Department of Information and Computer Science
Utrecht University

INFOB3CC: Data Parallelism (solutions)

Trevor L. McDonell

January 18, 2024

Introduction

Use these tasks to practice the topics of the lectures. You may have to do some research to read up on terms or topics not (yet) covered in the lectures.

Questions

1. What is data parallelism? How does it differ from task parallelism?

   > **Solution:** In task parallelism a problem is broken down into separate tasks which can be executed in parallel, and coordinate with each other. The (usually static) task decomposition dictates the parallel scalability. The tasks can range in granularity from instructions to statements, loops, functions, or larger.
   >
   > In data parallelism, the same function is applied to [subsets of] the data as a whole. Parallelism scales with the size of the data set.

2. Algorithmic skeleletons

   (a) What is the purpose of using programming patterns—or algorithmic skeletons—to talk about our code?

   > **Solution:** Algorithmic patterns give us a way to separate the semantics of the program (what we want to achieve) from the implementation of the program (how to achieve this, given a particular hardware architecture or programming language). In imperative languages such as C# these two characteristics are (unfortunately) closely intermixed, which makes thinking about parallelism more difficult.

3. In the lecture we discussed the stencil operation and separable filters. To gain some more understanding on the topic here is some additional reading material:

   - Computerphile: Separable Filters and a Bauble
     `https://edu.nl/8kjc6`

   (a) What is a convolution? How can they be implemented in terms of the stencil operator?

   > **Solution:** A convolution is a specialised kind of stencil, where the stencil consists of the kernel coefficients to be multiplied by the corresponding element of the input array and summed to the final value. For a two dimensional image I and kernel K, a convolution operator $\otimes$ can be defined as:

$$(I \otimes K)(x, y) = \sum_i \sum_j I(x + i, y + j)K(i, j)$$

Or a more visual example:

$$\begin{bmatrix} & I_1 & \\ I_2 & & I_3 \\ & I_4 & \end{bmatrix} \otimes \begin{bmatrix} & K_1 & \\ K_2 & & K_3 \\ & K_4 & \end{bmatrix} = I_1K_1 + I_2K_2 + I_3K_3 + I_4K_4$$

(b) What does it mean to implement the Gaussian blur as a separable filter? Why do we do this?

Solution: A separable filter is one which can be rewritten as the product of two or more (simpler) filters. We do this because a $n \times n$ Gaussian blur kernel can be separated into a $1 \times n$ convolution followed by a $n \times 1$ convolution (or vice versa). For large n this is significant.

(c) When can other stencil operators be implemented as separable filters?

Solution: Convolutions in particular can be easily separated, as it is easy to find two vectors whose outer product yields the original convolution matrix. For example, the horizontal Sobel operator:

$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & +1 \end{bmatrix}$$

(d) Explain why implementing the `stencil` operator in-place is difficult. Are there ways in which we can work around this limitation?

Solution: Since the `stencil` operation has access to a local neighbourhood of elements, updating values in-place can change the values which will be read in by later computations. Nevertheless, some stencil codes can be evaluated in-place, for example iterative codes using the red/black successive over-relaxation method.

(e) In a stencil computation, what is the *halo* region? How can the halo region be used when computing a stencil over multiple processors? In particular, think about if each processor has their own memory region which is not shared with the other processors.

Solution: The halo (or ghost) region refers to the area where the stencil pattern would be out-of-bounds, that is, along the edges of the array. In a multiprocessor system, this area could in fact be computed by a neighbouring processor. In order to distribute the stencil code over multiple processors (with discrete memory), the processors synchronise by communicating the values along the boundary with their neighbouring processors. The halo region represents the set of values which are computed by other processors, but are required by this processor in order to update the elements assigned to it.

(f) What is the tiling optimisation in the `stencil` pattern?

Solution: Since stencil codes compute using a local neighbourhood of values, tiling is an optimisation which amortises the cost of reading neighbouring elements over the computation

of multiple output elements. For example, a 3×3 stencil would require 9 input values to compute a single output value. If the computation is instead tiled to output four elements at once (in a 4×1 configuration), it would require 18 input elements to compute 4 outputs, halving read bandwidth requirements.

4. Data parallelism
   (a) What is the `map` pattern?

   Solution: The `map` operator applies a function `f` to every element of the dataset. The function is a computation on a value only, and is independent of the input and output index of that element; for example, it can not alter where the value is written to in the output. To allow safe parallel execution, the operator should have no side-effects.

