Lecture 6
Bulk Synchronous Parallel Homomorphisms
NII Lectures Series

Frédéric Loulergue$^1$

Université d’Orléans – LIFO – PaMDA Team

October-November 2013

$^1$and slides by Julien Tesson and Joeffrey Legaux
Outline

1. An Introduction to Program Calculation
2. Program Calculation and Parallel Programming
3. Calculating BSP Programs
4. BH Skeletons
5. Summary
This lecture is based on the following papers:


1 An Introduction to Program Calculation

2 Program Calculation and Parallel Programming

3 Calculating BSP Programs

4 BH Skeletons

5 Summary
Program calculation

- Specification
  or naive implementation

- Program transformation
  based on an equational theory

- Efficient implementation
A very simple example

**Specification:**

\[ \text{maximum} :: [a] =\geq a \]

\[ \text{maximum} = \text{hd} \circ \text{sort} \]
A very simple example

**Specification:**

\[
\begin{align*}
\text{maximum} & : [a] \Rightarrow a \\
\text{maximum} &= \text{hd} \circ \text{sort}
\end{align*}
\]

\[
\begin{align*}
\text{maximum} (a :: x) &= \\
&= \{ \text{def. of maximum} \}
\]

\[
\begin{align*}
\text{hd} (\text{sort} (a :: x)) &= \\
&= \{ \text{property of sort} \}
\end{align*}
\]

\[
\begin{align*}
\text{if} \ a > \text{hd} (\text{sort} x) \text{ then } a :: \text{sort} x \\
&= \{ \text{def. of hd} \}
\end{align*}
\]

\[
\begin{align*}
\text{if} \ a > \text{maximum} x \text{ then } a \\
&= \{ \text{define } x \uparrow y = \text{if } x > y \text{ then } x \text{ else } y \}
\end{align*}
\]
A very simple example

**Specification:**

\[
\text{maximum} :: [a] \Rightarrow a \quad \text{maximum} = \text{hd} \circ \text{sort}
\]

\[
\begin{align*}
\text{maximum} (a :: x) & = \{ \text{def. of maximum} \} \\
\text{hd} \ (\text{sort} \ (a :: x)) & = \{ \text{property of sort} \} \\
& \quad \text{hd} \ (\text{if} \ a > \text{hd} \ (\text{sort} \ x) \ \text{then} \ a :: \text{sort} \ x \\
& \quad \quad \quad \quad \quad \quad \text{else} \ \text{hd} \ (\text{sort} \ x) :: \text{insert} \ a \ (\text{tail} \ (\text{sort} \ x)))
\end{align*}
\]
A very simple example

**Specification:**

\[
\text{maximum} :: [a] \Rightarrow a \\
\text{maximum} = \text{hd} \circ \text{sort}
\]

\[
\text{maximum} (a :: x) = \\
\{ \text{def. of maximum} \} \\
\text{hd} (\text{sort} (a :: x)) = \\
\{ \text{property of sort} \} \\
\text{hd} (\text{if } a > \text{hd} (\text{sort} x) \text{ then } a :: \text{sort} x \text{ else } \text{hd} (\text{sort} x) :: \text{insert} a (\text{tail} (\text{sort} x))) = \\
\{ \text{by if law} \} \\
\text{if } a > \text{hd} (\text{sort} x) \text{ then } \text{hd} (a :: \text{sort} x) \text{ else } \text{hd} (\text{hd} (\text{sort} x) :: \text{insert} a (\text{tail} (\text{sort} x)))
\]
A very simple example

