並列ソフトウェアパイプライン

今田俊寛

目的

- **超スレッド並列環境に於いての(コンパイラによる)ソフトウェアパイプライン最適化を検討する。**

課題

- 一般に、ソフトウェアパイプラインは逐次最適化である。

- 何故か？
  - 一般にスレッド並列時には、各スレッドにおける配列のアクセスは飛び飛びの (stride) アクセスとなる。

  - 逐次最適化と同様にソフトウェアパイプライン化を施しても、データの先読み効果は無い。
    - (かつ、別のスレッドのデータを参照する事になる為、その時点で必要なデータの書き込みが完了している事を保証してやる必要がある)。
    - スレッド間に依存関係が付くだけ。
      - ⇒ 問題が多い。

解決法1(誘導変数の増分である記述法の導入)

- **まずは、前提として以下のような書き方を可能とする。**

```
#pragma omp parallel for
  for (i=0; i<max; i++) {
    d[i] = a[i] * b[i] + c[i];
  }
```

具体的なpragmaの記述法は、ここでは問題ではなく、このように記述する事で、並列化を指示出来る事が重要。

- **これは、並列記述上では**

```
for (i=tid; i<max; i+=maxthread) {
  d[i] = a[i] * b[i] + c[i];
}
```

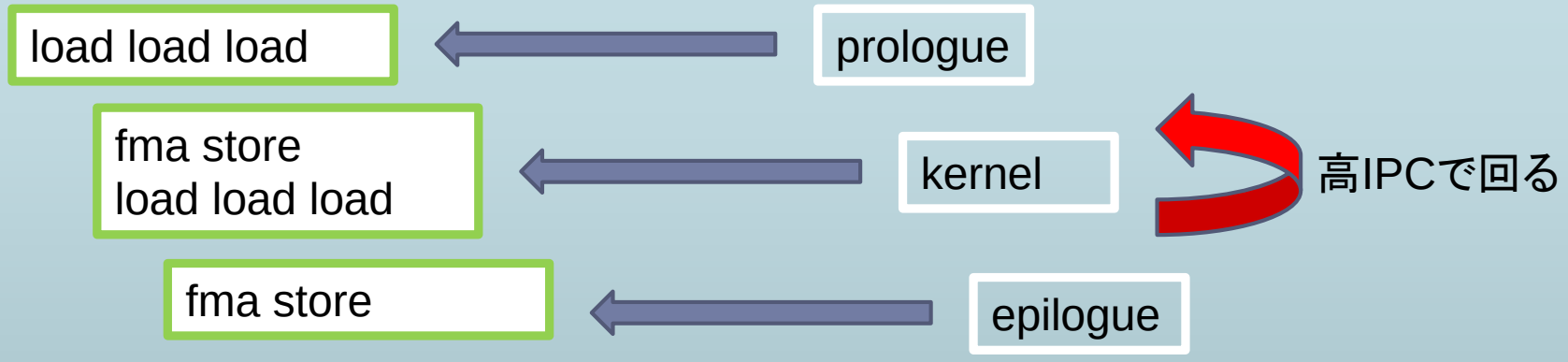
- **と等価であるとする。**

解決法2(逐次的ソフトウェアパイプライン化)

- **前スライドの以下のコードにpragmaを意識せずに、“そのまま”ソフトウェアパイプライン最適化を施す**

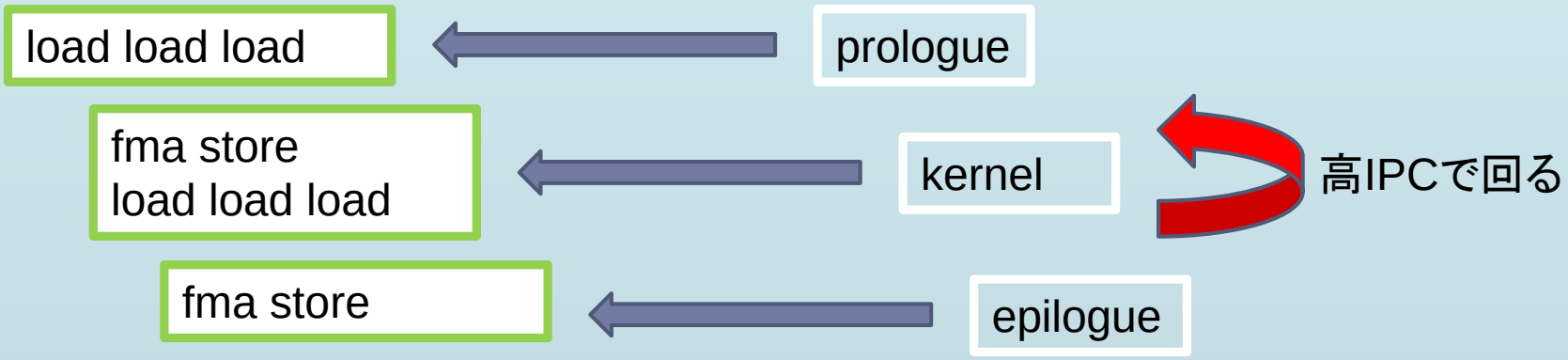
```
#pragma omp parallel for
  for (i=0; i<max; i++) {
    d[i] = a[i] * b[i] + c[i];
  }
```

