Fast Fourier Transforms in CHAPEL

Doru Thom Popovici, Franz Franchetti
ECE, Carnegie Mellon University
Motivation

Material Sciences

Quantum mechanics

Fourier Transform

Signal Processing

Synthetic Aperture Radar
Goal

World A

Low Performance

World B

> 30 MB of generated or handwritten code
Goal

World A

Low Performance

World B

Want best of both worlds

> 30 MB of generated or handwritten code
Outline

- Background

- Sequential and Parallel Implementation

- First Attempt in CHAPEL

- Summary
Background

- **Definition**

\[ y_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{2\pi j k n}{N}}, \quad k = 1..N \]

- **Fast implementations**

\[ y = DFT_n \cdot x \]

\[ DFT_n = \left[ \omega_n^{kl} \right]_{0 \leq k, l < n}, \quad \omega_n = e^{-\frac{2\pi j}{n}} \]
Sequential Recursive Implementation
Parallel Implementation

Communication  Parallel DFTs  Communication  Parallel DFTs  Communication

input

output
First attempt in CHAPEL

```chapel
var part = psize / radix;
var nsrcc_domain = src_domain by radix;
var ndest_domain = dest_domain #part;

for i in 1..radix { compute_dft(Y, X, nsrcc_domain + (i - 1) * cradix, ndest_domain + (i - 1) * part, ...); }
```

dft2(Y, X, src_domain, dest_domain);
dft4(Y, X, src_domain, dest_domain);
dft8(Y, X, src_domain, dest_domain);
dft16(Y, X, src_domain, dest_domain);
dft32(Y, X, src_domain, dest_domain);

generated

handwritten

variables, operations, communication, recursive stages, kernels
Single Core Intel Haswell 4770K

Performance [Mflop/s]

- Radix 2
- Radix 4
- Radix 8
- Radix 16
- Radix 32

Log size
Summary

- **Single threaded implementation**
  clean implementation; performance still lacking

- **Multiple threads and multiple nodes**
  work in progress; use domains and locals

- **Apply optimizations within CHAPEL**
  we know what is needed to optimize the Fourier transforms; add that knowledge within a compiler such as CHAPEL