**Specification:**

\[
\text{maximum} :: [a] \Rightarrow a \\
\text{maximum} = \text{hd} \circ \text{sort}
\]

\[
\begin{align*}
\text{maximum} (a :: x) & = \{ \text{def. of maximum} \} \\
\text{hd} (\text{sort} (a :: x)) & = \{ \text{property of sort} \} \\
\text{hd} (\text{if } a > \text{hd} (\text{sort} x) \text{ then } a :: \text{sort} x \\
\text{else } \text{hd} (\text{sort} x) :: \text{insert } a (\text{tail} (\text{sort} x))) & = \{ \text{by if law} \} \\
\text{if } a > \text{hd} (\text{sort} x) \text{ then } \text{hd} (a :: \text{sort} x) \\
\text{else } \text{hd} (\text{hd} (\text{sort} x) :: \text{insert } a (\text{tail} (\text{sort} x))) & = \{ \text{def. of hd} \} \\
\text{if } a > \text{hd} (\text{sort} x) \text{ then } a \text{ else } \text{hd} (\text{sort} x)
\end{align*}
\]
A very simple example

**Specification:**

\[
\text{maximum} :: [a] \Rightarrow a \\
\text{maximum} = \text{hd} \circ \text{sort}
\]

\[
\begin{align*}
\text{maximum} (a :: x) &= \{ \text{def. of maximum} \} \\
\text{hd} (\text{sort} (a :: x)) &= \{ \text{property of sort} \} \\
\text{hd} (\text{if } a > \text{hd(sort x)} \text{ then } a :: \text{sort x} \\
\text{else } \text{hd(sort x)} :: \text{insert a (tail(sort x))}) &= \{ \text{by if law} \} \\
\text{if } a > \text{hd(sort x)} \text{ then } \text{hd(a : sort x)} \\
\text{else } \text{hd(hd(sort x)} :: \text{insert a (tail(sort x)))} &= \{ \text{def. of hd} \} \\
\text{if } a > \text{maximum x} \text{ then } a \text{ else } \text{maximum x}
\end{align*}
\]
A very simple example

**Specification:**

\[
\text{maximum} :: [a] \Rightarrow a \\
\text{maximum} = \text{hd} \circ \text{sort}
\]

\[
\text{maximum} (a :: x) = \{ \text{def. of maximum} \}
\]

\[
\text{hd} (\text{sort} (a :: x)) = \{ \text{property of sort} \}
\]

\[
\text{if a > hd(sort x) then a :: sort x} \\
\text{else hd(sort x) :: insert a (tail(sort x))}
\]

\[
\{ \text{by if law} \}
\]

\[
\text{if a > hd(sort x) then hd(a : sort x) else hd(hd(sort x) :: insert a (tail(sort x)))}
\]

\[
\{ \text{def. of hd} \}
\]

\[
\text{if a > maximum x then a else hd(sort x)}
\]

\[
\{ \text{def. of maximum} \}
\]

\[
\text{if a > maximum x then a else maximum x}
\]

\[
\{ \text{define } x \uparrow y = \text{if } x > y \text{ then } x \text{ else } y \}
\]

\[
a \uparrow (\text{maximum } x)
\]
1. An Introduction to Program Calculation
2. Program Calculation and Parallel Programming
3. Calculating BSP Programs
4. BH Skeletons
5. Summary
Join lists can be easily distributed: on $p$ processeurs,

$$[x_0, \ldots; x_{n-1}]$$

could be seen as:

$$[x_0; \ldots; x_{\frac{n-1}{p}}] + + \ldots + + [x_{(n-1)\frac{n}{p}}; \ldots; x_{n-1}]$$

First Homomorphism Theorem:

*Every homomorphism could be written as the composition of a reduce and a map*
\[ h = \text{hom} \bigoplus f \ \text{id} \bigoplus \]

\[
\begin{align*}
\mathcal{P}_0 & \quad \ldots \quad \mathcal{P}_i & \quad \ldots & \quad \mathcal{P}_{p-1} \\
| & \quad | & \quad | & \quad | \\
h ( [x_0; \ldots; x_{n_0-1}] & \quad \ldots & \quad [x_{n_i-1}; \ldots; x_{n_i-1}] & \quad \ldots & \quad [x_{n_{p-2}}; \ldots; x_{n_{p-1}-1}] ) \\
= \{ \text{map phase} \} & \quad \bigoplus & \{ \text{reduce phase} \} \\
\bigoplus_{k=0}^{n_0-1} f \ x_k & \quad \ldots & \quad \bigoplus_{k=n_i-1}^{n_{i-1}-1} f \ x_k & \quad \ldots & \quad \bigoplus_{k=n_{p-2}}^{n_{p-1}-1} f \ x_k \\
= \bigoplus_{k=0}^{n_{p-1}-1} f \ x_k
\end{align*}
\]
With Coq

