Automatic Extraction of Parallelism for Embedded Software

Daniel Cordes

June 29th, 2011

Outline

1. Motivation / State-of-the-art / Ideas
2. ILP-based Task-Level Parallelization
3. ILP-based Loop-Level Parallelization
4. Conclusion & Future work

Outline

1. **Motivation / State-of-the-art / Ideas**
2. ILP-based Task-Level Parallelization
3. ILP-based Loop-Level Parallelization
4. Conclusion & Future work

Motivation

- Using multiple cores in one systems can...
  - ✓ reduce CPU frequencies / exec. time
  - ✓ save energy
  - ✓ enable further optimizations

Problem

- Most embedded application are written in sequential C
- Splitting program into tasks manually is...
  - ✗ error prone
  - ✗ time consuming

➔ Automatic parallelization beneficial

State-of-the-art

- Many semi- or fully automatic parallelization frameworks exist
- Common characteristics:
  - Detailed costs model are rarely used
  - Hard to exploit whether parallelization is beneficial
  - Due to complexity:
    - (Greedy or other) heuristics often used
  - Most frameworks not optimized for embedded devices

Idea

- **Focus** on characteristics of **embedded system** applications and architectures
  - Often streaming orientated applications
  - In general: limited OS support compared to host-architectures
  - NON-unified memory architectures → more expensive communication
- Use **integer linear programming** (ILP) for parallelization decisions
  - Clear, mathematical description of problem
  - Optimal due to its model

Outline

1. Motivation / State-of-the-art / Ideas
2. **ILP-based Task-Level Parallelization ***
3. ILP-based Loop-Level Parallelization
4. Future work

*D. Cordes, P. Marwedel, A. Mallik, *Automatic parallelization of embedded software using hierarchical task graphs and integer linear programming*, CODES/ISSS '10

ILP-based Task-Level Parallelization

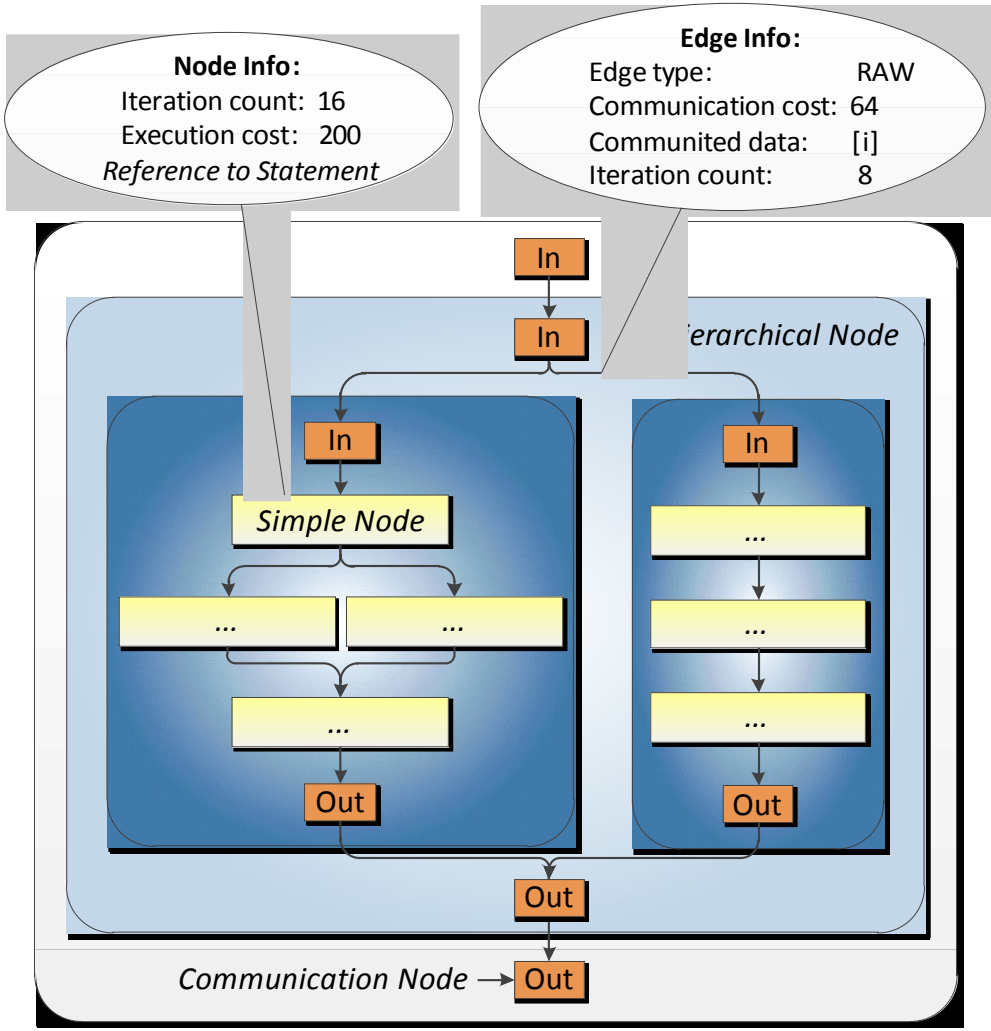
- Focus on characteristics of embedded system applications and architectures
- Reduce search space by introducing hierarchy to the task graph model
- Use integer linear programming (ILP) for parallelization decisions
- Use adequate cost model to balance tasks

