Systematic Development of Correct Bulk Synchronous Parallel Programs

Louis Gesbert[1], Zhenjiang Hu[2], Frédéric Loulergue[3], Kiminori Matsuzaki[4], Julien Tesson[3]

1 : MLstate, Paris, France
2 : National Institute of Informatics, Japan
3 : LIFO, University of Orléans
4 : Kochi University of Technology

December 10, 2010 – PDCAT

Outline of the Talk

Our Goal

To ease the development of correct and verifiable parallel programs with predictable performances

Parallel Programming

Automatic Parallelization

Structured Parallelism

- Bulk Synchronous Parallelism
- Declarative Parallel Programming
- Algorithmic Skeletons
- ...

Concurrent & Distributed Programming

Bulk Synchronous Parallelism (BSP)

Research on BSP

90' by Valiant & McColl

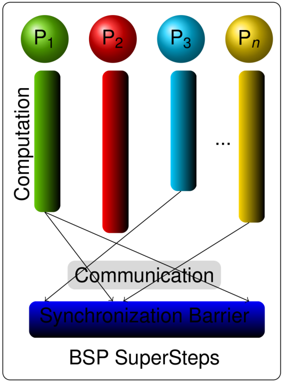
Three models

- abstract architecture
- execution model
- cost model

BSP computer

- p processor / memory pairs *(of speed r)*
- a communication network *(of speed g)*
- a global synchronisation unit *(of speed L)*

Execution model



Cost model

$$T(s) = \max_{0 \leq i < p} w_i + h \times g + L$$

where $h = \max_{0 \leq i < p}\{h_i^+, h_i^-\}$

The Bulk Synchronous Parallel ML Approach

- an efficient functional programming language with formal semantics and easy reasoning about the performance of programs (strict evaluation):

  ML **(Objective Caml flavor)**

- a restricted model of parallelism with no deadlock, very limited cases of non-determinism, a simple cost model:

  Bulk Synchronous Parallelism

The result is:

Bulk Synchronous Parallel ML (BSML)

Bulk Synchronous Parallel ML

Design principles

- Small set of parallel primitives
- Universal for bulk synchronous parallelism
- Global view of programs
- Simple formal semantics

BSML

a sequential functional language

+ a parallel data structure (non nestable)
+ parallel operations on this data structure

Papers and software

- `http://traclifo.univ-orleans.fr/BSML`

To Ease the Writing of BSML Programs . . .

- use the improved set of primitives:
  - W. Bousdira, F. Gava, L. Gesbert, F. Loulergue, and G. Petiot. **Functional Parallel Programming with Revised Bulk Synchronous Parallel ML**. In Koji Nakano, editor, *2nd International Workshop on Parallel and Distributed Algorithms and Applications (PDAA)*. IEEE Computer Society, 2010.

- or do not write any program!
  . . . write only **specifications** and **derive** programs

Development of Skeletal Parallel Programs

A lot of work on systematic derivation of skeletal parallel programs:

- **List homomorphism** plays an important role in this derivation
- There is a good correspondence between skeletons and list homomorphisms
- There is an theory, called **Constructive Algorithmics**, for construction of list homomorphisms

Can we apply the constructive algorithmics theory to derivation of BSP algorithms?

Derivation of BSP Programs

We aim to apply the homomorphic approach to systematic derivation of BSP algorithms.

- 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, a Specific Homomorphism for BSP Computation

List Homomorphism

Function h on lists is a list homomorphism, if

$$h(x +\!\!+ y) = (hx) \odot (hy)$$

for some associative operator $\odot$

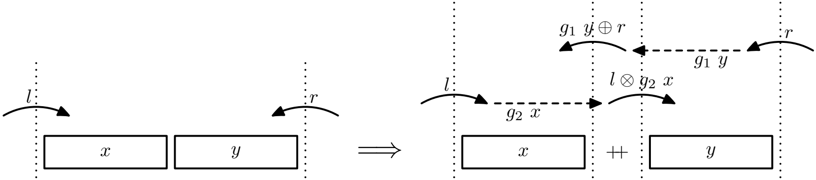
Properties

- Suitable for parallel computation in the D&C style:

$$\mathrm{sum}(x +\!\!+ y) = \mathrm{sum}\, x + \mathrm{sum}\, y$$

- Enjoy many nice algebraic properties

The BSP Homomorphism: Informally



The BSP Homomorphism: Formally

Definition (BH)

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

$$h\,[a]\,l\,r = [k\;a\;l\;r]$$
$$h\,(x \mathbin{+\!\!+} y)\,l\,r = h\,x\,l\,(g_r\,y \oplus_{\mathrm{r}} r) \mathbin{+\!\!+} h\,y\,(l \oplus_{\mathrm{l}} g_l\,x)\,r$$

where g_l and g_r are homomorphisms with associated associative operators $\oplus_{\mathrm{l}}$ and $\oplus_{\mathrm{r}}$

