2025全国大学生系统能力大赛编译系统设计赛（华为毕昇杯）技术培训会（第十场）

结构化并行编程模型及其实现机制

计卫星

北京师范大学 人工智能学院

2025/7/19

报告提纲

- 并行优化赛题解析
- 串行语言并行扩展
- 编程语言与结构化并行

并行优化赛题解析

主要任务

- 在已有SysY2022语言的基础上，参考OpenMP、Cilk或其他进行扩展
- 基于扩展后的语言完成编译器的设计和实现
- 设计测试用例验证和展示优化效果

具体要求

- 参赛队自行选择扩展的方向和并行优化技术，针对 ARM平台完成编译器的设计与实现，提交优化技术报告和编译器源码等相关文档
- 各参赛队自行决定所采用的优化技术等细节

并行优化赛题解析

评分标准

- 参赛队可以针对改进后的SysY语言重新设计实现编译器，或基于往届比赛的优秀作品、其他已开源编译器进行增量改进，使用往届优秀作品的参赛队，须在提交文档中明确说明增量工作内容，并提供原参赛队的合理授权说明。
- 参赛队应自行编写测试用例，尽可能从不同规模、不用应用场景展示新语言的使用方法，以及所完成的并行优化技术的性能优势。参赛队也可以对编译器设计赛道的公开测例进行改造后使用。
- 参赛队应提交优化技术报告和编译器源码，技术报告包括文法扩展内容、编译器实现方法、并行优化技术、测试用例和优化效果等。
- 本赛题的成绩由专家组根据参赛队提交的文档和答辩情况综合打分评定，考虑的因素包括但不限于提交作品的技术创新性、优化技术报告、技术方案及优化效果、参赛队答辩情况等。

报告提纲

北京师范大学 人工智能学院

- 并行优化赛题解析
- 串行语言并行扩展
- 编程语言与结构化并行

串行语言并行扩展

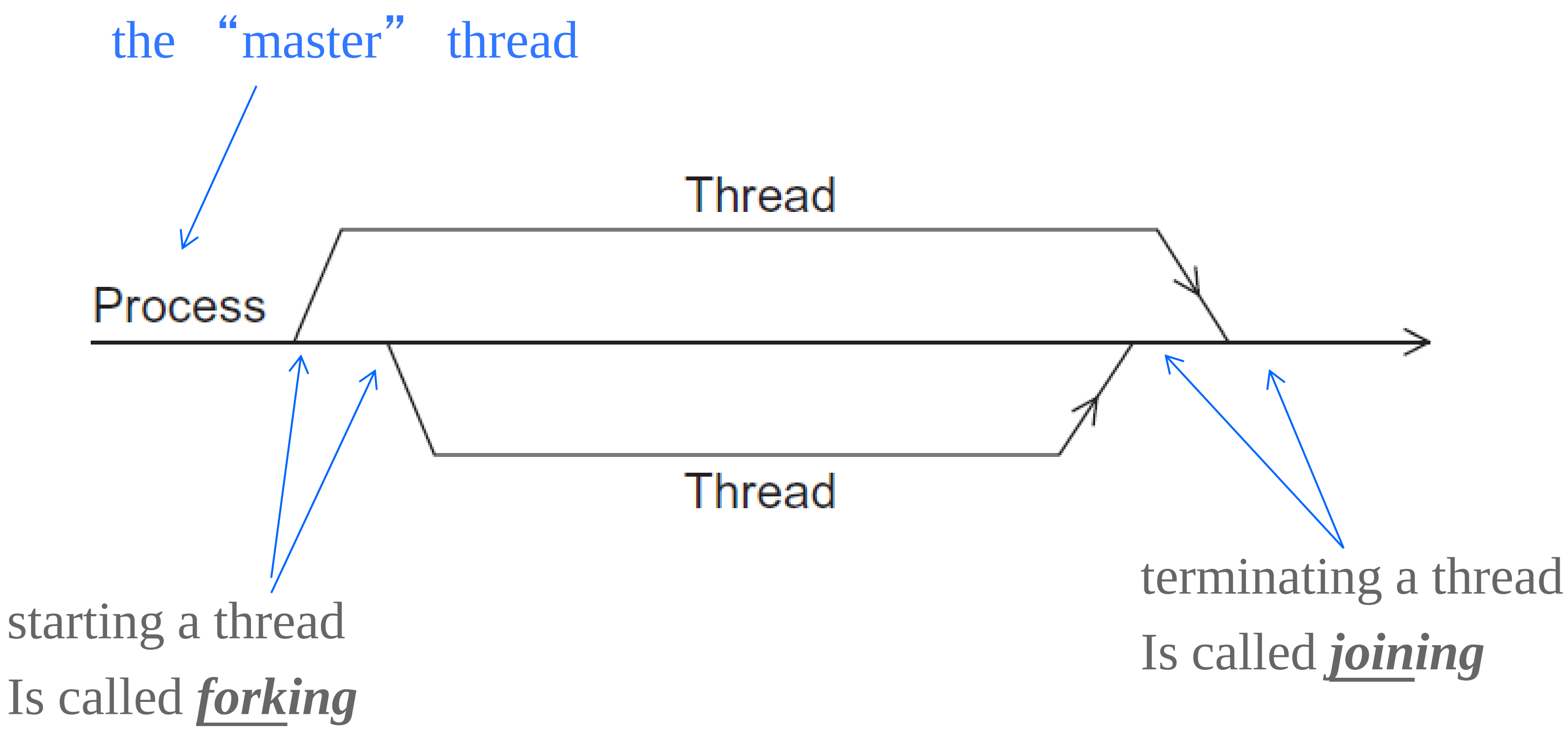
- 以库的方式扩展：POSIX Threads

```c
int main(void) {
  pthread_t threads[NUM_THREADS];
  int thread_args[NUM_THREADS];
  int i;
  int result_code;

  //create all threads one by one
  for (i = 0; i < NUM_THREADS; i++) {
    printf("In main: Creating thread %d.\n", i);
    thread_args[i] = i;
    result_code = pthread_create(&threads[i], NULL, perform_work, &thread_args[i]);
    assert(!result_code);
  }

  printf("In main: All threads are created.\n");

  //wait for each thread to complete
  for (i = 0; i < NUM_THREADS; i++) {
    result_code = pthread_join(threads[i], NULL);
    assert(!result_code);
    printf("In main: Thread %d has ended.\n", i);
  }

  printf("Main program has ended.\n");
  return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <pthread.h>
#include <unistd.h>

#define NUM_THREADS 5

void *perform_work(void *arguments){
  int index = *((int *)arguments);
  int sleep_time = 1 + rand() % NUM_THREADS;
  printf("Thread %d: Started.\n", index);
  printf("Thread %d: Will be sleeping for %d seconds.\n", index, sleep_time);
  sleep(sleep_time);
  printf("Thread %d: Ended.\n", index);
  return NULL;
}
```