Characteristics

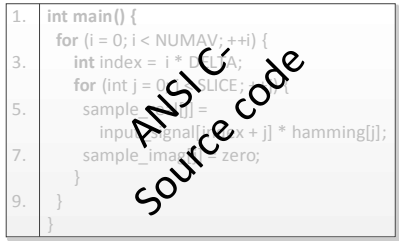
- Coarse grained parallelization technique
- Possible to limit number of concurrently executed tasks

Hierarchical Task Graph

- Too large search space for flat task graphs using ILP
- Introduction of hierarchy
  - → Only small number of nodes per hierarchical level
- Communication redirected through communication in- and out-nodes
  - → Decisions can be done locally
- Automatically extracted from source code
- Annotated with com / exec costs

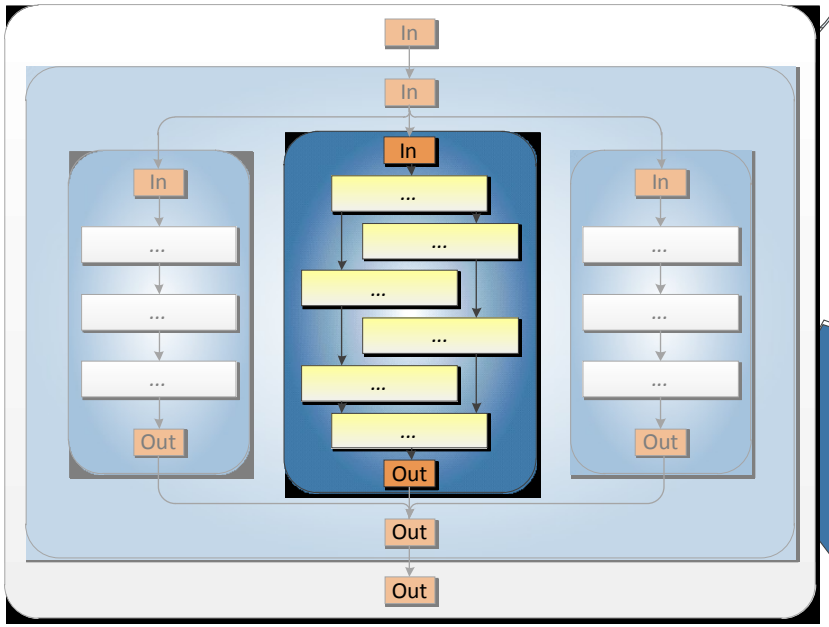


Parallelization Methodology

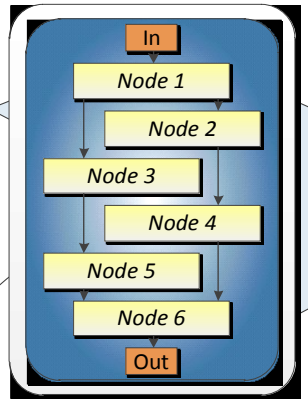


1. Extract hierarchical graph

2. Parallelize nodes bottom-up