- **非常に単純には以下のイメージとなる。**



解決法3(ソフトウェアパイプラインの本来の効果)

- **前スライドのイメージ**

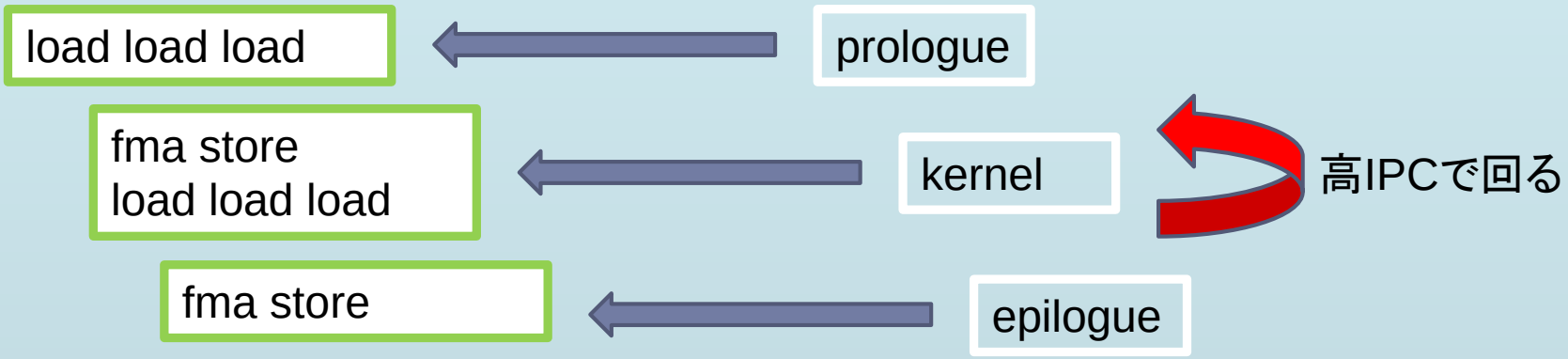


- **ここで、重要なのは、**

kernel は kernelのいくつか先の 回転のload を先読みする(MVE<Modulo Variable Expansion>の数に依存)と言う事。

解決法4(逐次的な最適化に於ける問題)

