Parallel Programming with Object Assemblies

Sławomir Kierat

Uniwersytet Warszawski

25 maja 2010

Plan prezentacji

1. Wstęp
   - Motywacja
   - Zaproponowane rozwiązanie
2. Chorus - podstawy
   - Sterta
   - Zespoły obiektów
   - Łączenie
   - Podziały
   - Przykład - triangulacja Delaunay
3. Chorus - formalnie
   - Składnia
   - Semantyka
   - Odporność na wyścig i zakleszczenie
4. JChorus
   - Składnia obiektowa
   - Części sekwencyjne i równoległe
   - Przykład
   - Implementacja JChorusa
5. Podsumowanie

OOPSLA 2009

- autorzy:
  - Roberto Lublinerman
  - Swarat Chaudhuri
  - Pavol Černý

Rysunek: Pavol Cerny

Rysunek: Swarat Chaudhuri

1 Wstęp
- Motywacja
- Zaproponowane rozwiązanie

2 Chorus - podstawy

3 Chorus - formalnie

4 JChorus

5 Podsumowanie

Programowanie równoległe

- zapotrzebowanie na aplikacje równoległe
- powszechność tanich wielordzeniowych maszyn
- wąskim gardłem wydajności staje się software
- obecne modele programowania:
  - zbyt niskopoziomowe
  - skomplikowane
  - podatne na błędy
  - źle się skalują
- zapotrzebowanie wysokopoziomowy model programowania równoległego, który jest:
  - prosty, intuicyjny
  - skalowalny

Programowanie równoległe w skomplikowanych aplikacjach

- nietrywialna równoległość
- operacje na zaawansowanych strukturach danych
- niemożliwe zapewnienie równoległości na etapie kompilacji
- obecnie brak odpowiedniego modelu dla problemów typu:
  - symulacje fizyczne
  - symulacje epidemiologiczne
  - udoskonalanie siatek (triangulacja)
  - tworzenie drzew rozpinających
  - symulacje n-ciał
  - analizy sieci społecznościowych
  - obliczenia na macierzach rzadkich

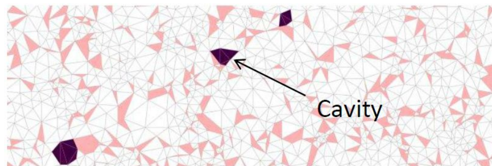
1. Wstęp
   - Motywacja
   - Zaproponowane rozwiązanie
2. Chorus - podstawy
3. Chorus - formalnie
4. JChorus
5. Podsumowanie

Zaproponowane rozwiązanie - Chorus

- główne cechy Chorusa:
  - lokalność
  - dynamizm
- cechy modeli znanych z Javy lub C#
  - globalna pamięć dzielona
    - ogranicza programowalność
    - ogranicza wydajność
  - problemy z wyścigiem i zakleszczeniem
    - dodatkowy wysiłek programisty
- model oparty o blokady
- model oparty o nieblokujące transakcje
- globalna sterta:
  - trudna do zanalizowania
  - statyczny podział za mało elastyczny

Sztandarowy przykład w prezentacji Chorusa

- Problem mesh refinement (udoskonalanie siatki)
- Siatka trójkątów - nie wszystkie trójkąty spełniają wymagania jakościowe
- lokalność wykonywanaych zmian
- Esencja problemu równoległości:
  - ubytki, które nie nachodzą na siebie można naprawiać rónolegle
  - potrzebny model synchronizacji oparty na lokalności i sąsiedztwie



Rysunek: Ubytki w siatce

Cechy Chorusa

- nie jest globalny jak model w Javie
- niestatyczny podział pamięci dzielonej
- brak niskopoziomowych mechanizmów
- obiektowy
- dynamiczne definiowanie lokalnej przestrzeni na stercie
- model oparty o zespoły z trzema możliwymi akcjami
  - operacje pisania/czytania w ramach jednego zespołu
  - łączenie zespołów
  - podział zespołów
- duża liczba zespołów - potencjalna duża równoległość
- zespoły - proxy do realizacji równoległości
- liczba zespołów nieograniczona