3. Transform to ILP



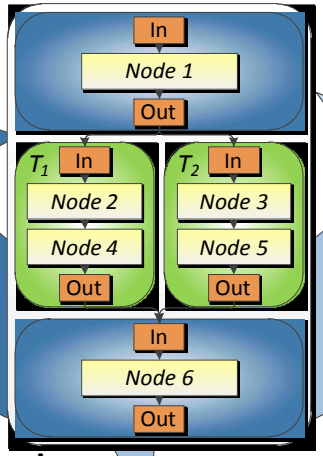
$$(1) \quad \min \sum x_i * c_i$$

$$(2) \quad \sum b_i * x_i \leq C$$

$$(3) \quad \forall x_i \leq 1$$

$$(4) \quad \forall x_i \geq 0$$

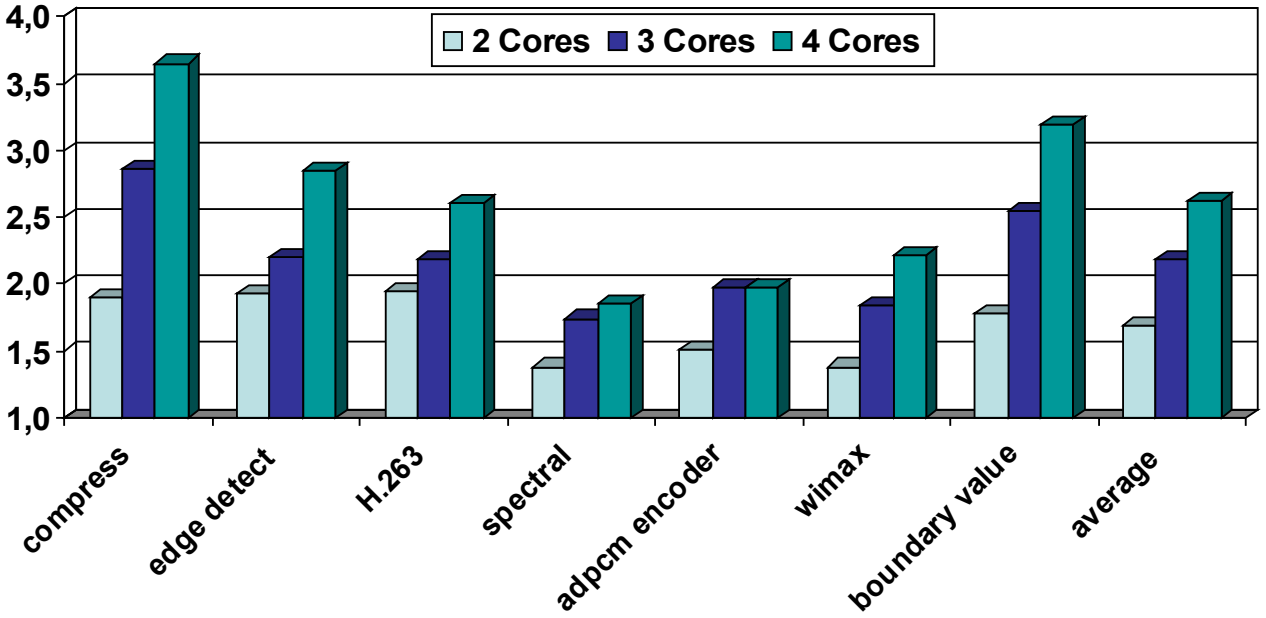
4. Solve ILP for different task limitations



5. Transform solutions to HTG

6. Attach results and continue with other nodes

Results



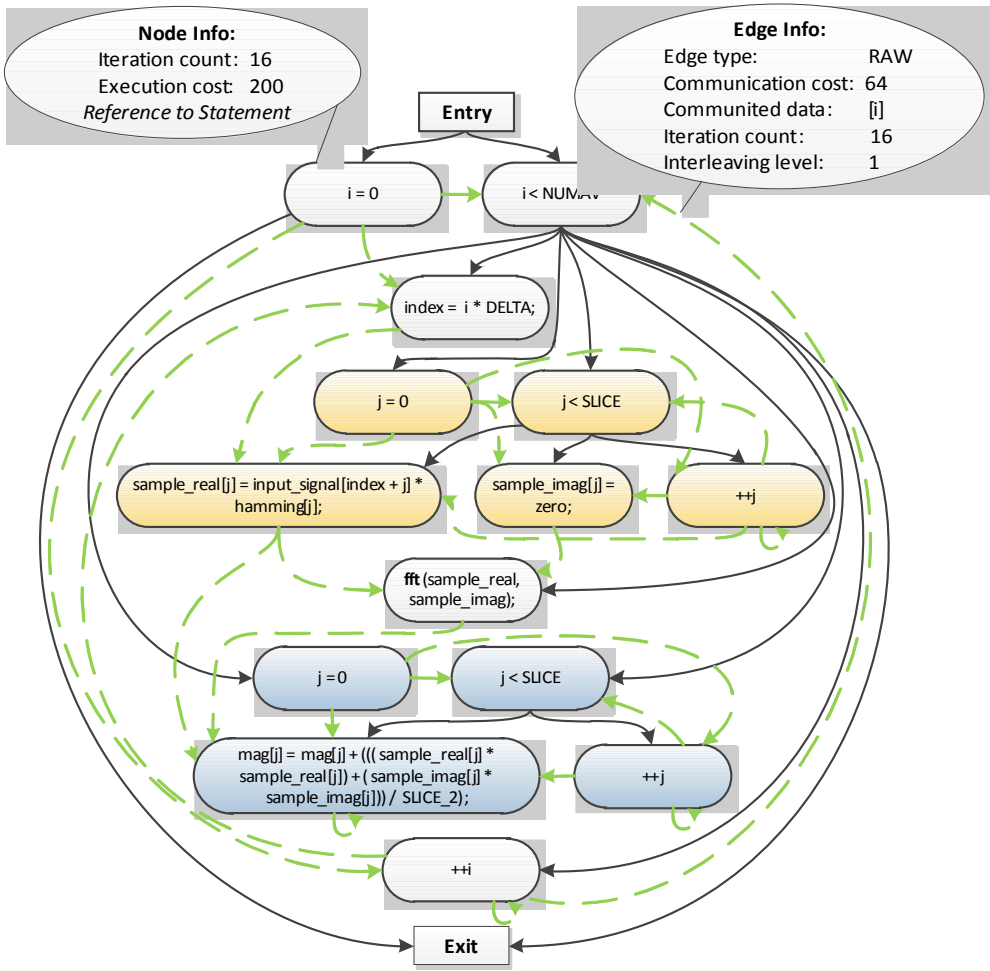
- Cycle accurate simulator: MPARM
- OS: RTEMS + Runtime Library
- Measured: Exec. time of application without OS init
- Speedup up to 1.9x / 2.9x / 3.7x
- Average speedup: 1.8x / 2.2x / 2.7x
- Work was part of MNEMEE European FP 7 project

