ProActive Parallel Suite

Grzegorz Chimosz

Wydział Matematyki, Informatyki i Mechaniki UW

8 stycznia 2009

Spis treści

Programowanie współbieżne i rozproszone

- Popularność architektur wielordzeniowych, wieloprocesorowych i rozproszonych
- ... i trudność wykorzystania ich potencjału
  - Duża część kodu zarządza dodatkowymi strukturami danych i sprawdza, czy nie pojawiły się przy tym błędy
  - Programowanie rozproszone – jeszcze więcej problemów
  - ProActive Parallel Suite przybywa z odsieczą!
- I jeszcze trzeba to uruchomić!

ProActive Parallel Suite

ProActive Parallel Suite to otwarty javowy middleware do obliczeń równoległych i rozproszonych.
Wspomaga:

- programowanie i projektowanie
- wrapping (C/C++, Fortran, MPI)
- uruchamianie i monitorowanie

Wspiera:

- gotowe modele współbieżności
- P2P
- przesyłanie plików
- równoważenie obciążenia
- i zapewnia odporność na błędy (wykonania)

O projekcie

Zaangażowani w projekt:

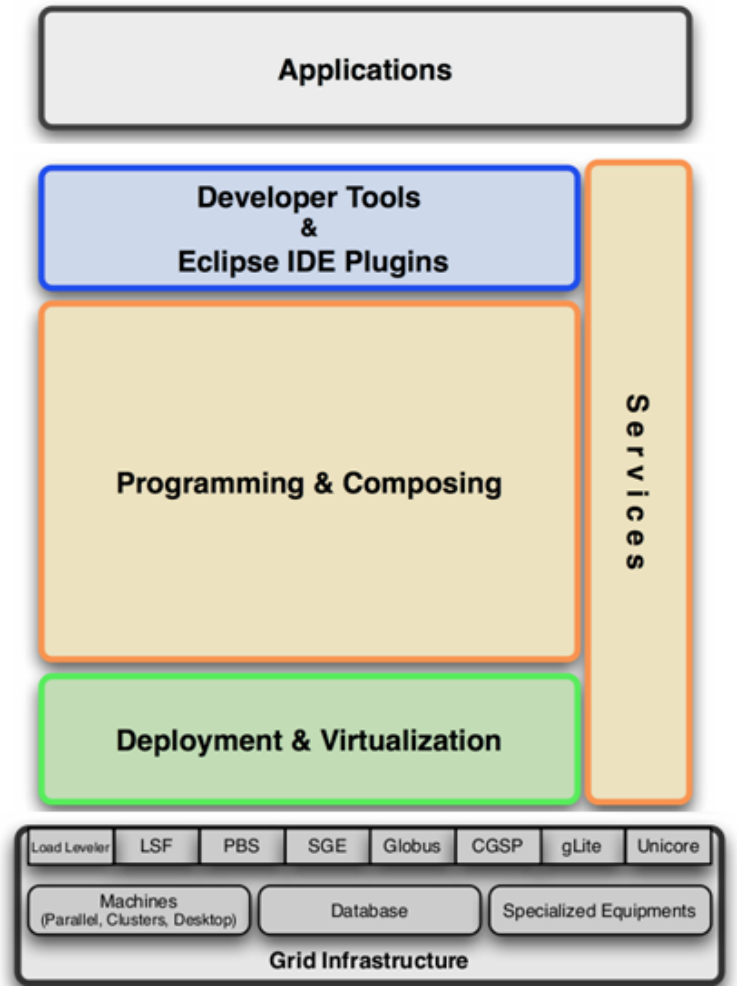
- ObjectWeb – `www.objectweb.org`
- INRIA – `www.inria.fr`
- CRNS – `www.cnrs.fr`
- Uniwersytet Nicejski – `http://portail.unice.fr`
- ActiveEon – `http://www.activeeon.com`

