

# 3点プロットの作り方 v17

---

塩谷 亮太

# 3点プロットのチェック・リスト

3点プロットを作ったら、以下が満たされているかを確認する：

（「形式に関するチェック」については、他のタイプのプロットでも同様に確認する）

## ■ 内容に関するチェック：

1. 背景，課題，提案の3項目から構成されている
2. 背景は，課題と提案に向けて話題を絞り込んでいる
3. 課題は，「～が悪い」「～が遅い」などの問題を直接示す文を含んでいる
4. 提案は，上記の問題が「どのように」「なぜ」解決されるのかを示す文を含んでいる

## ■ 形式に関するチェック：

5. 各箇条書きは複文を含んではならない
6. 1行を越えるような長い修飾節を含んだ文を含んではならない
7. 4つ以上の項目を並列に並べてはいけない
8. 箇条書きの親子関係で説明されている「階段」を作ってはいけない

# はじめに

- プロットとは：
  - ◇ 文章の論理構造を箇条書きの形でまとめたもの
  - ◇ 話の筋をまとめたもの，と考えてもよい
- 文章や発表スライドを書く前に，まずプロットを作る必要がある
  - ◇ これはいわば文章やスライドの設計図にあたるもの
  - ◇ いきなり文章を前から順に書き始めてはいけない
    - 論理的な構造をきちんと設計したあとで書き始める

# なぜプロットを作るのか？

- 話の筋を整理するため
  - ◇ 何が背景で、何が課題で、何をどう解決したのかを明確にする
- その筋に収束するよう文章を書くと、主張を明確に示すことが出来る
  - ◇ そうしないと、「言いたいことがなんとなく適当に並べられた良くわからないもの」が出来上がる
  - ◇ 設計図なしで建物を建てるとヒドい事になるのと同じ
- 論文の章構成レベルの設計をしているとも言える
  - ◇ これを先にやっておかないと、後から大きな手戻りが発生する
  - ◇ 文章の修正が細かい手直しではすまなくなる

# 3点プロットとは

- 3点プロットはいわば「プロットのプロット」
  - ◇ 「背景→課題→提案」の3要素が何なのかをまとめたもの
  - ◇ 各要素が何なのかをはっきりさせ、その流れを明確にする
  - ◇ これを膨らませて全体のプロットなどを作る
- いきなり全体のプロットを作るのは難しい
  - ◇ 考えることが多い
  - ◇ 直すのも大変
- 3点プロットは1ページに収まるので、お手軽

# もくじ

1. 3点プロットとは
2. 背景, 課題, 提案
3. 内容のまとめかた
4. 箇条書きの作り方

# 3点プロット： 背景，課題，提案の3点で話の筋をまとめる

## 1. 背景：主張全体の背景や問題を説明する

- ◇ 課題と提案に共通する背景や，それらが共通して取り組んでいる問題
- ◇ 一般的な話題から始めて，課題や提案へ向けて話題を絞り込む

## 2. 課題：解決しようとしている課題を説明する

- ◇ 既存手法の問題点
- ◇ 既存手法がない場合は，背景の中の着目する問題を掘り下げる

## 3. 提案：課題であげられた問題を解決する提案手法を説明する

- ◇ 課題に対する洞察や観察
- ◇ キーとなるアイデア
- ◇ なぜ & どのように課題を解決できるのか

# 例 1 : セットアソシアティブ・キャッシュ = 既存手法があるパターン

## ■ 背景：キャッシュ

- ◇ 目的：プロセッサとメイン・メモリ間の速度差の解消
- ◇ 構造：高速/小容量なメモリであり，メイン・メモリの一部を保持

## ■ 課題：既存のキャッシュは性能 or 複雑さに問題がある

- ◇ ダイレクト・マップ：単純だが，競合によるヒット率低下が大きい
- ◇ フルアソシアティブ：ヒット率は高いが，大量の比較器が必要

## ■ 提案：セットアソシアティブ・キャッシュ

- ◇ 手法：複数のラインを同時に保持するセットを用いる
- ◇ 効果：
  - ダイレクトマップと比べて競合にある程度耐性があるため，ヒット率が高い
  - フルアソシアティブに比べて比較器の数は大幅に少なく単純