串行语言并行扩展

- 以库的方式扩展：POSIX Threads



串行语言并行扩展

- 以Directive的方式扩展：OpenMP

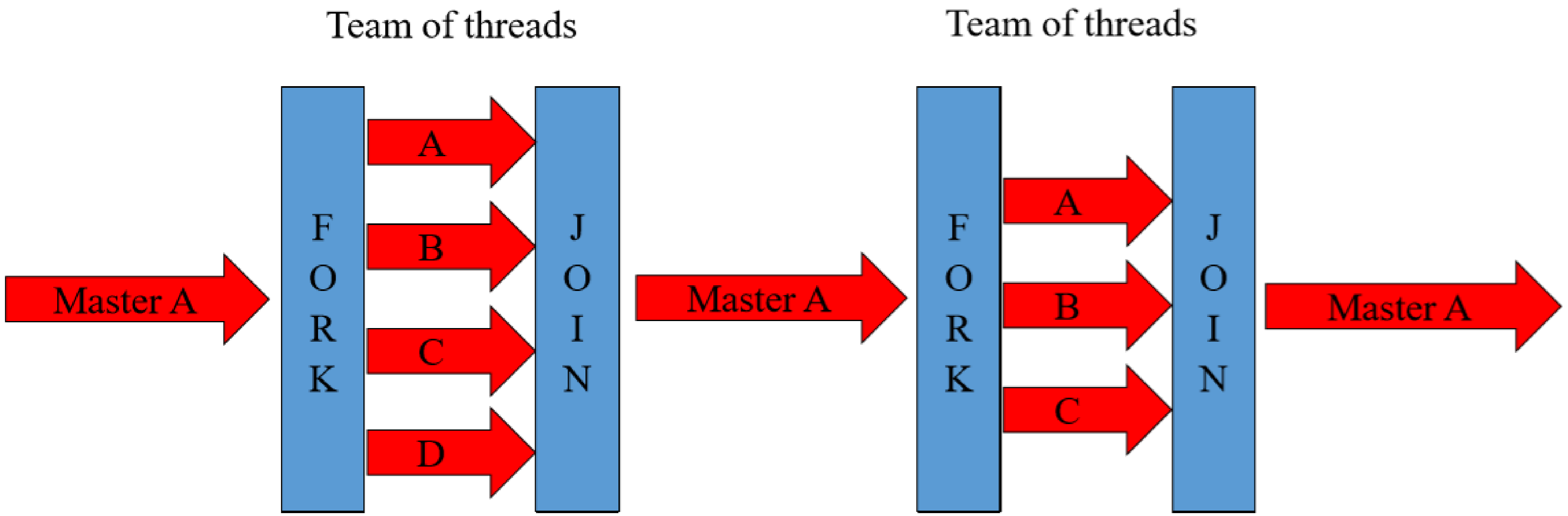
```
$ gcc -fopenmp hello.c -o hello
```

```c
#include <stdio.h>
#include <omp.h>

int main(void)
{
    #pragma omp parallel
    printf("Hello, world.\n");
    return 0;
}
```

```c
int main(int argc, char **argv)
{
    int a[100000];

    #pragma omp parallel for
    for (int i = 0; i < 100000; i++) {
        a[i] = 2 * i;
    }

    return 0;
}
```

串行语言并行扩展

- 以Directive的方式扩展：OpenMP



串行语言并行扩展

- 语言扩展：Cilk, Cilk++, Cilk Plus and OpenCilk

```
cilk int fib(int n) {
    if (n < 2) {
        return n;
    }
    else {
        int x, y;

        x = spawn fib(n - 1);
        y = spawn fib(n - 2);

        sync;

        return x + y;
    }
}
```

串行语言并行扩展

- 语言扩展：MATLAB

```
A = [1 1 0 0];
B = [1; 2; 3; 4];
```

```
C = B*A
```

```
C = 4×4

     1     1     0     0
     2     2     0     0
     3     3     0     0
     4     4     0     0
```

```
A = [1 3 5; 2 4 7];
B = [-5 8 11; 3 9 21; 4 0 8];
```

```
C = A*B
```

```
C = 2×3

    24    35   114
    30    52   162
```

串行语言并行扩展

- 语言扩展： **Python NumPy**

```
>>> a = np.array([[1, 0],
...               [0, 1]])
>>> b = np.array([[4, 1],
...               [2, 2]])
>>> np.matmul(a, b)
array([[4, 1],
       [2, 2]])
```

串行语言并行扩展

- 语言扩展： **Python NumPy**

numpy.dot

`numpy.dot(a, b, out=None)`

Dot product of two arrays. Specifically,

- If both *a* and *b* are 1-D arrays, it is inner product of vectors (without complex conjugation).
- If both *a* and *b* are 2-D arrays, it is matrix multiplication, but using `matmul` or `a @ b` is preferred.
- If either *a* or *b* is 0-D (scalar), it is equivalent to `multiply` and using `numpy.multiply(a, b)` or `a * b` is preferred.
- If *a* is an N-D array and *b* is a 1-D array, it is a sum product over the last axis of *a* and *b*.
- If *a* is an N-D array and *b* is an M-D array (where `M>=2`), it is a sum product over the last axis of *a* and the second-to-last axis of *b*:

```
dot(a, b)[i,j,k,m] = sum(a[i,j,:] * b[k,:,m])
```

It uses an optimized BLAS library when possible (see `numpy.linalg`).

dot

numpy.tensordot

`numpy.tensordot(a, b, axes=2)` [source]

Compute tensor dot product along specified axes.

