DiscoPoP: A Profiling, Analysis, and Visualization Tool for Parallelism Discovery

TECHNISCHE UNIVERSITÄT DARMSTADT

10th International Parallel Tools Workshop

Rohit Atre, Zia Ul-Huda, Mohammad Norouzi, Arya Mazaheri, Zhen Li,
Dr. Ali Jannesari, Prof. Felix Wolf

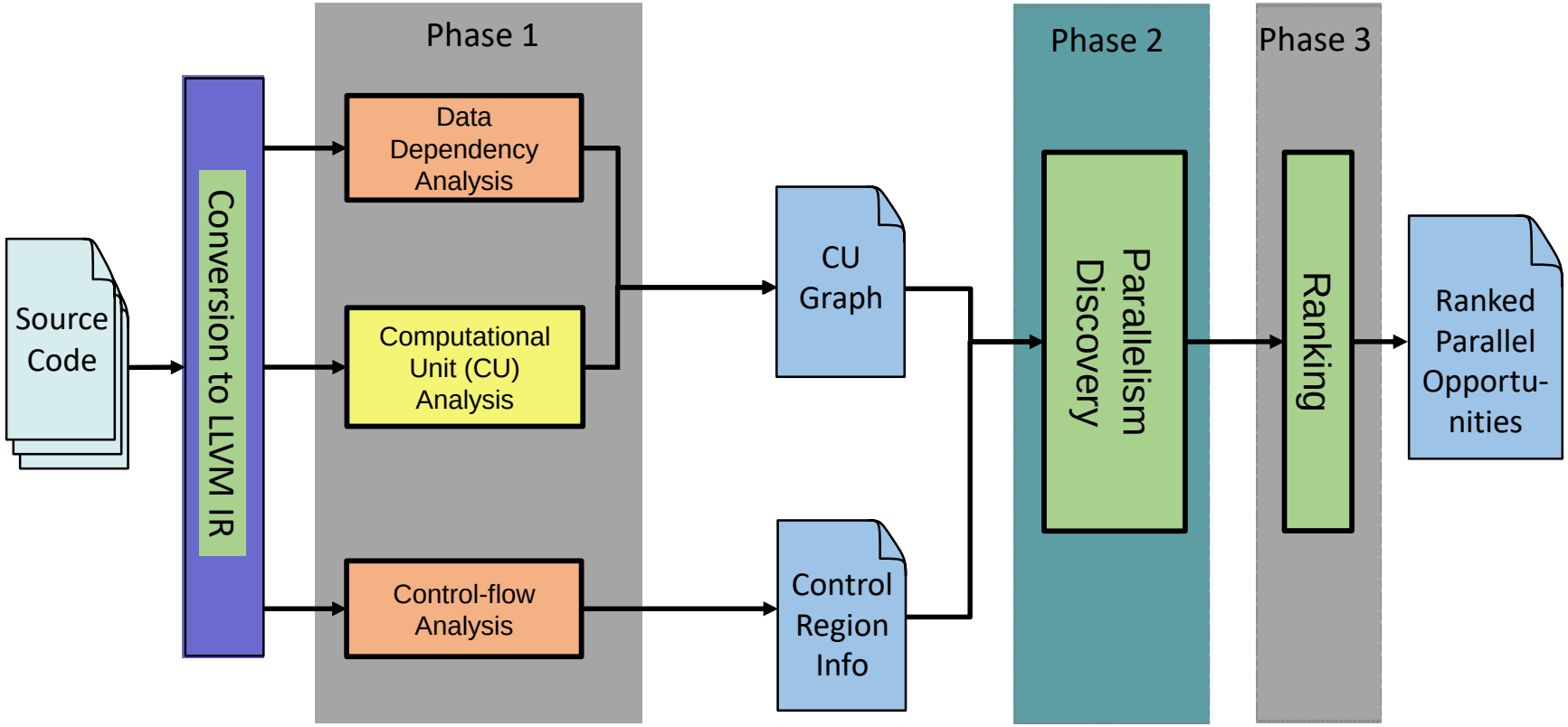
Introduction

TECHNISCHE UNIVERSITÄT DARMSTADT

- A large numbers of legacy programs need to be parallelized
- Transforming an existing sequential program into a parallel one is not easy
- DiscoPoP (Discovery of Potential Parallelism) is a tool to detect parallelism in sequential applications
  - Detects hotspots in the sequential applications
  - Gives hints to programmers: making parallelization process easy