Zobrazowanie na przykładzie

- operacje łączenia i podziału trójkątów mogą wykonywać się równolegle
- schemat działania
  - zdobądź swój własny region
  - operuj na nim
  - zwolnij go
- schemat działania na przykładzie
  - połącz trójkąty (zespoły) tworzące ubytek
  - oblicz nowy podział na trojkąty
  - podziel się na nowe trójkąty (zespoły)

1 Wstęp

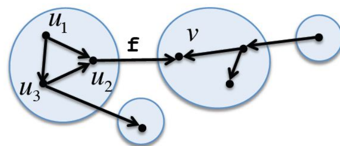
2 Chorus - podstawy
- Sterta
- Zespoły obiektów
- Łączenie
- Podziały
- Przykład - triangulacja Delaunay

3 Chorus - formalnie

4 JChorus

5 Podsumowanie

Sterta



Rysunek: Sterta

- charakter grafu skierowanego
- wierzchołki to obiekty (np. trójkąty)
- krawędzie to odpowiednie wskaźniki (np. sąsiedztwo)
- zespoły to zbiory wierzchołków wraz z krawędziami pomiędzy tymi wierzchołkami (np. ubytki)

1 Wstęp

2 Chorus - podstawy
- Sterta
- Zespoły obiektów
- Łączenie
- Podziały
- Przykład - triangulacja Delaunay

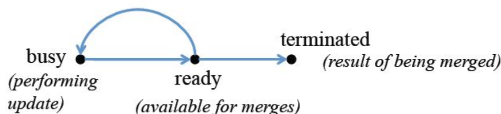
3 Chorus - formalnie

4 JChorus

5 Podsumowanie

Zespoły obiektów

- każdy obiekt należy do dokładnie jednego zespołu
- zespoły obiektów są izolowane od siebie - mechanizm zapewniający poprawność działania
- stany zespołów:
  - gotowy (stan początkowy) - niedeterministyczny wybór zadania
  - zajęty (wykonujący update)
  - zakończony (zmergowany)



Rysunek: Stany

- model wykonywania Guard:Update

1 Wstęp

2 Chorus - podstawy
- Sterta
- Zespoły obiektów
- Łączenie
- Podziały
- Przykład - triangulacja Delaunay

3 Chorus - formalnie

4 JChorus

5 Podsumowanie

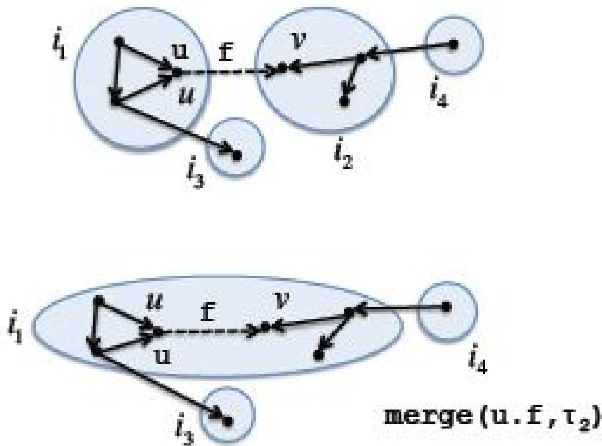
Łączenie

- jest to jedyna operacja synchronizacji - ograniczająca równoległość
- jeden z zespołów znika, drugi otrzymuje jego prace
- jedyny warunek konieczny - to bycie w stanie gotowy

Dostępne operacje

- $:: \mathit{merge}(u.f, \tau_2) : \mathit{Update}$
- $:: \mathit{merge}(u.f, \tau_2, v_1 := v_2) : \mathit{Update}$
- $:: \mathit{merge}(u.f, \tau_1', \tau_2[v_1' := v_1, v_2' := v_2]) : \mathit{Update}$
- $:: \mathit{merge}(u.f, \tau_2)\mathit{when}_g : \mathit{Update}$

Łączenie - zobrazowanie



1. Wstęp
2. Chorus - podstawy
   - Sterta
   - Zespoły obiektów
   - Łączenie
   - Podziały
   - Przykład - triangulacja Delaunay
3. Chorus - formalnie
4. JChorus
5. Podsumowanie