   (b) Give an example operation which can be implemented using `map`, and an example which can not.

   Solution: An example operation which can be implemented using `map` is computation of the Mandelbrot set. Any operation which is not completely independent at each element can not be computed in this way, such as finding the maximum value in an array.

   (c) How does `map` differ from the `stencil` operation?

   Solution: It differs from `stencil` as it only has access to the one input value, whereas `stencil` has access to a local neighbourhood of elements.

   (d) Are the `map` and/or `stencil` operations embarrassingly parallel? Why?

   Solution: Both are embarrassingly parallel operations as the computation of each element of the result is independent of all of the others.

5. Data motion and layout
   (a) What is the difference between the forward and backward permutation patterns?

   Solution: Backward permutation, or gather, is a parallel random read operation. Forward permutation, or scatter, is a parallel random write operation. Backwards permutation is always safe to execute in parallel, but forward permutation may lead to conflicts if multiple threads map their result to the same location.

   (b) For efficient code execution, the layout of data in memory must be considered. What is the difference between the AoS and SoA representations?

   Solution:

   - AoS: In the array of structures representation, the memory for each object is kept together. This means that the individual fields of successive objects in the array are scattered in memory, which prevents efficient vectorisation. For example:

     ```
     typedef struct Complex_t { float real, imag; } Complex;
     ```

     In an array of `Complex` numbers, the fields will be interleaved as `[real,imag,real,imag]`…

- SoA: In the structure of arrays representation, a separate array for each field of the structure is used. This keeps memory access contiguous over the individual fields of the structure, which is often required for vectorisation.

(c) Complex numbers are typically stored in the AoS representation. Give an example computation where this is a good representation, and an example where this is not the ideal representation.

Solution: Computations which require both components of the complex number for the computation (for example, computing the magnitude of the complex number) benefit from the AoS representation, since only a single memory read will be required to pull both values into the cache. If only one of the fields is used, then this memory bandwidth is wasted.

In the SoA representation, to return only the real (or imaginary) components of an array of complex numbers can be done in constant time, but in the AoS representation the values must be read and packed into a new array which is $\Theta(n)$.

6. Parallel scan

(a) What is the scan operation? What is the difference between the inclusive and exclusive scan?

Solution: The scan operation returns all of the partial reductions of an input sequence. The (left) inclusive scan at index `i` of the output is the reduction of all elements of the input up to and including index `i`. The (left) exclusive scan is the reduction of elements up to index `i-1`.

(b) How would you implement a parallel scan operation using multiple processors?

Solution: We can use the three-phase method as described in the lectures. For example, the exclusive left-to-right prefix-sum on a four processor system:

(a) Split the input into equal chunks and assign to each of the processors.

(b) The first three processors compute the reduction value of their chunk.

(c) A single processor computes an exclusive scan of the reduction values from step (2). We incorporate the initial value at this step.

(d) Each processor performs an exclusive scan of their input, where processor i uses the value at index i of the result from step (3) as their initial value (this is this chunk's carry-in value).

Alternatively, the chained scan could be used.

7. Matrix-vector multiplication

(a) Matrix-vector multiplication is an important operation in many physics and engineering applications. For example, a point p in three-dimensional space can be transformed into a different basis by multiplying by a matrix $\mathbf{A}$, to yield a new point $p' = \mathbf{A}y$, defined as:

$$\begin{bmatrix} ax + by + cz \\ dx + ey + fz \\ gx + hy + iz \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Describe how this operation could be implemented in terms of the data-parallel combinators we have discussed in class.

Solution: Each scalar value in the result is the dot-product of the vector with the corresponding row of the matrix. We can implement dot-product as a pair-wise multiplication of the two vectors (`map`) and then sum the result (`fold`). We apply the dot-product operation to each row of the matrix to compute the final result.

(b) What is *nested* data parallelism? Is your solution to previous question expressed as a nested or flat data-parallel style? Write it using only flat data-parallelism (in Accelerate).

Solution: The sample solution above is in a nested data-parallel style, as we are applying a parallel operation (dot product) in parallel to each row of the matrix (using map).