DiscoPoP Workflow

TECHNISCHE UNIVERSITÄT DARMSTADT



Outline

TECHNISCHE UNIVERSITÄT DARMSTADT

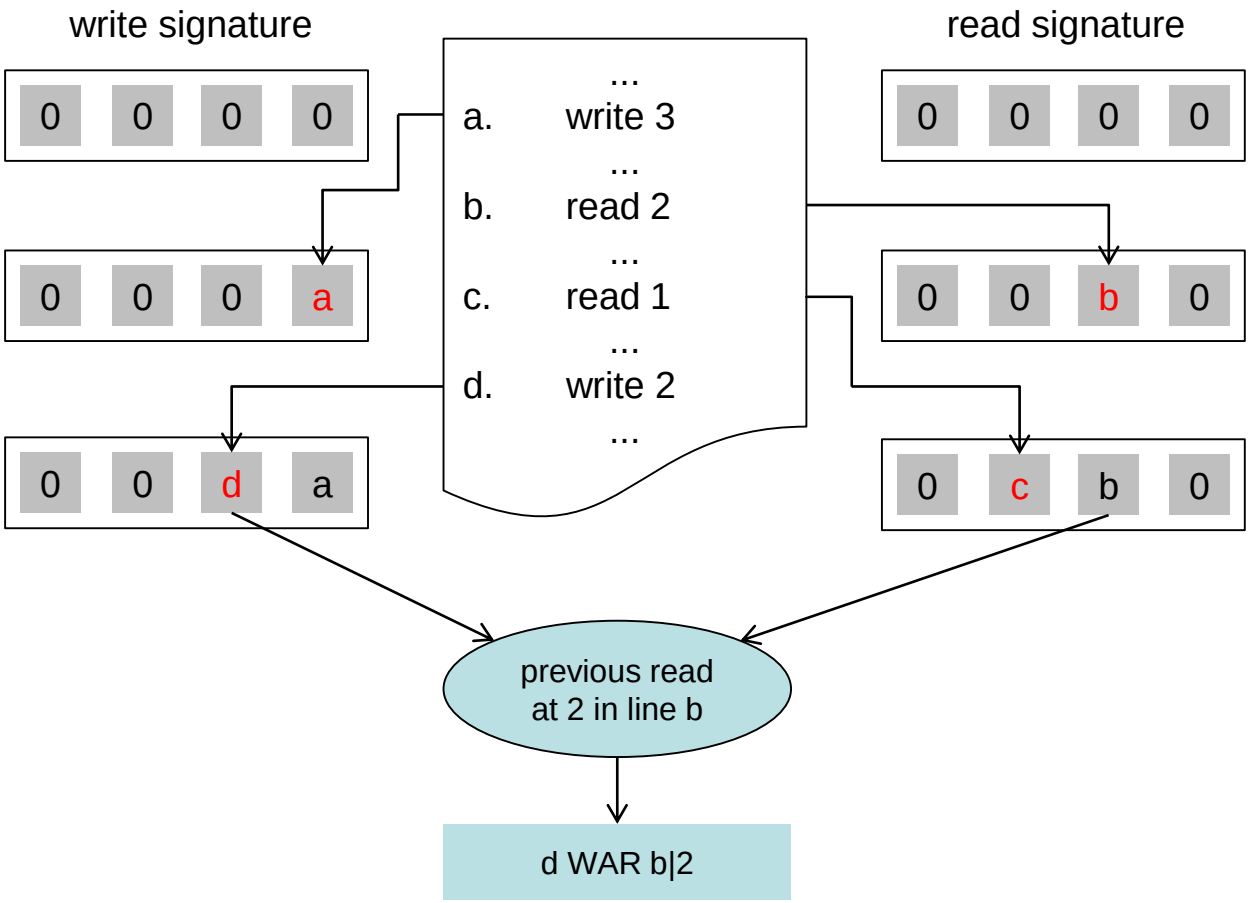
- Profiler
- Computational Units & Program graph
- Applications
- Evaluation
- Future Works

Profiling with signatures

TECHNISCHE UNIVERSITÄT DARMSTADT

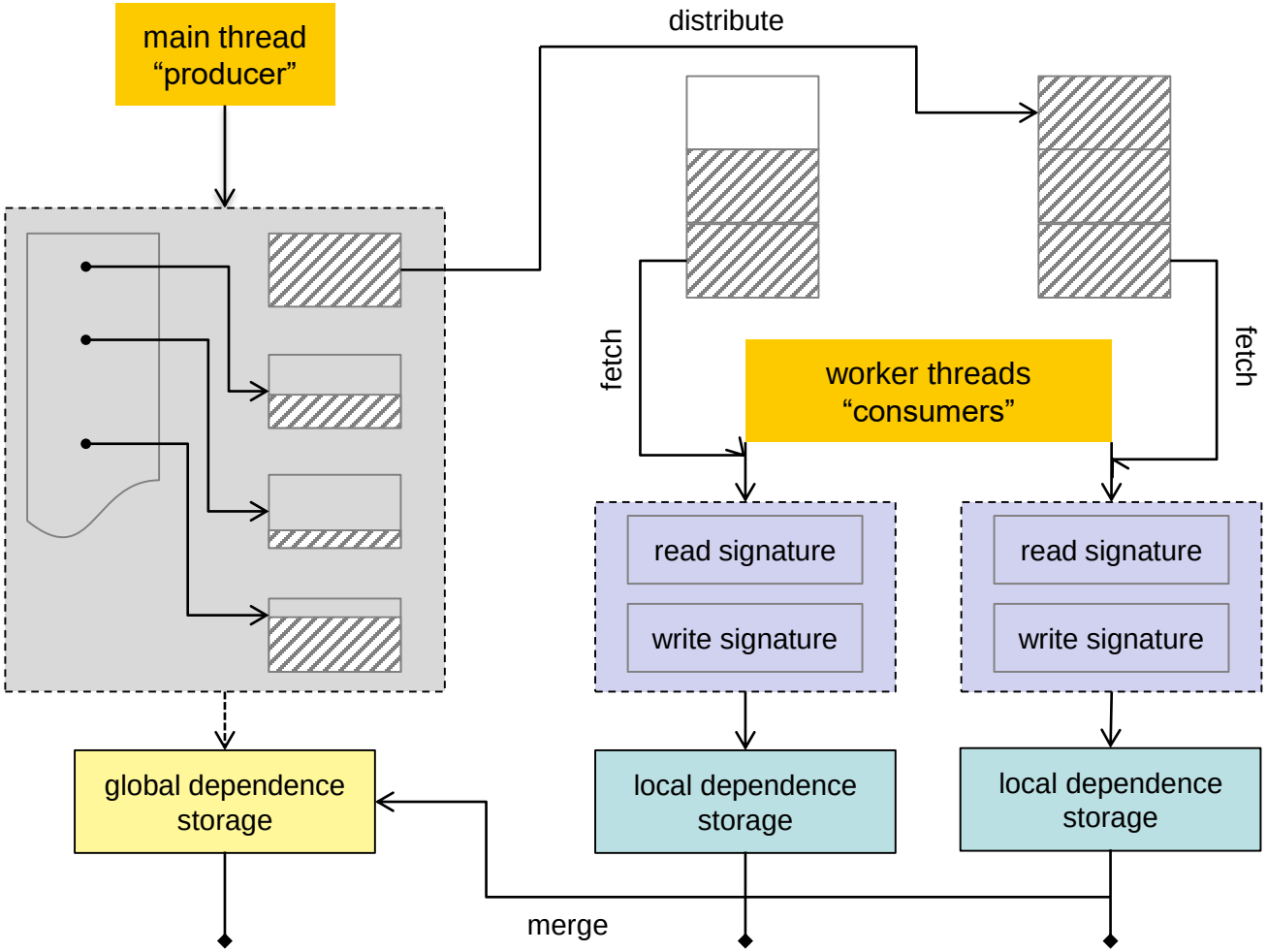
- A signature is usually implemented as a Bloom filter:
  - A fixed-size bit array
  - *k* different hash functions that together map an element to a number of array indices
- Two signatures: one for recording read operations and one for recording write operations