Given two tensors, *a* and *b*, and an array_like object containing two array_like objects, `(a_axes, b_axes)`, sum the products of *a*'s and *b*'s elements (components) over the axes specified by `a_axes` and `b_axes`. The third argument can be a single non-negative integer_like scalar, `N`; if it is such, then the last `N` dimensions of *a* and the first `N` dimensions of *b* are summed over.

Parameters: **a, b** : *array_like*

Tensors to "dot".

axes : *int or (2,) array_like*

- integer_like If an int N, sum over the last N axes of *a* and the first N axes of *b* in order. The sizes of the corresponding axes must match.
- (2,) array_like Or, a list of axes to be summed over, first sequence applying to *a*, second to *b*. Both elements array_like must be of the same length.

Returns: **output** : *ndarray*

The tensor dot product of the input.

报告提纲

北京师范大学
人工智能学院

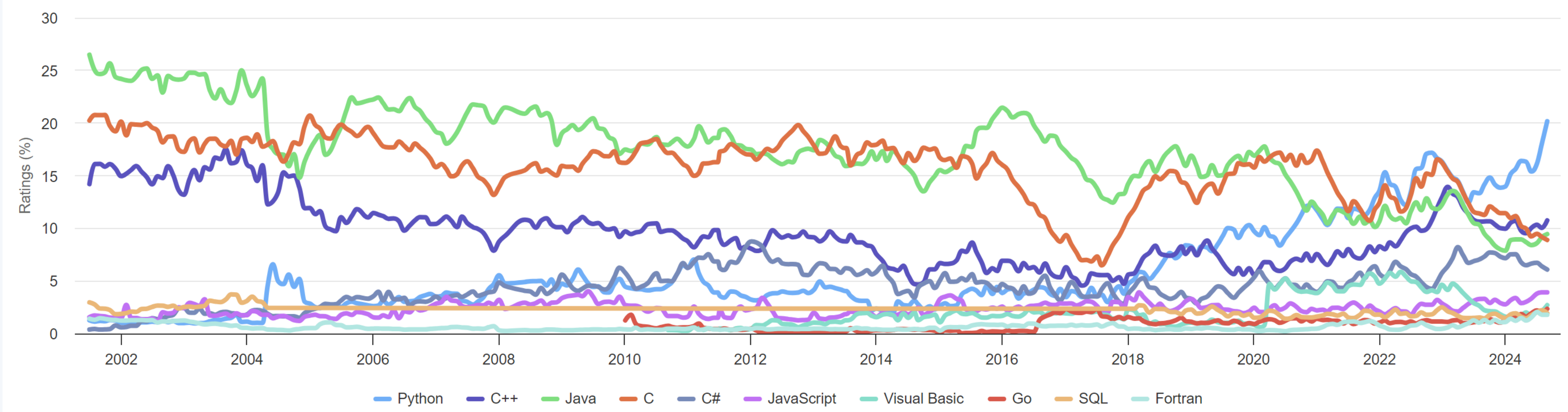
并行优化赛题解析

串行语言并行扩展

编程语言与结构化并行

编程语言与结构化并行

Sep 2024	Sep 2023	Change	Programming Language	Ratings	Change
1	1		Python	20.17%	+6.01%
2	3	^	C++	10.75%	+0.09%
3	4	^	Java	9.45%	-0.04%
4	**2**	v	**C**	**8.89%**	**-2.38%**
5	5		C#	6.08%	-1.22%



https://www.tiobe.com/tiobe-index/

编程语言与结构化并行

结构化

```c
void bfs_dump_imap(const char *prefix, struct super_block *s)
{
#ifdef DEBUG
	int i;
	char *tmpbuf = (char *)get_zeroed_page(GFP_KERNEL);

	if (!tmpbuf)
		return;
	for (i = BFS_SB(s)->si_lasti; i >= 0; i--) {
		if (i > PAGE_SIZE - 100) break;
		if (test_bit(i, BFS_SB(s)->si_imap))
			strcat(tmpbuf, "1");
		else
			strcat(tmpbuf, "0");
	}
	printf("%s: lasti=%08lx <%s>\n", prefix, BFS_SB(s)->si_lasti, tmpbuf);
	free_page((unsigned long)tmpbuf);
#endif
}
```

非结构化

```c
static int __init init_bfs_fs(void)
{
	int err = init_inodecache();
	if (err)
		goto out1;
	err = register_filesystem(&bfs_fs_type);
	if (err)
		goto out;
	return 0;
out:
	destroy_inodecache();
out1:
	return err;
}
```

https://github.com/torvalds/linux/blob/master/fs/bfs/inode.c

编程语言与结构化并行

北京师范大学 人工智能学院

结构化并行

```
int fib (int n) {
    if (n < 2) {
        return n;
    } else {
        int x, y;
        x = fib(n − 1);
        y = fib(n − 2);
        return x + y;
    }
}
```

```
int fib (int n) {
    if (n < 2) {
        return n;
    } else {
        int x, y;
        x = cilk_spawn fib(n − 1);
        y = fib(n − 2);
        cilk_sync;
        return x + y;
    }
}
```

```
cilk int fib (int n) {
    if (n < 2) return 1;
    else {
        int rst = 0;
        rst += spawn fib (n-1);
        rst += spawn fib (n-2);
        sync;
        return rst;
    }
}
```