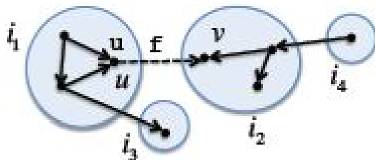
Podziały

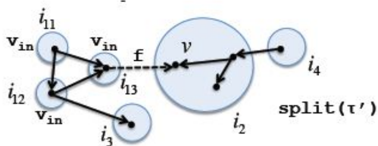
- jest to operacja odzyskiwania równoległości
- 2 typy:
  - podział jeden obiekt - jeden zespół
  - wydziel z zespołu zespół jednoobiektowy.

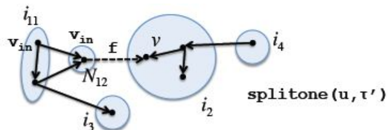
Dostępne operacje

- :: $split(\tau)$
- :: $splitone(u, \tau')$

Podział - zobrazowanie







Uwagi

- pozostałe operacje - to operacje na lokalnych obiektach.
- obiekty mogą mieć referencje do obiektów spoza regionu, ale nie mogą z nich korzystać (zgłaszany jest błąd)
- zespoły nigdy nie odwołują się do innych zespołów - operacje merge odbywają się na referencjach obiektów

1 Wstęp

2 Chorus - podstawy
- Sterta
- Zespoły obiektów
- Łączenie
- Podziały
- Przykład - triangulacja Delaunay

3 Chorus - formalnie

4 JChorus

5 Podsumowanie

Algorytm sekwencyjny

- zakładamy, że mamy wysokopoziomowe funkcje getBad itp.

kod sekwencyjny

```
1: Mesh m = /* read input mesh */
2: Worklist wl = new Worklist(m.getBad());
3: foreach Triangle t in wl {
4:   Cavity c = new Cavity(t);
5:   c.expand();
6:   c.retriangulate();
7:   m.updateMesh(c);
8:   wl.add(c.getBad()); }
```

- algorytm ma własność stopu

Algorytm w Chorusie

- Model w Chorusie:
  - obiekty = trójkąty; połączenia = sąsiedztwa
  - 2 rodzaje zespołów:
    - Triangle - odpowiada pojedynczemu trójkątowi - klasa inicjalna
    - Cavity - zespół trójkątów tworzący ubytek

kod w Chorusie

```
assembly Triangle:: ...
:: merge (v.f, Cavity, Triangle) when isBad: skip

assembly Cavity:: ...
:: merge (v.f, Cavity) when (not isComplete): skip
:: isComplete:
retriangulate(); split(Triangle)
```

1 Wstęp

2 Chorus - podstawy

3 Chorus - formalnie
- Składnia
- Semantyka
- Odporność na wyścig i zakleszczenie

4 JChorus

5 Podsumowanie

Składnia

$$
\begin{array}{lcl}
\mathit{Prog} & ::= & [\mathit{ADec}]^* \\
\mathit{ADec} & ::= & \texttt{assembly}\ \tau :: \texttt{local}\ [\mathtt{v}]^* \\
 & & \quad [::\ \mathit{Guard} : \mathit{Update}]^* \\
\mathit{Guard} & ::= & \texttt{merge}(\mathtt{v.f}, \tau_2\ [, \mathtt{v}_1 := \mathtt{v}_2]) \mid \\
 & & \texttt{merge}(\mathtt{v.f}, \tau_1', \tau_2\ [, \mathtt{v}_1' := \mathtt{v}_1]\ [, \mathtt{v}_2' := \mathtt{v}_1]) \mid \\
 & & \texttt{merge}(\mathtt{v.f}, \tau_2\ [, \mathtt{v}_1 := \mathtt{v}_2])\ \texttt{when}\ \mathit{Bexp} \mid \\
 & & \texttt{merge}(\mathtt{v.f}, \tau_1', \tau_2\ [, \mathtt{v}_1' := \mathtt{v}_1]\ [, \mathtt{v}_2' := \mathtt{v}_2]) \\
 & & \quad \texttt{when}\ \mathit{Bexp} \mid \mathit{Bexp} \\
\mathit{Update} & ::= & \mathtt{v} := \mathit{Exp} \mid \mathtt{v.f} := \mathit{Exp} \mid \mathit{Update}; \mathit{Update} \mid \\
 & & \texttt{skip} \mid \texttt{split}(\tau) \mid \texttt{splitone}(\mathtt{v}, \tau) \\
\mathit{Exp} & ::= & \mathtt{v} \mid \mathit{Exp}.\mathtt{f} \mid \texttt{error} \\
\mathit{Bexp} & ::= & \mathit{Exp} = \mathit{Exp} \mid \texttt{not}\ \mathit{Bexp} \mid \mathit{Bexp}\ \texttt{and}\ \mathit{Bexp}
\end{array}
$$