Outline

1. Motivation / State-of-the-art / Ideas
2. ILP-based Task-Level Parallelization
3. **ILP-based Loop-Level Parallelization**
4. Conclusion & Future work

ILP-based Loop-Level Parallelization

- HTG is both a blessing and a curse
- Problem: Hierarchy hides dependencies between different loop iterations
- -> Used a Program Dependence Graph (PDG)
- Build PDG for each loop and try to parallelize
- Finer grained approach than Task-Level parallelism

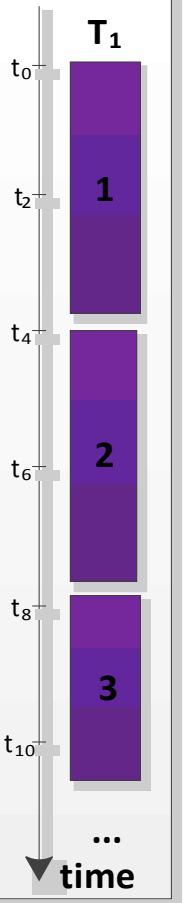


Parallelization example – Sequential execution

- Main loop of spectral benchmark
- All iterations executed sequentially

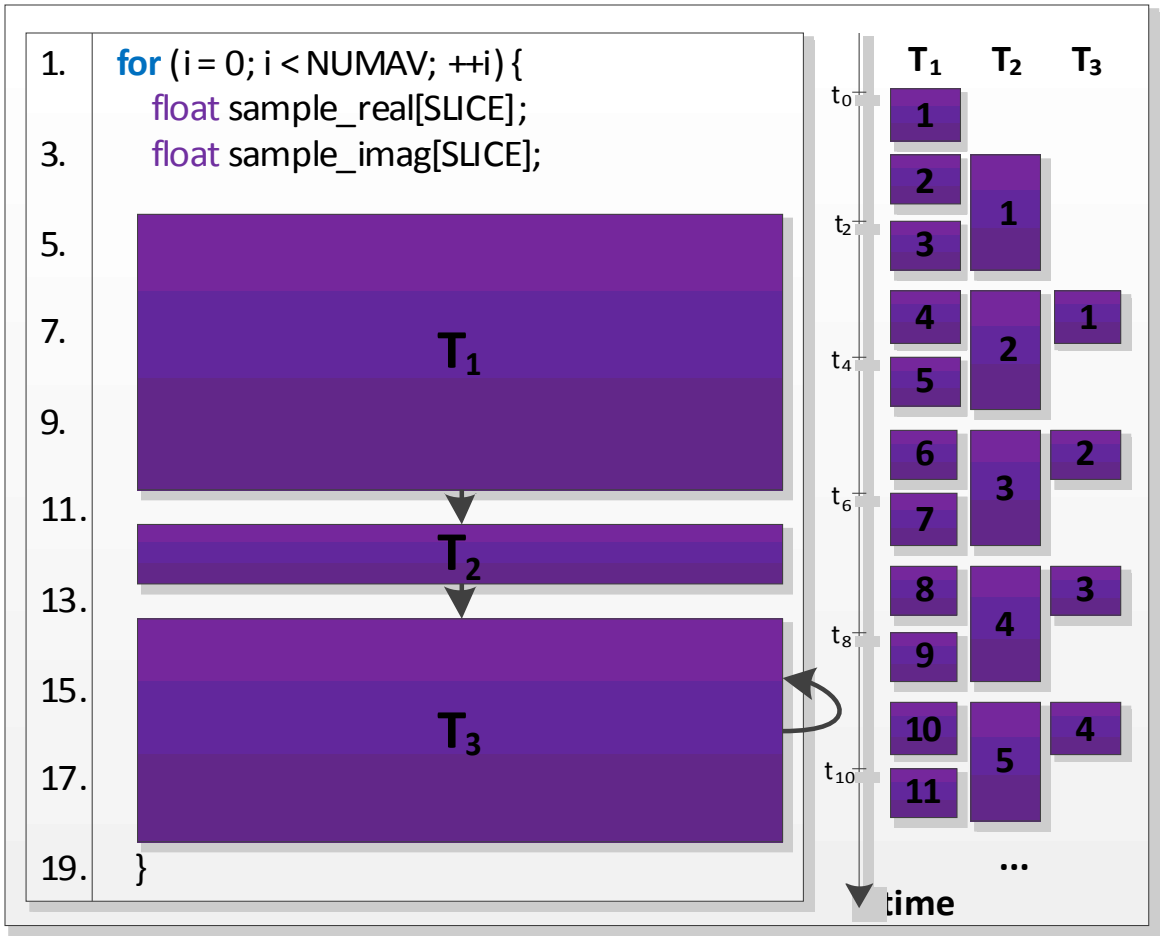
```
1.  for (i = 0; i < NUMAV; ++i) {
      float sample_real[SLICE];
3.    float sample_imag[SLICE];

5.    int index = i * DELTA;
      for (int j = 0; j < SLICE; ++j) {
7.     sample_real[j] =
          input_signal[index + j] * hamming[j];
9.    sample_imag[j] = zero;
      }
11.
      fft(sample_real, sample_imag);
13.
      for (int j = 0; j < SLICE; ++j) {
15.    mag[j] = mag[j] + ((( sample_real[j] *
            sample_real[j]) + ( sample_imag[j] *
17.         sample_imag[j])) / SLICE_2);
      }
19.  }
```