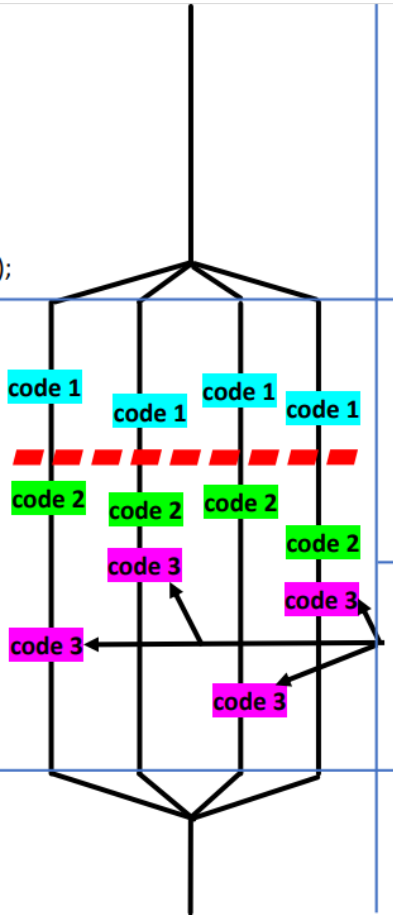
编程语言与结构化并行

非结构化

```
pthread_mutex_t mutex;
pthread_mutex_init(&mutex, NULL);

pthread_barrier_t barrier;
pthread_barrier_init(&barrier, NULL, num_threads);

pthread_t thread;
pthread_create(&thread, NULL, threadFunction, arg);
```

```
void * threadFunction(void* arg)
{
        code 1
        pthread_barrier_wait(&barrier);
        code 2

        pthread_mutex_lock(&mutex);
        code 3
        pthread_mutex_unlock(&mutex);
}
```

```
pthread_join(thread, NULL);
pthread_barrier_destroy(&barrier);
pthread_mutex_destroy(&mutex);
```



#include <pthread.h>

PThread Cheat Sheet

Made by Cristian Chilipirea

Any code 2 must execute **after all** code 1

When code 3 starts execution it must end before it can start on a different thread

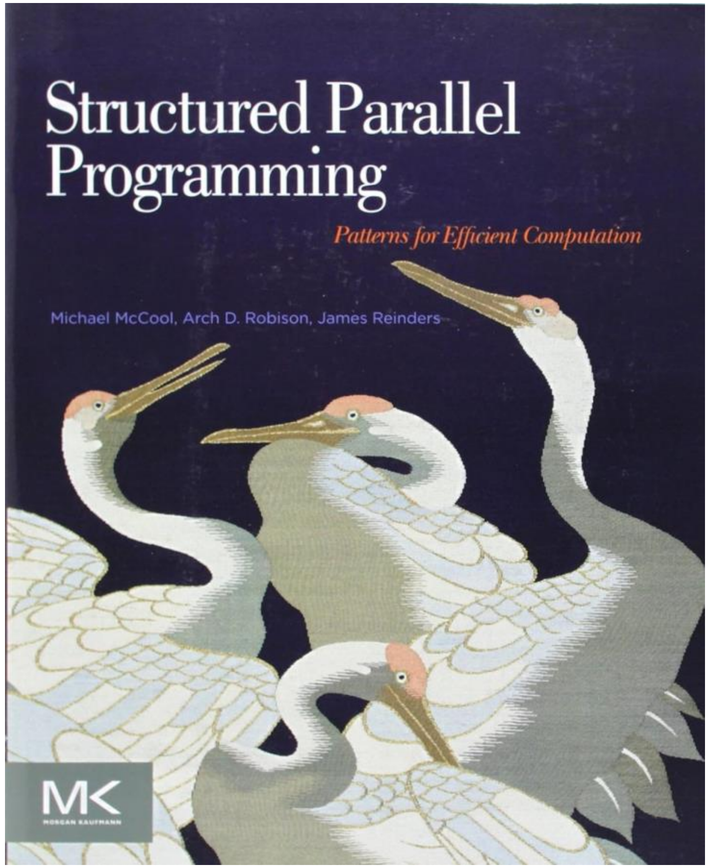
Compiling:
gcc main.c –fpthread
Number of cores:
cat /proc/cpuinfo

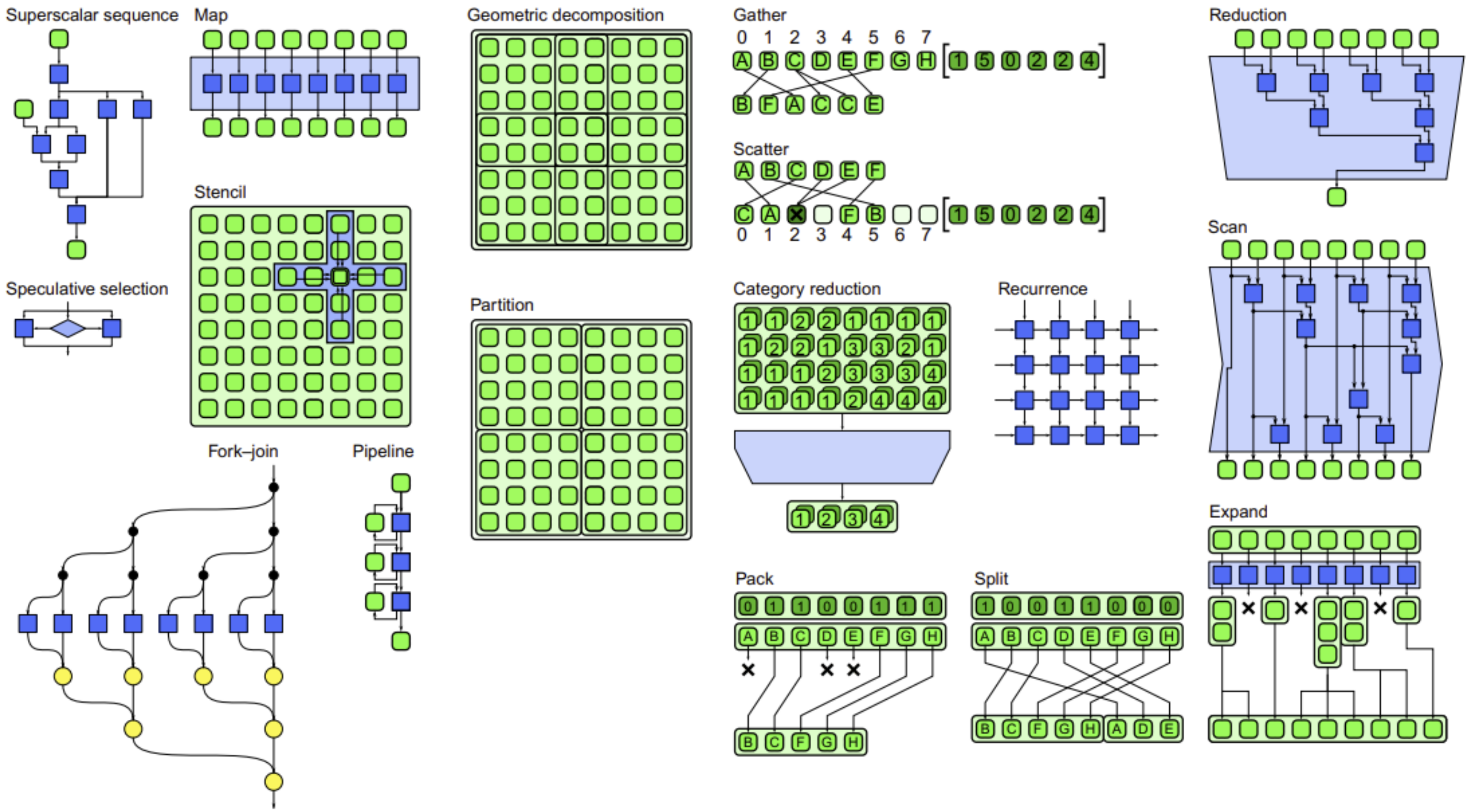
结构化

```
cilk int fib (int n) {
   if (n < 2) return 1;
   else {
      int rst = 0;
      rst += spawn fib (n-1);
      rst += spawn fib (n-2);
      sync;
      return rst;
   }
}
```

