An Operational Semantics for Skeletons

Marco Aldinucci

ISTI – CNR
National Research Council
Pisa, Italy

Marco Danelutto

Computer Science Dept.
University of Pisa
Pisa, Italy
Outline

- Skeletons
- Semantics – motivations
- The schema of semantics
- Axioms – rules
- Example
- Concluding remarks
Skeletons

- Skeletons are language constructs
  - well-defined input-output behavior
  - parallelism exploitation patterns
  - (sometimes) can be nested
  - several prepackaged implementations

- Two main families
  - Data Parallel (*map, reduce, scan ...*)
  - Task & Stream parallel (*farm, pipeline, ...*)
Motivations

- Usually **formal** functional semantics, **informal** parallel behavior
- Describe skeletons
  - in-out relationship (functional behavior)
  - parallel behavior
  - in uniform and precise way (non steady state)
  - in structural way
- Theoretical work motivated by concrete needs
  - Enable and automate performance-driven source-to-source optimizations
  - same in/out different parallel behaviors
  - Compare different skeleton sets expressive power
farm (pipe (seq f1) (seq f2))

e.g. with ASI
f1 = filter
f2 = render

channel, network ...

PE1 – PE4
sequential source code just plugged in
data items arrives in sequence, we cannot assume data is already distributed, data distribution cost is large, several farm scheduling policies are possible, as well as several data mappings

DipInf; 31/08/2003
pipe (map fc (seq f1 ) fd) (map gc (seq f2 ) gd)
Running example language: Lithium

- Stream and Data Parallel
  - farm, pipe
  - map, reduce, D&C, ...
  - Can be freely nested
  - All skeletons have a stream as in/out

- Java-based (skeletons are Java classes)

- Implemented and running [FGCS 19(5):2003]
  - http://www.di.unipi.it/~marcod/Lithium/ or sourceforge

- Macro data-flow run-time

- Support heterogeneous COWs

- Includes parallel structure optimization
  - performance-driven, source-to-source
The schema of semantics

- Axioms, three kind per skeleton:
  1. Describe skeletons within the steady state
  2. Mark the begin of stream *
  3. Manage the end of stream *

- Six rules:
  1. Two describing parallel execution (SP, DP)
     - Have a cost
  2. Four to navigate in the program structure
     - No cost, ensure strict execution order

- Look to SP/DP rules only to figure out program performance
The meaning of labels

Label represent an enumeration of PEs

Two kind of labels:

- On streams represent data mapping:
  \[ \langle x \rangle_3 \] means \( x \) is available on PE\(_3\)

- On arrows represent computation mapping
  \[ \rightarrow \quad 4 \] means such computation is performed by PE\(_4\)

Re-label \( \mathcal{O}(l,x) \) a stream means communicate it

- Semantics may embed an user-defined policy \( \mathcal{O}(l,x) \)
- Cost depend on label (topology) and data item \( x \) (size)
Axioms (steady state)

1. Apply inner skeleton $F \in \text{param}$ to the stream head $x$
2. Recur on the tail of the stream
3. Expressions 1 & 2 are joined by :: operator

\[ \text{Skel params } (x, \tau)_{\ell_1} \overset{\ell_1}{\rightarrow} F(x)_{\ell_2} :: \text{Skel params } (\tau)_{\ell_3} \]

a. The arrow label gets left-hand side stream label ()
b. Labels in the right-hand side may change (stream items may be bounced elsewhere)
Lithium axioms (for stream par skeletons)

\[ \text{seq } f\langle x, \tau \rangle_\ell \xrightarrow{\ell} (f \circ x)_\ell :: \text{seq } f\langle \tau \rangle_\ell \]

Embed seq code
Stream unfolded, Labels unchanged

\[ \text{pipe } \Delta_1 \Delta_2 \langle x, \tau \rangle_\ell \xrightarrow{\ell} \Delta_2 R_{\ell, x} \Delta_1 \langle x \rangle_\ell :: \text{pipe } \Delta_1 \Delta_2 \langle \tau \rangle_\ell \]

a. arrow label gets stream one – happens locally
b. label doesn’t change – keep 1st stage \( \Delta_1 \) locally
c. re-label \( R \) inserted in between 1st & 2nd stage – it will map 2nd stage elsewhere
d. tail is expected from the same source

\[ \text{farm } \Delta \langle x, \tau \rangle_\ell \xrightarrow{\ell} \Delta \langle x \rangle_{\mathcal{O}(\ell, x)} :: \text{farm } \Delta \langle \tau \rangle_{\mathcal{O}(\ell, x)} \]

a. stream item is distributed accordingly \( \mathcal{O} \) policy
b. a reference of tail of the stream follows the head
Lithium axioms (DP skeletons)

\[
\text{map } f_c \Delta f_d (x, \tau)e \xrightarrow{\ell} f_c (\alpha \Delta) f_d (x)e :: \text{map } f_c \Delta f_d (\tau)e
\]

