Parallel Programming in Haskell Almost For Free

An Embedding of Intel's Array Building Blocks

Bo Joel Svensson
Mary Sheeran
Chalmers University of Technology

Heterogeneous multi/many core systems

- Processors: Intel Sandy Bridge, Ivy Bridge and AMD APUs
  - multi core + integrated GPU.
  - Each core has SIMD capabilities.
- Discrete graphics: NVIDIA, AMD
  - Now about a thousand of compute elements.
  - CUDA, OpenCL.

Intel's Array Building Blocks

- Increase programmer productivity by:
  - Providing a high level programming model that abstracts from details such as:
    - Number of cores.
    - SIMD width.
  - JIT compile for the current architecture.
    - Platform independence.

Intel's Array Building Blocks

- What is ArBB?

ArBB Virtual Machine

Intel's Array Building Blocks

- What is ArBB?

Intel's Array Building Blocks

- What is ArBB?

A language embedded in C++
ArBB-VM C API
ArBB Virtual Machine

Intel's Array Building Blocks

- The ArBB-VM C API
  - A platform for implementing embeddings.
  - ArBB functions are created by calls to various API routines.
    - arbb_begin_function, arbb_end_function
    - arbb_op
      - add, sub, mul …
      - add_reduce, mul_reduce, …
      - add_scan, mul_scan, ...
      - scatter, gather, ...
    - Low level. The “assembly” language of the ArBB VM.
  - ArBB ops work on
    - Scalars.
    - Dense 1D, 2D and 3D vectors
    - (Irregular) Nested vectors.

Intel's Array Building Blocks

- The C++ Embedding

```
void arbb_matrix_vector(const dense<f32, 2>& a,
                        const dense<f32>& x,
                        dense<f32>& b) {
  b = add_reduce(a * repeat_row(x, a.num_rows()));
}
```

Intel's Array Building Blocks

- The C++ Embedding

```
void arbb_matrix_vector(const dense<f32, 2>& a,
                        const dense<f32>& x,
                        dense<f32>& b) {
  b = add_reduce(a * repeat_row(x, a.num_rows()));
}
```

- Uses C++ features.
  - Templates.
  - Overloading of operators.

Intel's Array Building Blocks and Haskell

- Haskell Bindings to ArBB-VM exist.
  - A very direct mapping of the ArBB-VM C API into Haskell.
  - Started working on an implementation of a backend to Data.Array.Accelerate using these bindings.

Intel's Array Building Blocks and Haskell

- ArBB-VM and Data.Array.Accelerate
  - API missmatch

- ArBB-VM
  - add_reduce, mul_reduce, …
  - gather, scatter, reverse, rotate, ...
  - 1D, 2D, 3D vectors

- Accelerate
  - fold
  - permute, backpermute
  - Arbitrary dimensionality

Intel's Array Building Blocks and Haskell

- EmbArBB
  - Taking the middle path approach.
  - Goal is to provide the same kind of functionality that the C++ embedding has to Haskell programmers.

Intel's Array Building Blocks and Haskell

- EmbArBB
  - Taking the middle path approach.
  - Goal is to provide the same kind of functionality that the C++ embedding has to Haskell programmers.

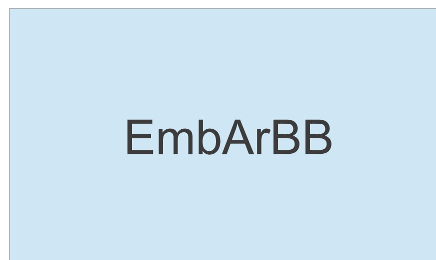
ArBB-VM Bindings

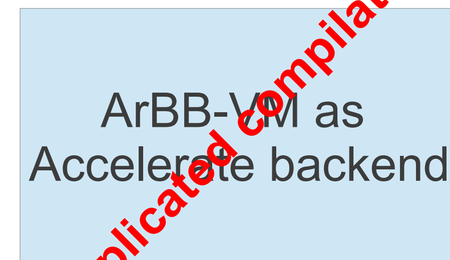
EmbArBB

ArBB-VM as Accelerate backend

Intel's Array Building Blocks and Haskell

- EmbArBB
  - Taking the middle path approach.
  - Goal is to provide the same kind of functionality that the C++ embedding has to Haskell programmers.