## 例 2 : 小泉くんの DATE = 既存手法があるパターン

- 背景 : 早いプリフェッチによるレイテンシの隠蔽
  - ◇ プリフェッチ : キャッシュにデータを先読みしておく技術
  - ◇ 多くの既存研究では十分に早くプリフェッチすることを重視
    - メモリ・アクセスのレイテンシを有効に隠蔽するため
    - 通常はなるべく遠い未来のアドレスを予測してプリフェッチ
- 課題 : 早すぎるプリフェッチ
  - ◇ 早すぎると, 使用される前にキャッシュから追い出される
  - ◇ これにより性能向上の機会が大きく失われている
- 提案 : プリフェッチを遅らせる
  - ◇ データが参照されるタイミングまであえて遅らせる

# 例 3 : 小田喜くんの輪講の例 = 既存手法があるパターン

(輪講なので具体的なアイデアがまだない事に注意)

## ■ 背景 : SIMT アーキテクチャにおける冗長な演算

- ◇ SIMT(D) では基本的には複数のデータに対して同じ演算を行う
- ◇ しかし SIMT では全く同じ冗長な演算を複数のレーンで行っている場合があります無駄

## ■ 課題 : スカラ化とその問題

- ◇ 冗長な演算を 1 つの演算にまとめるスカラ化が提案されている
- ◇ しかし, 従来のスカラ化では制約があり効果的に演算をまとめられない

## ■ 提案 : スカラ化の改良

- ◇ Temporal SIMT と動的なスカラ化の組み合わせにより実現

# 例 4 : 出川くんの ICCD = 既存手法がないパターン

- 背景：命令キャッシュ・ミス数を使った性能の見積もり
  - ◇ 従来，命令キャッシュに関わる研究ではミス数が主要な評価項目だった
  - ◇ 理由：ミス数が減ると基本的には実行時間が短くなるため
- 課題：精度とシミュレーション時間
  - ◇ 動機：
    - 現代の複雑化したプロセッサではミス数と実行時間が直接相関しない
    - 精度よい性能見積もりのためにはプロセッサ全体のシミュレーションが必要
  - ◇ 問題：しかしそのようなシミュレーションには長い時間かかる
- 提案：命令キャッシュ・ミス数に代わる新たな指針
  - ◇ 手法：新たな指針と，その指針を使った高速な性能見積もり
  - ◇ 効果：2桁短い時間でシミュレーションとほぼ同じ精度の性能見積もりを実現

# 例 5 : 木村さんの輪講 = 既存手法がないパターン

## ■ 背景 : ベクトル命令

- ◇ 単一の命令で可変長の複数データを処理する命令の方式
- ◇ データ並列性のある処理を対象
- ◇ 例 : RISC-V ベクトル拡張など

## ■ 課題 : ベクトル命令の実装コスト

- ◇ ベクトル命令では 1 つの命令が多数のアクセスを発生させる
- ◇ 従来の作り方で out-of-order プロセッサ上に実装すると, 複雑さが爆発する