Profiling with signatures



Parallel data-dependence profiling

TECHNISCHE UNIVERSITÄT DARMSTADT



Profiling multithreaded programs

TECHNISCHE UNIVERSITÄT DARMSTADT

- Allow program analyses for multithreaded applications
  - Communication pattern detection
  - Scheduling
  - Performance tuning

Outline

TECHNISCHE UNIVERSITÄT DARMSTADT

- Profiler
- Computational Units & Program graph
- Applications
- Evaluation
- Future Works

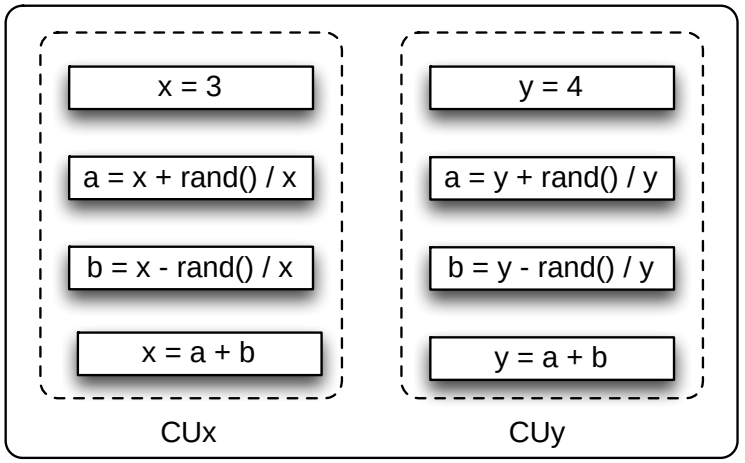
Computational Unit(CU)

TECHNISCHE UNIVERSITÄT DARMSTADT

- A collection of program statements
  - Follows the read-compute-write pattern

- Logical units to make larger tasks
  - Could be merged together
  - Assigned to threads
  - Building blocks for various patterns
    - Tasks in a taskpool
    - Stages of a pipeline

```
1       x = 3
2       y = 4
3       a = x + rand() / x
4       b = x -  rand() / x
5       x = a + b
6       a = y + rand() / y
7       b = y -  rand() / y
8       y = a + b
```



