Coping with Sparse Inputs on Enhanced Meshes – Semigroup Computation with COMMON CRCW Buses

Peter Damaschke
Fern Universität, Theoretische Informatik II
58084 Hagen, Germany
peter.damaschke@fernuni-hagen.de

Abstract

Consider an $\sqrt{n} \times \sqrt{n}$ processor mesh where, in addition to the local links, each row and column is enhanced by a COMMON CRCW bus. Assume that each processor stores an element of a commutative semigroup, and only $k < n$ entries (in arbitrary positions) are nonzero. We wish to compute the sum of all entries. For this problem we easily obtain a lower time bound of $\Omega(k^{1/4})$ if $k \leq n^{2/3}$. Our main result is an $O(k^{1/4} \log^2 k)$ time algorithm. It requires a composition of several data movement and compaction techniques which seem to be of general use for solving problems with sparse inputs scattered on the mesh, as it is typical e.g. for primal sketches in digital image processing.

1: Model, Motivation, and Problem

The mesh-connected computer with row and column buses (other denotations in the literature are: mesh or processor array with multiple broadcasting, enhanced mesh) has reached some attention as an architecture for parallel processing, especially suitable for digital geometry problems. It consists of a $\sqrt{n} \times \sqrt{n}$ grid of processors. Each processor is connected to its (at most four) neighbors by local links. Additionally, each row and column is equipped with a bus for long-distance communication. Each processor has local memory (of usually $O(\log n)$ bits) and applies in each time unit a global instruction to its own data. A processor can perform local computations and send and receive messages through the local links and buses. All processors in a row or column, respectively, can simultaneously read the message from the bus, but for technical reasons only one message per step can be broadcast by the bus. If only one processor per step is allowed to send a message, we speak of CREW buses, according to the terminology for PRAMs.

Since the first treatment in [9], many complexity results have been obtained for this architecture. Usually one assumes that the input has length $n$ and is pretiled onto the whole mesh, or it has length $k < n$ and stands initially in $k$ prescribed processors [2]. The mesh is often considered as a natural device for geometric problems on digital images ([9] [10] [3] and others) – every processor represents a pixel or a square of pixels and stores a color, in the simplest case only black or white. But the known mesh algorithms for geometric problems have complexities depending on the number $n$ of processors, even for sparse digital images with $k \ll n$ black pixels. On the other hand, in "continuous" computational geometry complexities are provided as functions of the number of given geometric features (points, line segments etc.). This discrepancy provokes the following metaproblem: Can we solve the fundamental computational problems for sparse inputs on the mesh in time depending only on the number $k$ of given features, regardless the size $n$ of the mesh? Note that the $k$ items stand in arbitrary positions now.

In [5] we give a general negative answer if CREW buses are presumed: Just the problem of informing some target processor about the coordinates of $k$ items has a complexity of $\Theta(n^{1/3})$, even if $k = 2$. (Case $k = 1$ is trivially solvable in constant time.) The results still holds if the processors have unbounded power and memory. The proof is a nontrivial adversary argument using only the prohibition of concurrent writing.

So what happens if we allow concurrent write access to the buses? Since a bus can carry only one message simultaneously, we must demand that concurrent processors write the same. Therefore we speak of COMMON CRCW buses. Such modifications of the mesh model were already suggested in early papers, e.g. [9]. But for problems with dense input this seems to be irrelevant: many provably optimal mesh algorithms require only CREW buses.

The aim of this paper is to demonstrate that the mesh with COMMON CRCW buses can quickly solve problems with sparse inputs, in the above sense. As a prototype problem we study semigroup computations. Let $(S, +)$ be a fixed commutative semigroup, w.l.o.g. containing a zero element. Examples are the sum, product, or maximum of numbers; for short we will always speak of "sums" throughout the paper. Given one element from $S$ in each processor, we wish to compute the sum of all these entries.

This problem is well-understood on enhanced meshes. As shown in [9], the time complexity is $\Theta(n^{1/3})$. Somewhat surprisingly, suitable non-square meshes can perform semi-
group computation even in $\Theta(n^{1/8})$ time [1]. An optimal
time bound for semigroup computation with $k$ nonzero en-
tries has been derived in [2], but the relevant items are as-
sumed to stand initially in prescribed processors.

