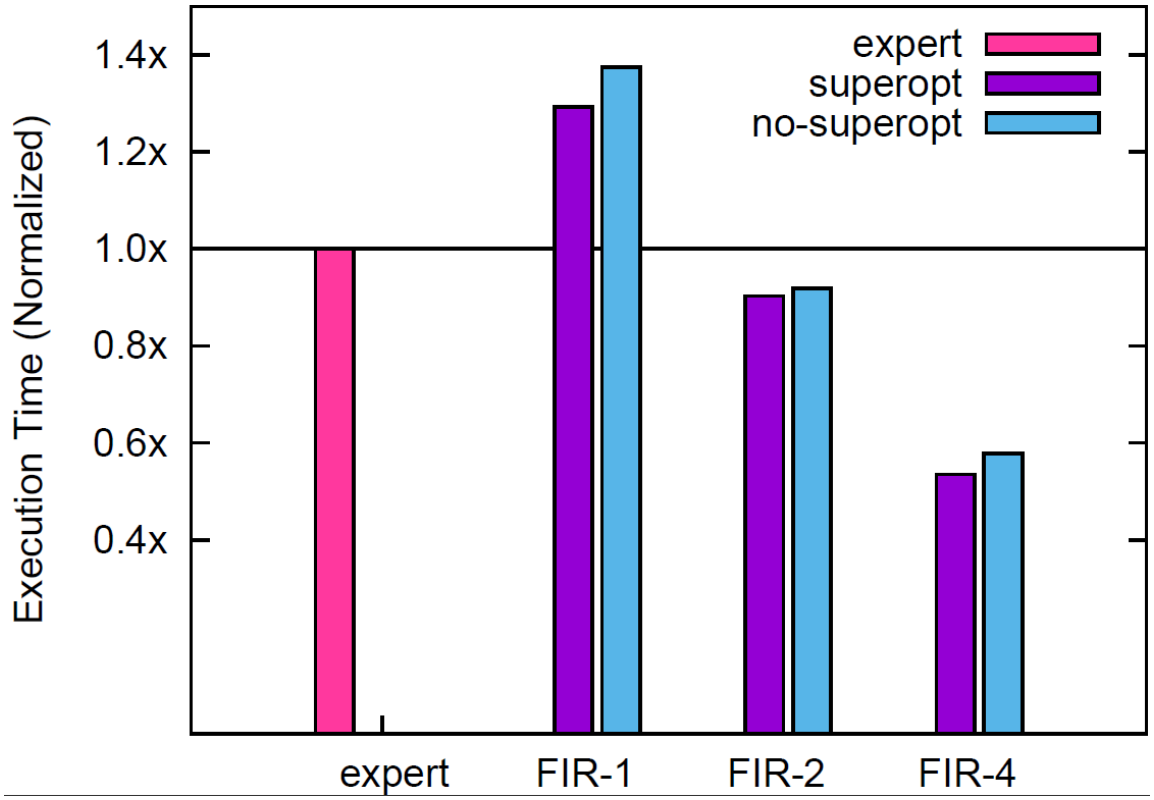

Number of Cores Used
26
22
18
14
10
expert
superopt
no-super

Single-Core Programs





FIR


Execution Time (Normalized)
1.4x
1.2x
1.0x
0.8x
0.6x
0.4x
expert
superopt
no-superopt
expert
FIR-1
FIR-2
FIR-4