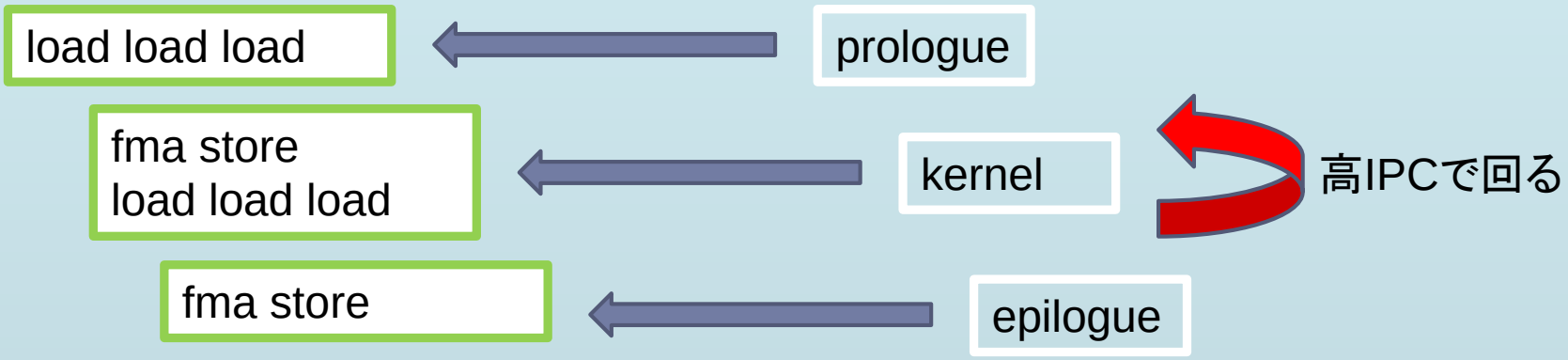
- **イメージ**



- **kernel は kernelのいくつか先の回転の load を先読みする**
  - **しかし、このままの逐次的な最適化イメージでは、各スレッドに於けるメモリアクセスの先読みにならないので、効果は期待出来ない。(かつ、各スレッドに依存関係が出来る為、逆効果(！)ですらある。)**

解決法5(誘導変数)

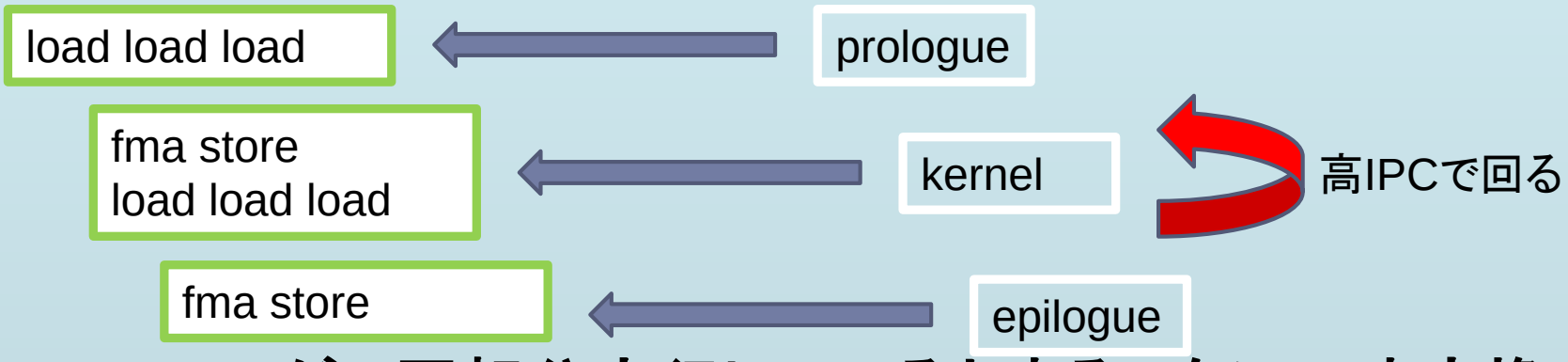
- **イメージ**



- **逐次イメージのままだと、(例えば)スレッド3のメモリアクセスをスレッド0が先読みするだけで嬉しくない！**
  - **8192スレッド並列なら、8192先の要素のメモリアクセスを先行して実行したい。**
  - **そこで、誘導変数の変換を行う。**

解決法6(誘導変数の変換)