In the present paper we consider the semigroup problem
with $k$ nonzero entries in $k$ unknown processors. Even the
number $k$ may be unknown in advance. With COMMON
CRCW buses we will obtain time bounds of $\Omega(k^{1/4})$ for
$k \leq n^{2/3}$, and $O(k^{1/4} \log^2 k)$.

As an illustrative example for the relevance of sparse
semigroup computation consider e.g. the computation of
the centroid of $k$ given points in the digital plane. How-
ever, semigroup computation is a basic problem occurring as
a subproblem in many contexts.

Whereas the earlier semigroup algorithms for items in
prescribed processors [9] [1] [2] are of quite simple regular
structure, we must now combine the data movement tech-
niques in a more refined way. Additionally (and most im-
portantly) we need a result from [11] concerning the parallel
compaction of scattered information. We are convinced that
the introduced approach is also applicable to the other basic
computational problems, rather than semigroup computa-
tion. For obtaining a first insight we choose the semigroup
problem as a subject of study because of its particular sim-
plicity.

We conclude this section with some terminology. We pre-
sume that the reader is familiar with the PRAM models, see
example [7]. Throughout the paper, the term mesh means the
$\sqrt{n} \times \sqrt{n}$ processor array with local links and a COMMON
CRCW bus in each row and column. Every processor knows
its coordinates $(i,j)$ ($1 \leq i, j \leq \sqrt{n}$) where $i$ and $j$ is the
number of the row and column, respectively, where the pro-
cessor is located. A submesh is the set of all processors
$(i,j)$ satisfying $a \leq i \leq b$ and $c \leq j \leq d$ for arbitrarily
given $a, b, c, d$. Let $i_1 < \ldots < i_r$ and $j_1 < \ldots < j_s$ be
arbitrarily chosen row and column numbers. A subgrid is
the set of all processors $(i,j)$ such that $i = i_\rho$ and $j = j_\sigma$
where $1 \leq \rho \leq r$ and $1 \leq \sigma \leq s$. So every submesh is a
subgrid, but not vice versa. With respect to a fixed subgrid,
$(\rho, \sigma)$ are called the relative coordinates of $(i,j)$. A subgrid
has width $w$ if $w$ is smaller than all $i_{\rho+1} - i_\rho$ and $j_{\sigma+1} - j_\sigma$.

Consider a fixed instance of the semigroup problem. An
item is a nonzero entry, and $k$ denotes the number of items.
A subgrid is called occupied if it includes all items of our
instance. So the minimal occupied subgrid contains at least
one item in each row and column, and no item is outside.
Two instances with equal sums are called equivalent.

The general semigroup problem is sensitive, i.e. for an
arbitrary partial input it is impossible to compute the sum
whithout knowing the last missing input value. But note
that this is not valid in all special semigroups, consider e.g.
$(S, \max)$ with a finite set $S$.

For proving a simple lower bound we adapt the well-
known information flow argument (see e.g. [1]) to our prob-
lem.

**Theorem 1** Any algorithm for general semigroup computa-
ton on the mesh requires $\Omega(k^{1/4})$ time if $k \leq n^{2/3}$.

**Proof** Consider instances with $k$ items in a $k^{1/2} \times k^{1/2}$ sub-
grid of width $k^{1/4}$. Obviously, in the first $k^{1/4}/2$ steps no
processor can become aware of more than one item by lo-
cal communication only. In the following we pretend that,
whenever some processor writes onto some bus, all pro-
cessors in the mesh (not only along the bus) receive this
message "for free". This concession might only reduce the
lower bound. An item written onto some bus in step $t$ is
called new if it has not been broadcast by buses before step
$t$. Clearly, while $t < k^{1/4}/2$ only $O(k^{1/4})$ buses can broad-
cast new items. Each of these buses can carry only one
semigroup element (i.e. an original item or a partial sum).
Due to sensitivity, a processor must know all the $k$ items in
order to compute the sum. This yields $k = O(k^{3/4}t)$, and
the assertion follows. \(\Box\)

Note that we obtain for larger $k$ the well-known $\Omega(n^{1/6})$
bound. The proof applies to any sensitive problem.

Now we wish to derive a possibly tight asymptotic upper
bound. One principal problem is to find quickly the $k$ items
being scattered somewhere on the mesh. In [11], P.Ragde
provided an algorithm for ordered compaction on the com-
mon CRCW PRAM with $m$ memory cells and $p \geq m$
processors. Based on perfect hashing, it moves $q$ items from
arbitrary memory cells into the first $q$ cells, preserving the
ordering of the items, in $O(q/\log q \log \log p)$ time, and there
is no need to know $q$ in advance.

