Slide 1

Nested Data Parallelism in Haskell

Manuel M. T. Chakravarty
University of New South Wales

Joint work with Gabriele Keller

① Nesl-style parallelism in Haskell
② Flattening Transformation

Slide 2

Parallel Arrays

Sparse matrix: $\begin{pmatrix} 5 & 0 & 8 \\ 0 & 0 & 0 \\ 9 & 0 & 0 \end{pmatrix} * \begin{pmatrix} -3 \\ 2 \\ 9 \end{pmatrix}$

$$\textbf{type}\ \mathit{SparseRow} \quad = [:(\mathit{Int}, \mathit{Float}):]$$
$$\textbf{type}\ \mathit{SparseMatrix} = [:\mathit{SparseRow}:]$$

E.g., $[:[:(0,5),(2,8):],[::],[:(0,9):]:]$

— Sparse matrix vector multiplication

$$\mathit{smvm} :: \mathit{SparseMatrix} \to [:\mathit{Float}:] \to \mathit{Float}$$
$$\mathit{smvm}\ \mathit{sm}\ \mathit{vec} =$$
$$[:\mathit{sumP}\ \underbrace{[:x * (\mathit{vec}\ !:\ \mathit{col}) \mid (\mathit{col}, x) \leftarrow \mathit{row}:]}_{\text{products of one row}} \mid \mathit{row} \leftarrow \mathit{sm}:]$$

Slide 3

New data type:

- ➜ Parallel array with α elements: $[:\alpha:]$
- ➜ For example,

$$\textbf{data}\ \mathit{RoseTree}\ \alpha = \mathit{Node}\ \alpha\ [:\mathit{RoseTree}\ \alpha:]$$

-- e.g., useful to implement Barnes-Hut N-body algorithm

List-like operations:

- ➜ Same special syntax, but with $[:\cdot:]$ brackets
  E.g., $[:x + y \mid x \leftarrow xs \mid y \leftarrow ys:]$
- ➜ Prelude functions with suffix P
  E.g., $\mathit{lengthP}\ xs,\ \mathit{mapP}\ (+1)\ xs$

Semantics:

- ➜ All elements are demanded simultaneously
  $\mathit{lengthP}\ [:1, \bot, 3:] = 3$, whereas $[:1, \bot, 3:]\ !:\ 0 = \bot$
- ➜ Finite length

Slide 4

How Shall We Implement Parallel Arrays?

① Very fine-grained multi-threading:
$[:\mathit{foo}\ x \mid x \leftarrow xs:]$ generates $\mathit{lengthP}\ xs$ threads

- ☑ Conceptually simple
- ☑ Provably efficient thread scheduling
- ✗ Thread granularity

② Flattening transformation:
$[:\mathit{foo}\ x \mid x \leftarrow xs:]$ becomes $\mathit{foo}^{\uparrow}\ xs$ where $\mathit{foo}^{\uparrow}$ lifted

- ☑ Improves array performance already on uniprocessors
- ☑ Portability (DM, GPUs & multicores with vector instructions)
- ✗ Requires sophisticated compiler technology

➨ Let's look more closely at flattening...

Slide 5

Flattening

What is flattening?

$$foo :: Int \to Int \to Int$$
$$foo\ x\ y = x * 2 + y$$

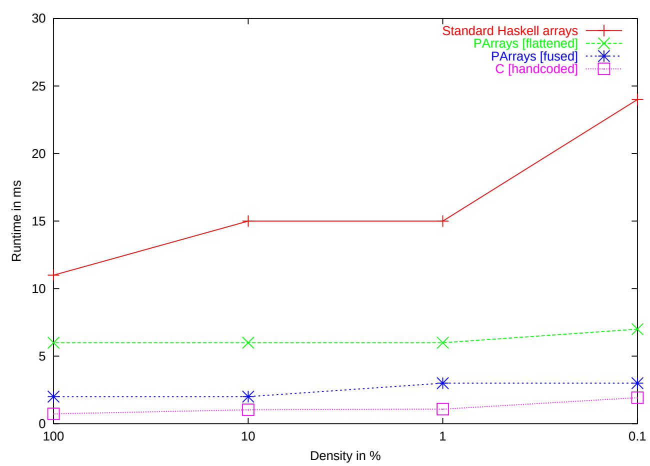
$$[{:}foo\ x\ y \mid x \leftarrow xs \mid y \leftarrow ys{:}] = foo^{\uparrow}\ xs\ ys$$

$$foo^{\uparrow} :: [{:}Int{:}] \to [{:}Int{:}] \to [{:}Int{:}]$$
$$foo^{\uparrow}\ xs\ ys = [{:}x * 2 + y \mid x \leftarrow xs \mid y \leftarrow ys{:}]$$
$$= xs *^{\uparrow} (replicateP\ (lengthP\ xs)\ 2) +^{\uparrow} ys$$

➜ In its full glory more tricky as it has to deal with recursion, higher-order functions, etc.