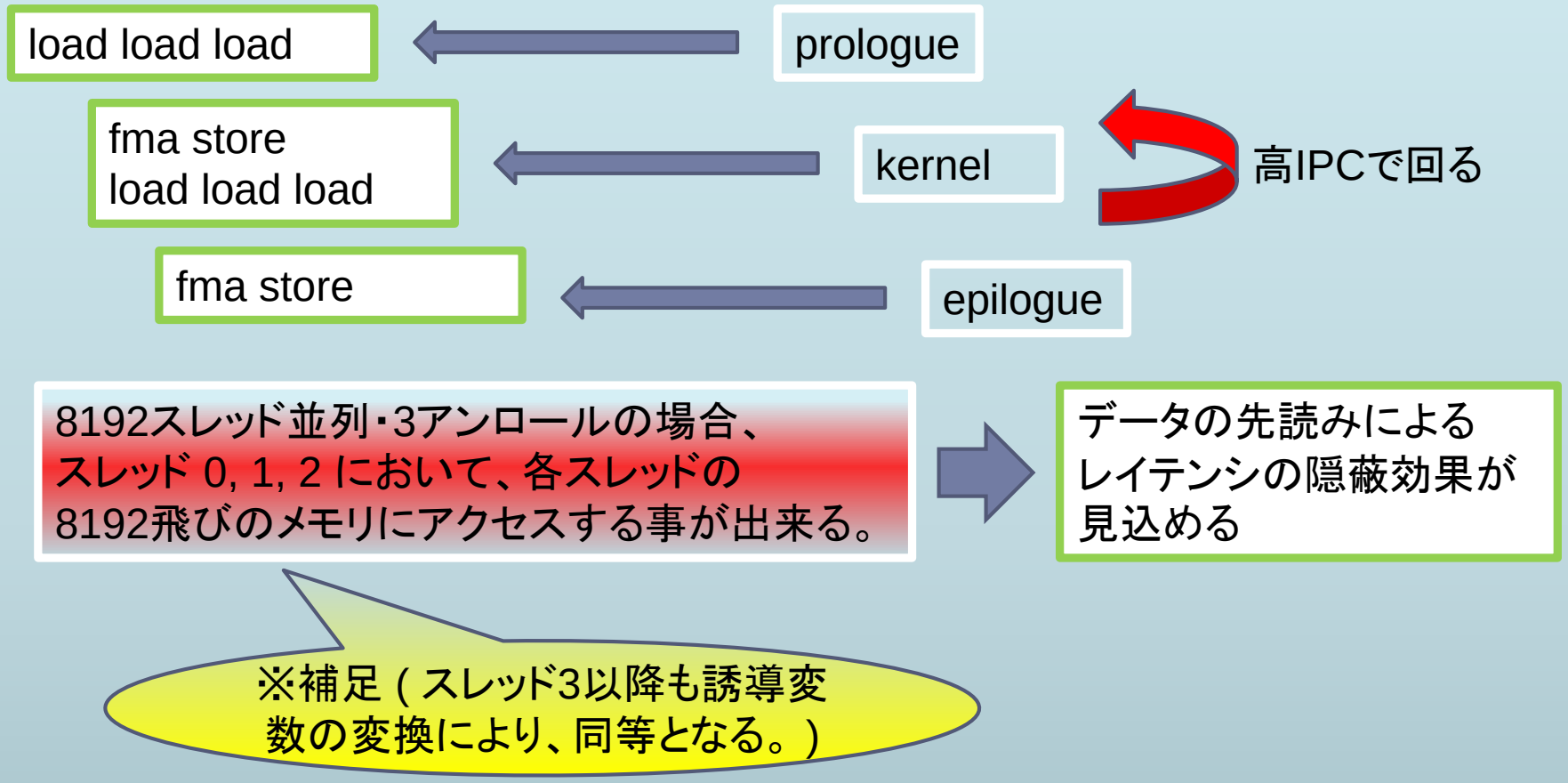
- イメージ



- kernel が 3回転分実行しているとする。次に示す変換を行う。
- 0 → 0　　3 → 1
- 1 → 8192　　4 → 8193
- 2 → 16384　　5 → 16385

