

Systems, Networks & Concurrency 2020



Data Parallelism

Uwe R. Zimmer - The Australian National University

Data Parallelism

References

- [Bacon98]
J. Bacon
Concurrent Systems
1998 (2nd Edition) Addison Wesley Longman Ltd, ISBN 0-201-17767-6
- [Ada 2012 Language Reference Manual]
see course pages or <http://www.ada-auth.org/standards/ada12.html>
- [Chapel 1.13 Language Specification Version 0.981]
see course pages or
http://chapel.cray.com/docs/latest/_downloads/chapelLanguageSpec.pdf
released on 7. April 2016

Data Parallelism

Vector Machines

Vectorization

```
type Real_Precision = Float
type Scalar = Real_Precision
type Vector = [Real_Precision]
scale :: Scalar -> Vector -> Vector
scale scalar vector = map (scalar *) vector
```



Data Parallelism

Vector Machines

Vectorization

```
type Real_Precision = Float
type Scalar = Real_Precision
type Vector = [Real_Precision]
scale :: Scalar -> Vector -> Vector
scale scalar vector = map (scalar *) vector
```

Potentially concurrent, yet:

Executed sequentially.





Data Parallelism

Vector Machines

Vectorization



```
import Control.Parallel.Strategies
type Real_Precision = Float
type Scalar = Real_Precision
type Vector = [Real_Precision]
scale :: Scalar -> Vector -> Vector
scale scalar vector = parMap rpar (scalar *) vector
```

Executed in parallel.

This may be faster or slower than a sequential execution.



Data Parallelism

Vector Machines

Vectorization



```
type Real is digits 15;
type Vectors is array (Positive range <>) of Real;
function Scale (Scalar : Real; Vector : Vectors) return Vectors is
  Scaled_Vector : Vectors (Vector'Range);
begin
  for i in Vector'Range loop
    Scaled_Vector (i) := Scalar * Vector (i);
  end loop;
  return Scaled_Vector;
end Scale;
```



Data Parallelism

Vector Machines

Vectorization



```
type Real is digits 15;
type Vectors is array (Positive range <>) of Real;
function Scale (Scalar : Real; Vector : Vectors) return Vectors is
  Scaled_Vector : Vectors (Vector'Range);
begin
  for i in Vector'Range loop
    Scaled_Vector (i) := Scalar * Vector (i);
  end loop;
  return Scaled_Vector;
end Scale;
```

Buzzword collection:
AltVec, SPE, MMX, SSE,
NEON, SPU, AVX, ...

Translates into
CPU-level vector operations

Combined with
in-lining, loop unrolling and caching
this is as fast as a single CPU will get.



Data Parallelism

Vector Machines

Vectorization



```
const Index = {1 .. 100000000},
Vector : [Index] real = 1.0,
Scale : real = 5.1,
Scaled : [Vector] real = Scale * Vector;
```

Function is
"promoted"



Data Parallelism

Vector Machines

Vectorization



```
const Index = {1 .. 100000000},
Vector      : [Index] real = 1.0,
Scale      : real = 5.1,
Scaled     : [Vector] real = Scale * Vector;
```

Function is
"promoted"

Translates into CPU-level vector operations
as well as multi-core or
fully distributed operations



Data Parallelism

Vector Machines

Reduction



```
type Real_Precision = Float
type Vector = [Real_Precision]
equal :: Vector -> Vector -> Bool
equal v_1 v_2 = foldr (&&) True $ zipWith (==) v_1 v_2
```



Data Parallelism

Vector Machines

Reduction



```
type Real_Precision = Float
type Vector = [Real_Precision]
equal :: Vector -> Vector -> Bool
equal v_1 v_2 = foldr (&&) True $ zipWith (==) v_1 v_2
```

Potentially concurrent, yet:

Executed lazy sequentially.



Data Parallelism

Vector Machines

Reduction



```
type Real_Precision = Float
type Vector = [Real_Precision]
equal :: Vector -> Vector -> Bool
equal = (==)
```

Potentially concurrent, yet:

Executed lazy sequentially.



Data Parallelism

Vector Machines

Reduction

A

```
type Real is digits 15;
type Vectors is array (Positive range <>) of Real;
function "=" (Vector_1, Vector_2 : Vectors) return Boolean is
  (for all i in Vector_1'Range => Vector_1 (i) = Vector_2 (i));
```

Translates into
CPU-level vector operations

^<-chain is evaluated lazy sequentially.



Data Parallelism

Vector Machines

Reduction

A

```
type Real is digits 15;
type Vectors is array (Positive range <>) of Real;
function "=" (Vector_1, Vector_2 : Vectors) return Boolean is
  (for all i in Vector_1'Range => Vector_1 (i) = Vector_2 (i));
```

Translates into
CPU-level vector operations

^<-chain is evaluated lazy sequentially.