Computational Unit (CU) – Example

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1 int x = 3;
2 for (int i = 0; i < MAX_ITER; ++i) {
3 int a = x + rand() / x;
4 int b = x - rand() / x;
5 x = a + b;
6 }
```

Identify variables global to each region

Add the global variable **x** to **read-set** if it is read

Add the global variable **x** to the **write-set** when written

Region 1

- Every **read** on a **global variable** should **happen before** a corresponding **write** on it
- For every read instruction that violates, a new CU is created

Program Graph (Output of phase-1)

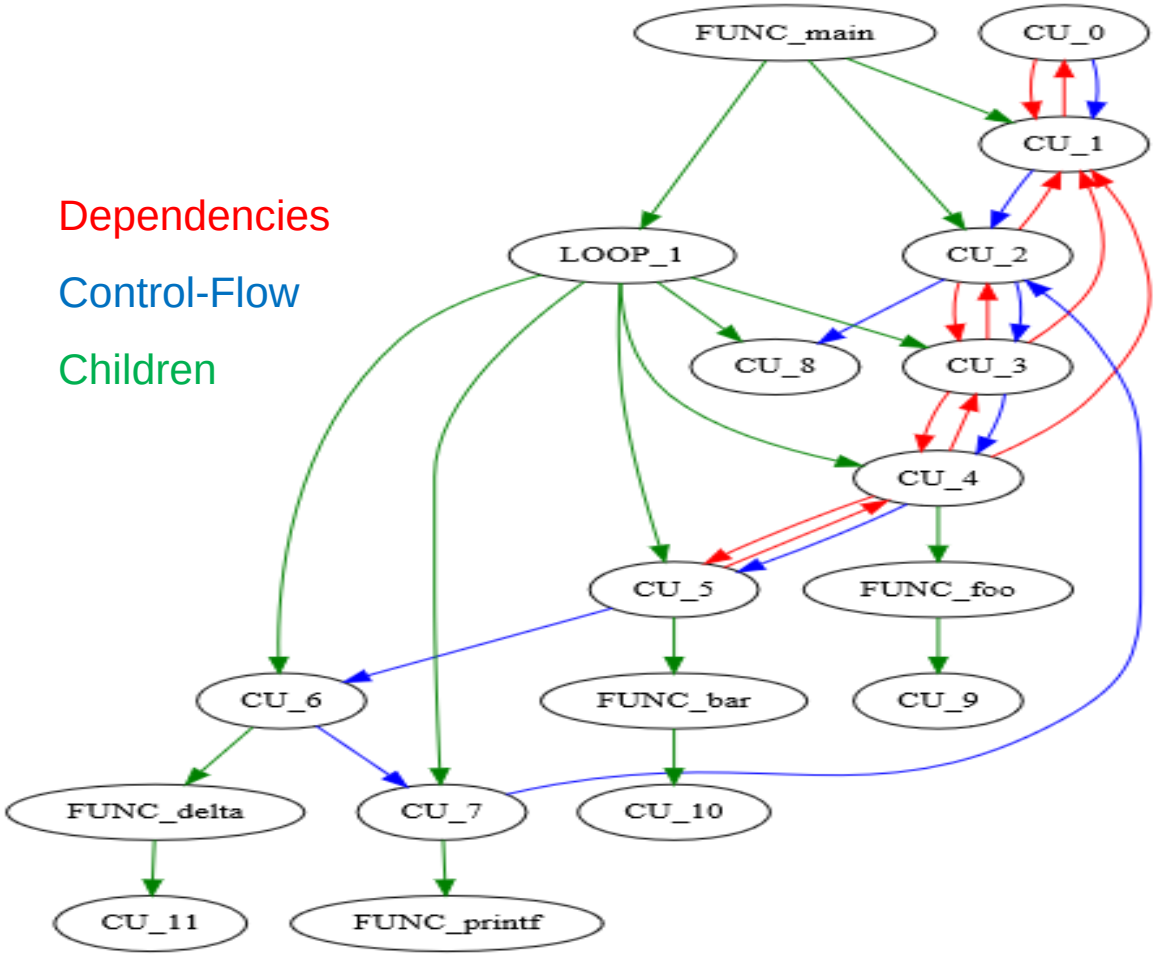
TECHNISCHE UNIVERSITÄT DARMSTADT