In Accelerate you can do this by expanding the rank-two argument arrays into rank-three arrays by replicating them across a new dimension, which corresponds to the index space of matrix multiplication:

$$(AB)_{i,j} = \sum_{k=1}^{n} A_{i,k} B_{k,j}$$

where i and j correspond to `rowsA` and `colsB` in the code below. The summation index k corresponds to the innermost axis of the replicated arrays, along which axis we perform the summation after an element-wise multiplication of the replicated elements of `arr` and `brr`.

```
mXm :: Num e => Acc (Matrix e) -> Acc (Matrix e) -> Acc (Matrix e)
mXm arr brr
  = fold (+) 0
  $ zipWith (*) arrRepl brrRepl
  where
    I2 rowsA colsA = shape arr
    I2 rowsB colsB = shape brr
    ---
    arrRepl = replicate (lift $ Z :. All   :. rowsB :. All) arr
    brrRepl = replicate (lift $ Z :. colsA :. All   :. All)
            $ transpose brr
```

(c) A sparse matrix is one in which only the non-zero elements of the matrix are stored. Sparse matrices appear in many applications; see the following site for same example datasets, along with some very nice visualisation: `https://sparse.tamu.edu/about`

For example, the Hardesty3 dataset, which arose out of a computer vision application, consists of only 0.000065% non-zero entries. To store every value explicitly would require around 250 terabytes, but the non-zero values require only 160 megabytes.

In the lectures we described a method for computing a sparse-matrix vector multiply. Suppose we want to parallelise this operation by having threads individually compute each value of the output; that is, thread one computes the first element of the result vector, thread two computes the second element, et cetera. How do we determine the range of values in the sparse matrix each thread should operate over?

Solution: In the compressed sparse row representation, the sparse matrix is represented as a vector of all the non-zero elements together with their column index, along with a segment descriptor recording the number of non-zero elements in each row.

An exclusive prefix-sum of the segment descriptor will compute the offset to the start of each row of the matrix. Thread `i` thus computes with the matrix elements between index `i` and index `i+1` of the offsets vector.

8. Segmented operations

(a) A segmented `scan` can be defined in terms of a regular (non-segmented) `scan`, by the use of an operator lifting function $\oplus^s$. Anneke wishes to implement the segmented plus operator as follows:

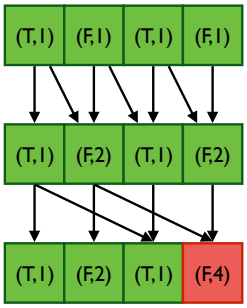
$$(f_x, x) \ +^s \ (f_y, y) = \text{if } f_y \text{ then } (true, y) \text{ else } (false, x + y)$$

where x and y are the array values and f_x and f_y are the corresponding flag values. The operator to the `scan` function should satisfy some mathematical property or properties. Which property/properties does/do not hold for this implementation, but should?

Solution: Associativity

(b) Explain (or draw) a parallel execution of (segmented) `scan` using Anneke's operator, which demonstrates that the operator does not work correctly.

Solution: In a parallel reduction, Anneke's operator will combine results across flag boundaries. Example: `[T,F,T,F]`:



(c) Can you construct a segmented version of the `map` operator, using only a single regular (non-segmented) `map`? Motivate your response.

Solution: Yes. The segmented `map` simply ignores the flag value.

(d) Can you construct a segmented version of the `fold` operator, using only a single regular (non-segmented) `fold`? Motivate your response.

Solution: No. A `fold` returns only a single value, whereas a segmented `fold` requires one return value per segment.

(e) Run length encoding is a form of lossless data compression in which sequences of the same data value (the runs) are stored as a single value together with the count of how many times that value appeared. For example, the string `MMMOOOOOOOOOOWWWWWWOOOOWWW` can be encoded as the array `[(3,'M'), (9,'O'), (6,'W'), (4,'O'), (3,'W')]`.

Using the parallel array functions discussed in the course, provide an implementation of how a run-length encoded string, encoded in an array of (count, value) pairs as shown above, can be decoded into the full uncompressed string. A high-level overview or pseudocode implementation is sufficient. Describe your solution, and include the operator used with the array function(s) of your answer.