编程语言与结构化并行

结构化并行



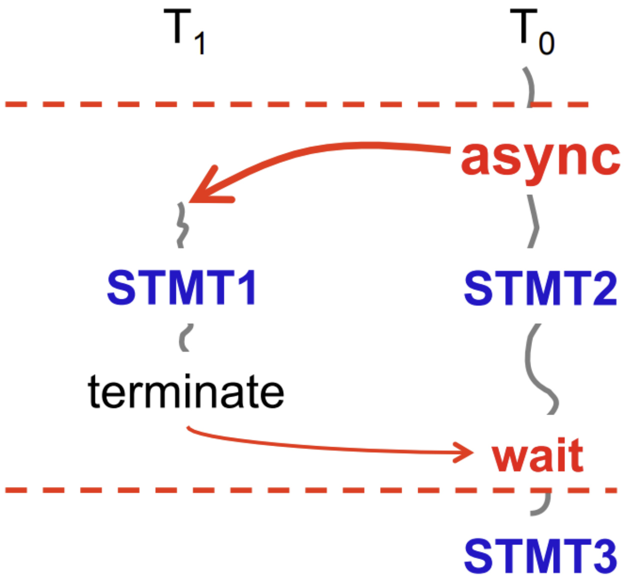


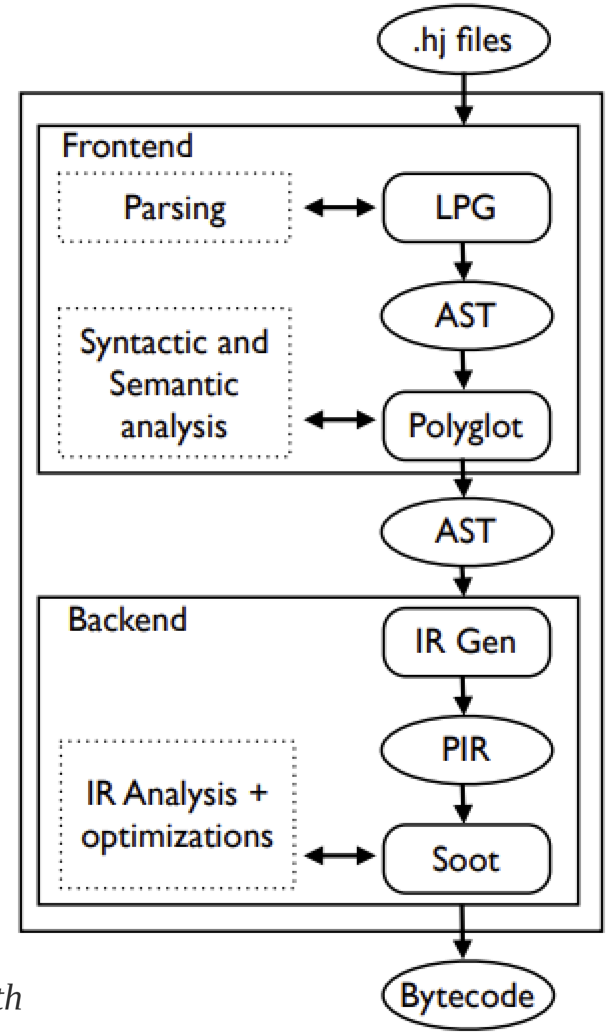
Efficient parallel algorithms using a composable, structured, scalable, and machine-independent approach to parallel computing.

编程语言与结构化并行

Habanero-Java

```
//Task T0(Parent)
finish {     //Begin finish
  async
    STMT1;  //T1(Child)
  //Continuation
  STMT2;     //T0
} //Continuation //End finish
STMT3;       //T0
```





Vincent Cavé, Jisheng Zhao, et al. Habanero-Java: the new adventures of old X10. In Proceedings of the 9th International Conference on Principles and Practice of Programming in Java (PPPJ '11)

编程语言与结构化并行

Deterministic Parallel Ruby

```
def quickSort(a)
  return a if a.length <= 1
  m = a.length/2
  pivot = a[m]
  left = mid = right = nil
  co ->{left = quickSort(a.map{|v| v <  pivot?v:nil}.compact)},
     ->{mid  =           a.map{|v| v == pivot?v:nil}.compact)},
     ->{right= quicksort(a.map{|v| v >  pivot?v:nil}.compact)}
  return left + mid + right
end
```