Strona domowa projektu:
`proactive.inria.fr`

Wspierane technologie

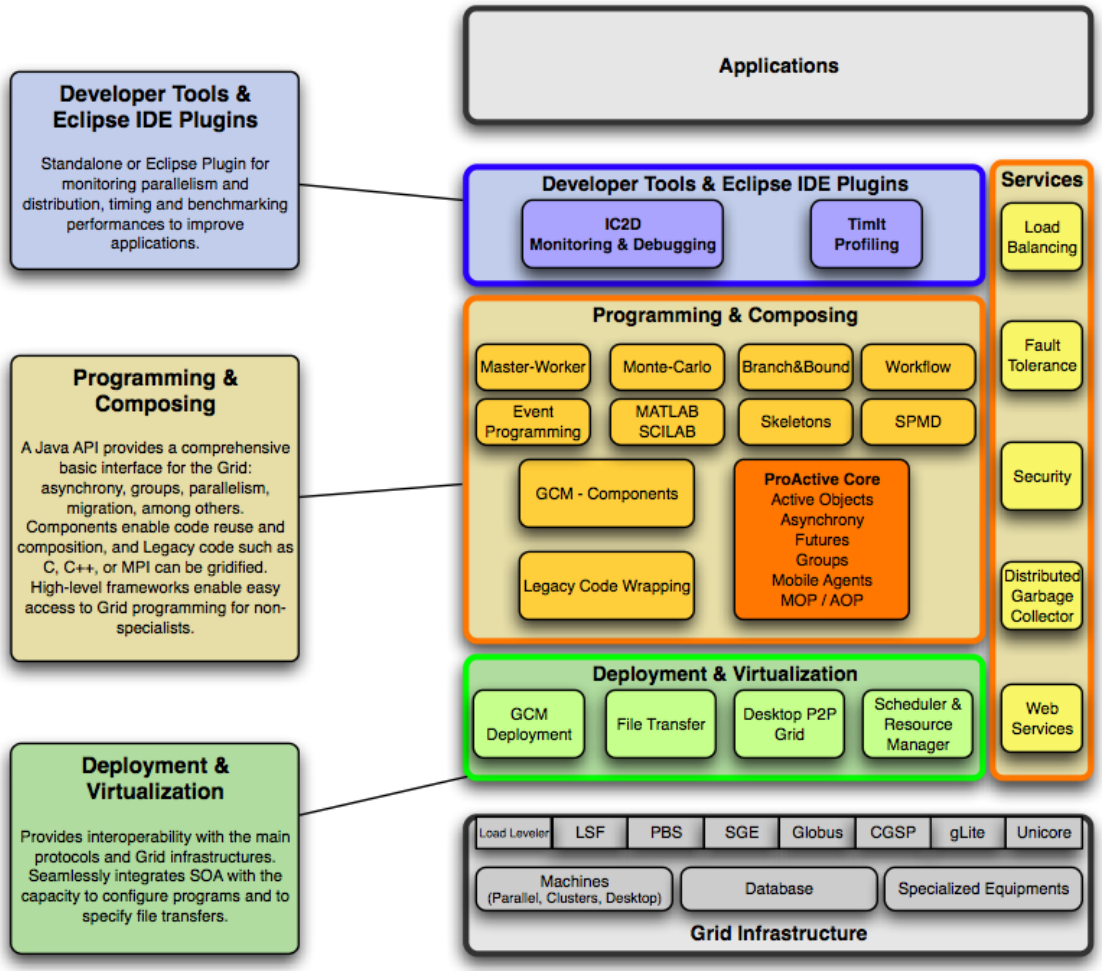
- Systemy operacyjne
  - Linux, Windows, Solaris, MacOS, SGI Irix
- Maszyny wirtualne Javy
  - Sun, SGI, BEA, JRockit
- Protokoły sieciowe
  - ssh, rsh, sshGSI, rcp, scp, Unicore, Globus Gram
- Chmury obliczeniowe
  - Amazon EC2

Architektura ProActive Parallel Suite



Źródło: [1]

Architektura ProActive Parallel Suite



Źródło: [2]

Obiekty aktywne (ang. *active objects*)

Obiekt aktywny składa się z:

- dokładnie jednego wątku
- kolejki żądań
- potencjalnie wielu obiektów „pasywnych”
- pośrednika (ang. *proxy*)