```c
#include <stdio.h>

int foo(int in, int d){
    return in * d;
}

int bar(int in, int d){
    return in + d;
}

int delta(int in, int d){
    return in -d;
}

int main( void)
{
    int a,b,c,i;
    int d = 20;
    for (i=0; i<100; i++) {
        a = foo(i, d);
        b = bar(a, d);
        c = delta(b, d);
        printf("%d\n", c);
    }
    return 0;
}
```

Dependencies

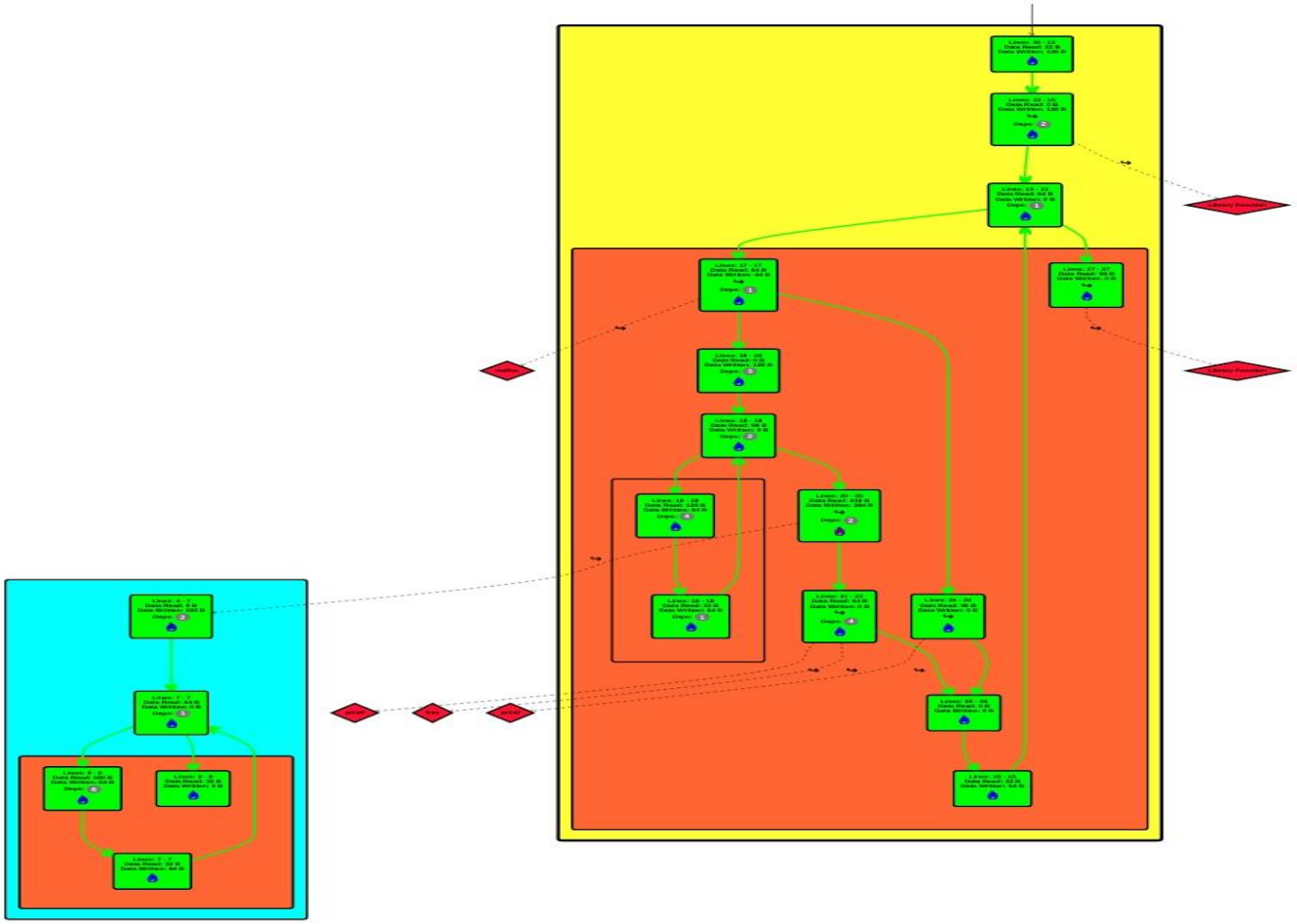
Control-Flow

Children



Program Graph Visualization

TECHNISCHE UNIVERSITÄT DARMSTADT



Outline

TECHNISCHE UNIVERSITÄT DARMSTADT

- Profiler
- Computational Units & Program graph
- Applications
- Evaluation
- Future Works

Applications

TECHNISCHE UNIVERSITÄT DARMSTADT

- Detection of parallel design patterns
  - Output of phase-1 is used for detecting:
    - Pipeline
    - Multiloop pipeline
    - Task parallelism
    - Geometric decomposition
    - Do-all loops
    - Reduction
  - Different approaches are used to detect these patterns

Parallel Patterns

TECHNISCHE UNIVERSITÄT DARMSTADT

Technique	Patterns
Template matching	Pipeline, Do-all
Linear regression analysis	Multi-loop pipeline, Reduction
Breadth first serach	Task parallelism

Pipeline

TECHNISCHE UNIVERSITÄT DARMSTADT

- Exists only if stages are executed many times
  - Loops, recursions and functions with multiple loops
- Graph Matrix is computed from CU Graph of the hotspot
- Pipeline Matrix is created based on the number of CUs in Graph Matrix
- Pipeline Matrix have specific properties like chain dependences and Forward dependence weights

Multi-loop Pipeline

TECHNISCHE UNIVERSITÄT DARMSTADT

- Iterations of one loop depends on iterations of another loop
- Each loop can be a stage of a pipeline
- We profile loops and gather iteration dependence data

```
for (. . .)// Loop x
      a[i] = foo(i);