- Lectures 2 and 3: homomorphism theory in Coq
- Lectures 3 and 4:
  - parallel implementations of map and reduce in BSML
  - verification of BSML programs in Coq
- Lecture 5:
  - support for program calculation in Coq
  - support for parallel program calculation in Coq
    (correspondance with algorithmic skeletons)
Outline

1. An Introduction to Program Calculation
2. Program Calculation and Parallel Programming
3. Calculating BSP Programs
4. BH Skeletons
5. Summary
Calculation of BSP Programs

How to apply program calculation to BSP programs?

▷ What is the relationship between homomorphisms and BSP algorithms?
  ▷ In skeletal programming: we use homomorphisms to hide data communication
  ▷ In BSP programming: we want to use homomorphisms to expose data communication

▷ How to systematically derive homomorphisms that are suitable for the BSP model?

Solution:
BH specific homomorphisms-like for BSP computation
due to Pr. Zhenjiang Hu
The BSP Homomorphism: Informally

BSP Homomorphism

- inspired by homomorphisms
- adapted to the BSP model:
  compute, gather needed informations, compute

Sequential semantics
The BSP Homomorphism: Formally

Definition (BH)

$h$ is a BSP Homomorphism, or BH, if it can be written as:

\[ h[a]l r = [ka]l r \]
\[ h(x \oplus y)l r = h x l (g_r y \otimes_r r) \oplus h y (l \oplus_l g_l x) r \]

where $g_l$ and $g_r$ are homomorphisms with associated associative operators $\oplus_l$ and $\otimes_r$

Conditions

- the homomorphisms condition can be weakened
- discussion in Julien Tesson’s PhD thesis
Writing Specifications

For writing specifications:

- recursive definitions
- well-known collective operators: map, fold, scan, . . .
- communication operators: shift, permute, . . .
- a new collective operator: mapAround

mapAround

- is to map a function to each element of a list
- is allowed to use information of the sublists in the left and right of the element

\[
\text{mapAround} \ f \ [x_1, x_2, \ldots, x_n] = \\
[ f ([], x_1, [x_2, \ldots, x_n]), f ([x_1], x_2, [x_3, \ldots, x_n]), \\
\ldots, f ([x_1, x_2, \ldots, x_{n-1}], x_n, []) ].
\]
Theorem (Parallelisation $\text{mapAround}$ with $BH$)

For a function

$$h = \text{mapAround } f$$

if we can decompose $f$ as $f(\text{ls}, x, \text{rs}) = k(\text{g}_1 \text{ls}, x, \text{g}_2 \text{rs})$, where:

- $k$ is any function,
- $g_i$ is a composition of a projection with a homomorphism

then $h$ is a BSP Homomorphism
Example 1: The Tower Building Problem

Specification

tower \((x_L, h_L) (x_R, h_R)\) \(xs = \) mapAround visibleLR \(xs\)

where visibleLR \((ls, (x_i, h_i), rs) = \) visibleL \(ls \ x_i \wedge \) visibleR \(rs \ x_i\)

visibleL \(ls \ x_i = \) maxAngleL \(ls \ < \ \frac{h + h_i - h_L}{x - x_L}\)

visibleR \(rs \ x_i = \) maxAngleR \(rs \ < \ \frac{h + h_i - h_R}{x_R - x}\)

maxAngleL \([],\) \(-\infty\)

maxAngleL \(([(x, h)] \++ xs) = \ \frac{h - h_L}{x - x_L} \uparrow \) maxAngleL \(xs\)

and the function \(\text{maxAngleR}\) can be similarly defined.
Example 2: All Nearest Smaller Values