## ■ 提案 : ベクトル命令の実装の複雑さを下げる

- ◇ 部分的な in-order 実行の導入による複雑さの緩和
- ◇ in-order/out-of-order 部の軽量な同期方法の提案

# もくじ

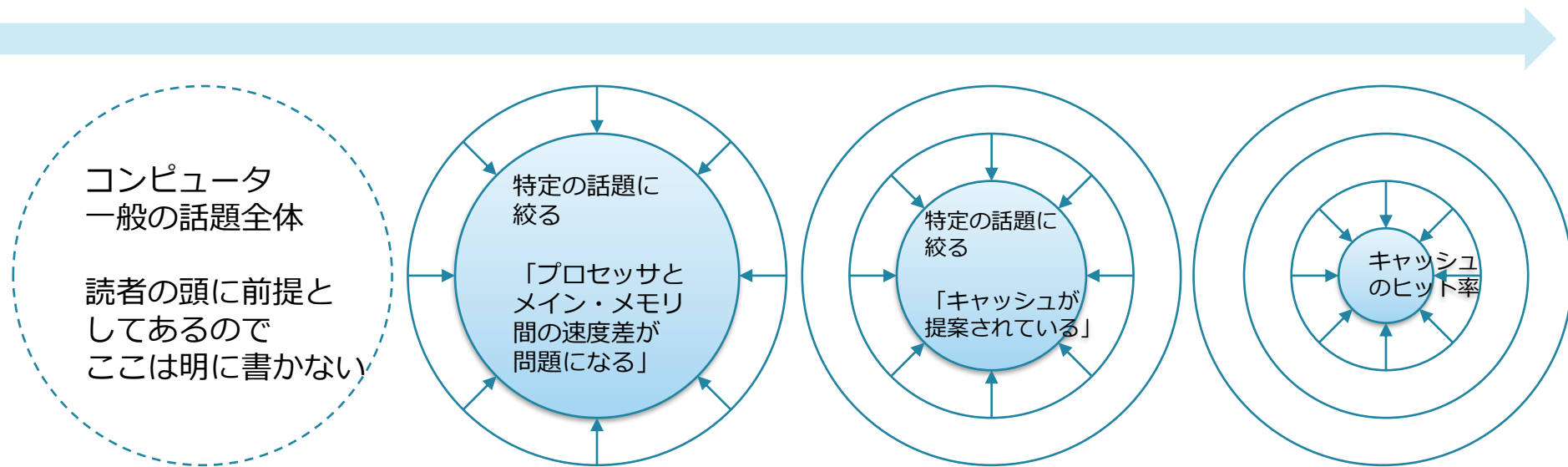
1. 3点プロットとは
2. 背景, 課題, 提案
3. 内容のまとめかた
4. 箇条書きの作り方

# 背景について

- 「背景」はプロット全体の議論の場を設定する
  - ◇ まず、何についての話題なのか
  - ◇ その話題において何が主に問題となるのかや、目的は何なのか
  - ◇ 「課題」や「提案」で話す内容に話題を絞っていく
- たとえば、前述の例の課題の場合：
  - ◇ 「キャッシュ」
  - ◇ 「早いプリフェッチによるレイテンシの隠蔽」
  - ◇ 「SIMT アーキテクチャにおける冗長な演算」
  - ◇ 「命令キャッシュ・ミス数を使った性能の見積もり」
  - ◇ 「ベクトル命令」

# 背景の話題の絞り方

- 一般的な事項から話題を絞っていく
  - ◇ 「課題」や「提案」の話題がちょうど含まれるところまで絞る
  - ◇ 絞りきったところで,
    - それを一言にまとめたものを「背景」のトップレベルに書く
    - その下に絞っていく過程をぶら下げる



# 背景を絞る例（小泉くんの DATE の背景）

- 背景：早いプリフェッチによるレイテンシの隠蔽
  1. メイン・メモリのアクセス・レイテンシが大きな問題に
  2. プリフェッチ：キャッシュにデータを先読みしておく技術
  3. 多くの既存研究では十分に早くプリフェッチすることを重視
    - メモリ・アクセスのレイテンシを有効に隠蔽するため
    - 通常はなるべく遠い未来のアドレスを予測してプリフェッチ
- 上記の背景は、以下のようにして話題を絞り込んでいる
  - ◇ 「1. メモリのレイテンシが問題」 →
  - ◇ 「2. プリフェッチによる解決」（レイテンシ問題の解決法の1つ） →
  - ◇ 「3. 早いプリフェッチの重視」（プリフェッチの性質の1つ）
- このようにして、その後の「課題：早すぎるプリフェッチ」と「提案：プリフェッチを遅らせる」に繋げている



# 課題について

- 「課題」は、良くない事を示すネガティブな語句や文を必ず含む
  - ◇ 「～が悪い」「～できない」など
  - ◇ これにより、なにが問題であるのかを明示する
- たとえば、前述の例の課題の場合：
  - ◇ 「従来のスカラ化では制約があり効果的にまとめられない」
  - ◇ 「性能向上の機会が大きく失われている」
  - ◇ 「複雑さが爆発する」
  - ◇ 「シミュレーションには長い時間かかる」