for (. . .)// Loop y
      b[i] = bar(a[i]);
```

Example code

Variable	Iteration # of Loop x (I_x)	Iteration # of Loop y (I_y)
a[0]	0	0
a[1]	1	1
…	…	…
a[n]	n	n

Results of profiled run

Fusion & Reduction

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Loop fusion:**
  - Fusion of loops ***x*** and ***y*** can occur if:
    - Both loops ***x*** and ***y*** are do-all loops
    - There are no loop carried dependences.
- **Reduction**
  - State-of-the-art compilers may miss reduction due to pointer aliasing or array referencing
  - Dynamic analysis helps overcome limitations of static analysis
  - Detection approach same as multi-loop pipeline
    - Profile iterations of a single loop

Applications

TECHNISCHE UNIVERSITÄT DARMSTADT

- SPMD/MIMD type task parallelism
  - Detection of independent sets of CUs that can run in parallel with each other
  - Detection of parallelism between different region levels
  - Detection of synchronization points between different parallel tasks

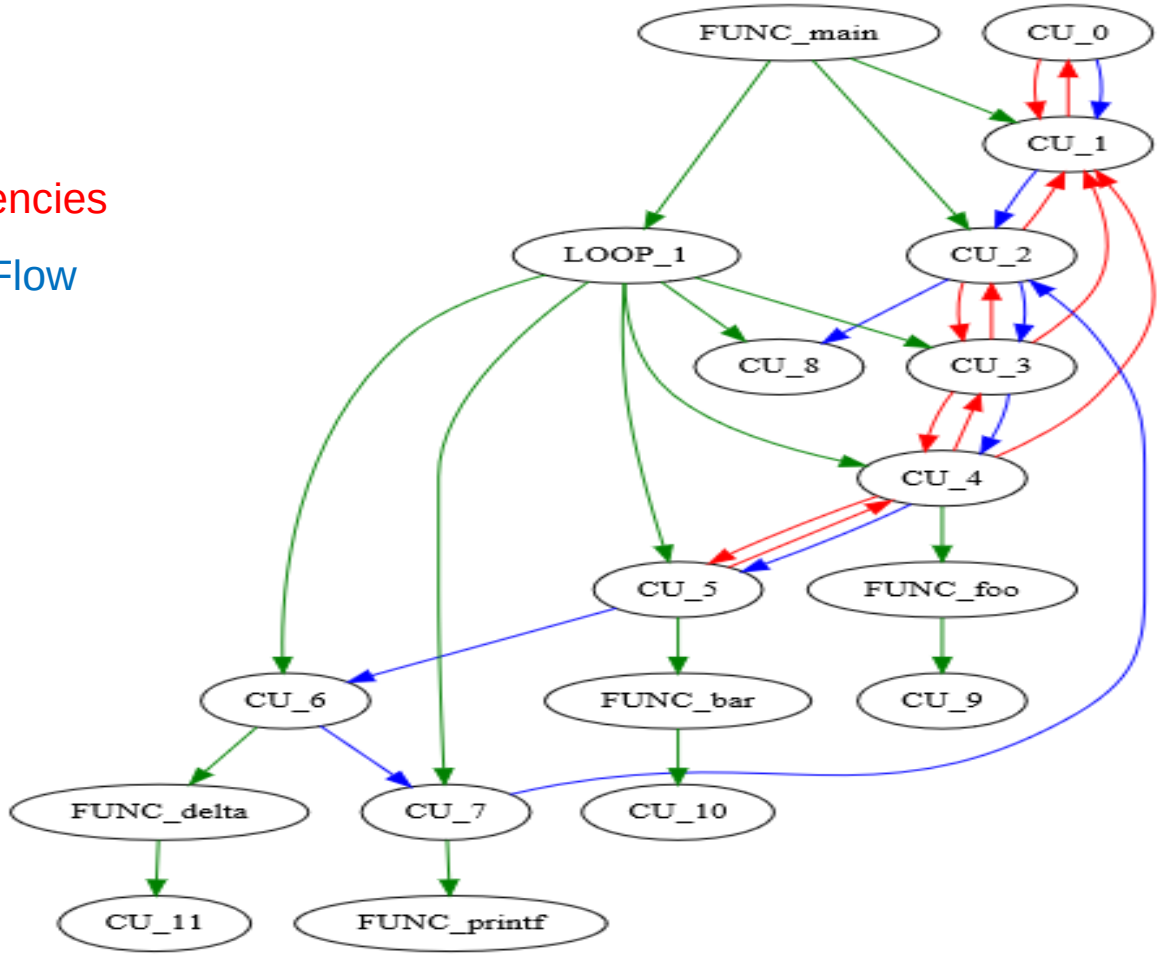
Task Parallelism

TECHNISCHE UNIVERSITÄT DARMSTADT

Dependencies

Control-Flow

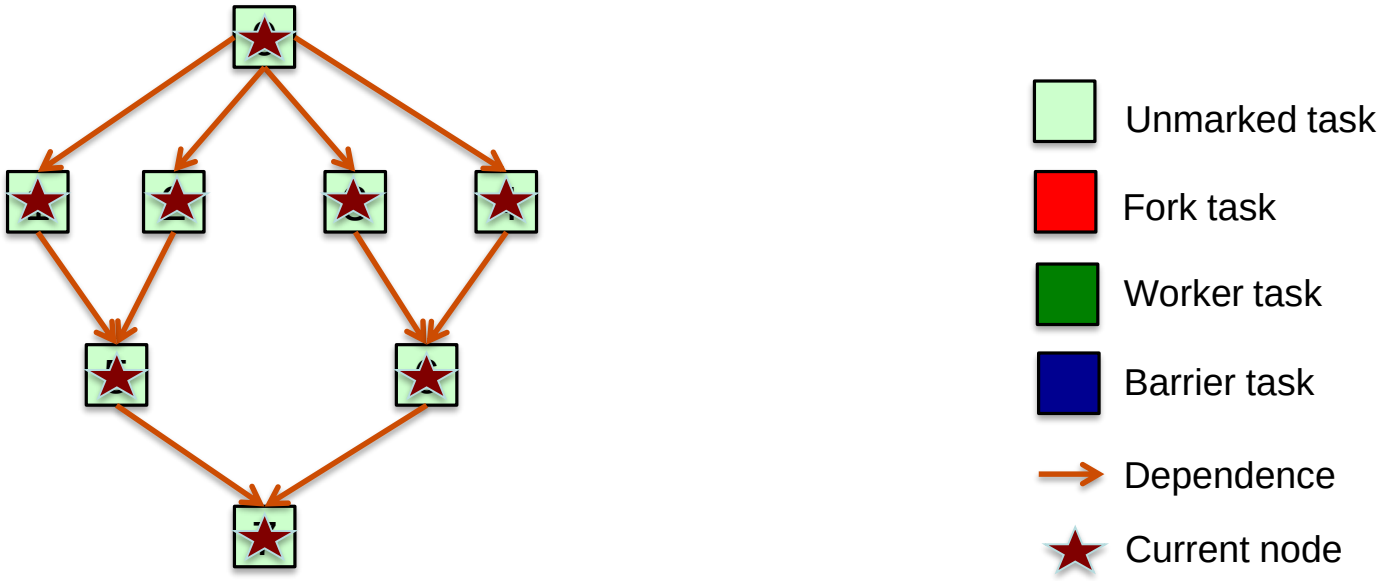
Children



Task Parallelism

TECHNISCHE UNIVERSITÄT DARMSTADT

- Using Breadth-first search we classify the CUs into Fork, Worker and Barrier CUs



Simplified CU Graph – Task Parallelism

- Two barriers can run in parallel if there is no directed path between them

CU Instantiation

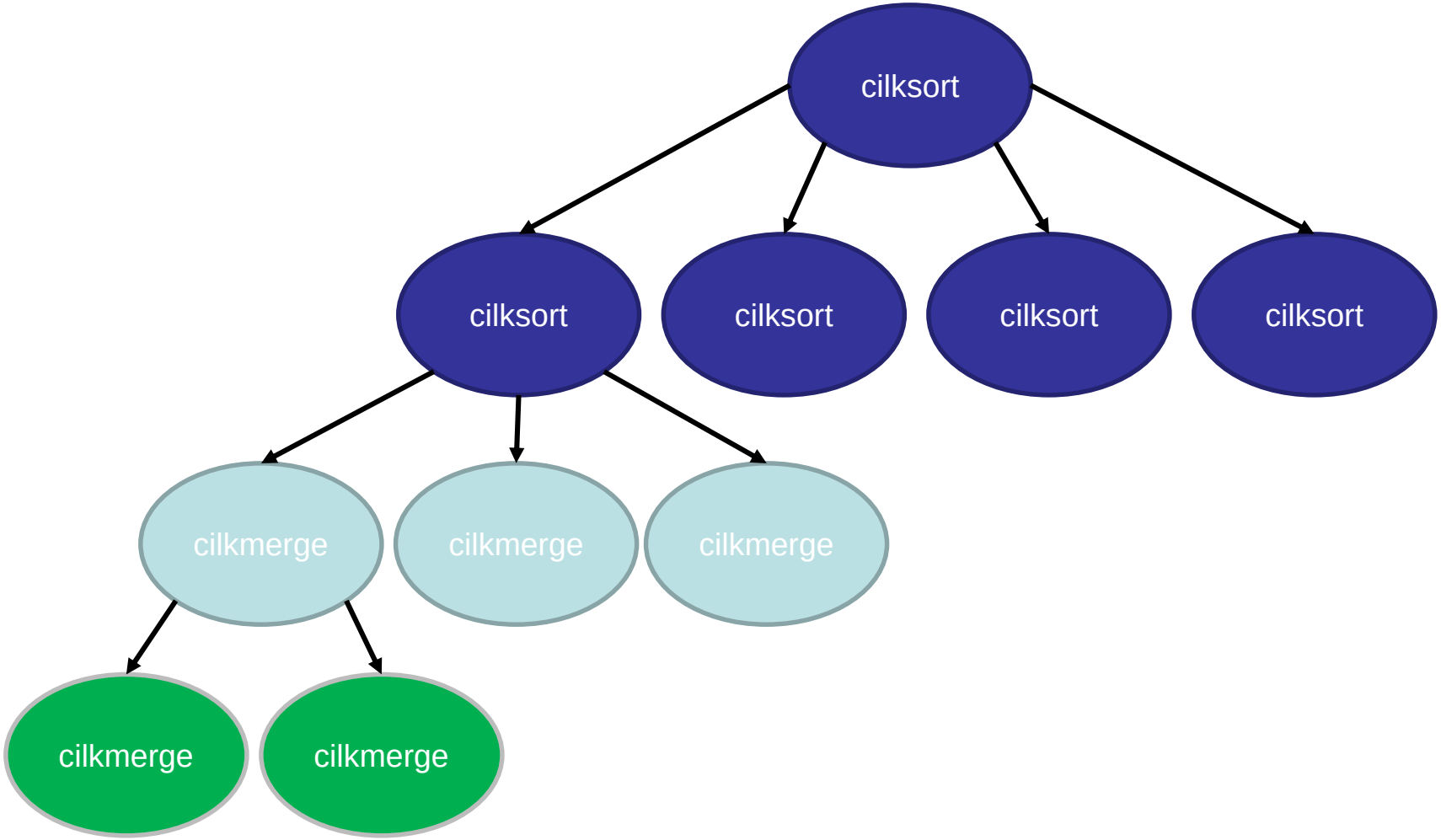
TECHNISCHE UNIVERSITÄT DARMSTADT

