Cerebras CS-2 上のプログラミングを支援する並列スケルトン

明治大学理工学部　若杉直椰　岩崎英哉

概要

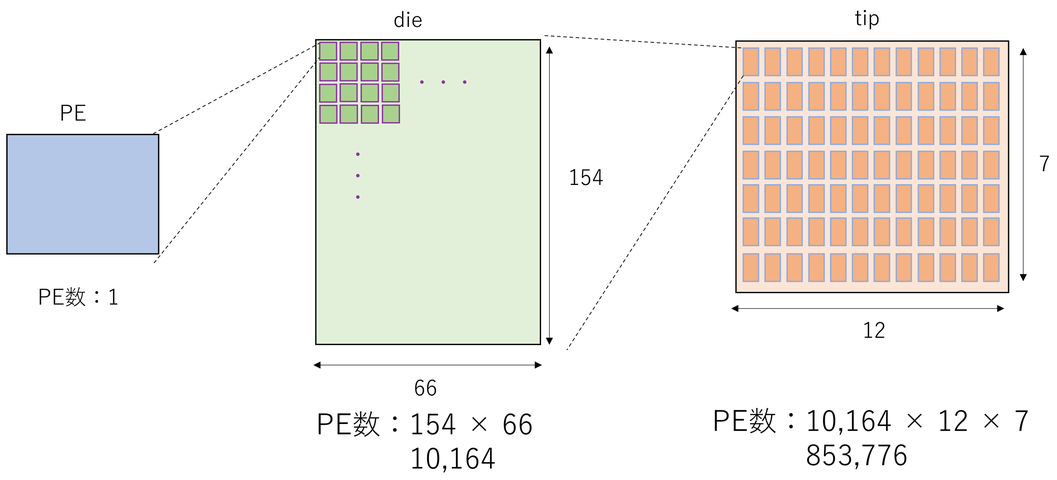
コーディング難易度の高い，機械学習用アクセラレータ Cerebras CS-2 のプログラム開発を支援する．

- **ドメイン固有言語 (DSL)** を導入し，**並列スケルトン**を用いて簡潔な記述を可能とした．
- DSLプログラムをCerebras CS-2上で動作するコードに変換する変換器を実装した．
- いくつかのプログラムを用いて，有用性を評価した．

Cerebras CS-2: アーキテクチャ

大型半導体ウェハスケールエンジン(WSE-2)を搭載した，分散メモリ型のコンピュータ．

- ダイを切り出すことなく1枚の巨大なチップとすることで，チップ内部で大規模な並列計算を可能にしている [1] .
- 密結合なPE同士で通信が完結するため，低レイテンシ通信が可能である．隣接PEへの通信は**1サイクル**で可能．



Cerebras CS-2 : プログラム開発

以下の三つのコードを書く必要がある

- ホストコード (Python) : ホストとデバイス (CS-2) の間のデータの授受，CS-2上での関数実行を命令する．
- レイアウト (CSL) : 使用するPEのレイアウトや，データの通信経路を決める．
- PEプログラム (CSL) : 具体的なPE内での処理を記述する．

CSLはCS-2上のプログラムを記述する独自の構文を持つ言語であり，**煩雑でハードルが高い**．

提案手法1 : 並列スケルトンの提供

並列スケルトン [2] は，並列プログラミングにおける頻出パターンを抽象化してライブラリ関数としたもの．

⟹ ユーザは，並列スケルトンを組み合わせて，並列性を意識することなくプログラムを記述できる．

1次元データ用の並列スケルトンとして，以下を提供する．

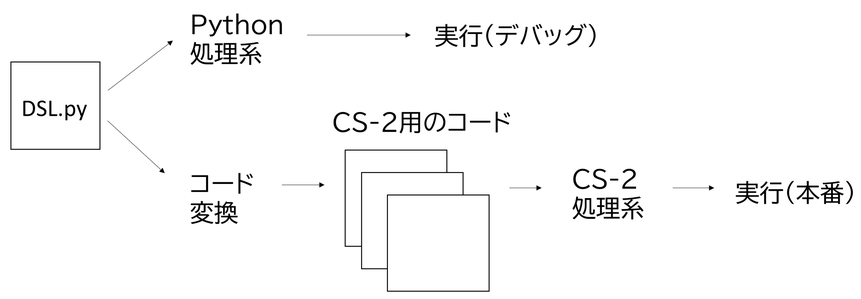
- map : リストに一様に関数適用したリストを返す．
- map_ow : リストを直接書き換えるmap．
- zipwith : 2つのリストに一様に関数適用したリストを返す．
- zipwith_ow : リストを直接書き換えるzipwith．
- reduce : 結合的な二項演算子を用いて要素を集積する．