Data Parallelism

Vector Machines

Reduction

A

```
type Real is digits 15;
type Vectors is array (Positive range <>) of Real;
function "=" (Vector_1, Vector_2 : Vectors) return Boolean is (Vector_1 = Vector_2);
```

Infinite recursion

Translates into
CPU-level vector operations

^<-chain is evaluated lazy sequentially.



Data Parallelism

Vector Machines


Reduction

A

```
type Real is digits 15;
type Vectors is array (Positive range <>) of Real;
function Equal (Vector_1, Vector_2 : Vectors) return Boolean is (Vector_1 = Vector_2);
```

Translates into
CPU-level vector operations

^<-chain is evaluated lazy sequentially.



Data Parallelism

Vector Machines

Reduction

A

```


type Real is digits 15;
type Vectors is array (Positive range <>) of Real;
function Equal (Vector_1, Vector_2 : Vectors) return Boolean renames "=";

```

Translates into
CPU-level vector operations

^-chain is evaluated lazy sequentially.

© 2020 Uwe R. Zimmer, The Australian National University page 412 of 758 (chapter 5: "Data Parallelism" up to page 427)



Data Parallelism

Vector Machines

Reduction

A

```


type Real is digits 15;
type Vectors is array (Positive range <>) of Real;
function "=" (Vector_1, Vector_2 : Vectors) return Boolean is
  (for all i in Vector_1'Range => Vector_1 (i) = Vector_2 (i));

```

Translates into
CPU-level vector operations

^-chain is evaluated lazy sequentially.


© 2020 Uwe R. Zimmer, The Australian National University page 413 of 758 (chapter 5: "Data Parallelism" up to page 427)



Data Parallelism

Vector Machines

Reduction




```

const Index = {1 .. 100000000},
      Vector_1, Vector_2 : [Index] real = 1.0;
proc Equal (v1, v2) : bool
  {return && reduce (v1 == v2);}

```

Function is
 "promoted"


© 2020 Uwe R. Zimmer, The Australian National University page 414 of 758 (chapter 5: "Data Parallelism" up to page 427)



Data Parallelism

Vector Machines

Reduction



```

const Index = {1 .. 100000000},
      Vector_1, Vector_2 : [Index] real = 1.0;
proc Equal (v1, v2) : bool
  {return && reduce (v1 == v2);}


```

^-operations are
 evaluated in a **concurrent
 divide-and-conquer**
 (binary tree) structure.

Translates into **CPU-level vector operations**
 as well as **multi-core or**
fully distributed operations


Function is
 "promoted"

© 2020 Uwe R. Zimmer, The Australian National University page 415 of 758 (chapter 5: "Data Parallelism" up to page 427)

 **Data Parallelism**

Vector Machines


Reduction



```
const Index = {1 .. 10000000},
      Vector_1, Vector_2 : [Index] real = 1.0;
proc Equal (v1, v2) : bool
  {return v1 == v2;}
writeln (Equal (Vector_1, Vector_2));
```


Type mismatch


© 2020 Uwe R. Zimmer, The Australian National University page 416 of 758 (chapter 5: "Data Parallelism" up to page 427)

 **Data Parallelism**


Vector Machines

General Data-parallelism







© 2020 Uwe R. Zimmer, The Australian National University page 417 of 758 (chapter 5: "Data Parallelism" up to page 427)

 **Data Parallelism**


Vector Machines

General Data-parallelism







© 2020 Uwe R. Zimmer, The Australian National University page 418 of 758 (chapter 5: "Data Parallelism" up to page 427)

 **Data Parallelism**

Vector Machines

General Data-parallelism






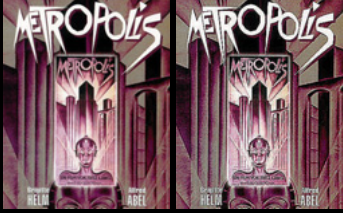

```
const Mask : [1 .. 3, 1 .. 3] real = ((0, -1, 0), (-1, 5, -1), (0, -1, 0));
```

© 2020 Uwe R. Zimmer, The Australian National University page 419 of 758 (chapter 5: "Data Parallelism" up to page 427)

Data Parallelism

Vector Machines

General Data-parallelism

```

const Mask : [1 .. 3, 1 .. 3] real = ((0, -1, 0), (-1, 5, -1), (0, -1, 0));
proc Unsharp_Mask (P, (i, j) : index (Image)) : real
  {return + reduce (Mask * P [i - 1 .. i + 1, j - 1 .. j + 1]);}


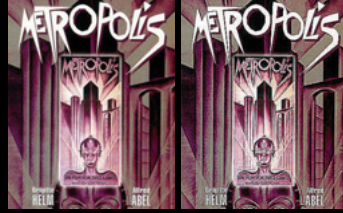