https://github.com/RB-DPR/RB-DPR

编程语言与结构化并行

Deterministic Parallel Ruby

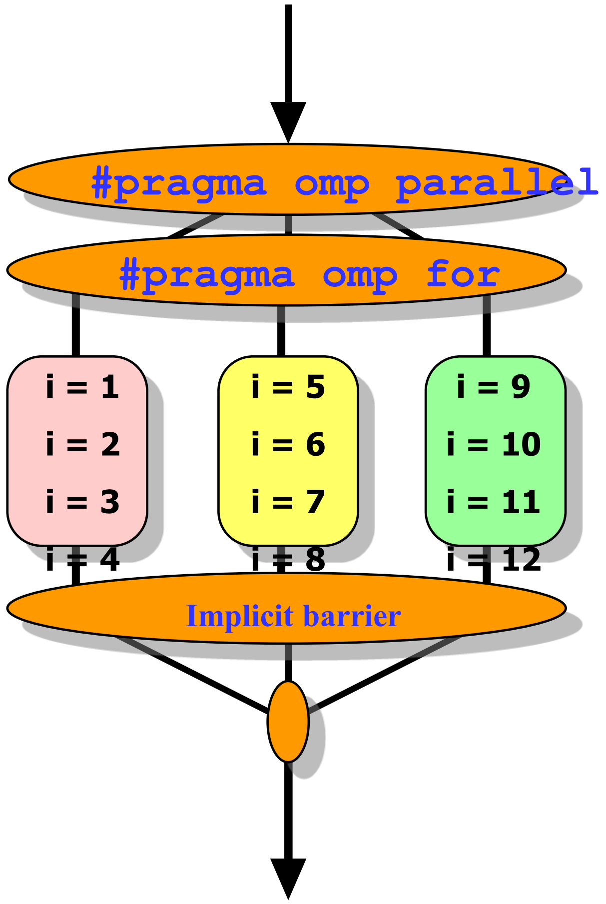
```
a=[1,2,3,4]
b=[4,3,2,1]
(0...a.size).each{ |i|
  a[i] = a[i] + b[i]
}
```

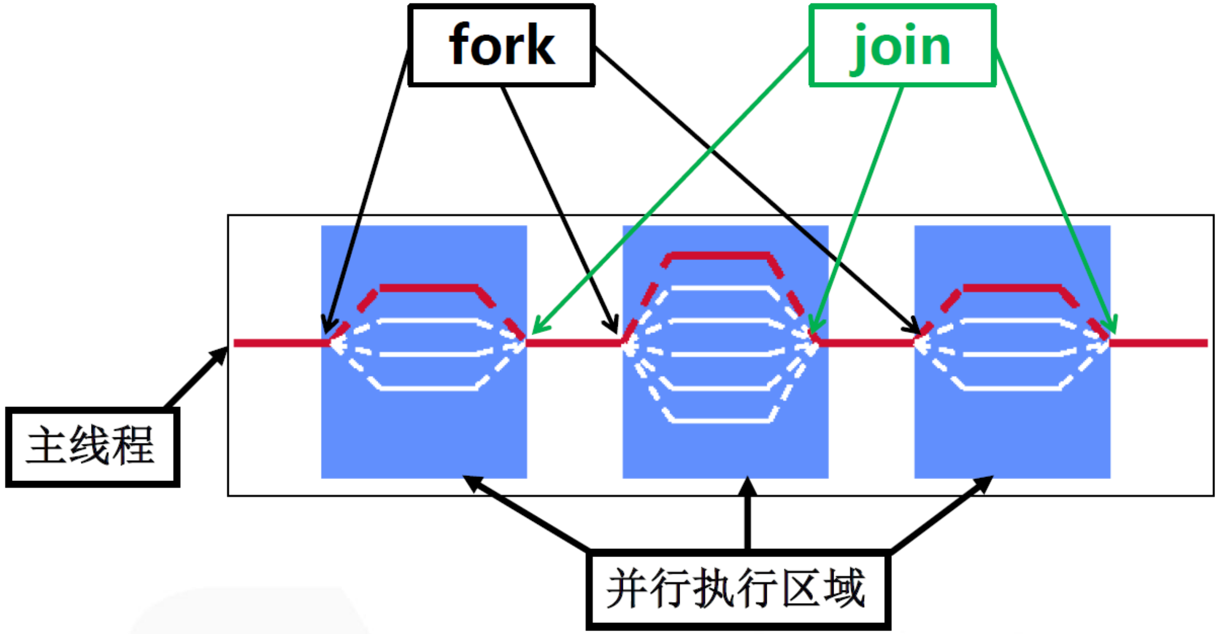
```
a=[1,2,3,4]
b=[4,3,2,1]
(0...a.size).all{ |i|
  a[i] = a[i] + b[i]
}
```