Foreach data, find the first smaller value at the left and at the right.
ANSV derivation

\[ ansv \; as = \; mapAround \; nsv \; as \]

where

\[ nsv \; (ls, x, rs) = (nsv_L \times ls, nsv_R \times rs) \]

\[ nsv_L \times [] = -\infty \]
\[ nsv_L \times (ls \; ++ \; [l]) = \text{if} \; l < x \; \text{then} \; l \; \text{else} \; nsv_L \times ls \]

\[ nsv_R \times [] = -\infty \]
\[ nsv_R \times ([r] \; ++ \; rs) = \text{if} \; r < x \; \text{then} \; r \; \text{else} \; nsv_R \times rs \]

MapAround parallelisation requirements

\( nsv \) is not here in the form \( k(gl \; ls, \; x, \; gr \; rs) \)
ANSV derivation

\[ nsv_R \land rs = pickup_R \lor (candidates_R \land rs) \]

- \textit{candidates}_R select potential values regardless of the searched value before exchange
- \textit{pickup}_R do select the final value of interest

MapAround parallelisation requirements

\[ nsv = k(candidates_l \land ls, x, candidates_r \land rs) \]

- \textit{candidates}_R is an homomorphism
- \textit{k} just applies \textit{pickup}_L and \textit{pickup}_R
Example 3: One Dimensional Heat Diffusion Simulation

- Heat equation:
  \[
  \frac{\partial u}{\partial t} - \kappa \frac{\partial^2 u}{\partial x^2} = 0 \quad \forall t, \quad u(0, t) = l \quad \forall t, \quad u(1, t) = r
  \]

  - \(\kappa\) is the heat diffusivity of the metal,
  - \(l\) and \(r\) some constants (the temperature outside the metal)

- A discretised version:
  \[
  u(x, t+dt) = \frac{\kappa dt}{dx^2} \times (u(x+dx, t) + u(x-dx, t) - 2 \times u(x, t)) + u(x, t)
  \]
Exemple 4: One Dimensional Heat Diffusion Simulation

- Usual imperative implementation:

```c
for(int i=0; i<n; i++)
    uu[i] += kappa*dt/(dx*dx)*( (i==(n-1)?r:u[i+1]) +
                                  (i==0?l:u[i-1]) - 2*u[i] );
```

- A recursive, pure functional implementation:

```ocaml
let rec heatSeq l r dt dx kappa (u : float list) : float list =
    match u with
    | [] | ui :: u' -> match u' with
                                        | [] | [ kappa*dt/(dx*dx)*(r +. l -. ui -. ui) +. ui ]
                                        | ui plus1 :: -> (kappa*dt/(dx*dx)*(ui plus1 +. l -. ui -. ui) +. ui) :: (heatSeq ui r dt dx kappa u')
```

F. Loulergue SyDPaCC – Lecture 6 October-November 2013
Exemple 4: One Dimensional Heat Diffusion Simulation

- Usual imperative implementation:

```c
for(int i=0; i<n; i++)
    uu[i] += kappa*dt/(dx*dx)*( (i===(n-1)?r:u[i+1]) +
                                 (i==0?l:u[i-1]) - 2*u[i] );
```

- A recursive, pure functional implementation:

```ocaml
let rec heatSeq l r dt dx kappa (u : float list) : float list =
    match u with
    | []         -> []
    | ui :: u'    ->
      match u' with
      | []         -> [ kappa*dt/(dx*dx)*. ( r +. l -. ui-. ui ) +. ui ]
      | ui_plus1 :: _ ->
        (kappa*dt/(dx*dx)*. (ui_plus1 +. l -.ui-.ui) +. ui)::
        (heatSeq ui r dt dx kappa u')
```