Solution: Example solution:

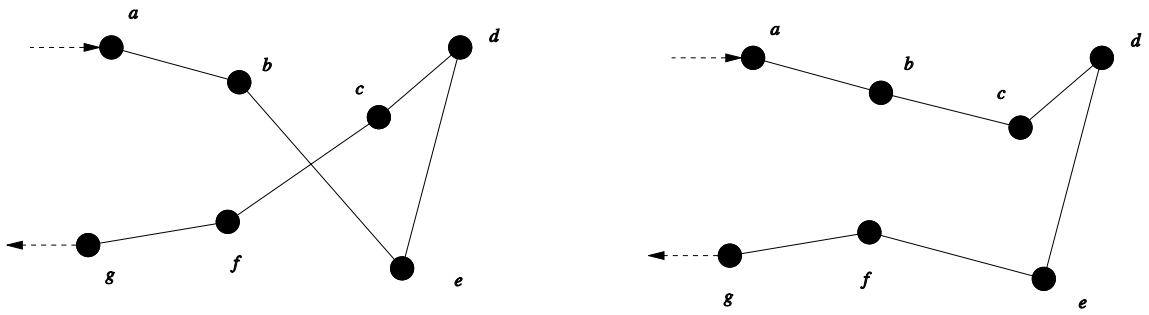
```
rle_decode :: Elt e => Acc (Vector (Int, e)) -> Acc (Vector e)
```

```
rle_decode enc =
  let
      (seg, xs)        = A.unzip enc
      T2 offset len    = scanl' (+) 0 seg
      heads            = permute const
                           (fill (I1 (the len)) undef)
                           (\ix -> I1 (offset!ix))
                           xs
   in
   scanl1Seg const heads seg
```

9. In this question we consider the schedule of a truck which delivers parcels. The route is stored in an array, where the first element of the array is the parcel to deliver first, the second element is the parcel to deliver second, and so on.

   In the following questions consider which of the parallel array combinators discussed in the course can be used to implement the described functionality.

   (a) Some routes can be optimised using the procedure 2-Opt. This method flips the delivery order for part of the route. For example, consider the following delivery schedule:



   The route on the left can be optimised using 2-Opt to the one on the right, that is:

$$[a, b, e, d, c, f, g] \xrightarrow{2-\text{Opt}} [a, b, c, d, e, f, g]$$

   If the start and end indices of the part of the route to flip are given (in the example above, the indices for c and e), which function can be used to perform 2-Opt? Two answers are possible; select them both!

   A. `fold`
   B. `gather`
   C. `map`
   D. `scanl`
   E. `scanr`
   F. `scatter`
   G. `stencil`
   H. `zipWith`

   (b) Is there a reason to prefer one of the approaches in the previous question to the other? Motivate your answer.

> **Solution:** Gather is always safe to execute in parallel. Scatter must deal with the function not being injective and/or surjective. Scattered writes also cause more inter-core communication when threads access the same cache line, even if there is no collision.

(c) Using the function `drivingTime :: Parcel -> Parcel -> Time` we can compute the time to drive between any two parcel delivery addresses.[1] Which of the following operations can be used to compute an array where each element contains the driving time from the previous address? For example, the fourth element of the array will contain the driving time from the third parcel delivery point to the fourth delivery point. Select the correct response.

A. `fold`
B. `gather`
C. `map`
D. `scanl`
E. `scanr`
F. `scatter`
G. `stencil`
H. `zipWith`

(d) It is important for the driver to know how long the deliveries will take. Given an array containing the time to drive between each parcel delivery address, what is the most efficient way to compute the total driving time for the route? Select the correct response.

A. `fold`
B. `gather`
C. `map`
D. `scanl`
E. `scanr`
F. `scatter`
G. `stencil`
H. `zipWith`

(e) We want to inform clients of the expected time at which their parcel will be delivered. Given an array containing the time to drive between each parcel delivery address, how do we compute the expected delivery time of each parcel? Select the correct response.

A. `fold`
B. `gather`
C. `map`
D. `scanl`
E. `scanr`
F. `scatter`
G. `stencil`
H. `zipWith`

(f) As the parcels can be very heavy, the total mass of the delivery truck should be taken into account when computing the driving time between delivery locations. Given the total mass of all of the parcels in the delivery truck, the function `slowdownFactor :: Mass -> Double` returns a multiplicative factor which determines how much slower the delivery truck is compared to an empty truck. For example, a value of 1.0 is the normal driving speed, while a value of 2.0 means the delivery will take twice as long (the truck must drive at half the speed due to the extra mass). After each parcel is delivered, the truck is lighter, and thus for subsequent deliveries can drive faster. Taking this new information into account, how do we compute:

i. The expected delivery time for each parcel; and
ii. The total time to deliver all parcels on the route?

Provide an implementation using the array functions discussed in the course (`fold`, `gather`, `map`, `scanl`, `scanr`, `scatter`, `stencil` and `zipWith`). A high-level overview or pseudocode implementation is sufficient. You may assume a function `parcelMass :: Parcel -> Mass`, which gives the mass of each parcel. Describe your solution, and include the operator used with the array function(s) of your answer.