We shall simulate this algorithm on the mesh and, with
help of this, we determine the minimal occupied subgrid for
our given instance, i.e. we inform each processor about its
relative coordinates. Since the at most $k \times k$ processors
in our occupied subgrid are responsive then by their rela-
tive coordinates, we can afterwards apply the standard tech-
niques in this subgrid, thus obtaining a complexity bound
depending on $k$ only.

**Lemma 2** An $O(t)$ time computation on a common
CRCW PRAM with $m$ memory cells and $p$ processors can
also be executed on an $m \times p$ mesh in $O(t)$ time without
using the local links.

**Proof** We devote a processor in the first row of the mesh to
each PRAM processor, and a processor in the first column
to each memory cell. (So processor $(1,1)$ is covered twice
which doesn’t matter.) W.l.o.g. assume that the PRAM
performs local computations, read and write operations in
distinct steps, and a processor is concerned with at most
one cell each time (but possibly several processors with the

---

2: The Complexity of Sparse Semigroup Computation
same cell). We have to simulate every PRAM step in constant time on the mesh. This is straightforward in all three cases and can be omitted here. Only note that, in fact, there arise no write conflicts on the buses. □

Lemma 3 Given an instance with \( k \) items, the minimal occupied subgrid can be determined in \( O(\log k) \) time. That means, in this time we can tell all processors in the minimal occupied subgrid their relative coordinates.

Proof Each processor containing an item writes some special symbol onto its row bus. Reading from the row buses, each processor in the first column knows whether or not its row contains at least one item. A processor in the first column stores a row indicator iff its row is nonempty.

By Lemma 2 we can use Ragde’s ordered compaction algorithm to move the \( q \leq \min\{k, \sqrt{m}\} \) row indicators into the first \( q \) processors of the leftmost column, preserving their ordering. Note that a \( q \times \sqrt{m} \) subgrid is available for this, hence we need \( O(\log q/\log \log \sqrt{m}) = O(\log k) \) time. This is the only time-consuming procedure, all other steps can be done in constant time. Supplying the row indicators with their original row numbers, we thus obtain in the first \( q \) processors an ordered list of the occupied rows. Clearly, the position of a row indicator in this list is the row component of the relative coordinates. We now additionally supply the row indicators with these list positions.

Next distribute the row indicators to \( q \) different columns using the row buses, and move them back to their original rows through the column buses. Finally, every processor containing a row indicator sends its list position to all processors of its row.

Proceed similarly with the columns. Obviously, a processor belongs to the minimal occupied subgrid iff it has received two numbers from the row and column indicators, and these are exactly the relative coordinates. □

In the following let \( r \) denote the largest relative coordinate value. So we have an occupied subgrid of size \( r \times r \), obtained from the minimal occupied subgrid by adding some empty rows or columns.

Once the relative coordinates are determined, we could now simply give an \( O(r^{1/2}) \) algorithm, adopting parts of the semigroup algorithm from [9]. If our occupied subgrid is completely full \( (k = r^2) \) then this is \( O(k^{1/4}) \). But in the other extremal case \( k = r \) (each row or column has only one item) this solution would be bad – we only get \( O(k^{1/2}) \). In the latter case it is better to ask for the positions of the items in each row by binary search, and to remove and add the items found in this way. But if some rows are fully occupied and others are sparse, both approaches give time bounds with avoidable high exponents. The general difficulty is that we obtain from applying Lemma 3 neither \( k \) (which may be any number from \( r \) to \( r^2 \)) nor the distribution of the items in our occupied subgrid.

Our next idea is to combine the two approaches for the full and sparse case appropriately. The next three lemmas establish some basic procedures for this.

Lemma 4 Assume all items stand in some \( r \times r \) subgrid. Then we can transform the given instance in \( O(w) \) time into an equivalent instance, contained in an occupied subgrid of size \( s \times s \) \((s \leq r)\) and width \( w \).

Proof Partition the entire mesh into submeshes of size \( w \times w \), sum up the items in each submesh in \( O(w) \) time, and store the partial sums in the left upper corners of the submeshes. For this we only need the local links, so we can proceed in all submeshes in parallel. Obviously, the number of occupied rows and columns cannot increase. □

Lemma 5 If all items in a row stand in \( r \) processors with known identification numbers \( 1, \ldots, r \) then we can remove and add \( q \) of them in \( O(q \log r) \) time, using only the row bus.