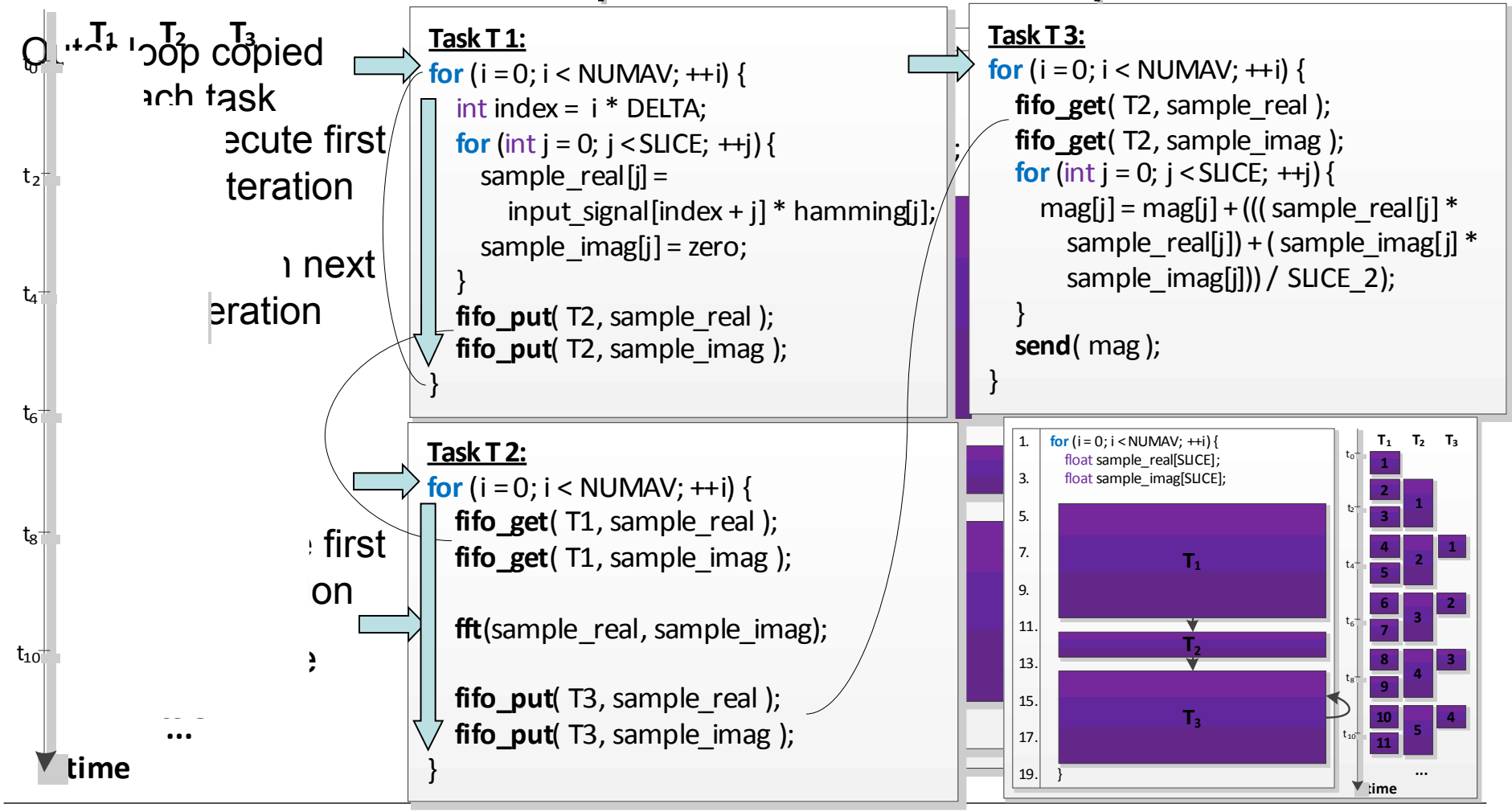


Parallelization example – Horizontal splits

- Split statements into disjunctive parts
- Creates a pipeline of calculations
- Execute 1 iteration
- Communicate data
- Continue with next iteration