F. Loulergue  SyDPaCC – Lecture 6  October-November 2013
Example 3: One Dimensional Heat Diffusion Simulation

(* heat: float→ float→ float→ float→ float→ (float list)par→ (float list)par *)

let heat l r dt dx gamma u =
  let leftBounds, rightBounds = getBounds l r u in
  ⟨ heatSeq $leftBounds$ $rightBounds$ dt dx gamma $u$ ⟩

getBounds: float→ float→ (float list)par→ (float par)* (float par)

⟨ ⋯, ⋯, ⋯, $a_1^j$, ⋯, $a_n^j$, ⋯, ⋯, ⋯ ⟩
Example 3: One Dimensional Heat Diffusion Simulation

Property of heatSeq

\[
\begin{align*}
\text{heat} \; [] \; l \; r &= \; [] \\
\text{heat} \; [a] \; l \; r &= \; [\text{formula} \; a \; l \; r] \\
\text{heat} \; (x \; ++ \; y) \; l \; r &= \; \text{heat} \; x \; l \; (\text{hd} \; y \; r) \; ++ \;
\text{heat} \; y \; (\text{last} \; x \; l) \; r
\end{align*}
\]
Example 3: One Dimensional Heat Diffusion Simulation

Property of heatSeq

\[
\begin{align*}
\text{heat} \; [] \; l \; r &= [] \\
\text{bh} \; [] \; l \; r &= [] \\
\text{heat} \; [a] \; l \; r &= \text{[formula} \; a \; l \; r] \\
\text{bh} \; [a] \; l \; r &= \text{[k} \; a \; l \; r] \\
\text{heat} \; (x \; +++ \; y) \; l \; r &= \text{heat} \; x \; l \; (\text{hd} \; y \; r) \; +++ \\
&\quad \text{heat} \; y \; (\text{last} \; x \; l) \; r \\
\text{bh} \; (x \; +++ \; y) \; l \; r &= \text{bh} \; x \; l \; (g_r \; y \; \otimes_r \; r) \; +++ \\
&\quad \text{bh} \; y \; (l \; \oplus_l \; g_l \; x) \; r
\end{align*}
\]
Example 3: One Dimensional Heat Diffusion Simulation

Property of heatSeq

\[
\begin{aligned}
\text{heat } [] &| l r = [] \\
\text{bh } [] | l r &= [] \\
\text{heat } [a] | l r &= \text{[formula } a | l r] \\
\text{bh } [a] | l r &= [k a | l r] \\
\text{heat } (x ++ y) | l r &= \text{heat } x | l (hd y | r) ++ \text{heat } y (last x | l) | r \\
\text{bh } (x ++ y) | l r &= \text{bh } x | l (g_r y \otimes_r r) ++ \text{bh } y (l \oplus_l g_l x) | r
\end{aligned}
\]
Example 3: One Dimensional Heat Diffusion Simulation

**Property of heatSeq**

\[
\begin{align*}
\text{heat } [] \mid l \mid r &= [ ] \\
\text{bh } [] \mid l \mid r &= [ ] \\
\text{heat } [a] \mid l \mid r &= \text{[formula } a \mid l \mid r] \\
\text{bh } [a] \mid l \mid r &= \text{[k } a \mid l \mid r] \\
\text{heat } (x + + y) \mid l \mid r &= \text{heat } x \mid l (\text{hd_option } y \ll l \mid r) + + \\
&\quad \text{heat } y (\text{last_option } x \gg l) \mid r \\
\text{bh } (x + + y) \mid l \mid r &= \text{bh } x \mid l (g_r y \otimes_r r) + + \\
&\quad \text{bh } y (l \oplus_l g_l x) \mid r
\end{align*}
\]

with \( l \ll r = \begin{cases} 
  l & \text{if } l \neq \text{None} \\
  r & \text{otherwise}
\end{cases} \)

**BH conditions**

- last_option (resp. hd_option) is a homomorphism with operator \((\gg\gg)\) (resp. \((\ll\ll))\)
Example 4: sparse-matrix vector multiplication