- Var - uniwersum zmiennych zespołu.
- T - uniwersum klas zespołów z klasa inicjalną ι
- $v, v_1, v_2, v_1', v_2' \in Var$
- $\tau, \tau_1', \tau_2 \in T$

- zmienne należą do odpowiedniej klasy
- żadne zespoły nie są deklarowane dwa razy
- istnieje deklaracja dla klasy inicjalnej

1. Wstęp
2. Chorus - podstawy
3. Chorus - formalnie
   - Składnia
   - Semantyka
   - Odporność na wyścig i zakleszczenie
4. JChorus
5. Podsumowanie

Definicje

- Loc - zbiór lokacji, F - nazwy etykiet

Definicja

Sterta - etykietowany graf skierowany $O \subseteq Loc$, $E \subseteq OxFxO$ taki, że dla każdego $u \in O$ i $f \in F$ istnieje co najwyżej jedna krawędź $(u, f, v) \in E$

Definicja

sterta H jest regionem w stercie G, jeżeli jest podgrafem generowanym przez podzbiór wierzchołków z G.

Definicja

Zespół - to krotka $N =< i, \tau, H, \mu, S >$, $i \in ID$, $\tau \in T$, H region w G, $\mu : Var(\tau) \rightarrow H \cup error$, S update lub ϵ

Definicje cd

Definicja

Stan programu - to krotka $S = < G, \Gamma >$, Γ to zbiór stanów zespołów

- definiujemy semantykę operacyjną z operacją przejścia $\rightarrow$ pomiędzy stanami $< G, \Gamma >$ - G nie musi być stertą całego programu
- przejście: $< G, \Gamma > \rightarrow < G', \Gamma' >$
- wartości w semantyce to obiekty, true lub false
- termy pomocnicze mogą być albo wartością albo termem $< N, e >$ N - stan, e wskaźnik lub wyrażenie boolowskie
- przepływ programu to ciąg stanów, z stanem terminalnym na końcu

Reguły przejścia

(ASSEMBLY-STEP)
$$\frac{N = \langle i, \tau, H, \mu, S\rangle \in \Gamma \qquad \langle H, \{N\}\rangle \longrightarrow \langle H', \Gamma'\rangle}{\langle G, \Gamma\rangle \longrightarrow \langle G[H \rightsquigarrow H'], \Gamma \setminus \{N\} \cup \Gamma'\rangle}$$

(MERGE-1)
$$\frac{\begin{array}{c} N_1 = \langle i_1, \tau_1, H_1, \mu_1, \epsilon\rangle \in \Gamma \quad N_2 = \langle i_2, \tau_2, H_2, \mu_2, \epsilon\rangle \in \Gamma \quad i_1 \neq i_2 \quad \mathtt{v}, \mathtt{v}_1 \in \mathit{Var}(\tau_1) \quad \mathtt{v}_2 \in \mathit{Var}(\tau_2) \\ (\mathtt{merge}(\mathtt{v.f}, \tau_2, \mathtt{v}_1 := \mathtt{v}_2) : S) \in \mathit{Act}(\tau_1) \quad \mu(\mathtt{v}) = u \quad u \xrightarrow{G, \mathtt{f}} v \quad v \text{ is in } H_2 \end{array}}{\langle G, \Gamma\rangle \longrightarrow \langle G, \Gamma \setminus \{N_1, N_2\} \cup \{\langle i_1, \tau, H_1 \sqcup_G H_2, \mu_1[\mathtt{v}_1 \mapsto \mu_2(\mathtt{v}_2)], S\rangle\}\rangle}$$