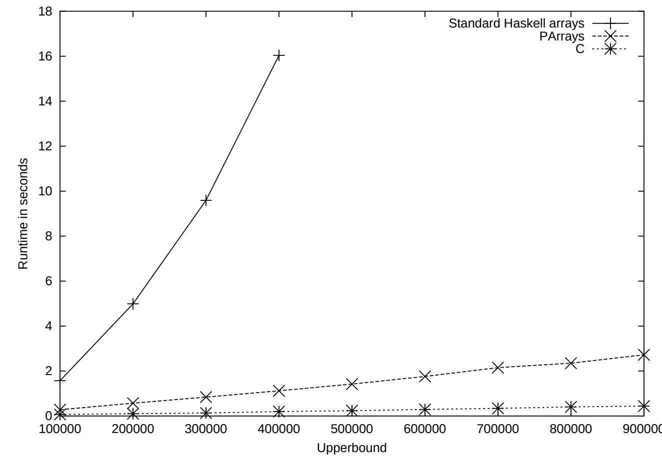
Slide 6

Sequential performance with flattening:



Sparse-Matrix Vector Multiplication: 160,000 non-zeros

Slide 7



Prime sieve

Slide 8

Portability of flattening...

...to distributed-memory machines:

➜ There surely will be clusters of multicores

➜ Flattening gives us a handle on controlling load balancing

➜ Flat arrays are easier to partition than nested arrays (and other irregular structures)

...to graphical processing units (GPUs):

➜ Recently became interesting for general-purpose computing

➜ Stream processing on GPUs generalises classic vector processing

➜ Flattening seems attractive to widen application domain

...to multicores with vector instructions:

➜ Use multiple cores, hyperthreads, and vector instructions simultaneously

Slide 9

Implementing Flattening for Haskell

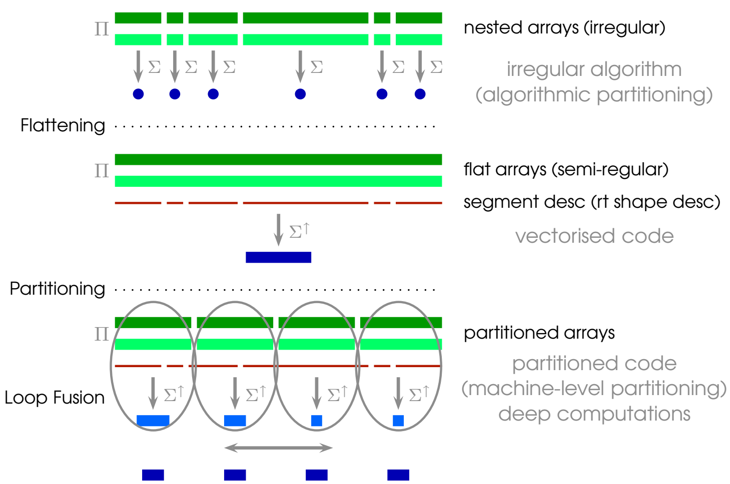
Flattening itself:

① Flattening of data structures
② Vectorising functions
③ Rewriting of closures to support vanilla and vectorised versions of functions

Supporting transformations:

➜ Type-indexed array primitives
➜ Partitioning into threads (guided by types)
➜ Equational array fusion (improves locality of access)

Slide 10



Slide 11

Conclusions

Nested data parallelism:

➜ Generalises regular data parallelism
➜ Convenient programming model for a wide range of applications
➜ Fits nicely into functional programming languages

Flattening:

➜ Transforms nested into flat data parallelism
➜ Already improves sequential array performance in Haskell
➜ Promises portability
➜ Requires further transformations (array fusion)
➜ Requires significant implementation effort