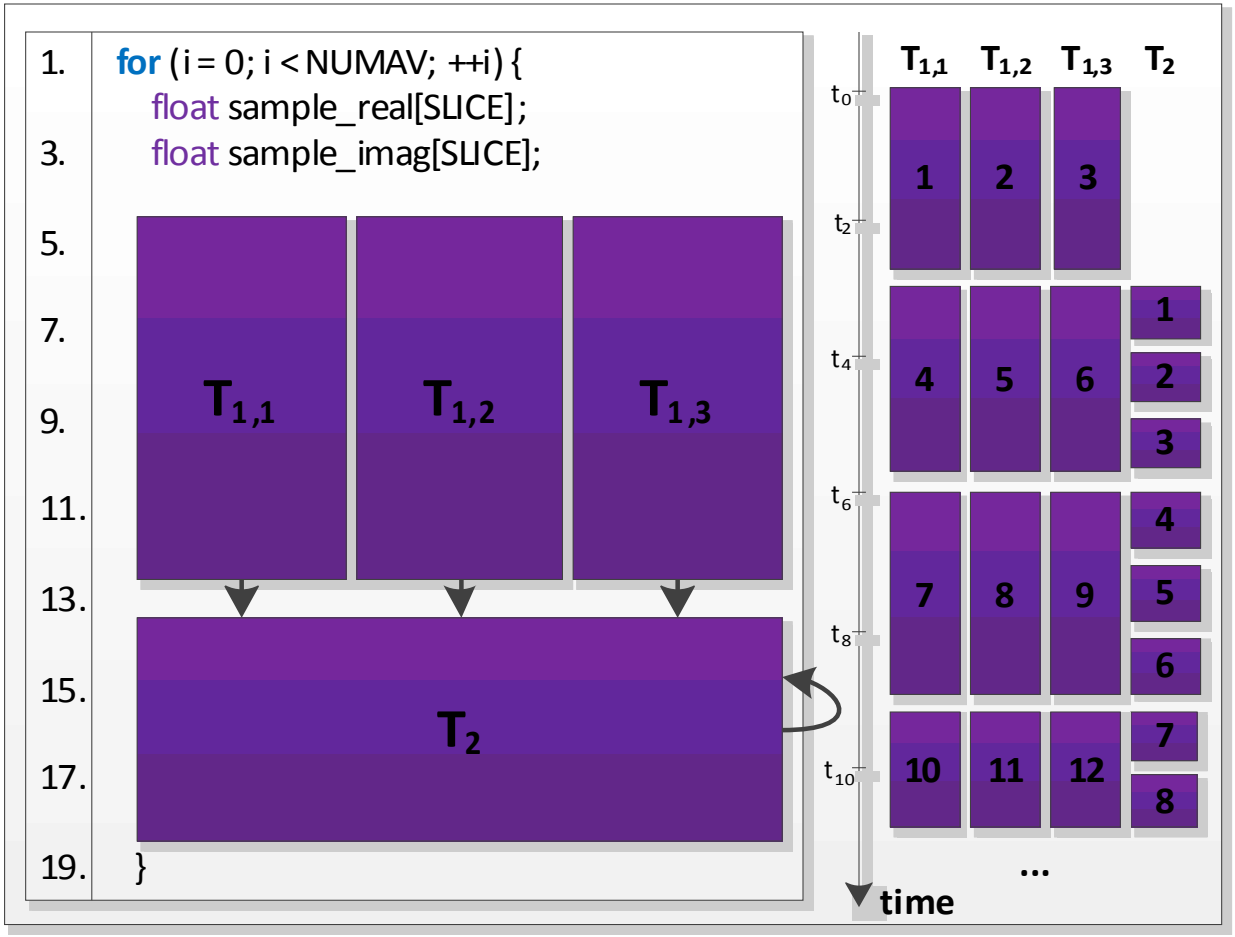


Parallelization example – Horizontal splits



Parallelization example – Horizontal+Vertical splits

- Horizintal pipeline-cuts + vertical cuts
- T1 is divided into 3 threads
- Iter. {1,4,7,..}, {2,5,8,..} and {3,6,9,..} executed in parallel



Parallelization example – Horizontal splits

Task T1_1:

```
for (i = 0; i < NUMAV; i += 3) {
  int index = i * DELTA;
  for (int j = 0; j < SLICE; ++j) {
    sample_real[j] =
      input_signal[index + j] * hamming[j];