(MERGE-2)
$$\frac{\begin{array}{c} N_1 = \langle i_1, \tau_1, H_1, \mu, \epsilon\rangle \in \Gamma \quad N_2 = \langle i_2, \tau_2, H_2, \mu_2, \epsilon\rangle \in \Gamma \quad i_1 \neq i_2 \quad \mathtt{v}_1', \mathtt{v}_2' \in \mathit{Var}(\tau_1') \quad \mathtt{v}_1 \in \tau_1, \mathtt{v}_2 \in \tau_2 \\ (\mathtt{merge}(\mathtt{v.f}, \tau_1', \mathtt{v}_1' := \mathtt{v}_1, \mathtt{v}_2' := \mathtt{v}_2) : S) \in \mathit{Act}(\tau_1) \quad \mu_1(\mathtt{v}) = v \quad v \xrightarrow{G, \mathtt{f}} w \quad w \text{ is in } H' \end{array}}{\langle G, \Gamma\rangle \longrightarrow \langle G, \Gamma \setminus \{N, N'\} \cup \{\langle i, \tau', H \sqcup_G H', \mu_{Init}(\tau', w)[\mathtt{v}_1' \mapsto \mu_1(\mathtt{v}_1), \mathtt{v}_2' \mapsto \mu_2(\mathtt{v}_2)], \epsilon\rangle\}\rangle}$$

(EXP-1)
$$\frac{N = \langle i, \tau, H, \mu, S\rangle \qquad \mu(\mathtt{v}) = u}{\langle N, \mathtt{v}\rangle \dashrightarrow u}$$

(EXP-2)
$$\frac{N = \langle i, \tau, H, \mu, S\rangle \qquad \langle N, e\rangle \dashrightarrow u \qquad u \xrightarrow{H, \mathtt{f}} v \qquad v \text{ is in } H}{\langle N, e.\mathtt{f}\rangle \dashrightarrow v}$$

(EXP-3)
$$\frac{\langle N, e\rangle \dashrightarrow \mathtt{error}}{\langle N, e.\mathtt{f}\rangle \dashrightarrow \mathtt{error}}$$

(EXP-4)
$$\frac{N = \langle i, \tau, H, \mu, S\rangle \qquad \langle N, e\rangle \dashrightarrow u \qquad u \xrightarrow{H, \mathtt{f}} v \qquad v \text{ is not in } H}{\langle N, e.\mathtt{f}\rangle \dashrightarrow \mathtt{error}}$$

(GUARDS-WITHOUT-MERGES)
$$\frac{N = \langle i, \tau, H, \mu, \epsilon\rangle \qquad (g : S) \in \mathit{Act}(\tau) \qquad \langle N, g\rangle \dashrightarrow \mathit{true}}{\langle H, \{N\}\rangle \longrightarrow \langle H, \{\langle i, \tau, H, \mu, S\rangle\}\rangle}$$

(ASSIGN-1)
$$\frac{N = \langle i, \tau, H, \mu, \mathtt{v} := e\rangle \qquad \langle N, e\rangle \dashrightarrow u}{\langle H, \{N\}\rangle \longrightarrow \langle H, \{\langle i, \tau, H, \mu[\mathtt{v} \mapsto u], \epsilon\rangle\}\rangle}$$

(ASSIGN-2)
$$\frac{N = \langle i, \tau, H, \mu, \mathtt{v.f} := e\rangle \qquad \mu(\mathtt{v}) = u \qquad \langle N, e\rangle \dashrightarrow v \qquad H' = H[(u, \mathtt{f}, _) \rightsquigarrow (u, \mathtt{f}, v)]}{\langle H, \{N\}\rangle \longrightarrow \langle H', \{\langle i, \tau, H', \mu, \epsilon\rangle\}\rangle}$$