Solution: Solution sketch:

(a) `scanr (+)` to compute the weight of the truck after each delivery

[1] In the Accelerate library, as used in the third practical, the function `drivingTime` would have an `Exp` type; we ignore this detail for brevity, here and through the remainder of the paper.

(b) `zipWith (\w t -> t * slowdown w)` using the array from (1) and array of unadjusted delivery times between each address.

(c) `scanl (+) 0` to compute the expected delivery times

(d) `fold (+) 0` to compute the total delivery time (or `scanl'` above)

10. A furniture shop has developed an application to manage the stock in their warehouse. Each product consists of one or more parts. A part may also be used for different products. For instance, different tables may use the same legs combined with a different table top.

    To further increase the capacity of the system, the furniture shop considers to use data parallelism to process orders in bulk. Instead of checking the stock level for each product one at a time, the system will now check whether every part for every product of the entire order is in stock at once.

    In the following questions answer using the data parallel operations discussed in the course (`map`, `zipWith`, `fold`, `scanl`, `scanr`, `permute`, `backpermute`, and `stencil`). Nested data parallelism is not allowed. Write code in Haskell.

    (a) An order is given as an array of products. Write a function which computes an array of the parts required to fulfil the order.

    You may use $n(i)$ to denote the number of parts required for product i, and $f(i, k)$ to denote the k-th part of item i, where $0 \leq k < n(i)$.

Solution:

```
parts :: Acc (Vector Product) -> Acc (Vector Part)
parts prods =
  let
      parts_per_prod               = map n prods
      T2 prod_offset total_parts   = scanl' (+) 0 parts_per_prod

      -- a head flags array with a 1 at the start of each segment and a zero
      -- everywhere else
      head_flags = permute const
                           (fill (I1 (total_parts+1)) 0)
                           (\i -> I1 (prod_offset ! i))
                           (fill (shape prods) 1)

      -- a sequence [0,1...] starting at each head flag; this gives the index
      -- k for each part i, to use in the function f
      idxs  = map (subtract 1)
            $ map snd
            $ scanl1 (segmented (+))
            $ zip head_flags
            $ fill (I1 total_parts) 1

      -- for each product i repeats the index i by n(i) times.
      iotas = map snd
            $ scanl1 (segmented const)
            $ zip head_flags
            $ permute const
                      (fill (I1 (total_parts+1)) undef)
```

```
                    (\i -> I1 (prod_offset ! i))
          $ enumFromN (shape prods) 0
in
zipWith f (gather iotas prods) idxs
```

(b) Given an array of parts required for an order, we need to compute the total number of each part which is required. Write a function which computes an array such that the value at index i of the array contains the count of the number of parts of kind i required for this order.

Solution: This is building a histogram. We can assume that there the furniture shop keeps track of n different parts, and each is part i is identified by a unique number in the range $[0, n)$.

```
part_counts :: Acc (Vector Part) -> Acc (Vector Int)
part_counts parts =
  permute (+1) (fill (I1 n) 0) (\i -> I1 (parts ! i)) parts
```

(c) Given the output of the previous question, and an array containing the current stock level for every part in the warehouse, where both are represented as an array where the i-th element denotes the count for parts of kind i. Write a function which determines whether everything for the order is in stock.

Solution:

```
in_stock :: Acc (Vector Int) -> Acc (Vector Int) -> Acc (Scalar Bool)
in_stock current required
  = fold (&&) True
  $ zipWith (>=) current required
```

(d) If all of the parts are in stock for the order, how can we compute the new stock levels, after reserving the parts required for this order?

Solution:

```
zipWith (-)
```

11. The scale and connectivity of the global air travel network increases the risk for infectious diseases to spread. Epidemic control measures can be applied to air travel networks to minimise the risk of large-scale contagion. In order to design the most effective outbreak control measures, we must build a model of the dynamics of infection which can then be used to evaluate the impact of various outbreak control policies.