```

© 2020 Uwe R. Zimmer, The Australian National University page 420 of 758 (chapter 5: "Data Parallelism" up to page 427)

Data Parallelism

Vector Machines

General Data-parallelism

```

const Mask : [1 .. 3, 1 .. 3] real = ((0, -1, 0), (-1, 5, -1), (0, -1, 0));
proc Unsharp_Mask (P, (i, j) : index (Image)) : real
  {return + reduce (Mask * P [i - 1 .. i + 1, j - 1 .. j + 1]);}
const Sharpened_Picture = forall px in Image do Unsharp_Mask (Picture, px);




```

© 2020 Uwe R. Zimmer, The Australian National University page 421 of 758 (chapter 5: "Data Parallelism" up to page 427)

Data Parallelism

Vector Machines

General Data-parallelism

Translates into **CPU-level vector operations**
as well as **multi-core** or
fully distributed operations

```

const Mask : [1 .. 3, 1 .. 3] real = ((0, -1, 0), (-1, 5, -1), (0, -1, 0));
proc Unsharp_Mask (P, (i, j) : index (Image)) : real
  {return + reduce (Mask * P [i - 1 .. i + 1, j - 1 .. j + 1]);}
const Sharpened_Picture = forall px in Image do Unsharp_Mask (Picture, px);


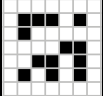

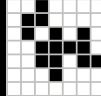
```

© 2020 Uwe R. Zimmer, The Australian National University page 422 of 758 (chapter 5: "Data Parallelism" up to page 427)

Data Parallelism

Vector Machines

General Data-parallelism

```

const Mask : [1 .. 3, 1 .. 3] real = ((0, -1, 0), (-1, 5, -1), (0, -1, 0));
proc Unsharp_Mask (P, (i, j) : index (Image)) : real
  {return + reduce (Mask * P [i - 1 .. i + 1, j - 1 .. j + 1]);}
const Sharpened_Picture = forall px in Image do Unsharp_Mask (Picture, px);


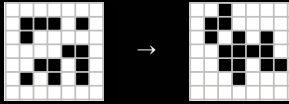
```

© 2020 Uwe R. Zimmer, The Australian National University page 423 of 758 (chapter 5: "Data Parallelism" up to page 427)

Data Parallelism

Vector Machines

General Data-parallelism


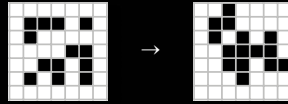
Cellular automaton transitions from a state S into the next state S' :
 $S \rightarrow S' \Leftrightarrow \forall c \in S: c \rightarrow c' = \tau(S, c)$, i.e. all cells of a state transition *concurrently* into new cells by following a rule τ .

© 2020 Uwe R. Zimmer, The Australian National University page 424 of 758 (chapter 5: "Data Parallelism" up to page 427)

Data Parallelism

Vector Machines

General Data-parallelism

Cellular automaton transitions from a state S into the next state S' :
 $S \rightarrow S' \Leftrightarrow \forall c \in S: c \rightarrow c' = \tau(S, c)$, i.e. all cells of a state transition *concurrently* into new cells by following a rule τ .


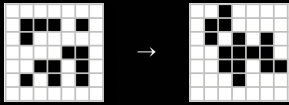
```
Next_State = forall World_Indices in World do Rule (State, World_Indices);
```

© 2020 Uwe R. Zimmer, The Australian National University page 425 of 758 (chapter 5: "Data Parallelism" up to page 427)

Data Parallelism

Vector Machines

General Data-parallelism

Cellular automaton transitions from a state S into the next state S' :
 $S \rightarrow S' \Leftrightarrow \forall c \in S: c \rightarrow c' = \tau(S, c)$, i.e. all cells of a state transition *concurrently* into new cells by following a rule τ .

```
Next_State = forall World_Indices in World do Rule (State, World_Indices);
```

John Conway's **Game of Life** rule:

```
proc Rule (S, (i, j) : index (World)) : Cell {
  const Population : index ({0 .. 9}) =
    + reduce Count (Cell.Alive, S [i - 1 .. i + 1, j - 1 .. j + 1]);
  return (if Population == 3
    || (Population == 4 && S [i, j] == Cell.Alive) then Cell.Alive
    else Cell.Dead);
}
```

© 2020 Uwe R. Zimmer, The Australian National University page 426 of 758 (chapter 5: "Data Parallelism" up to page 427)

Data Parallelism

Summary

Data Parallelism

- **Data-Parallelism**
 - Vectorization
 - Reduction
 - General data-parallelism
- **Examples**
 - Image processing
 - Cellular automata

© 2020 Uwe R. Zimmer, The Australian National University page 427 of 758 (chapter 5: "Data Parallelism" up to page 427)