(SEQUENCE)
$$\frac{N = \langle i, \tau, H, \mu, S\rangle \qquad \langle H, \{N\}\rangle \longrightarrow \langle H', \{\langle i, \tau, H', \mu', \epsilon\rangle\}\rangle}{\langle H, \{\langle i, \tau, H, \mu, (S; S')\rangle\}\rangle \longrightarrow \langle H', \{\langle i, \tau, H', \mu', S'\rangle\}\rangle}$$

(SPLIT)
$$\frac{N = \langle i, \tau, H, \mu, \mathtt{split}(\tau')\rangle \qquad H \text{ has objects } u_1, \ldots, u_n \qquad i_{u_1}, \ldots, i_{u_n} \text{ are globally unique fresh IDs}}{\langle H, \{N\}\rangle \longrightarrow \langle H, \{\langle i_{u_j}, \tau', \{u_j\}, \mu_{Init}(\tau', u_j), \epsilon\rangle : 1 \leq j \leq n\}\rangle}$$

1 Wstęp

2 Chorus - podstawy

3 Chorus - formalnie
- Składnia
- Semantyka
- Odporność na wyścig i zakleszczenie

4 JChorus

5 Podsumowanie

Wyścig

- brak wyścigu zapewnia nam izolacja zespołów
- w przypadku operacji łączenia mamy zapewnione, że zespoły są w stanie gotowości, zatem również nie są wykonywane żadne operacje

Zakleszczenie

- zakleszczenie - sytuacja, gdy jeden proces czeka na zasoby od drugiego i vice versa (ogólnie istnieje cykl)
- zakleszczenie w języku zespołów - każdy zespół chce się zmergować z drugim, ale nie mogą tego zrobić
- sytuacja niemożliwa, bo zespół w stanie gotowy nie może zapobiec swojemu połączeniu

1 Wstęp

2 Chorus - podstawy

3 Chorus - formalnie

4 JChorus
- Składnia obiektowa
- Części sekwencyjne i równoległe
- Przykład
- Implementacja JChorusa

5 Podsumowanie

Składnia obiektowa

- klasy nie są beztypowe, lecz zgodne z semantyką Javy
- Zespoły są specjalnymi klasami
  - blok action deklaruje wszystkie dozory zespołu
  - składnia Guard : {Update}
- deklaracja odpowiedniego konstruktora umożliwia przesyłanie podczas łączenia danych z ginącego zespołu
- dodatkowe operacje:
  - merge(L,τ,x):{Update}
  - splitmany(L,τ,p1,p2,p3,...)
  - L - kolekcja obiektów
  - x - etykieta krawędzi
  - p1,p2,p3... - parametry
- odwołanie do obiektu spoza zespołu zgłasza wyjątek NonLocalException
- funkcja zmiany klasy - become(τ,p1,p2,p3,...)

1 Wstęp

2 Chorus - podstawy

3 Chorus - formalnie

4 JChorus
- Składnia obiektowa
- Części sekwencyjne i równoległe
- Przykład
- Implementacja JChorusa

5 Podsumowanie

Części sekwencyjne i równoległe

- równoległe programy mają również cześci sekwencyjne
- W JChorusie piszemy programy sekwencyjne z wywołaniami:
  - parallel(τ(p1,p2,p3,...))
  - gdy wszystkie zespoły będą w stanie końcowym wykonanie wraca do sekwencyjnej części kodu.
- dodatkowo dodano obiekty read-only, do których jest dostęp do odczytu z każdego zespołu (zapis kończy się wyjątkiem)
- możliwe jest rzutowanie obiektów na readOnly
- rzutowanie w drugą stronę jest możliwe tylko w sekwencyjnej części kodu

