社会シミュレーションを並列化するための新しい言語機構の研究へ向けて

夏 澄彦　佐藤 芳樹　千葉 滋 @東京大学

社会シミュレーションを並列化して、
HPC上で動かしたい！！

社会シミュレーションの特徴

- 計算粒度が大きく、依存関係が複雑になることが多い
- 社会シミュレーションの研究者はコンピュータの専門家ではない

並列化しづらい

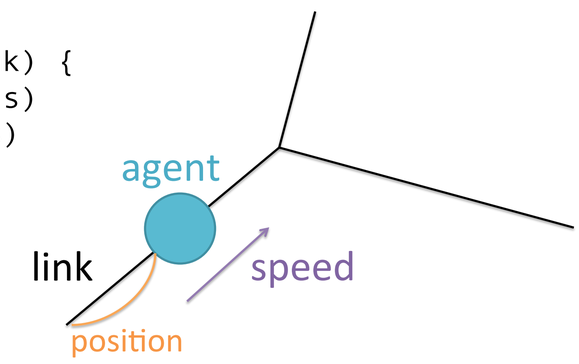
目標

計算の依存関係を記述し、容易に並列化できる言語機構の開発

現在、産業総合研究所と連携し、歩行者シミュレータ「CrowdWalk」をHPC環境へ移植中

例）歩行者シミュレーション

- 歩行者（agent）はグラフ構造の道路（link）を辿り、ゴールまで進む
- 単位時間ごとにagentの速度、位置を再計算し、移動させる



1. 並列化しよう

Java 8から導入される並列コレクションとlambdaを利用

```
class Agent {
  void update() {
    speed = calcSpeed()
    nextPlace = calcNextPlace()
    if (place.link != nextPlace.link) {
      nextPlace.link.agents.add(this)
      place.link.agents.remove(this)
    }
    place = nextPlace
  } }

agents.map(agent -> agent.update)
```

```
void update() {
  speed = calcSpeed()
  nextPlace = calcNextPlace()
  if (place.link != nextPlace.link) {
    synchronized (nextPlace.link.agents) {
      nextPlace.link.agents.add(this)
      place.link.agents.remove(this)
    }
  }
  place = nextPlace }

agents.parallel().map(agent -> agent.update)
```

```
class Agent {
  Double speed,
  Place place,
  Place nextPlace }

class Place {
  Link link,
  Double position }

class Link {
  List<Agent> agents }
```

2. 衝突回避を導入したい

- linkにはcapacityがある
- 先にlinkに辿り着いたagentが優先され、入りきれなかったagentはlink手前で停止する

linkに辿り着いた順では無く、Threadが割り当てられた順に実行される（シミュレーションモデル×、再現性×）

```
void tryToMoveToNextPlace() {
  if (nextPlace.link.agents.size < nextPlace.link.capacity) {
    nextPlace.link.agents.add(this)
    place.link.agents.remove(this)
    place = nextPlace
  } else {
    place.position = link.length } }

void update() {
  speed = calcSpeed()
  nextPlace = calcNextPlace()
  if (place.link != nextPlace.link) {
    synchronized (nextPlace.link.agents) {
      tryToMoveToNextPlace() } } }

agents.parallel().map(agent -> agent.update)
```

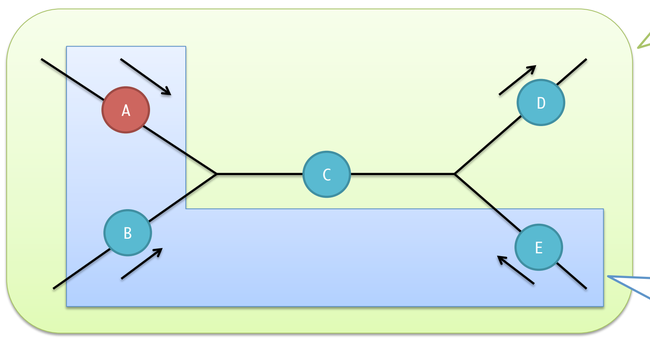
並列コレクションの処理の一部で、部分集合を指定し、バリア同期や実行の順序関係を制御できる必要がある

```
void update() {
  speed = calcSpeed()
  nextPlace = calcNextPlace()
  if (place.link != nextPlace.link) {
    linkLeftTime = (place.link.length - place.distance) / speed
    waitOthers (nextPlace.link.neighborAgents) {
      orderIf (another -> nextPlace.link == another.nextPlace.link,
               another -> linkLeftTime - another.linkLeftTime) {
        tryToMoveToNextPlace()
      } } } }

agents.parallel().map(agent -> agent.update)
```

aspect

```
void update() {
  speed = calcSpeed()
  nextPlace = calcNextPlace()
  if (place.link != nextPlace.link) {
    tryToMoveToNextPlace()
  }
  position = nextPosition }

agents.parallel(call(tryToMoveToNextPlace).around {
  linkLeftTime = (place.link.length - place.distance) / speed
  waitOthers (nextPlace.link.neighborAgents) {
    orderIf (another -> nextPlace.link == another.nextPlace.link,
             another -> linkLeftTime - another.linkLeftTime) {
      proceed()
    } } }).map(agent -> agent.update)
```

- **`waitOthers (agents) { ... }`**
  指定したagentsがこのブロック直前まで実行が終わるまでwaitする



- **`orderIf (condition, comparable) { ... }`**
  conditionがtrueになるような他のagentsがいた場合、そのagentsはcomparable順にブロックが逐次実行される