EmbArBB Programming

```
matVec :: Exp (DVector Dim2 Float)
       -> Exp (DVector Dim1 Float)
       -> Exp (DVector Dim1 Float)
matVec m v = addReduce rows
           $ m * (repeatRow (getNRows m) v)
```

EmbArBB Programming

- Vectors
  - `DVector d a`
  - `NVector a`

EmbArBB Programming

- Vectors
  - `DVector d a`
  - `NVector a`

DVectors

```
{1,2,3,4,5,6,...}

{{1,0,0},
 {0,1,0},
 {0,0,1}}

{{{1,1,1},{2,2,2},{3,3,3}},
 {{4,4,4},{5,5,5},{6,6,6}}}
```

EmbArBB Programming

- Vectors
  - `DVector d a`
  - `NVector a`

NVectors

```
{{1,2},{3,4,5,6},{7},{9,10}}
```

DVectors

```
{1,2,3,4,5,6,...}

{{1,0,0},
 {0,1,0},
 {0,0,1}}

{{{1,1,1},{2,2,2},{3,3,3}},
 {{4,4,4},{5,5,5},{6,6,6}}}
```

EmbArBB Programming

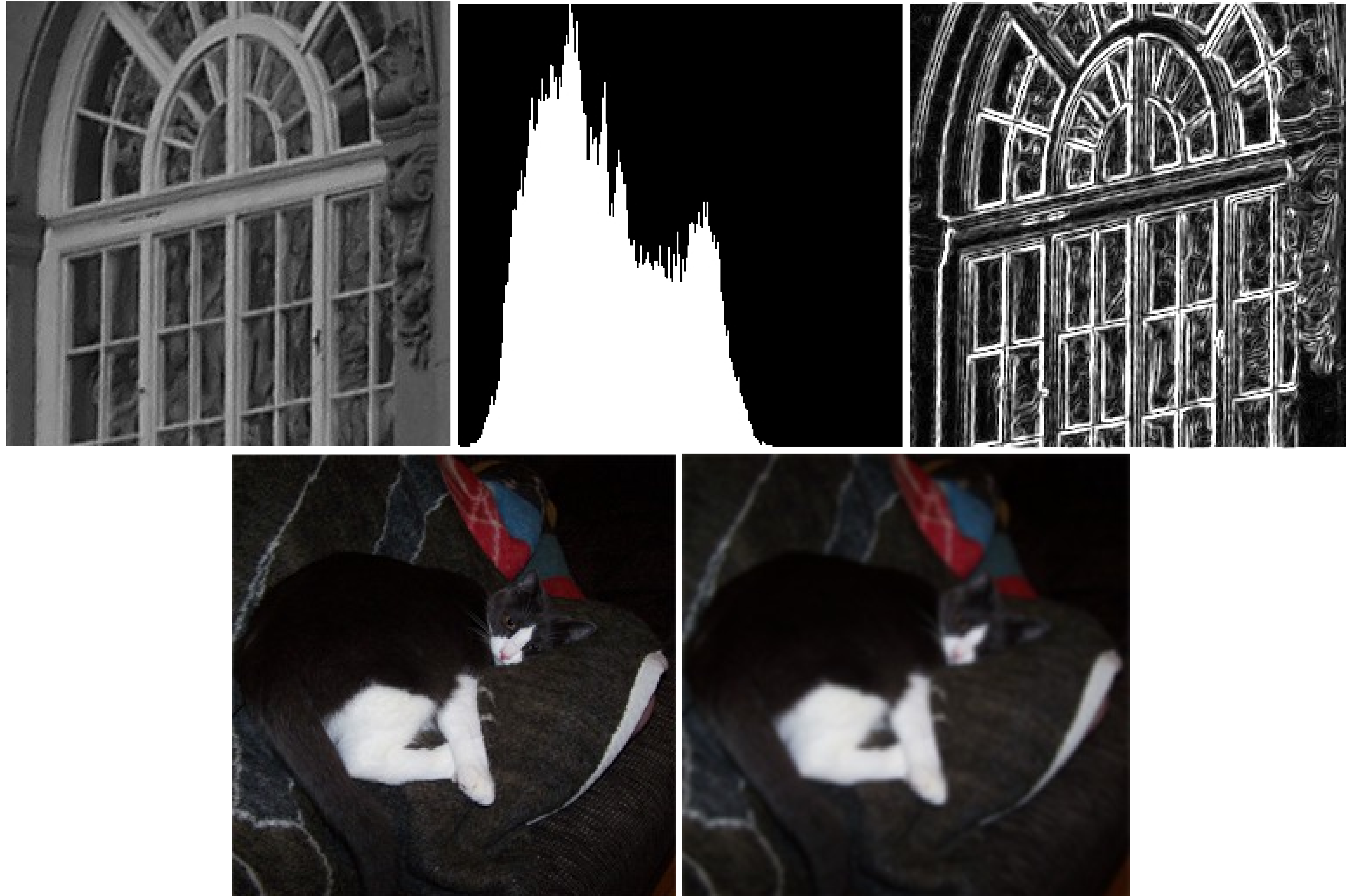
- Library of functions

```
addReduce :: Num a
          => Exp USize   -- rows, cols or pages
          -> Exp (DVector (t:.Int) a)
          -> Exp (DVector t a)

repeatRow :: Exp USize            -- #repetitions
          -> Exp (DVector Dim1 a) -- row
          -> Exp (DVector Dim2 a)

getNRows :: Exp (DVector (t:.Int:.Int) a)
         -> Exp USize
```