Proof For any \( i, j \) with \( 1 \leq i \leq j \leq r \) we can ask in one step whether at least one of the processors with number from \( i \) to \( j \) contains an item. For this, we send \( i, j \) to each processor of the row. Then any processor compares its number \( c \) with \( i \) and \( j \). If \( i \leq c \leq j \) and an item is present then it writes a special symbol onto the bus. So we can identify in \( O(q \log r) \) time \( q \) processors containing items, simply by binary search. The details are straightforward. Whenever an item is detected, we send it through the bus to some target processor where all arriving items are added. □

We can apply Lemma 5 in parallel in many rows and store all partial sums in the first column such that they can be finally added in logarithmic time.

Lemma 6 If an occupied \( r \times r \) subgrid of width \( w \) is known then we can complete the semigroup computation in \( O(w + r/w) \) time.

Proof Partition each row of the occupied subgrid into \( w \) subrows of size \( r/w \). Move the contents of these subrows into the next \( w \) (empty) rows of the mesh, each subrow into a separate empty row. Since the processors know their relative coordinates and can be informed about \( r \) and \( w \), they can simply compute the destinations of their items. Using only the local links for distributing the items, this can be executed for all rows in parallel in \( O(w) \) time.

Now any row of the mesh contains at most \( r/w \) items in \( r/w \) distinguished processors which still know the relative coordinates of the original positions of their items. So we can compute in a straightforward way the partial sums of the rows in \( O(r/w) \) time, using only the row buses.

At this moment we have at most \( wr \) items in the first column of the mesh. Simulating a PRAM due to Lemma 2 we finally add them in \( O(\log(wr)) \) time. With \( O(\log(wr)) = O(\log w + \log r) = O(w + \sqrt{r}) \) we get the asserted time bound. □

The next central lemma shows, roughly speaking, how to construct smaller and fuller occupied subgrids, that is, we increase the exponent \( a \) in an estimation of kind \( r^a \leq k \).
Lemma 7 Suppose we know a lower bound $r^a \leq k (1 \leq a \leq 2)$ for the initial number $k$ of items, and an occupied $r \times r$ subgrid of width $r^{a/4}$ is already determined. Further let $b$ be some real number with $0 < b < a/4$. Then we can transform our instance of the semigroup problem into an equivalent instance such that: All items stand in some $s \times s$ subgrid with $s \leq r$, the term $s^{1+a/4+b}$ is a new lower bound for $k$, and the transformation needs $O((a/4 - b)^{-1} k^{1/4} \log k)$ time.

Proof The transformation algorithm works for each row of the considered occupied subgrid independently in parallel, since it uses only row buses and local links. In the following, all considerations are done with one fixed row, called a principal row. The initially empty rows outside our occupied $r \times r$ subgrid are called auxiliary rows.

The algorithm runs in successive rounds $i = 0, 1, 2, \ldots$ Before the $i$-th round we have a consecutively enumerated subset $P_i$ of $r^c$ processors $(0 \leq c_i \leq 1)$ in the principal row, containing all items of this principal row. Initially we have $c_0 = 1$, and the numbers are simply the column components of the relative coordinates. Next we specify what happens in the $i$-th round.

Partition $P_i$ into $r^{a/4}$ blocks of size $r^{c_i-a/4}$ and move the contents of each block into a separate row among the next $r^{a/4}$ auxiliary rows in $O(r^{a/4})$ time. Again, the correct destinations of items are easily computable from the processor numbers and $r, a, c_i$. If $c_i < a/4$ then we get only one block; we need not distinguish this as a separate case. Using Lemma 5, detect in each auxiliary row up to $r^{a/4}$ items, remove and add them in $O(r^{a/4} \log r)$ time.

As earlier, establish a row indicator in the leftmost processor of every auxiliary row still containing at least one item. In a straightforward way, assign consecutive numbers to these row indicators in $O(r^{a/4})$ time. Particularly, the last row indicator knows the number of still nonempty auxiliary rows. If this number is larger than $r^b$ then move the remaining items back and stop the entire process for our principal row.