https://github.com/RB-DPR/RB-DPR

编程语言与结构化并行

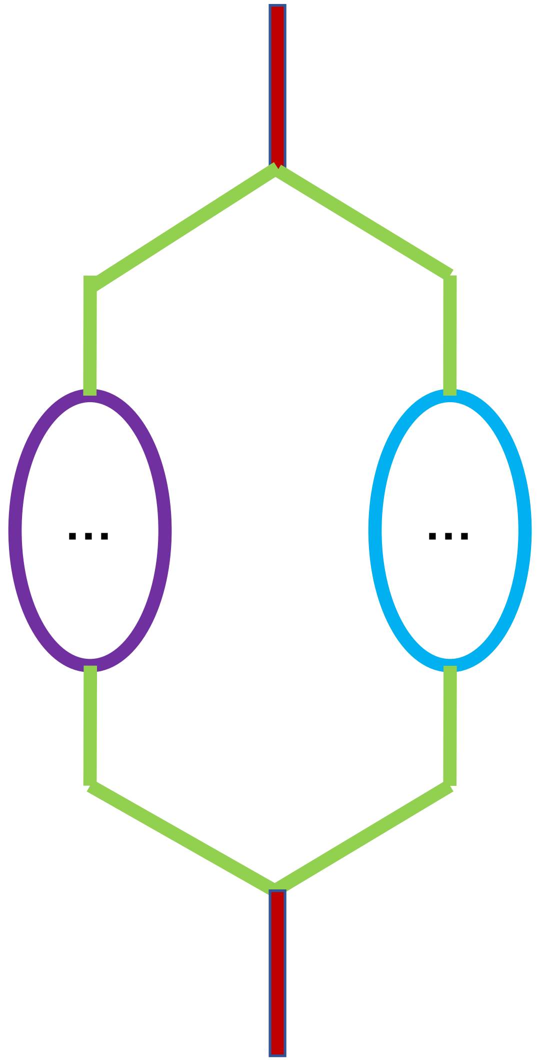
OpenMP





Ruby结构化并行及实现

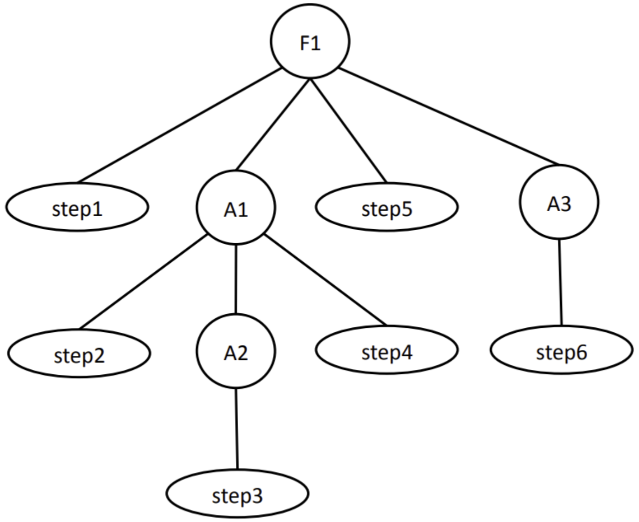
```
class Book
     attr :name
     attr :price
     …
end
…
ob = Array.new(100)  #old books
pb = Array.new(200)  #popular books
…
co ->{
     (0...100).all {|i|
       ob[i].price *= 0.8
     }
  },
  ->{
     (0...200).all {|j|
       pb[j].price *= 1.2
     }
  }
…
```



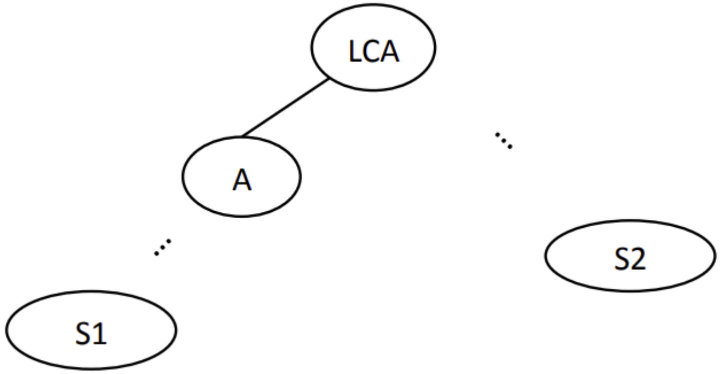
Ruby结构化并行及实现

Dynamic Program Structure Tree (DPST)

```
finish { // F1
  S1;
  S2;      } step1
  async { // A1
    S3;
    S4;    } step2
    S5;
    async { // A2
      S6; } step3
    } // async A2
    S7;
    S8;    } step4
  } // async A1
  S9;
  S10;     } step5
  S11;
  async { // A3
    S12;
    S13;   } step6
  } // async A3
} // finish F1
```



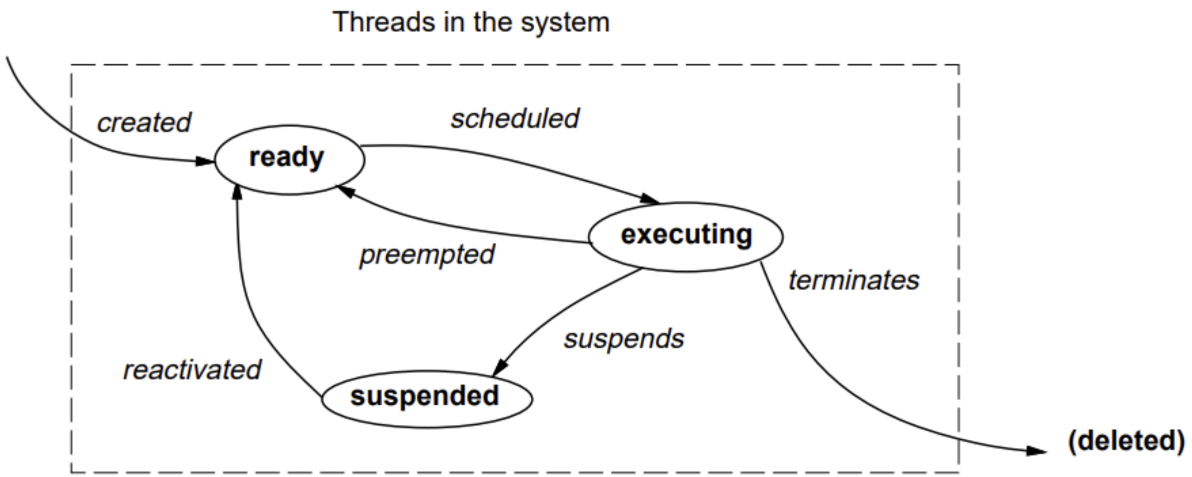
Lowest Common Ancestor (LCA)

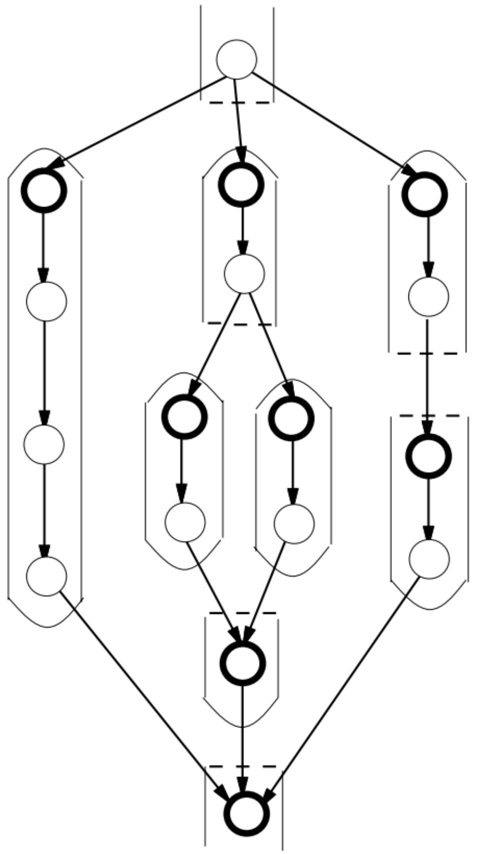


Raghavan Raman, Jisheng Zhao, Vivek Sarkar et al. Scalable and precise dynamic data race detection for structured parallelism. PLDI

Ruby结构化并行及实现


```
In parallel for i = 1 to n
   Temporary B[n]
   In parallel for j = 1 to n
      F(B,i,j)
   Free B
```





Girija J. Narlikar and Guy E. Blelloch. Space-efficient scheduling of nested parallelism. ACM Trans. Program. Lang. Syst.