EmbArBB Programming: Interfacing with Haskell

```
main =
  withArBB $
  do
     f <- capture matVec
     let m1 = V.fromList [2,0,0,0,
                          0,2,0,0,
                          0,0,2,0,
                          0,0,0,2]
         v1 = V.fromList [1,2,3,4]

     m <- copyIn $ mkDVector m1 (Z:.4:.4)
     v <- copyIn $ mkDVector v1 (Z:.4)

     r1 <- new (Z:.4) 0

     execute f (m :- v)  r1

     r <- copyOut r1

     liftIO$ putStrLn$ show r
```

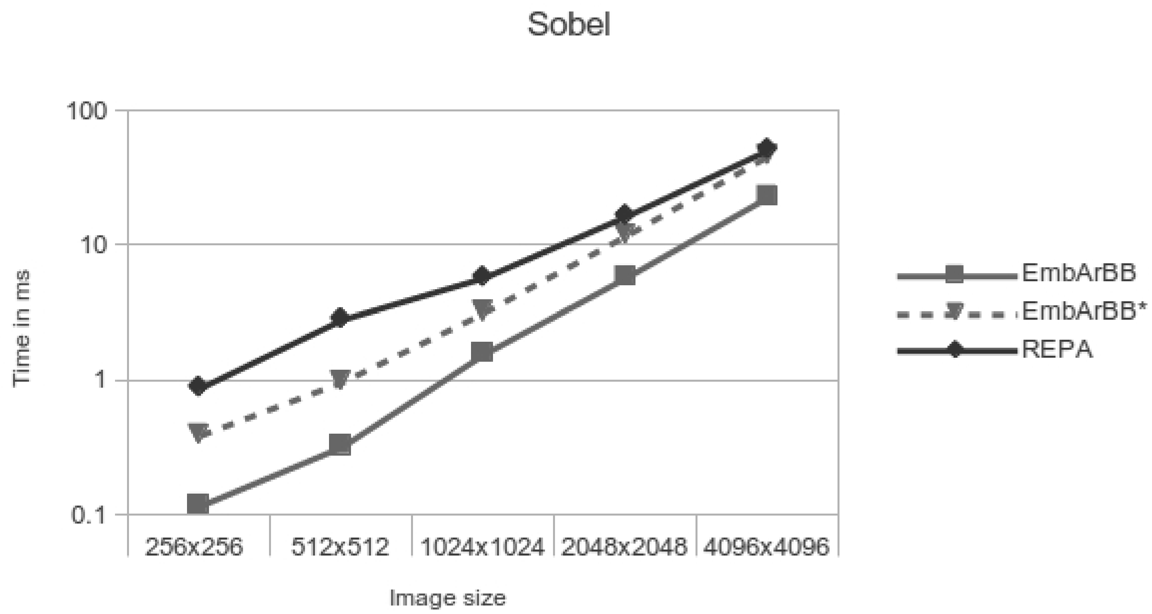
EmbArBB: Benchmarks

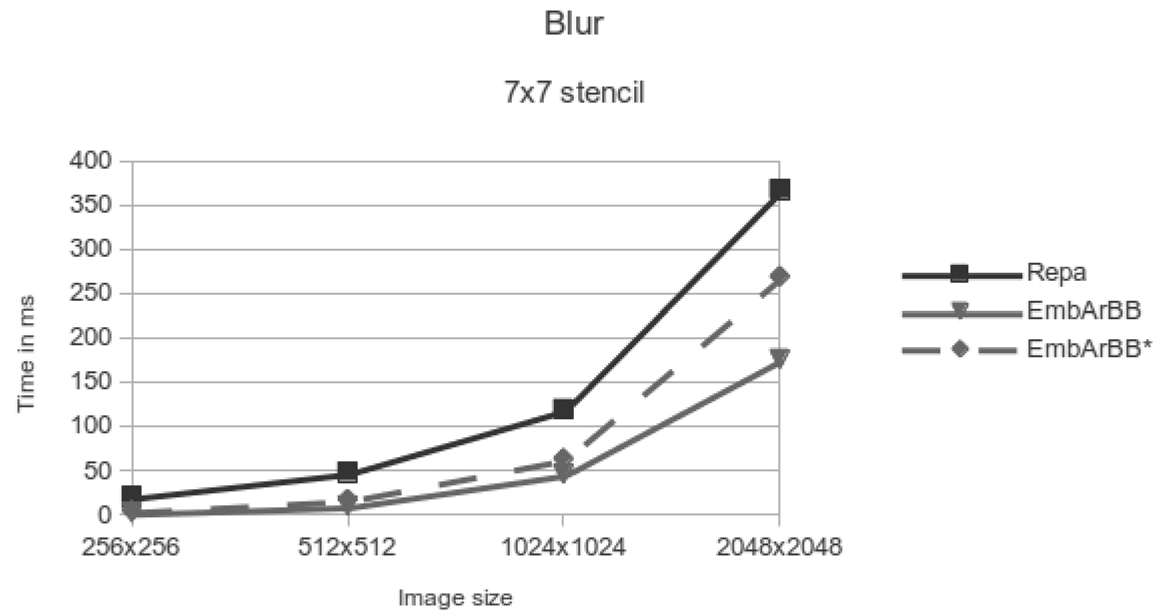


EmbArBB and REPA Programming differences

- In the paper EmbArBB is compared to the REPA embedded language.
  - Similarities: both have shape polymorphic library functions.
  - Differences: The same as in the ArBB / Data.Array.Accelerate case.
    - Specific reductions, add, mul .. in ArBB.
    - …