Consider the case that at most $r^b$ auxiliary rows remain nonempty. Clearly, the total number of remaining items is bounded then by $r^{c_i+b-a/4}$. Construct a new subset $P_{i+1} \subset P_i$ of size $r^{c_i+b-a/4}$ containing all these items. The only problem is to delete the processor numbers from these row indicators in $O(r^{a/4})$ time. To realize this, remember that we had $r^b$ nonempty auxiliary rows, and from each of them we already removed $r^a$ items. Hence, if $r'$ denotes the number of surviving principal rows, we have $k \geq r'r^{a/4+b}$. (Note that $k$ was the initial, not the actual number of items.)

In a second stage proceed similarly with the columns of our occupied subgrid in $O((a/4 - b)^{-1} k^{1/4} \log k)$ time. For the same reasons as above, each of the $r''$ surviving principal columns possessed at least $r^{a/4+b}$ items before the whole transformation, hence also $k \geq r''r^{a/4+b}$. With $s := \max\{r', r''\} \leq r$ we obtain an occupied $s \times s$ subgrid and $k \geq s^{1+a/4+b} \geq s^{1+a/4+b}$. □

Now we are ready to prove our upper bound.

Theorem 8 Semigroup computation with $k$ items on the mesh can be performed in $O(k^{1/4} \log^2 k)$ time.

Proof Let $k$ be the initial number of items, that means, $k$ is not changed during computation. Set $a := 1$. Repeat the following loop until $a \geq 2 - \delta$ is reached (we specify $\delta$ later). Observe that $r^a \leq k$ is a loop invariant.

- Determine the minimal occupied subgrid by Lemma 3 in $O(k \log k)$ time. Augment it to a minimal square subgrid of size $r \times r$.

- Construct a new occupied $s \times s$ subgrid of width $w = s^{a/4}$ and set $r := s$. By Lemma 4 this needs $O(r^{a/4}) = O(k^{1/4})$ time.

- For a suitable $b$ that we specify later, compute a new occupied subgrid such that $k \geq r^{1+a/4+b}$ by Lemma 7 in $O((a/4 - b)^{-1} k^{1/4} \log k)$ time.

The last step dominates the complexity of the loop. When $a \geq 2 - \delta$ is achieved we apply Lemma 4 once again with $w := r^{(a-2)/4} \leq k^{1/4}$ and terminate the computation by applying Lemma 6. The time for the latter is bounded by $O(w + r/w) = O(k^{1/4+b})$. For every actual $a$ in the loop we choose $b := \frac{a}{2} - \frac{1}{2}$. By this we get $2 - (1 + a/4 + b) = 2 - (\frac{3}{2}a + \frac{7}{8}) = \frac{3}{2}(2 - a)$ and $(a/4 - b)^{-1} = 6/(2 - a)$. The first formula says that in each loop the distance $2 - a$ is reduced by a constant factor. The second conclusion implies, together with Lemma 7, that each loop needs $O((2 - a)^{-1} k^{1/4} \log k)$ time. (Note that we have $2 - a > \delta$ in the loops.)

Now choose $\delta := \log \log r'/\log r'$ where $r'$ is the first $r$ value. Since $r' \leq k \leq r''$, we have $\delta^{-1} = O(\log k/\log k)$. Moreover, $2 - a \leq \delta$ is reached after $O(\log \delta^{-1}) = O(\log \log k)$ loops. Altogether we need $O(k^{1/4} \log^2 k)$ time for the loops.
For the completion we obtained above an $O\left(k^{1/4+\delta}\right)$ bound. Since
\[
    k^\delta \leq k^{2\log \log k / \log k} = 2^{2 \log \log k} = \log^2 k
\]
this is also $O(k^{1/4} \log^2 k)$. □

3: Conclusions

There is still a $\log^2$ gap between our lower and upper bound. Because of simplicity of the proof of Theorem 1 we conjecture that the lower bound can be raised. The upper bound itself is more of theoretical interest as an asymptotic result, than of practical relevance. The algorithm is quite complicated, in contrast one can design much simpler algorithms with worst-case time below $O(k^{1/2})$; this is left as an exercise to the reader. Note that $k^{1/4} \log^2 k < k^{1/2}$ only for very large $k$. Nevertheless, the introduced techniques might be useful for solving problems on sparse digital images efficiently on enhanced meshes. This opens a broad field of research. For instance, what is in our model the complexity of computing convex hulls, shortest paths, nearest neighbors, or of several pattern recognition tasks?

References