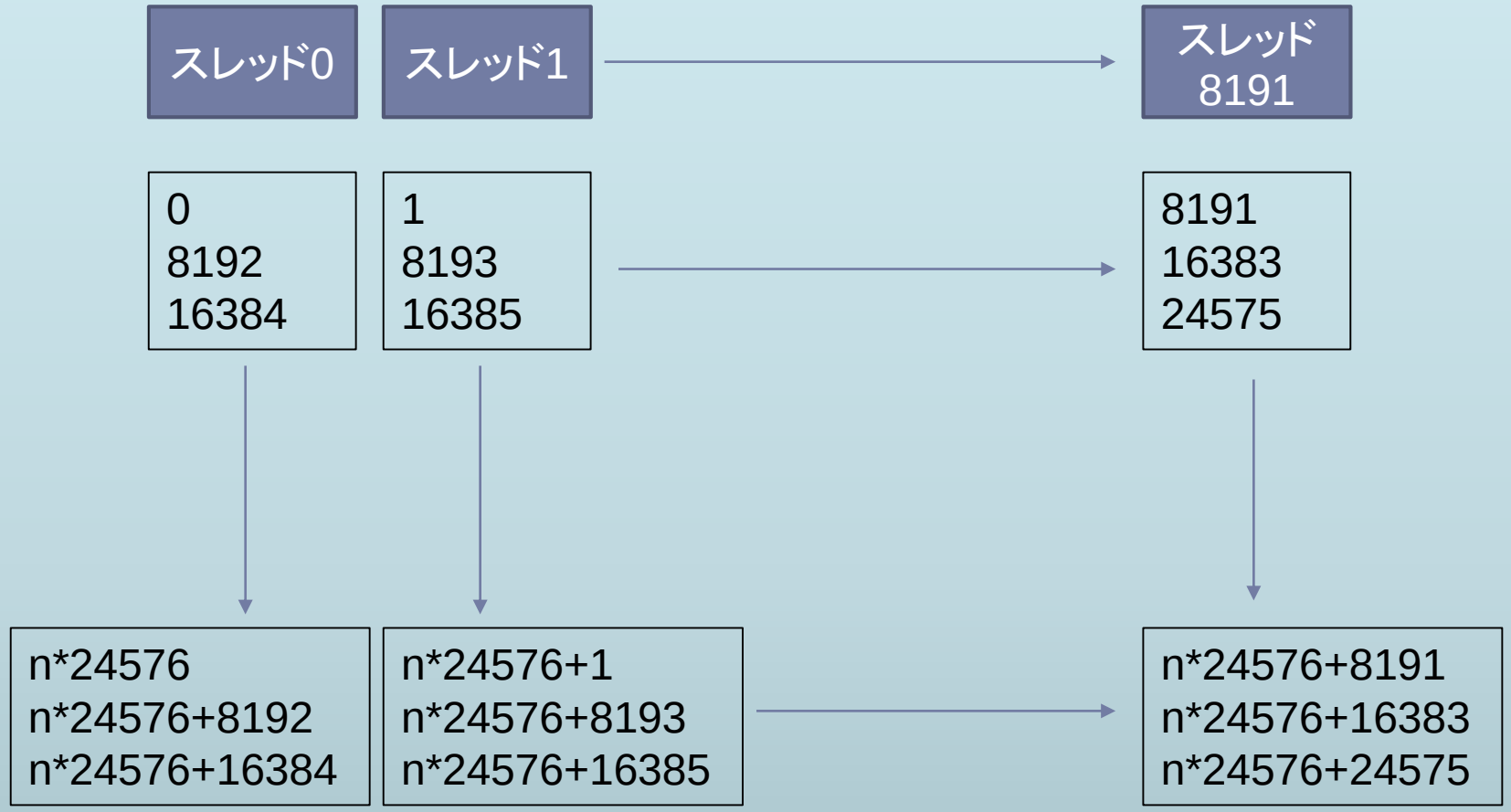
解決法7(誘導変数の変換により何が起きるか)