Obiekty aktywne – własności

Własności obiektów aktywnych:

- brak współdzielenia obiektów pasywnych – głębokie kopie przy wywołaniu metod, co może prowadzić do utraty spójności
- obiekty aktywne jeżeli są parametrami wywołania – są przekazywane przez referencje
- asynchroniczne żądania
- tworzenie odroczonych obiektów i oczekiwanie z konieczności

Obiekty aktywne – sposoby tworzenia

Utworzenie nowej instancji

```
1 A a = (A)ProActive.
2   newActive(A.class.getName(), params, node);
```

Klasa

```
1 class pA extends A implements RunActive { ... }
```

Uaktywnienie istniejącego obiektu

```
1 A a = new A (obj, 7);
2 /* ... */
3 a = (A)ProActive.turnActive (a, node);
```

Oczekiwanie z konieczności (ang. *wait by necessity*)

Wywołanie metody składa się z dwóch etapów:

1. **żądanie (nazwa, argumenty)**
   - powoduje utworzenie odroczonego obiektu (ang. *future object*)
   - po obliczeniu wywoływany obiekt odeśle wynik
2. **odpowiedź (wynik wywołanej metody)**
   - wywołujący może kontynuować swoją pracę, nawet jeżeli wynik nie jest dostępny – asynchroniczność

Oczekiwanie z konieczności (ang. *wait by necessity*)

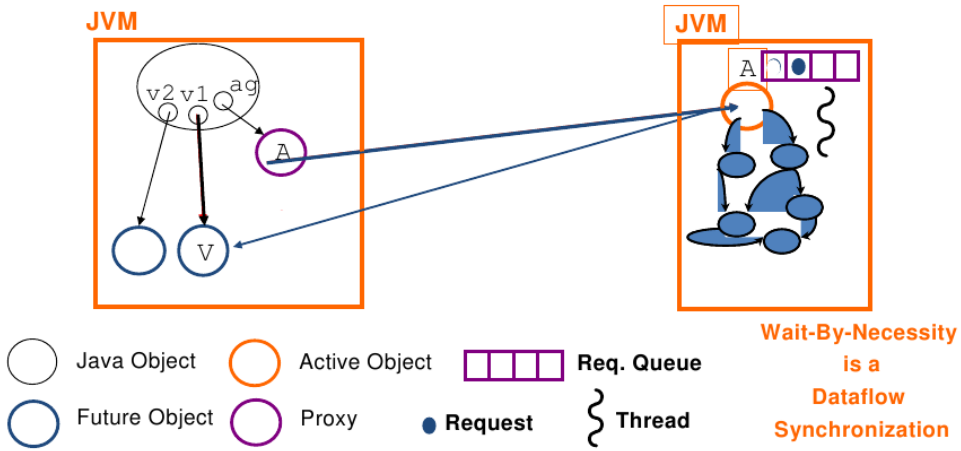
Wywołanie metody składa się z dwóch etapów:

1. **żądanie (nazwa, argumenty)**
   - powoduje utworzenie odroczonego obiektu (ang. *future object*)
   - po obliczeniu wywoływany obiekt odeśle wynik
2. **odpowiedź (wynik wywołanej metody)**
   - wywołujący może kontynuować swoją pracę, nawet jeżeli wynik nie jest dostępny – asynchroniczność