\[
d\&c f_{tc} f_c \Delta f_d (x, \tau)e \xrightarrow{\ell} d\&c f_{tc} f_c \Delta f_d (x)e :: d\&c f_{tc} f_c \Delta f_d (\tau)e
\]

\[
d\&c f_{tc} f_c \Delta f_d (y)e = \begin{cases} 
\Delta (y)e \\
f_c (\alpha (d\&c f_{tc} f_c \Delta f_d)) f_d (y)e 
\end{cases} \quad \text{iff } (f_{tc} y) \quad \text{otherwise}
\]
Lithium rules overview

\[
\begin{align*}
E_i & \xrightarrow{\ell_i} E'_i & \forall i \ 1 \leq i \leq n \\
\exists i, j & \ 1 \leq i, j \leq n, \ l_i = l_j \Rightarrow i = j & \text{sp} \\
\langle \nu \rangle_\bot & :: E_1 :: \cdots :: E_n :: \Gamma \overset{\bot}{\longrightarrow} \langle \sigma \rangle_\bot :: E'_1 :: \cdots :: E'_n :: \Gamma
\end{align*}
\]

\[
\begin{align*}
f_d \langle x \rangle_e & \xrightarrow{\ell} \langle y_1 \rangle_e, \cdots \langle y_n \rangle_e \ \rho \ & \Delta \langle y_i \rangle_e \xrightarrow{\ell} \langle z_i \rangle_e \\
& \quad \quad \quad f_c \langle x_1 \rangle_e, \cdots \langle z_n \rangle_e \ \rho \xrightarrow{\ell} \langle x \rangle_e, \ i = 1 \ldots n & \text{dp}
\end{align*}
\]

\[
\begin{align*}
f_c (\alpha \Delta) \ f_d \langle x \rangle_e \xrightarrow{\ell} \langle x \rangle_e
\end{align*}
\]

\[
\begin{align*}
\Delta \langle x \rangle_{\ell_1} & \xrightarrow{\ell_2} \langle y \rangle_{\ell_3} & \text{relabel} \\
R_\ell & \Delta \langle x \rangle_{\ell_1} \xrightarrow{\ell_2} \langle y \rangle_{\ell}
\end{align*}
\]

\[
\begin{align*}
E_1 & \xrightarrow{\ell} E_2 & \text{context} \\
\Delta E_1 & \xrightarrow{\ell} \Delta E_2
\end{align*}
\]

\[
\begin{align*}
\langle \sigma \rangle_{\ell_1} :: \langle \tau \rangle_{\ell_2} \overset{\bot}{\longrightarrow} \langle \sigma, \tau \rangle_\bot & \text{ join} \\
\langle \epsilon \rangle_\bot :: \langle \tau \rangle_{\ell_2} \overset{\bot}{\longrightarrow} \langle \tau \rangle_{\ell_2} & \text{ join}_\epsilon
\end{align*}
\]
sp rules details

\[ E_i \xrightarrow{\ell_i} E'_i \quad \forall i \ 1 \leq i \leq n \quad \land \quad \exists i, j \ 1 \leq i, j \leq n, \ell_i = \ell_j \Rightarrow i = j \quad sp \]

\[ \langle v \rangle \bot :: E_1 :: \cdots E_n :: \Gamma \xrightarrow{\bot} \langle \sigma \rangle \bot :: E'_1 :: \cdots E'_n :: \Gamma \]

Many semantics for each program

- \( i=j=1 \) always possible, i.e. no stream parallelism is exploited
- All of them are "functionally confluent", describe the same in-out relationship
- All of them describe the same parallel behavior, but with different degrees of parallelism
Example (2-ways-2-stages pipeline)

\[\text{farm}(\text{pipe} (\text{seq } f_1) (\text{seq } f_2)) \langle x_1, x_2, x_3, x_4, x_5, x_6, x_7 \rangle\]

- Iterate the same operation on the whole stream
- pipe now disappeared
- two different labels on streams: 0 and 1
- two different labels on \( R \): 02, 12
- farm now disappeared

- Apply pipe inner skeletons (stages) to the item
- A re-labeling operation \( R \) is introduced in the middle
Example (continued)

\[
(e)_1 :: \text{seq} f_2 \text{ seq } f_1 (x_1)_0 :: \text{seq} f_2 \text{ seq } f_1 (x_2)_1 :: \text{seq} f_2 \text{ seq } f_1 (x_3)_0 :: \text{seq} f_2 \text{ seq } f_1 (x_4)_1 :: \\
\text{seq} f_2 \text{ seq } f_1 (x_5)_0 :: \text{seq} f_2 \text{ seq } f_1 (x_6)_1 :: \text{seq} f_2 \text{ seq } f_1 (x_7)_0
\]