    We will use an SIR model[2] to represent the evolution of an outbreak in a given population over time, using a set of differential equations specifying the proportion of the population in each possible state an individual can assume: susceptible (S), infected (I), and recovered (R). We have the following discrete

[2]Chen, Nathan, Rey, David, and Gardner, Lauren. Multiscale Network Model for Evaluating Global Outbreak Control Strategies. In *Journal of the Transporation Research Board*, 2017. http://dx.doi.org/10.3141/2626-06

form equations:

$$S_{i,t+1} = S_{i,t} - \frac{\beta_i I_{i,t} S_{i,t}}{N_{i,t}} + \sum_{j \in \theta(i)} S_{ji,t} - \sum_{j \in \theta(i)} S_{ij,t} \tag{12}$$

$$I_{i,t+1} = I_{i,t} + \frac{\beta_i I_{i,t} S_{i,t}}{N_{i,t}} - \gamma I_{i,t} + \sum_{j \in \theta(i)} I_{ji,t} - \sum_{j \in \theta(i)} I_{ij,t} \tag{13}$$

$$R_{i,t+1} = R_{i,t} + \gamma I_{i,t} + \sum_{j \in \theta(i)} R_{ji,t} - \sum_{j \in \theta(i)} R_{ij,t} \tag{14}$$

where $\theta(i)$ is the set of nodes (cities) which have a direct connection to node i, β is the probability of contact between a susceptible and an infected person causing infection, and γ the proportion of infected people recovering per day (the inverse of this value is the number of days it takes for an infected individual to recover). In this question we consider the spread of influenza, for which we can treat these last two terms as constants: $\beta = 0.15$ and $\gamma = 1/7$.

For example, equation (12) says that the number of susceptible individuals in city i at time $t+1$ is given by the number of susceptible individuals at time t (first term), minus the number of susceptible individuals which became infected through contact with an infected individual (second term), then modified by the number of susceptible individuals entering (third term) and leaving (fourth term) the city.

In the following questions you will sketch how this model can be implemented using the data parallel operations discussed in the course (`map`, `zipWith`, `fold`, `scanl`, `scanr`, `permute`, `backpermute`, and `stencil`). Nested data parallelism is not allowed. Write code in Haskell.

(a) At each time step t, the number of individuals in each category (S, I, and R) is modified by the rate at which individuals travel between cities. We can model this with an $N \times N$ adjacency matrix `travelFlows`, where the value at index `Z :. j :. i` represents the number of passengers travelling from city i to city j, where $0 \leq i < N$ and $0 \leq j < N$.

Give a function which computes in data parallel:

i. The total number of individuals departing each city

ii. The total number of individuals arriving at each city

> **Solution:** This is just summing along the rows/columns of the matrix respectively.
>
> Note also that (ii) gives the third term from equations 12-14, and (i) gives the fourth term from equations 12-14.

(b) Given a vector N (the total population at each node i) and vectors S, I, and R containing the current number of individuals in each category respectively at time t, write a function to compute in data parallel each of the vectors S, I, and R at time $t + 1$.

> **Solution:** Once you realise that the previous question computes the second and third terms, the rest is just `zipWith`.

(c) The *infection attack rate* is the percentage of the population which contracts the disease in an at-risk population during a specified time interval. Write a function to compute in data parallel the attack rate at each city for a given time step.

> **Solution:** Also just `zipWith`

(d) In order to model the dynamics of the infection, we iteratively apply equations (12)–(14). You are given matrices S', I', and R' containing the number of individuals in each category for every time step of the simulation. Write a function to compute in data parallel the *peak prevalence* of the infection in each city.

Solution: `fold max` over the time dimension of the matrix

(e) Infectious disease spread is an example of exponential growth because the number of cases on a given day is proportional to the number of cases in the previous day. However, exponential growth like this can not continue forever, and must start slowing down at some point. Given the matrices S', I', and R' from the previous question, write a function to compute in data parallel the inflection point of the curve, the time t at which the *growth factor* first drops to (at or below) one.

Solution: We have:

$$\text{growth factor} = \frac{\Delta N_d}{\Delta N_{d-1}}$$

where ΔN_i gives the number of new cases on day i. So `zipWith` or `stencil` is used to compute ΔN and then again to compute the growth factor. `scanr` is then used to determine the point where this drops below zero.

```
inflection_point :: Acc (Vector Float) -> Acc (Scalar Int)
inflection_point growth_factor
  = backpermute Z_ (::. 0)
  $ map snd
  $ scanr1 (\(T2 v1 t1) (T2 v2 t2) ->
              v1 > 0 && v2 <= 0 ? (T2 v2 t2, T2 v1 t1))
  $ zip growth_factor
  $ enumFromN (shape growth_factor) 0    -- time
```