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

$$\begin{array}{l} mapAround\ f\ [x_1, x_2, \ldots, x_n] = \\ \quad [\ f\ ([], x_1, [x_2, \ldots, x_n]), f\ ([x_1], x_2, [x_3, \ldots, x_n]), \\ \qquad \ldots, f\ ([x_1, x_2, \ldots, x_{n-1}], x_n, [])\ ]. \end{array}$$

mapAround is a BSP Homomorphism

Theorem (Parallelisation *mapAround* with *BH*)

For a function

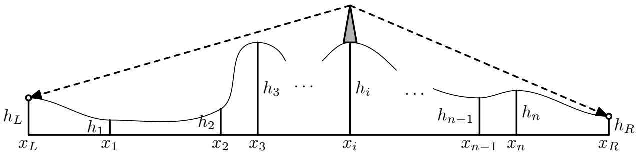
$$h = mapAround\ f$$

if we can decompose f *as* $f\ (ls, x, rs) = k\ (g_1\ ls, x, g_2\ rs)$, *where:*

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

then h *is a BSP Homomorphism*

An Example: The Tower Building Problem



Specification

$$
\begin{array}{l}
tower\ (x_L, h_L)\ (x_R, h_R)\ xs = mapAround\ visibleLR\ xs \\
\qquad \textbf{where}\ visibleLR\ (ls, (x_i, h_i), rs) = visibleL\ ls\ x_i \wedge visibleR\ rs\ x_i \\
\qquad\qquad visibleL\ ls\ x_i = maxAngleL\ ls < \frac{h+h_i-h_L}{x-x_L} \\
\qquad\qquad visibleR\ rs\ x_i = maxAngleR\ rs < \frac{h+h_i-h_R}{x_R-x}
\end{array}
$$

$$
\begin{array}{lcl}
maxAngleL\ [\,] & = & -\infty \\
maxAngleL\ ([(x,h)] +\!\!+\ xs) & = & \frac{h-h_L}{x-x_L} \uparrow maxAngleL\ xs
\end{array}
$$

and the function *maxAngleR* can be similarly defined.

BSML Programs?

- There is a BSML implementation of BH as a higher-order function
- How are we sure BSML implementation actually realizes BH specification?

The Coq Proof Assistant

The Curry-Howard correspondance

Programming World	Logical World
Type	Theorem
Program	Proof

In practice

Coq can be seen as

- a **functional** programming language
- with a **rich type system** able to express logical properties
- plus a language of tactics to build proof terms

Coq allows **program extraction from proofs**

The SDPP Framework in Coq I

About BH

- 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**

The SDPP Framework in Coq II

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**

The SDPP Framework in Coq III

Examples

- Tower Building Problem
- Maximum Prefix Sum Problem
- Array Packing Problem

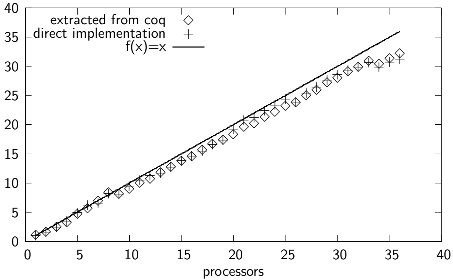
Some statistics

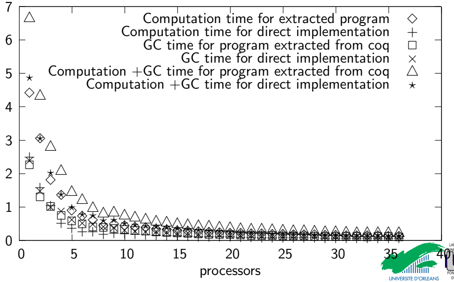
Part	**Spec**	**Proof**
Support	1959	2791
BH	427	1185
Spec	188	186
BSML	895	1088
Examples	254	100
	3723	**5318**

Available at

`http://traclifo.univ-orleans.fr/SDPP`

Experiment: The Tower Building Problem





Conclusion and Ongoing Work

Systematic Development of BSP Programs

Problem Specification

↓

Derivation based on Proved Transformation Theory

↓

Algorithm in BH

↓

Program extraction from Coq-proved BSML implementation of BH

↓

Certified BSP Parallel Programs in BSML

- A new skeleton and its theory for deriving BSP algorithms
- All proofs and formalisations done in the Coq proof assistant
- Experiments with programs extracted from proofs

Ongoing work

- New applications
- More automation for Coq proofs
- Reasoning about BSP costs

Future Work

Verified frameworks for the systematic development of parallel programs from **specifications** to **assembly code**

- New skeletons and their theories
- **Verified** compilers:
  - for BSML
  - for Algorithmic Skeleton C
- Programs extraction and experiments