- This formula no longer can be reduced by axioms
- \text{sp} rule can be applied:

"Any rightmost sequence of expressions can be reduced provided streams exploits different labels"

- In this case the longest sequence includes two expressions, i.e. the max. par degree is 2 (matching the double-pipeline startup phase)
Example (continued)

Due to the re-labeling we have 4 adjacent expressions exploiting different labels: 02, 12, 0, 1 – i.e. a max. parallelism degree of 4

The step can be iterated up to the end of stream

Max parallelism degree 4 since no more than 4 different labels appear adjacently (easy to prove)
Example (continued)

By iterating SP rule we eventually get

\[
\langle \epsilon \rangle \perp :: \langle f_2 (f_1 x_1) \rangle_{02} :: \langle f_2 (f_1 x_2) \rangle_{12} :: \langle f_2 (f_1 x_3) \rangle_{02} :: \langle f_2 (f_1 x_4) \rangle_{12} :: \langle f_2 (f_1 x_5) \rangle_{02} :: \langle f_2 (f_1 x_6) \rangle_{12} :: \langle f_2 (f_1 x_7) \rangle_{02} \]

That can be joined to form the output stream

\[
\langle f_2 (f_1 x_1), f_2 (f_1 x_2) f_2 (f_1 x_3), f_2 (f_1 x_4) f_2 (f_1 x_5), f_2 (f_1 x_6), f_2 (f_1 x_7) \rangle \perp
\]

- Count parallelism
- Count communications
- or reason about it
Summary

Operational semantics for skeletons
- Describes both functional and parallel behavior
- User-defined mapping/scheduling
- User-defined comm/comp costs
- General, easy to extend
- No similar results within the skeleton community

Enable performance reasoning
- Skeleton normal-form [PDCS99, FGCS03, web]
- Provably correct automatic optimizations

Formally describe your brand new skeleton and its performance
Mammography app. (lithium)

Completion time (secs)

Number of Processing Elements

Normalized Form (data + task parallel application)
Non Normal Form (data + task parallel application)

raw

optimized

15 – 20% better
Thank you

Questions?

www.di.unipi.it/~aldinuc
Stream skeletons

farm
- functionally the identity!
- a.k.a. parameter sweeping, embarrassingly parallel, replica manager ...
- instead for some other group it is apply-to-all

pipe
- parallel functional composition
- pipe $f_1 f_2 < x >$ computes $f_2 (f_1 x )$
- $f_1, f_2$ executed in parallel on different data items
Describe skeletons

- Usually functional behavior only described
- Parallel behavior does matter for performance
- Usually performance described by cost formulas

\[ T(\text{scan } Op) = \left( \frac{n}{p} - 1 \right) t_{Op} + g(p - 1) \text{comm\_size} + l + \left( \frac{n}{p} + p - 2 \right) t_{Op} \]

- Doesn’t describe the behavior just the cost
- What happens if Op is parallel?
  - Not compositional
  - Handmade for each architecture
  - Data layout not described
Axioms (begin/end of the stream)

Begin of stream marking:

\[ \text{Skel params} \ (x, \tau) \xrightarrow{\perp} \{\epsilon\}_\perp :: \text{Skel params} \ (x, \tau)_\perp \]

End of stream management:

\[ \text{Skel params} \ (x)_{\ell_1} \xrightarrow{\ell_1} \mathcal{F}(x)_{\ell_2} \]
An example of reduction

\[
\begin{align*}
\text{seq } f_1 \langle x_1 \rangle_0 & \xrightarrow{0} \langle f_1 \ x_1 \rangle_0 & \text{relabel} \\
\mathcal{R}_{02} \text{ seq } f_1 \langle x_1 \rangle_0 & \xrightarrow{0} \langle f_1 \ x_1 \rangle_{02} & \text{context} \\
\text{seq } f_2 \mathcal{R}_{02} \text{ seq } f_1 \langle x_1 \rangle_0 & \xrightarrow{0} \text{ seq } f_2 \langle f_1 \ x_1 \rangle_{02} & \\
\langle e \rangle_\perp & :: \text{ seq } f_2 \mathcal{R}_{02} \text{ seq } f_1 \langle x_1 \rangle_0 & :: \text{ seq } f_2 \mathcal{R}_{12} \text{ seq } f_1 \langle x_2 \rangle_1 & :: \text{ seq } f_2 \mathcal{R}_{02} \text{ seq } f_1 \langle x_3 \rangle_0 & :: \cdots & \xrightarrow{1} \\
\langle e \rangle_\perp & :: \text{ seq } f_2 \langle f_1 \ x_1 \rangle_{02} & :: \text{ seq } f_2 \langle f_1 \ x_2 \rangle_{12} & :: \text{ seq } f_2 \mathcal{R}_{02} \text{ seq } f_1 \langle x_3 \rangle_0 & :: \cdots
\end{align*}
\]