2次元データについても同様のものを提供する．

提案手法2 : DSLの導入

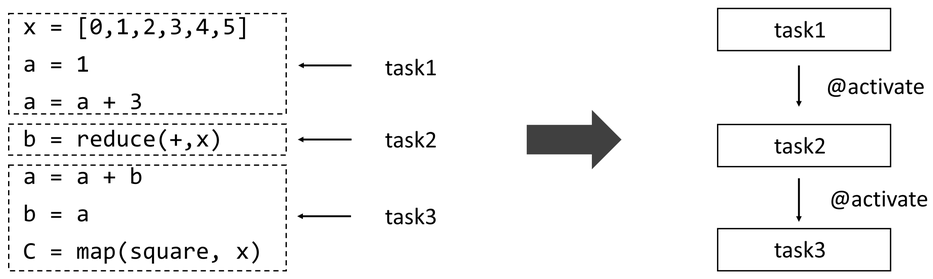
DSLを**Pythonの内部DSL**として定義する．

- CSLの煩雑な構文を回避する．
- Pythonの処理系による動作確認・デバッグを可能とする．



コード変換器の実装

CSLはタスクに基づく**継続渡し形式**でコードを記述する．
⟹ コード変換器は，DSLプログラムの制御フローを解析し，コードブロックをタスクへ割り当てる．



性能評価

提案したDSLによって，コードを簡潔に書くことができた．

```
param memcpy_params: comptime_struct;
param c2d_params: comptime_struct;
param data_per_pe: u16;

const sys_mod = @import_module("<memcpy/memcpy>", memcpy_params);
const mpi_x = @import_module("<collectives_2d/pe>", .{ .dim_params = c2d_params.x });
const mpi_y = @import_module("<collectives_2d/pe>", .{ .dim_params = c2d_params.y });

var data: [data_per_pe] f32;
var result: [data_per_pe] f32;

var data_ptr: [*]f32 = &data;
var result_ptr: [*]f32 = &result;

var recv_x = @zeros([data_per_pe]f32);
var recv_y = @zeros([data_per_pe]f32);

var data_dsd = @get_dsd(mem1d_dsd, .{ .tensor_access = |i|{data_per_pe} -> data[i] });

var recv_re_result = @zeros([1]f32);
var send_re_result = @zeros([1]f32);
var re_result: f32 = 0.0;
var re_count: u16 = 1;
var px: u16;
var py: u16;

const reduce_id:              local_task_id = @get_local_task_id(9);
const y_reduce_id:            local_task_id = @get_local_task_id(10);
const x_bcas_id:              local_task_id = @get_local_task_id(11);
```

PEプログラム(CSL)

```
param WIDTH: i16 ;
param HEIGHT: i16 ;
param DATA_PER_PE: i16 ;
param LENGTH: i16 ;

const c2d = @import_module("<collectives_2d/params>");

const memcpy = @import_module( "<memcpy/get_params>", .{
    .width = WIDTH,
    .height = HEIGHT,
});

layout {

  @set_rectangle(WIDTH,HEIGHT);

  var py:i16 = 0;
  while(py < HEIGHT) : (py += 1){
    var px:i16 = 0;
    while(px < WIDTH) : (px += 1){
      const memcpy_params = memcpy.get_params(px);

      const c2d_params = c2d.get_params(px,py,.{
        .x_colors = .{
        @get_color(0),
        @get_color(1)
        },
        .x_entrypoints = .{
        @get_local_task_id(2),
```

レイアウト(CSL)

```
#!/usr/bin/env cs_python
import argparse
import json
import numpy as np

from cerebras.sdk.runtime.sdkruntimepybind import SdkRuntime
from cerebras.sdk.runtime.sdkruntimepybind import MemcpyDataType
from cerebras.sdk.runtime.sdkruntimepybind import MemcpyOrder

parser = argparse.ArgumentParser()
parser.add_argument("--name", help="the test name")
parser.add_argument("--cmaddr", help="IP:port for CS system")
args = parser.parse_args()

with open(f"{args.name}/out.json", encoding='utf-8') as json_file:
    compile_data = json.load(json_file)
compile_params = compile_data["params"]
width =  int(compile_params["WIDTH"])
height =  int(compile_params["HEIGHT"])
data_per_pe = int(compile_params["DATA_PER_PE"])
length = int(compile_params["LENGTH"])

data = [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0]
data = np.array(data,dtype=np.float32)

memcpy_dtype = MemcpyDataType.MEMCPY_32BIT
memcpy_order = MemcpyOrder.ROW_MAJOR

runner = SdkRuntime(args.name, cmaddr=args.cmaddr)
```

ホストコード(Python)

```
import skelton

from dataclass import Data

WIDTH = 2
HEIGHT = 2
def f (x):
    return x - ave
def g (x):
    return x * x

data1 = Data(1)
data1.data = [1,2,3,4,5,6,7,8]
ave = skelton.reduce("add",data1,8)
ave = ave / 8
a = skelton.map(f,data1,8)
skelton.map_ow(g,a,8)
re = skelton.reduce("add",a,8)
print(re)
```

DSL

3つのプログラムを対象に，コード長を比較した．

コードの種類	DSL	手書きのコード				
プログラム名	DSL	code.csl	layout.csl	run.py	手書きコードの総和	DSL / 総和[%]
行列積	12	143	43	34	220	5.5
分散	16	158	43	34	235	6.8
モンテカルロ	17	146	44	41	231	7.4

コード長を**約10%**まで減少させることができた．

分散を求めるプログラムにおいて，シミュレータを用いて実行し，サイクル数を比較した．

コードの種類	サイクル数 [回]	割合
生成コード	7532	121
手書きのコード	6225	100

生成コードのサイクル数は**約1.2倍**となった．

今後の課題

- map，zipwithの融合等による，実行速度の向上
- 並列スケルトンの拡張

参考文献

[1] Takaaki Miyajima, Ryunosuke Matuzaki, Leon Fukuoka. ”STREAM Benchmark on Cerebras WSE-2”. ISC 2024. Poster.
[2] 岩崎英哉, 胡振江. ”スケルトン並列プログラム”. 情報処理. Vol46. No10 pp 1158-1162 (2005)