1 Wstęp

2 Chorus - podstawy

3 Chorus - formalnie

4 JChorus
- Składnia obiektowa
- Części sekwencyjne i równoległe
- Przykład
- Implementacja JChorusa

5 Podsumowanie

Kod W JChorusie

```
 1: assembly Triangle {
 2:  Triangle(TriangleObject t) {
 3:     if (t.isBad())
 4:         become(Cavity, t); // become a Cavity
 5:    }
 6: } /* end Triangle */

 7: assembly Cavity {
 8:   action  { // expand cavity
 9:      merge(outgoingedges, Cavity, TriangleObject t) : {
10:          outgoingedges.remove(t);
11:          frontier.add(t);
12:          build(); }
13:    }

14:   Set members; Set border;
15:   Queue frontier;  // current frontier
16:   List outgoingedges;     // outgoing edges on which
                              // to merge
17:   TriangleObject initial;

18:   Cavity(TriangleObject t) {
      ... initialize data fields....
19:      frontier.enqueue(t);
20:      build(); }
      ...
```

```
21:     void build() {
22:         while (frontier.size() != 0) {
23:             TriangleObject curr = frontier.dequeue();
24:             try {
25:                if (isMember(curr)) members.add(curr);
26:                else border.add(curr);
                   // add triangles using BFS
27:                for (TriangleObject n: curr.neighbors())
28:                   if (notSeen(n)) frontier.add(n);
29:             } catch(NonLocalException e)
                { // triangle not in assembly,
                      // add to merge list
30:                  outeredges.add(e.getObject());    }
31:          }
32:          if (outeredges.isEmpty()) {
33:              retriangulate(); split(Triangle);
34:          }
35:     }

36:     void retriangulate() { ... }
37:     boolean isMember(TriangleObject t) {... }
38:     boolean notSeen(TriangleObject t) {... }
39: } /* end Cavity */

40: shared TriangleObject {
41:     Point p1, p2, p3;
42:     Triangle s1, s2, s3;

43:     Point circumCenter() {...}
44:  }

50: assembly Loader {
51:     Loader(String filename) {
52:         ...
53:          ... new Triangle(p1, p2, p3);
54:         ...
55:         split(Triangle);
56:     }
57:  } /* end Loader */
```

1 Wstęp

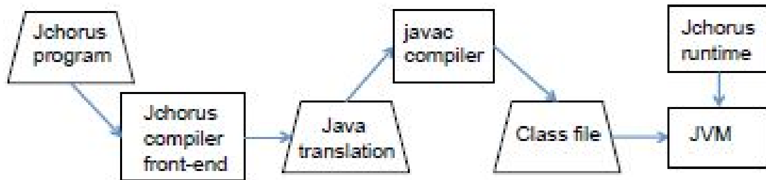
2 Chorus - podstawy

3 Chorus - formalnie

4 JChorus
- Składnia obiektowa
- Części sekwencyjne i równoległe
- Przykład
- Implementacja JChorusa

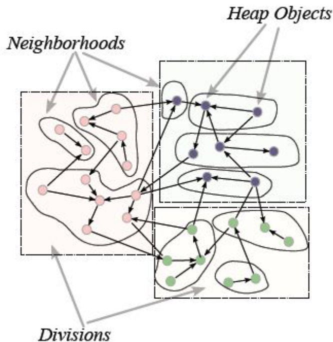
5 Podsumowanie

Kompilacja



- Problem odwzorowania wielu wątków na skończoną liczbę procesorów
- Kompilator przekształca program w JChorusie na program w Javie
- Kluczowe pojęcie - dywizja - zbiór zespołów

Dywizje



- liczba dywizji powinna odpowiadać liczbie procesorów
- każda dywizja ma dwie kolejki:
  - Workq(D) - kolejka oczekujących zespołów
  - Blockq(D) - kolejka do zewnętrznych połączeń
- Dywizje są ułożone w topologię pierścienia oraz przesyłają sobie żeton
- Tylko dywizja z żetonem może wykonać zewnętrzne połączenie

Analiza zakleszczenia

- Merge w ramach jednej dywizji:
  - brak zakleszczenia - sytuacja sekwencyjna
- Merge w ramach różnych dywizji:
  - również brak zakleszczenia, gdyż w końcu zespół wyląduje w kolejce Blockq

Podsumowanie

- Esencją Chorusa jest wysokopoziomowa równoległość oparta na pojęciu zespołu
- Są prowadzone prace nad wydajniejszą implementacją JChorusa
- Planowane jest również wykonanie większej liczby testów
- Oczekiwane są opinie użytkowników Chorusa
- Postępuje również rozwój teoretycznej części modelu
- Planowane jest także powstanie specjalnego zrębu ułatwiającego pracę

Dziękuję za uwagę