并行IR及实现

- 目前中间语言缺少关于并行语义的描述
  - 并行在编程语言层面已经有体现
  - 中间语言层面缺少并行语义描述的问题
    - 已有适用于串行程序的优化技术无法适用于并行程序
    - 并行程序中的串行部分优化也受到影响
  - 并行中间语言主要用于
    - 捕获并行语义（并行循环）
    - 描述变量属性（共享/私有）
    - 并行同步抽象

并行IR及实现

序号	文献/工具	基础编译器	IR表示	优化方法	是否开源
1	Tapir	LLVM	在LLVM IR基础上新增指令detach、reattach和sync	公共子表达式消除、循环不变代码外提、尾递归消除、并行循环调度、不必要的同步删除和微小任务消除	https://github.com/wsmoses/Tapir-Meta.git
2	HPVM	LLVM	HPVM IR	将数据映射到GPU常量内存、内存Tiling、LLVM相关优化	https://gitlab.engr.illinois.edu/llvm/hpvm-release/-/releases
3	Trireme	LLVM	LLVM IR、HPVM IR	基于已有的LLVM和HPVM优化方法	否
4	LIFT	LIFT	LIFT IR	Barrier消除、控制流图简化	https://gitlab.com/michel-steuwer/cgo_2017_artifact
5	ApproxHPVM	LLVM	ApproxHPVM IR	HPVM优化、精确感知的动态调度调优方法	https://gitlab.engr.illinois.edu/llvm/hpvm-release/-/releases
6	MLIR	LLVM	MLIR	基于应用中编译器的优化方法	https://mlir.llvm.org
7	LLVM IR并行扩展	LLVM	LLVM IR	基于LLVM已有优化	否
8	HPVM2FPGA	LLVM	HPVM IR	自动输入缓冲、自动 ivdep 插入、循环展开、贪婪循环融合、自动节点融合、自动任务并行	https://gitlab.engr.illinois.edu/llvm/hpvm-release/-/releases

并行IR及实现

现有工作—Tapir

Tapir在LLVM编译器的IR中添加了三个指令——detach、reattach和sync，以实现在程序的控制流图中非对称地表示逻辑上并行的任务。

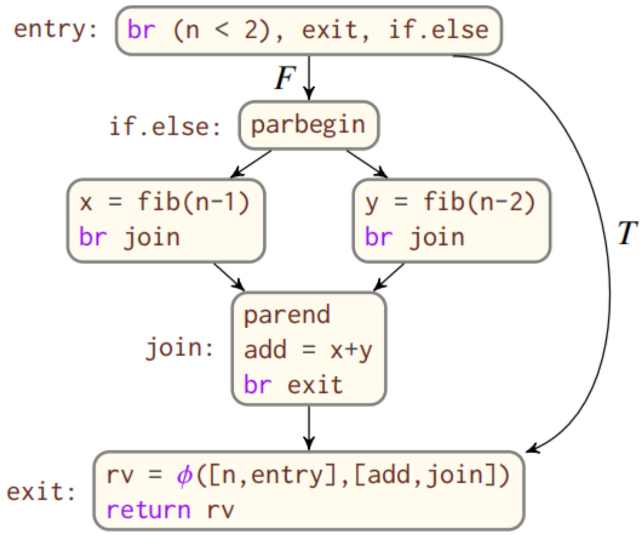
a

```c
int fib(int n) {
  if (n < 2) return n;
  int x, y;
  x = cilk_spawn fib(n - 1);
  y = fib(n - 2);
  cilk_sync;
  return x + y;
}
```

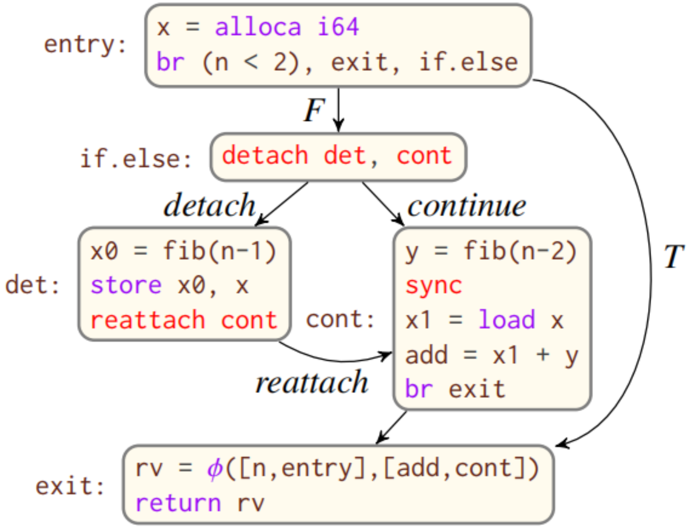
b

```c
int fib(int n) {
  if (n < 2) return n;
  int x, y;
  #pragma omp task shared(x)
  x = fib(n - 1);
  #pragma omp task shared(y)
  y = fib(n - 2);
  #pragma omp taskwait
  return x + y;
}
```

c



d



带有非对称并行的Tapir CFG与传统CFG对比

[1] Schardl et al. Tapir: Embedding fork-join parallelism into LLVM's intermediate representation. PPoPP 2017: 249-265.

并行IR及实现

- 大部分工作基于LLVM IR，对其他IR的并发和优化研究较少
  - 很多方法基于LLVM IR进行扩展
- 对并行模式的支持有限
  - Tapir仅支持fork-join模式，不支持Map、Reduce并行等
  - LIFT仅支持GPU上的数据并行实现和优化
- 工具的可用性和可扩展性有待提升
  - ApproxHPVM仅用于机器学习和图像处理程序。
  - LIFT仅支持CPU架构，Trireme不支持GPU、DPU等架构

谢谢！