```
void cilksort(ELM *low, ELM *tmp, long size)
{
  ...
    cilksort(A, tmpA, quarter);
    cilksort(B, tmpB, quarter);
    cilksort(C, tmpC, quarter);
    cilksort(D, tmpD, size - 3 * quarter);

    cilkmerge(A, A + quarter - 1, B, B + quarter - 1,
tmpA);
    cilkmerge(C, C + quarter - 1, D, low + size - 1,
tmpC);
    cilkmerge(tmpA, tmpC - 1, tmpC, tmpA + size - 1,
A);
  ...
}
```

```
void cilkmerge(ELM *low1, ELM *high1,
               ELM *low2, ELM *high2,
               ELM *lowdestif {
…
cilkmerge(low1, split1 - 1,
               low2, split2,
               lowdest);
cilkmerge(split1 + 1, high1,
               split2 + 1, high2,
               lowdest+lowsize+2);
…
}
```

CU Instantiation - PET

TECHNISCHE UNIVERSITÄT DARMSTADT



Applications

TECHNISCHE UNIVERSITÄT DARMSTADT

- Energy efficient parallelism
  - Energy consumption per CU
  - Energy efficient pattern detection
  - Energy efficient task formation

Energy Efficient Parallelism

TECHNISCHE UNIVERSITÄT DARMSTADT

Energy optimization

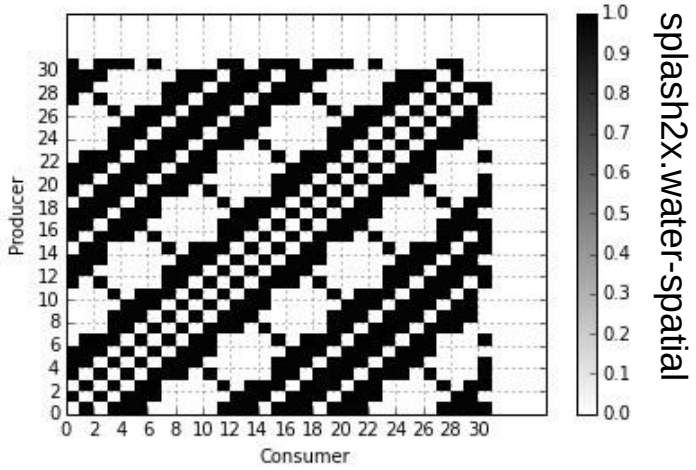
- Reduce memory accesses
- Which openmp constructs to use?

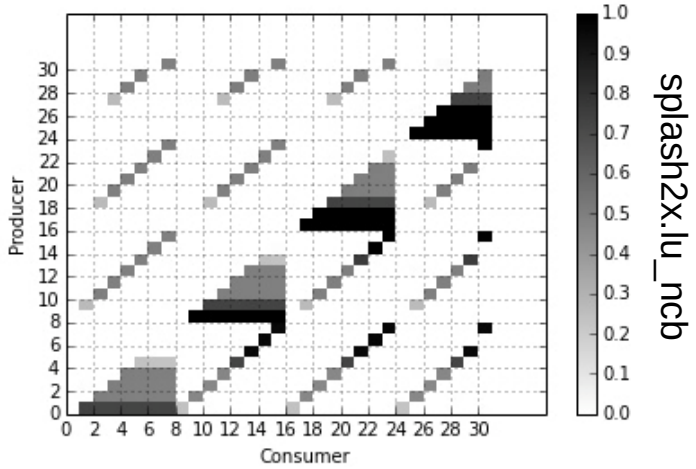
Energy efficient task formation

- Considering CU attributes (data size, memory access frequency, etc.) to form tasks
- Which openmp constructs to use?

Applications

- Detecting communication patterns
  - Produce communication pattern of splash2x applications on multicore platform based on profiled data dependences
  - shows the communication intensity between producer and consumer threads.
  - Critical to understand the performance of parallel applications





Outline

TECHNISCHE UNIVERSITÄT DARMSTADT

- Profiler
- Computational Units & Program graph
- Applications
- Evaluation
- Future Works

Evaluation

TECHNISCHE UNIVERSITÄT DARMSTADT

Pipeline

Application	Pipelines Detected	Pipelines Implemented	Speedup
Parsec:Bodytrack	1	1	1.22
Parsec:Blackscholes	0	0	NA
Parsec:Dedup	2	2	2.15
Parsec:Ferret	2	2	6.14
libVorbis	2	1	8.4

Evaluation

TECHNISCHE UNIVERSITÄT DARMSTADT

Multi-loop pipeline & fusion

Program	LOC	Detected Pattern	# of inter-dependent loops	Speedup	# of Threads
Polybench:ludcmp	135	Multi-loop pipeline	2	14.06	32
Polybench:reg_detect	137	Multi-loop pipeline	2	2.26	16
Parsec:fluidanimate	3987	Multi-loop pipeline	4	1.50	3
Starbench:rot-cc	578	Fusion	2	16.18	32
Polybench:correlation	137	Fusion	4	10.74	32
Polybench:2mm	153	Fusion	2	13.50	32

Evaluation

TECHNISCHE UNIVERSITÄT DARMSTADT

Task parallelism

Program	LOC	Detected Pattern	Estimated Speedup	Speedup	# of Threads
Bots:fib	32	Task parallelism	3.25	13.25	32
Bots:sort	305	Task parallelism	2.11	3.67	32
Bots:strassen	399	Task parallelism	3.5	8.93	32
Polybench:3mm	166	Task parallelism + Do-all	1.5	13.93	16
Polybench:mvt	114	Task parallelism + Do-all	1.96	11.39	32
Polybench:fdtd-2d	142	Task parallelism	2.17	5.19	8

- Recursion in *fib and strassen* is the cause for the big difference in estimated and actual speedups
  - DiscoPoP does not track the depth of recursion when profiling

Future works

TECHNISCHE UNIVERSITÄT DARMSTADT

- Optimize DiscoPoP profiler
  - Add static dependence analysis
  - Generate coarser CUs

Future works

TECHNISCHE UNIVERSITÄT DARMSTADT

- Support for OpenMP based parallel applications
  - Currently profiler only supports Pthread based applications
  - OpenMP is the industry standard for multi-core applications
  - Addition of support for openMP based parallel applications in DiscoPoP profiler

Future works

TECHNISCHE UNIVERSITÄT DARMSTADT

- Energy profiling
  - Needed for energy efficient task formation and pattern detection
  - Dynamic: instrumentation methods
  - Static: Energy per instruction (EPI)

Future works

TECHNISCHE UNIVERSITÄT DARMSTADT

- Detect energy efficient parallel patterns
  - Detect available parallel patterns in a code section
  - Considering energy efficiency parameters
    - Number of cores
    - Number of tasks
    - Energy balance of detected parallel tasks
    - Data size of tasks
    - Communication frequency
    - …

Current status

TECHNISCHE UNIVERSITÄT DARMSTADT

- An efficient parallelized dependence profiler is available
  - Supports Pthread based parallel applications
- Successful detection of several parallel design patterns
- Adoption of new version of LLVM is in progress
- Profiler soon to be published as an open source tool

TECHNISCHE UNIVERSITÄT DARMSTADT

Thank You!