# Parallel Algorithms for Image Processing: Practical Algorithms with Experiments*

(Extended Abstract)

Armin Bäumker, Wolfgang Dittrich

## Abstract

*In this paper we design and analyse parallel algorithms with the goal to get exact bounds on their speed-ups on real machines. For this purpose we employ the BSP* model [3] which is an extension of Valiant's BSP model [13] and rewards blockwise communication. Further we use Valiant's notion of $c$-optimality. Intuitively a $c$-optimal parallel algorithm for $p$ processors tends to speed-up $p/c$, where the communication time is asymptotically smaller than the computation time. We consider a basic problem in Image Processing, Connected Component Labeling for two and three dimensional images. Our algorithms are randomized and 2-optimal with high probability for a wide range of BSP* parameters where the range becomes larger with growing input sizes. Our algorithms improve on previous results as they either need an asymptotically smaller amount of data to be communicated or fewer communication rounds. We further report on implementation work and experiments.*

## 1: Introduction

**The problem:** We present parallel algorithms for the Connected Component Labeling Problem for two and three dimensional colour images. A 2-dimensional image is a $N$ x $N$ array of coloured pixels and a 3-dimensional one is a $N$ x $N$ x $N$ array of coloured voxels, colours are from $\{1, \ldots, k\}$, $k$ a constant. Two pixels $v$ and $w$ are adjacent if and only if pixel $v$ lies in north, east, south, or west of $w$, in the case of voxels, adjacency is also given via back and front connections.

For a given colour $c$, the adjacency relation defines a graph $G_c$ on the pixels or voxels of colour $c$ in a natural way.

The task of the Connected Component Labeling problem is to label the pixels (voxels) such that each connected component, of each $G_c$, $c \in \{1, \ldots, k\}$, gets a unique label that identifies the component.

In [11] Connected Component Labeling is cited as an imported object recognition problem in DARPA Image Understanding benchmarks. It can be applied to several problems in physics [1]. The 3D case can be applied to solve problems motivated by medical applications like processing a 3D scan of a computer tomograph.

**The BSP* model:** The algorithms presented in this paper are designed for the BSP* model introduced in [3]. It is an extension of Valiant's BSP (Bulk-synchronous parallel) model [13]. The BSP* model consists of:

- a number of processor/memory components,
- a router that can deliver messages point to point among the processors, and
- a facility to synchronize all processors in barrier style.

A computation on this model proceeds in a succession of *supersteps* separated by synchronizations. For clarity we distinguish between *communication* and *computation supersteps*. In computation supersteps processors perform local computations on data that is available locally at the beginning of the superstep. In communication supersteps all the necessary exchange of data between the processors is done by the router. An instance of the BSP* model is characterized by the parameters $p$, $L$, $B$ and $g$.

- The parameter $p$ is the number of processor/memory components.
- $L$ is the minimum time between successive synchronisation operations. Thus $L$ is the minimum time for a superstep.
- $B$ is the minimum size the messages must have in order to fully exploit the bandwidth of the router.
- $g$ is the ratio of the total throughput of the whole system in terms of basic computational operations to the throughput of the router in terms of messages of size $B$ delivered.

For a computation superstep with at most $t$ local operations on each processor we charge $\max\{L, t\}$ time units. For an $h$-relation, i.e. a routing request where each processor sends and receives at most $h$ messages, we charge $\max\{g \cdot h \cdot \lceil \frac{s}{B} \rceil, L\}$ time units in a communication superstep, if the messages have maximum size $s$.

Note that messages of size smaller than $B$ are treated as if they were of size $B$. Thus in the BSP* model it is worthwhile to send large messages of size at least $B$, i.e. use *blockwise communication*. This feature models the behaviour of many routers of real parallel machines which support the exchange of large messages and achieve much higher throughput for large messages compared to small messages.

**$c$-Optimality:** As optimality criterion for our algorithms we use the notion of $c$-optimality [13]. Let $A^*$ be the best sequential algorithm on the RAM for the problem under consideration, and let

*email: $\{$abk,dittrich$\}$@uni-paderborn.de, Fax: +49-5251-603514. Supported in part by DFG-Sonderforschungsbereich 376 "Massive Parallelität: Algorithmen, Entwurfsmethoden, Anwendungen" and by DFG Leibniz Grant Me872/6-1

$T(A^*)$ be its worst case runtime. Let $c$ be a constant with $c \geq 1$. In order to be $c$-*optimal* (with respect to $p$, $L$, $g$ and $B$) a BSP* algorithm $A$ has to fulfill the following requirements:

- The ratio between the time spent for computation supersteps of $A$ and $T(A^*)/p$ has to be in $c + o(1)$.
- The ratio between the time spent for the communication supersteps of $A$ and $T(A^*)/p$ has to be in $o(1)$.

All asymptotic bounds refer to the problem size as $n \to \infty$. Intuitively, the speed-up of a $c$-optimal algorithm tends to $p/c$.

**Known Results:** Sequentially Connected Component Labeling can be done in time $cN^2$ and $cN^3$ for 2D and 3D images, respectively, with a standard depth-first or breath-first search algorithm, for a small constant $c$.