Aktywne obiekty w praktyce

Przykład

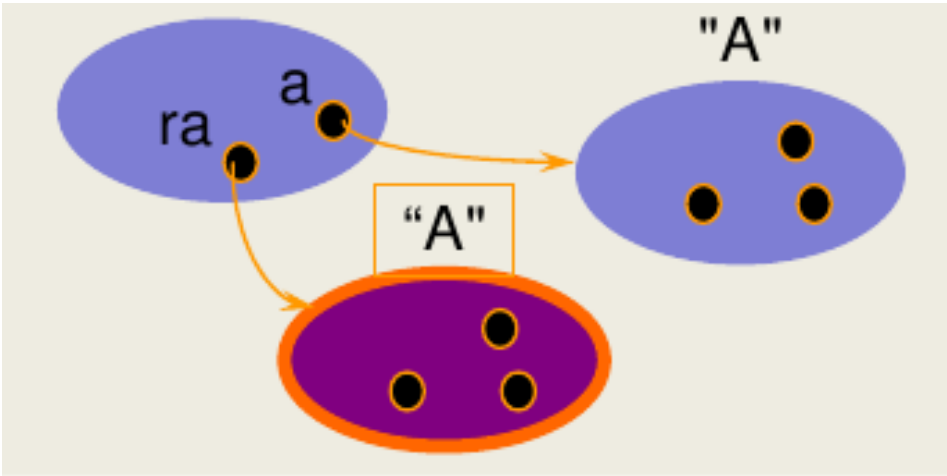
```
1 A ag = newActive (A.class.getName(), [...],
      VirtualNode);
2 V v1 = ag.foo (param);
3 V v2 = ag.bar (param);
4 /* ... */
5 v1.bar(); //Wait-By-Necessity
```



Źródło: [3]

Przezroczystość mechanizmu wykonania

- Polimorfizm
  - obiektów aktywnych i zwykłych używa się dokładnie tak samo
  - polimorficzne są też obiekty odroczone
- Oczekiwanie z konieczności
  - zapewnia zupełnie przezroczystą synchronizację
  - obiekty odroczone są zupełnie przezroczyste



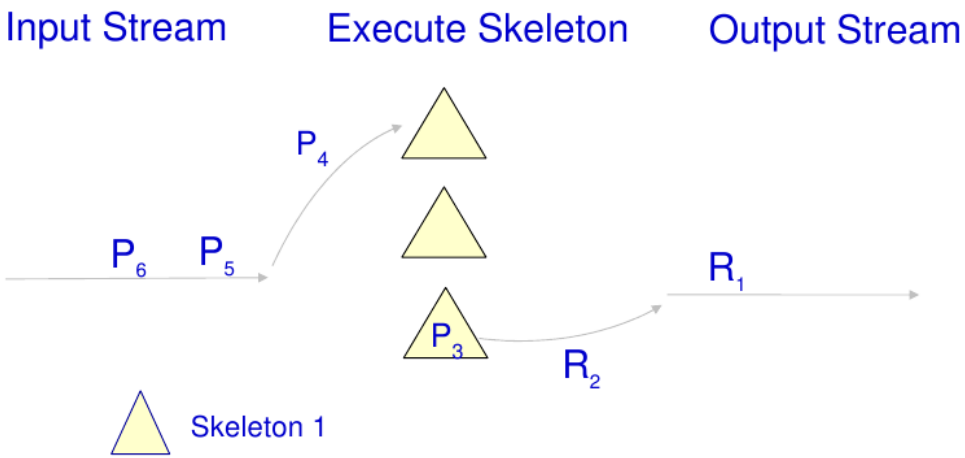
Źródło: [3]

Gotowe schematy przetwarzania

- Kolejkujące dane
  - seq
- Rozpraszające zadania
  - farm
  - pipe
  - if
  - while
  - for
- Rozpraszające dane
  - divide and conquer
  - map
  - fork