# 項目間の関係

- 「課題」に書く内容は、「背景」で提示した問題に対応させる
  - 1. 既存手法がある場合：
    - 「背景」で提示した問題を解決する
    - （基本的にはこの形になることが多い）
  - 2. 既存手法がない場合：
    - 「背景」の特定の問題に着目して掘り下げる
- 「提案」では、「背景」と「課題」で提示した問題に対応させる
  - ◇ 基本的には「課題」で提示した問題を解決する方法を書く
  - ◇ そうすれば、普通は自然と「背景」の問題にも対応する

# 応用：4点プロット

## ■ 洞察を加えた4点プロットの形でもよい

- ◇ 背景

- ◇ 課題

- ◇ 洞察

  - 課題に対する新しい観察結果や、課題の核心の新しい解釈など

- ◇ 提案

## ■ 洞察は課題や提案の下にぶら下がることもある

- ◇ 小泉くんの例での「早すぎるプリフェッチ」は洞察でもある

# もくじ

1. 3点プロットとは
2. 背景, 課題, 提案
3. 内容のまとめかた
4. 箇条書きの作り方

# とりあえず、この形にまとめる事を目指す

「～」となっているところは、一つの文ないしは名詞による説明

## 1. 背景：「～」

- ◇ 「～」 = 背景を説明する 1 つの文
- ◇ 「～」 …

## 2. 課題：「～」

- ◇ 「～」
- ◇ 「～」

## 3. 提案：「～」

- ◇ 「～」
- ◇ 「～」

# 作り方の例

- 作り方の例

1. ボトムアップな作り方
2. 課題から掘り下げる作り方

- やりやすいようにやれば良いし, 上記を組み合わせても良い

- ◇ 他にも色々なやり方があると思う

# ボトムアップな方法

1. まずは思いつく関連しそうな項目をたくさん書き出してみる

- ◇ 名詞や短文の形にして並べてみる

2. 各項目を整理

1. 関係する項目同士をくくって親子にまとめる

- それらを一言で表した, まとめの短文（親）を作る
- 親の下に, それらを子項目としてインデントして置く

2. 内容が冗長なものをマージしたり削除する

3. ある程度まとまったら, それらを背景, 課題, 提案に分類する

- ◇ このとき, 「項目間の関係」のページで説明した関連を意識する
- ◇ うまく分類して関係を説明できない時は, 本質的に関係ない可能性がある

# 課題から掘り下げる方法

- まず「課題」は何であるのかから考える
  - ◇ なにが問題なのかを端的にまとめる
  - ◇ 「～が悪い」「～が遅い」「～の効率がよくない」などの形の1文に出来るとよい
- 次にそれを「提案」がどのように解決しているのか考える
  - ◇ 上でまとめた問題が、「どのように」「なぜ」解決されているのかをまとめる
- それら「課題」と「提案」に話題を絞り込んでいくような「背景」を考える



# もくじ

1. 3点プロットとは
2. 背景, 課題, 提案
3. 内容のまとめかた
4. 箇条書きの作り方

# 文を短くする

## ■ 複文を使うのは原則禁止

- ◇ 複数の文を繋げて 1 つの文にしてしまうと、関係が良くわからなくなりがち
- ◇ 最初は単文のみで作る
  - どうしても複文を入れる場合は、1 行に収まる長さまで
  - 3 文以上からなる複文は常に禁止

## ■ 同様の理由により、長い修飾節を持った文も使わない

- ◇ 長くなってしまう場合は、インデントをしてぶら下げると良い

# 箇条書きの親子関係の作り方

- インデントされた子要素の部分と、その親の関係
  - ◇ 子項目は、その親項目のなんらかの詳細を説明する
  - ◇ 親項目は、その子項目をまとめた内容となる
- 項目の「属性」や「話題」に従ってくくっていくと良い