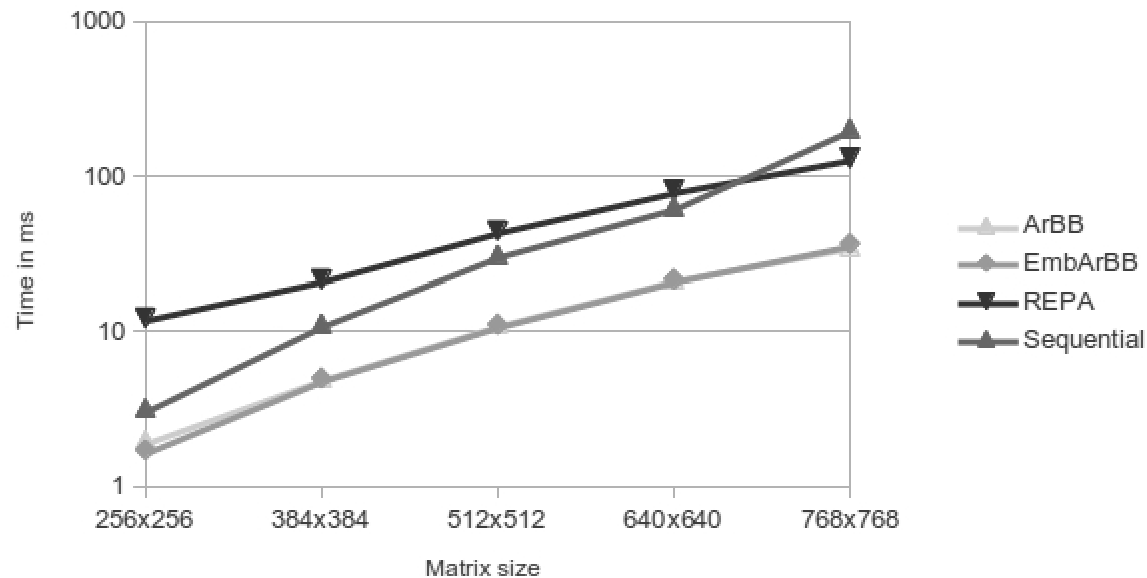
EmbArBB: The benchmark results





EmbArBB: The benchmark results

ArBB/C++ compared to EmbArBB



Just a few words about the Implementation

```
data Expr = Lit Literal
          | Var Variable

          | Index0 Expr
          | ResIndex Expr Int

          | Call (R GenRecord) [Expr]
          | Map  (R GenRecord) [Expr]

          | While ([Expr] -> Expr)
                  ([Expr] -> [Expr])
                  [Expr]

          | If Expr Expr Expr
          | Op Op [Expr]
```

Just a few words about the Implementation

```
data a :. b = a :. b

data Z = Z

type Dim0 = Z
type Dim1 = Dim0 :. Int
type Dim2 = Dim1 :. Int
type Dim3 = Dim2 :. Int
```

Just a few words about the Implementation

```
addReduce :: Num a
          => Exp USize
          -> Exp (DVector (t:.Int) a)
          -> Exp (DVector t a)
addReduce (E lev) (E vec) =
  E $ Op AddReduce [vec,lev]
```

Just a few words about the Implementation

- Phantom types supplies a typed interface.
- Deeply embedded language.
- Uses the StableName library for sharing detection.
- Took influence from Nikola when implementing Call and Map (similar to the vapply approach in Nikola).

Related Work

- REPA & Data.Array.Accelerate
  - Are more expressive languages. (higher order functions).
  - Arbitrary dimensionality on arrays.
  - Accelerate has GPU execution.
  - REPA executes on many threads.
- Nikola
  - Was used as inspiration.
  - GPU execution.

Future Work

- Add support for tuples as elements in vectors.
  - The C++ embedding supports vectors of structs via some AOS to SOA transformation.
- See if it is possible to support higher dimensionality of vectors.
  - Needs to compile down to operations on 1D, 2D, 3D vectors.
- Improve overall robustness.
  - EmbArBB is work in progress.

Conclusion

- EmbArBB
  - Good performance at little implementation effort.
  - Pleasant and simple programming model.
  - Threaded and Vectorized code (SIMD + threads) for free.
  - Portable:
    - Multi core CPUs, MIC.
    - Will it also support GPU execution in the future?
      - Intel integrated GPUs and or Discrete GPUs?
  - Closely tied to Intel ArBB.
    - The usefulness of EmbArBB very much depends on Intel's future plans for ArBB.