Schemat *farm*

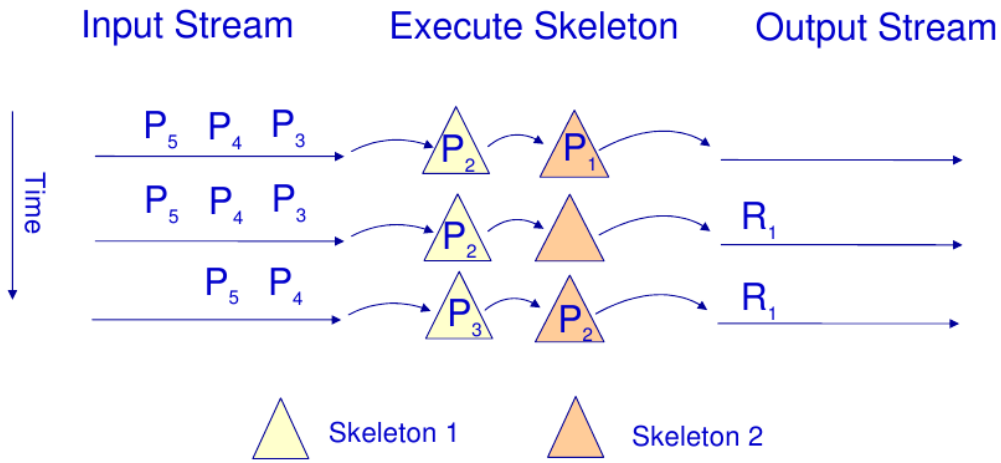
- Różne dane są przetwarzane równolegle



Źródło: [3]

Schemat *pipe*

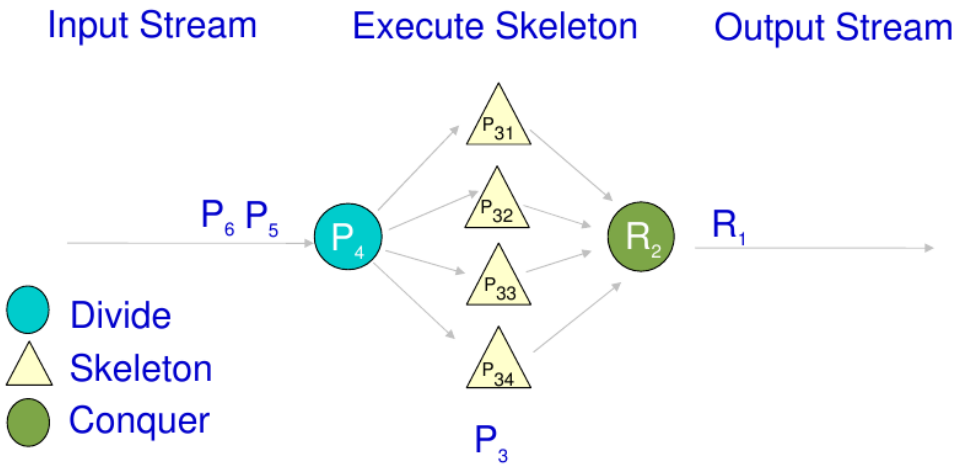
- Reprezentuje obliczenia podzielone na etapy
- Etapy są obliczane równolegle



Źródło: [3]

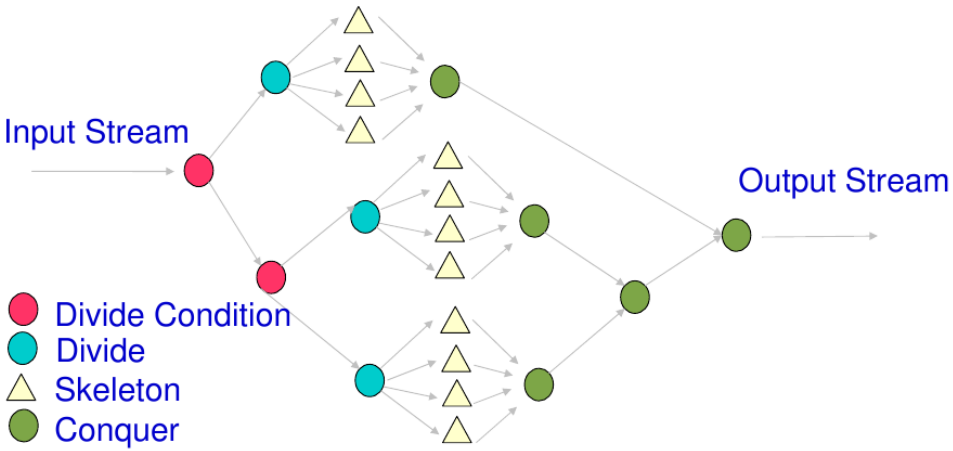
Schemat *map*

- Dzieli dane na mniejsze części, do każdej przykłada funkcję i scala wynik



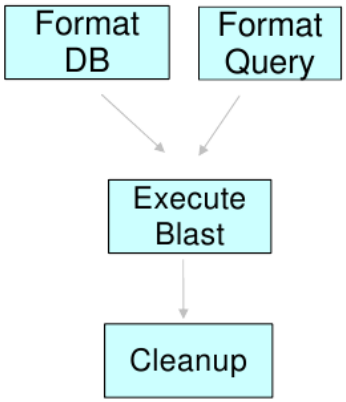
Źródło: [3]