# 属性による親子関係の確認


- 各項目の先頭に一言にまとめた属性をつけて作ると確認しやすい
  - ◇ 子から見て親の何であるのかを属性としてつける
  - ◇ 「問題：」「理由：」「結果：」「目的：」「例：」「詳細：」など
- 属性をつけた例：
  - ◇ 背景：ベクトル命令
    - 詳細：単一の命令で可変長の複数データを処理する命令の方式
    - 目的：データ並列性のある処理を対象
    - 例：RISC-V ベクトル拡張などの形で実装されている
- 属性は、プログラミング言語における型の概念に似ている
  - ◇ 構造体や配列と似たような構造化の考え方ができる
  - ◇ 「違う型のものを配列に入れてはいけない」のような概念が応用できる

# 共通の属性をくくる

- 同じレベルにある複数の同じ属性の項目は、くくって1つにする
  - ◇ 複数の同じ属性（型）のものが別の属性（型）と同じレベルに並んではいけない
- たとえば、2つの「例：」が「詳細：」と同じレベルにあるような以下の場合：
  - ◇ 「例：」をくくって1つの配列にする
  - ◇ 背景：ベクトル命令
    - **詳細**：単一の命令で可変長の複数データを処理する命令の方式
    - **例**：RISC-V ベクトル拡張
    - **例**：NEC SX
  - ◇ 背景：ベクトル命令
    - **詳細**：単一の命令で可変長の複数データを処理する命令の方式
    - **例**：
      - \* RISC-V ベクトル拡張
      - \* NEC SX



# 共通の話題をくくる

- 同じレベルにある複数の同じ話題の項目も、くくって1つにする
    - ◇ 属性とは直行している
  - たとえば、「ベクトル命令」について話している以下のような場合：
    - ◇ それらを「ベクトル命令」でくくる
    - ◇ 詳細：ベクトル命令とは単一の命令で可変長の複数データを処理する命令の方式
    - ◇ 例：ベクトル命令には以下のような実装の例がある
      - RISC-V ベクトル拡張
      - NEC SX
- 
- ◇ ベクトル命令：
    - 詳細：単一の命令で可変長の複数データを処理する命令の方式
    - 例：実装の例：
      - \* RISC-V ベクトル拡張
      - \* NEC SX

# インデントにぶらさげる項目数

- 1つの項目にぶら下げる項目は3つまで
  - ◇ 4つ以上の項目が同じレベルに並んでいる場合は、それらをインデントにまとめる
  - ◇ 3つより多いと関係が曖昧になるし、理解しづらい

# 箇条書きの親子関係における「階段」

- $A \rightarrow B \rightarrow C$  のような演繹の関係を下のような「階段」のような箇条書きに  
してしまいがちだが、これは良くない

- ◇ (親子関係は基本的に 概要 $\leftrightarrow$ 詳細 を表すもの)

- ◇ A : ~

- B : ~

- \* C : ~

- このような場合は  $A \rightarrow B \rightarrow C$  の主張を一言にまとめたものを (X) を作り、  
その下にぶらさげる

- ◇ X :

- A : ~

- B : ~

- C : ~



# 演繹の関係にある要素の書き換えの例

## A→B→C を X の下に展開

各インデントレベルに 1 つだけ項目がある階段が出来てしまっている

- A : 人間の脳の一時記憶の大きさには限りがある
  - ◇ B : このため, 人間は 5 ~ 7 個以上の事柄を一度に把握できない
    - C : したがって, 余裕を持って 3 個程度以内にするのがよい

X に全体をまとめる一言を入れて, その下に並列にぶら下げると良い

- X : 1 項目にぶら下げる項目の数は少なめにする
  - ◇ A : 人間の脳の一時記憶の大きさには限りがある
  - ◇ B : このため, 人間は 5 ~ 7 個以上の事柄を一度に把握できない
  - ◇ C : したがって, 余裕を持って 3 個程度以内にするのがよい

# まとめ

- 3点プロットの作り方について説明
  - ◇ 話の筋を背景, 課題, 提案の3点で, 1ページにまとめたもの
  - ◇ コンパクトなので考えやすい
- これが出来たら, 次に全体のプロットなどを作る