    sample_imag[j] = zero;
  }
  fft(sample_real, sample_imag);
  fifo_put( T2, sample_real );
  fifo_put( T2, sample_imag );
}
```

Task T1_2:

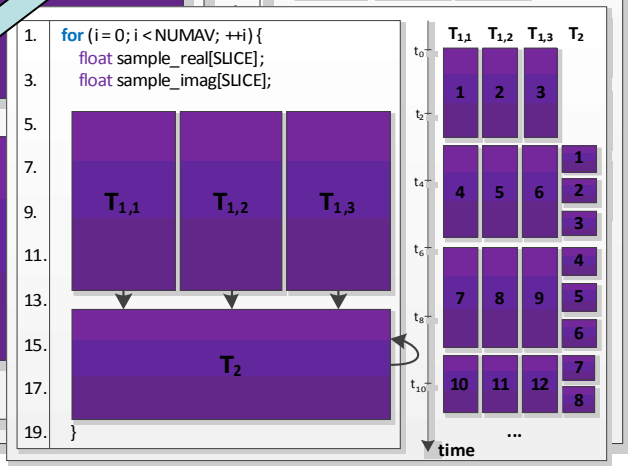
```
for (i = 1; i < NUMAV; i += 3) {
  int index = i * DELTA;
  for (int j = 0; j < SLICE; ++j) {
    sample_real[j] =
      input_signal[index + j] * hamming[j];
    sample_imag[j] = zero;
  }
  fft(sample_real, sample_imag);
  fifo_put( T2, sample_real );
  fifo_put( T2, sample_imag );
}
```

Task T1_3:

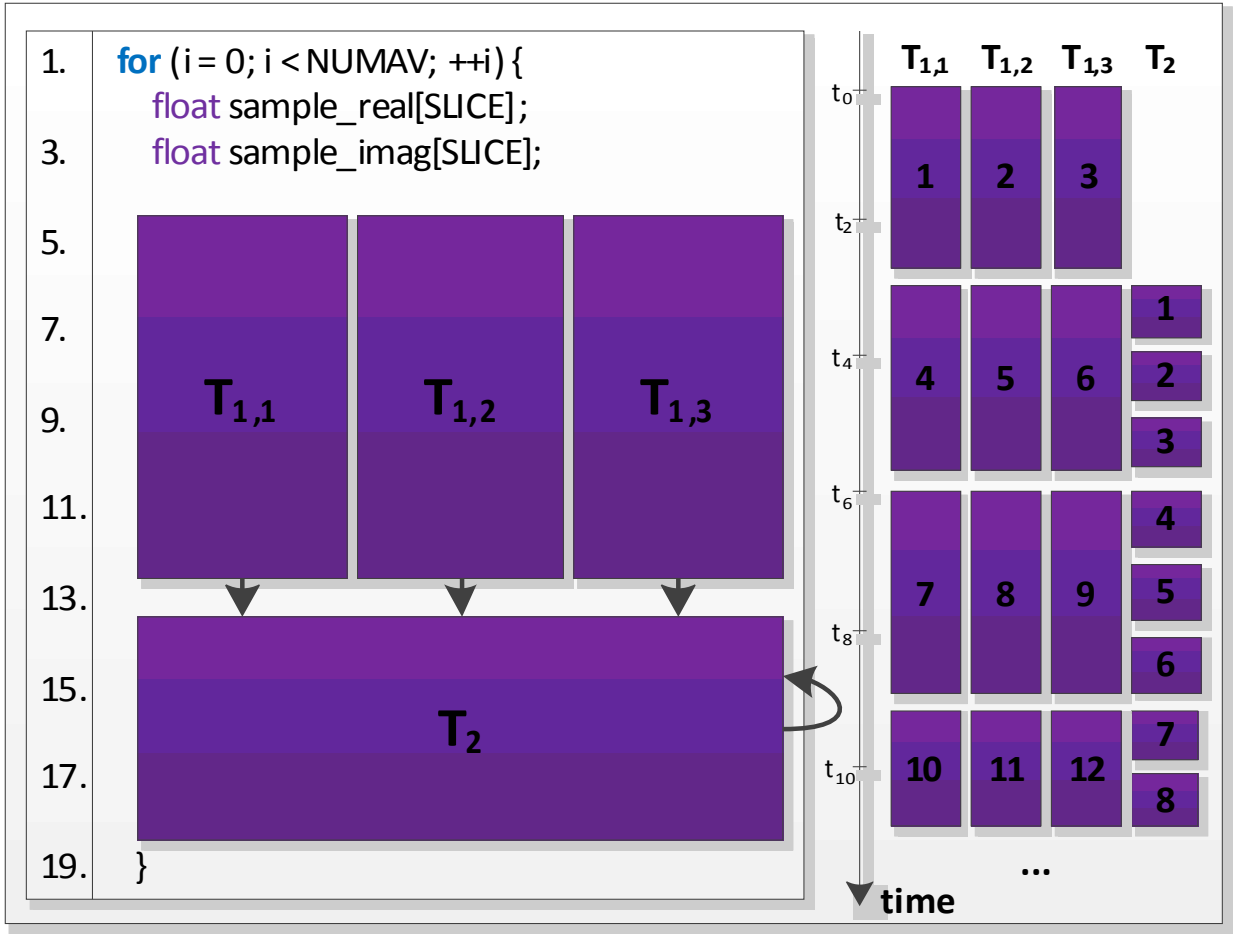
```
for (i = 2; i < NUMAV; i += 3) {
  int index = i * DELTA;
  for (int j = 0; j < SLICE; ++j) {
    sample_real[j] =
      input_signal[index + j] * hamming[j];
    sample_imag[j] = zero;
  }
  fft(sample_real, sample_imag);
  fifo_put( T2, sample_real );
  fifo_put( T2, sample_imag );
}
```

Task T2:

```
for (i = 0; i < NUMAV; ++i) {
  fifo_get( i == 0 ? T1_1 : ..., sample_real );
  fifo_get( i == 0 ? T1_1 : ..., sample_imag );
  for (int j = 0; j < SLICE; ++j) {
    mag[j] = mag[j] + ((( sample_real[j] *
      sample_real[j]) + ( sample_imag[j] *
      sample_imag[j])) / SLICE_2);
  }
  send( mag );
}
```



Parallelization example – Horizontal+Vertical splits



Parallelization Methodology

- Extract a Sub-PDG for each loop
- Parallelize each loop(-nest)
  - Transform dependencies, execution times, execution counts etc. to ILP
  - Extract horizontal + vertical splits at the same time
    → avoid local optimum
- Combine best solutions of all loops
  - Observe limit of concurrently executed tasks

Outline

1. Motivation / State-of-the-art / Ideas
2. ILP-based Task-Level Parallelization
3. ILP-based Loop-Level Parallelization
4. **Conclusion & Future work**

Conclusion & Future work

- Presented two ILP-based parallelization techniques
  - A coarse-grained task-level methodology
  - A finer-grained loop-level methodology
- Approaches orthogonal to each other
  - Combine both approaches
  - Technical problem: Hierarchical vs. flat graph
- Future work
  - Better support for heterogeneous platforms
  - Also look at evolutionary algorithms

Thank you very much for your attention!

Questions?