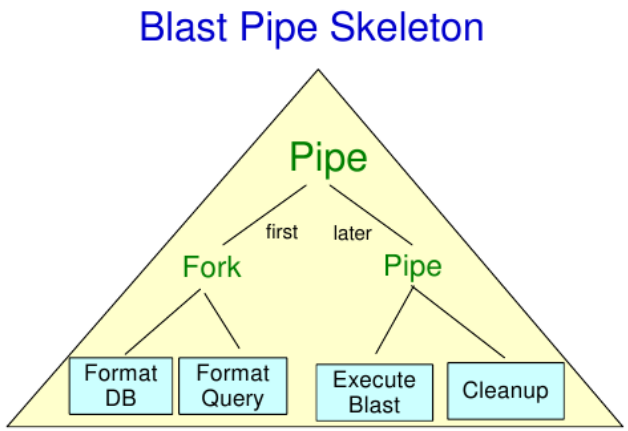
Schemat *divide and conquer*



Źródło: [3]

Użycie schematu na przykładzie *BLAST*





Źródło: [3]

Użycie schematu na przykładzie *BLAST*

Schemat dla BLAST

```
1 Pipe blastPipe = new Pipe(
2                  new Fork(
3                      new Seq(new ExecuteFormatDB()),
4                      new Seq(new ExecuteFormatQ())),
5                  new Pipe(
6                     new Seq(new ExecuteBlast()),
7                     new Seq(new CleanBlast())));
```

Schemat *dziel i zwyciężaj* dla BLAST

```
1 Skeleton root = new DaC(
2         new DivideDB(), new DivideCondition(),
3                  blastPipe, new ConquerResults());
```

Użycie schematu na przykładzie *BLAST*

Dziel

```
1 public class DivideCondition implements Condition {
2   public boolean evalCondition(Blast param) {
3     File file = param.getDatabaseFile();
4     return file.length() > param.getMaxDBSize();
5   }
6 }
```

Użycie schematu na przykładzie *BLAST*

I zwyciężaj

```
1  public class DivideDB implements Divide{
2    public Vector divide(Blast param) {
3      Vector<BlastParameters> children = new Vector<
           BlastParameters>();
4      Vector<File> files = divideDBFile(param);
5      for (File newDBFile : files) {
6        Blast newParam = new Blast(param);
7        param.setDBName(newDBFile);
8        children.add(newParam);
9      }
10     return children;
11   }
12 }
```

Użycie schematu na przykładzie *BLAST*

Przygotowanie do obliczeń

```
1 Skeleton root = /* ... */;
2 ResourceManager manager =
3             new ProActiveManager("descriptor.xml", "vn-
                  name");
4 Calcium calcium = new Calcium(manager);
5 Stream stream = calcium.getStream(root);
```

Użycie schematu na przykładzie *BLAST*

Załadowanie danych

```
1 stream.input(new Blast("db-file","query-file1"));
2 stream.input(new Blast("db-file","query-file2"));
3 stream.input(new Blast("db-file","query-file3"));
```

Odczytanie wyniku

```
1 Blast res1 = stream.getResult();
2 Blast res2 = stream.getResult();
3 Blast res3 = stream.getResult();
```

Przykład

ProActive w działaniu.

Bibliografia


ProActive in 5 minutes.
[online: http://proactive.inria.fr/index.php?page=proactive_in_five_min], 2008.


ProActive features.
[online: http://proactive.inria.fr/userfiles/image/ProActive_Features_Full.png], 2008.


Denis Caromel.
ProActive Tutorial.
[online: http://proactive.inria.fr/userfiles/file/presentation/Tutorial-ProActive450slides-December-21-2007.pdf], 2007.