- **イメージ**



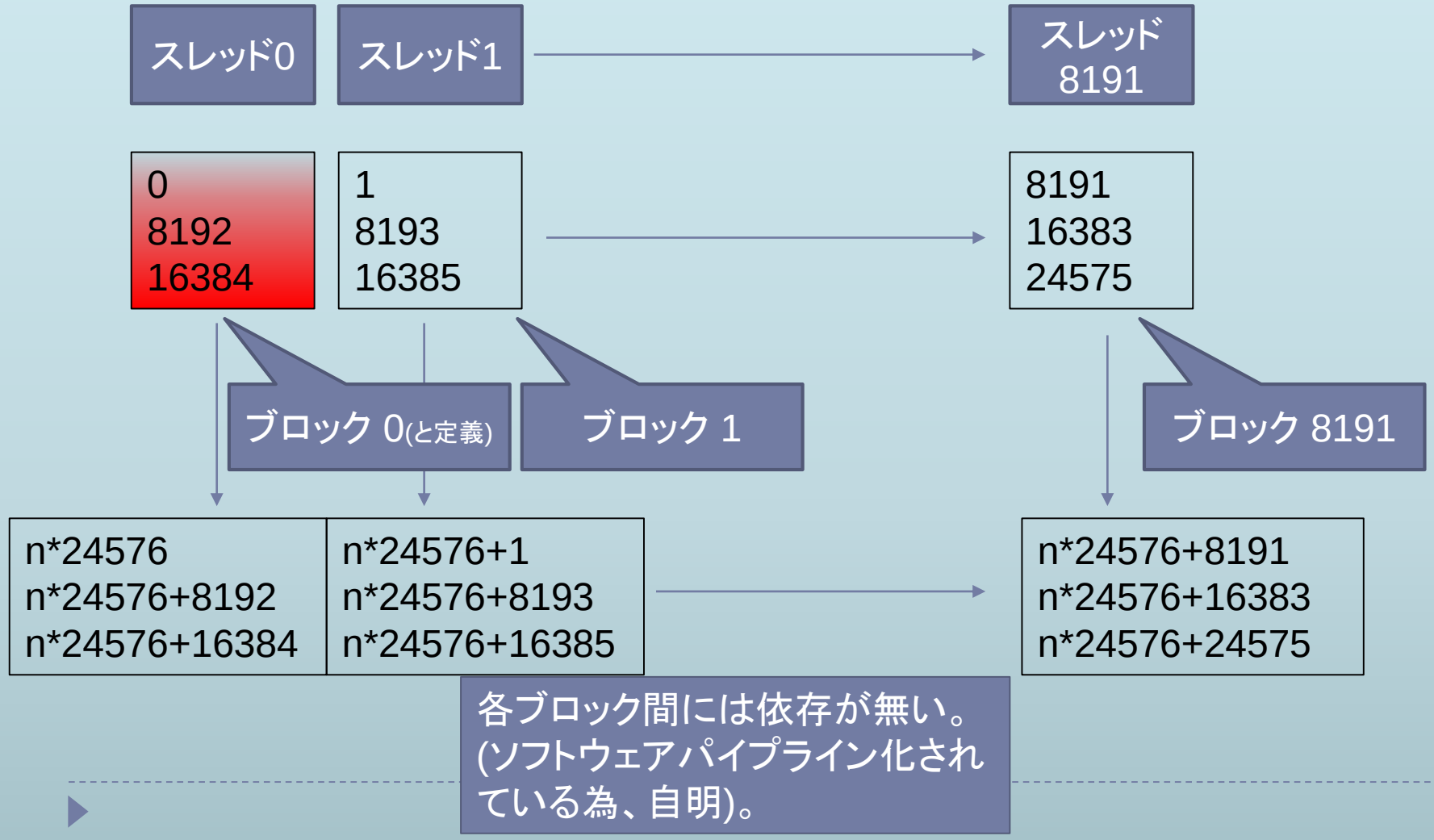
解決法8(各スレッドの実行イメージ)

- **イメージ**



解決法9(前提条件)

- イメージ



解決法10

前スライドより・・・

各ブロック間には依存が無い。

各ブロックは、ソフトウェアパイプラインにより、アンロール(厳密にはMVE)されている。
従って、8192 分離れた要素へのアクセスを先行して行える。

一般に、逐次(横？)方向の最適化である
ソフトウェアパイプラインを
並列(縦？)方向に行える！

ソフトウェアパイプラインなので、ある回転において load した値は次以降(1以上後)の回転で使われる。
また、(レジスタの足りる範囲で)任意のレイテンシを隠蔽可能。
※参照 https://www.ssken.gr.jp/MAINSITE/download/newsletter/2012/20120820-sci-1/lecture-05/SSKEN_sci2012-1_oinaga_presentation.pdf のページ18

何が新しい？

- **明示的なスレッド並列では・・・**
  - 配列アクセス・誘導変数の増分値が飛び飛びとなる。
  - その為、一般的なソフトウェアパイプライン化による性能向上を図るのは困難であった。

- **逐次的な記述から並列プログラムを自動生成する仕組みを導入**
  - 本質的には並列前提のプログラム
  - → 逐次化(pragmaの利用を想定)
  - → 逐次最適化(ソフトウェアパイプライン)
  - → 並列プログラムに逆変換(誘導変数の変換)

- **MIMDを活かした最適化**

結論

- 並列プログラムを簡単に行いたい。
  - 逐次プログラムからpragma等で自動並列化させる仕組みを用意する。
  - ここまでなら普通にある考え方(それ程新しくない)。
- 今回は逐次プログラムとして記述(+α)する事で、並列環境でのソフトウェアパイプライン最適化を可能とする。
- 結果として、記述の容易さ(見かけ上、コードは逐次)・性能上のメリット(メモリアクセスの先読み効果)が得られる。

終わり