Sparse matrix: array representation of triples \((y, x, a)\):

- \(y\): the row-index of the nonzero element,
- \(x\): the column-index of the nonzero element, and
- \(a\): the value of the nonzero element.

\[
A = \begin{pmatrix}
1.1 & 2.2 & 0 \\
0 & 1.3 & 1.4 \\
0 & 0 & 3.5
\end{pmatrix}
\]

\(as = [(0, 0, 1.1), (0, 1, 2.2), (1, 1, 1.3), (1, 2, 1.4), (2, 2, 3.5)]\)

After multiplication: first element of each row contains the solution, others contain a dummy value

\[
mult \ as \ [3.0, 4.0, 1.0] = [(0, 0, 12.1), (0, 1, \Box), (1, 1, 6.6), (1, 2, \Box), (2, 2, 3.5)]
\]
Specification with \textit{mapAround}

What is needed from the left or from the right?

- element is the first one in the row:
  - compare the row-index with that of the left element

- result value needs:
  - partial sum of the rightward values in the row
  - multiplied by the vector

- from the right:
  - the row-index of the right element
  - the partial sum in the row (of right element)
Example 4: sparse-matrix vector multiplication III

\[
\text{mult as } v = \text{mapAround (f v) as }
\]
\[
\text{where } f \ v \ (ls, (y, x, a), rs) = \\
\text{let } y_l = g_l \ ls; (y_r, s_r) = g_r \ v \ rs \\
\text{in if } (y_l == y) \text{ then } (y, x, \square) \\
\quad \text{elseif } (y_r == y) \text{ then } (y, x, v\langle x \rangle \ast a + s_r) \\
\quad \text{else } (y, x, v\langle x \rangle \ast a) \\
\]

where \( v\langle i \rangle \): \( i \)th element of the vector \( v \)
Example 4: sparse-matrix vector multiplication IV

**Function $g_l$**
Takes the row-index of the last element in a list

$$g_l = (\gg, \lambda(x, y, a).y) \text{ where } a \gg b = b,$$

and any value (here we use $-1$) is a left unit of the operator $\gg$. 
Example 4: sparse-matrix vector multiplication V

Function $g_r \, v$

$$g_r \, v \, [(y, x, a)] = (y, a \times v(x))$$
$$g_r \, v \, [as \, ++ \, (y, x, a)] = \text{let} \, (y', s) = g_r \, v \, as \, as \, ++ \, (y, x, a) \, \text{in} \, (y', \text{if} \, y' == y \, \text{then} \, s + a \times v[x] \, \text{else} \, s)$$

we have:

$$g_r \, v \, [(y, x, a)] = (y, a \times v(x))$$
$$g_r \, v \, (ls \, ++ \, rs) = g_r \, v \, ls \, \circ \, g_r \, v \, rs$$

where $(y_l, s_l) \circ (y_r, s_r) = \text{if} \, y_l == y_r \, \text{then} \, (y_l, s_l + s_r) \, \text{else} \, (y_l, s_l)$

A right unit of the operator $\circ$ is $(-1, 0)$. 
Example 4: sparse-matrix vector multiplication VI

\[ \text{mult as } v = \]
\[ BH(k \ v, ((\circ, \lambda(y, x, a).(y, a \ast v\langle x\rangle))), ((\gg, \lambda(x, y, a).y)))) \] as

where \( k \ v (y_l, (y, x, a), (y_r, s)) = \text{if } y == y_l \text{ then } (y, x, \Box) \]
\[ \text{elseif } y == y_r \text{ then } (y, x, a \ast v\langle x\rangle + s) \]
\[ \text{else } (y, x, a \ast v\langle x\rangle) \]

\[ a \gg b = b \]
\[ (y_l, s_l) \circ (y_r, s_r) = \text{if } y_l == y_r \text{ then } (y_l, s_l + s_r) \text{ else } (y_l, s_l) \]
Outline

1. An Introduction to Program Calculation
2. Program Calculation and Parallel Programming
3. Calculating BSP Programs
4. BH Skeletons
5. Summary
BH Skeleton Implementations

There are two implementations:

- a C++ skeleton for OSL – Orléans Skeleton Library
- a BSML implementation verified in Coq
Parallel implementation

Summaries Computation using \( g_i \) and \( g_r \)

Summaries fusion using \( \oplus_i \) and \( \otimes_r \)

Local \( BH \) computation
Orléans Skeleton Library

- C++ algorithmic skeletons library
- currently implemented on top of MPI
- follows the BSP model

Parallel data structures: distributed arrays

Template: DArray<A>

- computation skeletons: map, zip, reduce, ...
- Communication skeletons: shift, permute, ...
- Distribution management skeletons: redistribute, getPartition, flatten
- Expression template mechanism: loop fusion at compile time
Loop fusion

Calculating $2 \times A + (B \times C)$ on arrays

Simple function calls

4 loops, 3 temporaries

Construction of complex type at compilation

A single loop
Skeleton signature

```cpp
DArray<typename K::result_type>
bh(K k, Homomorphism<T, L> * hl, Homomorphism<T, R> * hr,
    L l, R r, const DArray<T>& temp);
```

Homomorphism example

```cpp
class HAdd: public Homomorphism<int, int> {
    public:
        HAdd() { neutral = 0;}
        inline int f(const int& i) {return i;}
        inline int o(const int& i1, const int& i2) {return i1 +i2;}
};
```
BH applies to

- distributed arrays
- skeleton expressions that produce arrays
- triggers the fusion mechanism when relevant

Efficient implementation

- local applications of the homomorphisms (linear time)
- global exchange of the local summaries
- local applications of the main function (linear time)
class candidatesL:
    public Homomorphism <int, std::vector<int> >
// ..

class candidatesR:
    public Homomorphism<int, std::vector<int> >
// ..

std::vector<int> empty(0);

DArray<int> input = // ...

DArray<std::pair<int, int> > result =
    osl::bh(new candidatesL(), new candidatesR(),
            empty, empty, empty, input);
BH in Coq I

BH Skeleton in BSML

- Formalisation of BH definition
- Computational definitions of BH & proofs of equivalence
  - sequential very inefficient
  - sequential
  - parallel
  - sequential optimised
  - parallel optimised

- extraction of the BSML implementation of BH
About specifying programs

- Proof of the correctness of BSML versions of communication operators (shifts, permute)
- Formalisation of mapAround
- Proof that mapAround is a BH
- Proof that any homomorphism is a BH
- Formalisation of what does it means for a sequential function to be parallelisable

$\Rightarrow$ composition of derivations, communication operators
Demonstration

Using SDPP version nii2013
http://traclifo.univ-orleans.fr/SDPP
Experiments: The Tower Building Problem

Extraction from Coq

- extracted from coq
- direct implementation
- \( f(x) = x \)
Experiments: ANSV & Matrix multiplication

Hand-written OSL versions

![Graph showing speedup vs number of cores for different versions of Sparse Matrix-Vector Multiplication: Ideal curve, ANSV, and Sparse Matrix-Vector Multiplication. The graph illustrates how the speedup increases with the number of cores for each version.]
1. An Introduction to Program Calculation

2. Program Calculation and Parallel Programming

3. Calculating BSP Programs

4. BH Skeletons

5. Summary
Summary

- Program calculation is useful for developing parallel programs in a systematic way
- BSP Homomorphisms are dedicated to bulk synchronous parallel program calculations
- We provide Coq support for program calculation and bulk synchronous parallel program calculation
- BH skeletons exist as:
  - part of the C++ Orléans Skeleton Library
  - a parallel function in BSML, verified in Coq
- Applications using BSP homomorphisms (and Coq):
  - tower building
  - array packing
  - all nearest smaller values
  - 1D heat equation