Cypher et al. [5] present an algorithm for the Connected Component Labeling Problem on $N$ x $N$ 2D images. It needs time $O(\log^2 N)$ on an $N$-processor Hypercube and Shuffle Exchange network.

Bader and JáJá [2] present an architecture independent parallel algorithm for the 2D image case. Their algorithm uses the Block Distributed Memory (BDM) model. This model is quite similar to the BSP* model. Their algorithm requires $O(N^2/p)$ computation time and $O(N)$ communication time. A modified version of their algorithm for the 3D case requires $O(N^3/p)$ computation and $O(N^2)$ communication time. Implementations and experimental results based on SPLIT-C for a variety of machines are also presented in [2].

There are several other algorithms but most of them are architecture specific, and do not port easily to other platforms. Therefore we do not compare them to our algorithms which are designed for a more general model. An example for an architecture specific algorithm is that of Cypher et al. [6].

**New Results:** We present and analyse two parallel algorithms for Connected Component Labeling of $N$ x $N$ and $N$ x $N$ x $N$ colour images on the BSP* model. Let $T_2$ and $T_3$ be the sequential runtimes for the best algorithms for the 2D and 3D case, respectively. Then we have the following result:

**Theorem 1.1.** For every integer $d > 0$, there is a BSP* algorithm for the Component Labeling Problem on $N$ x $N$ images that needs

- computation time $2 \cdot \frac{T_2}{p} + O(d \cdot N/p^{1/2} + L \cdot d \cdot \log p)$
- communication time $O(g \cdot d \cdot N/B \cdot p^{1/2} + L \cdot d \cdot \log p)$

with probability $1 - e^{N/p^{\frac{1}{2}+\frac{1}{d}}}$, if $N \geq p^{\frac{1}{2}+\frac{1}{d}+\epsilon}$, for an arbitrary $\epsilon > 0$, and $B \leq N/p^{\frac{1}{2}+\frac{1}{d}}$.

**Theorem 1.2.** For every integer $d > 0$, there is a BSP* algorithm for the Component Labeling Problem on $N$ x $N$x $N$ images that needs

- computation time $2 \cdot \frac{T_3}{p} + O(d \cdot N^2/p^{2/3} + L \cdot d \cdot \log p)$
- communication time $O(g \cdot d \cdot N^2/B \cdot p^{2/3} + L \cdot d \cdot \log p)$

with probability $1 - e^{N^2/p^{\frac{2}{3}+\frac{1}{d}}}$, if $N^2 \geq p^{\frac{2}{3}+\frac{1}{d}+\epsilon}$, for an arbitrary $\epsilon > 0$, and $B \leq N^2/p^{\frac{2}{3}+\frac{1}{d}}$.

Thus our algorithms are 2-optimal for the above parameter constellations. The algorithms use the same approach as described in Cypher et al. [5]. Unfortunately, if we would adapt the algorithm in [5] canonically to our model we would have to perform $\Theta(\log^2 N)$ communication rounds and could not use blockwise communication. Our algorithm performs only $O(\log p)$ communication rounds and uses blockwise communication as it is rewarded by the BSP* model.

An implementation of the algorithm in [2] on the BSP* model would communicate $O(N)$ data in the 2D case and $O(N^2)$ data in the 3D case. Thus, our results improve upon the results in [2] by a factor of $\Theta(p^{1/2})$ in communication time in the 2D case and by a factor of $\Theta(p^{2/3})$ in the 3D case. In other words: our algorithms tolerate a higher value of the parameter $g$ than the algorithm in [2]. It is therefore more suitable for parallel systems with low communication bandwidth.

## 2: Algorithms for the Connected Component Labeling Problem on two and three dimensional images

### 2.1: Outline of the algorithms

In the following we give a short overview of our BSP* algorithms. We focus on the 2D case and point out the main differences to the 3D case. For a detailed description see [4]. The image is divided into $p$ equal-sized squares, the *sub-images*. The outermost pixels of each sub-image are called *the border* of that sub-image. The sub-images are numbered row-major, see Figure 1. The sub-image with number $i$ is stored on processor $P_i$. Two processors are called *adjacent processors*, if they store adjacent parts of the image.

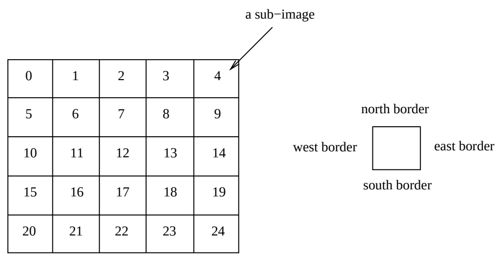

Figure 1: The subdivision and distribution of the image.

The algorithms consist of the following four phases. *Phase 1:* Each processor computes independently the connected component labeling for its sub-image. *Phase 2:* A graph G is constructed, which we call the *global connectivity graph*. A vertex of $G$ represents a connected component of a sub-image, that touches the border of its sub-image. Since at most $2Np^{1/2}$ pixels belong to the border, $G$ has at most that much vertices. Two such vertices are connected, if they contain adjacent pixels with the same colour. So $G$ has $O(Np^{1/2})$ edges. In the 3D case $G$ has $O(N^2 \cdot p^{1/3})$ vertices and edges. Note that the size of $G$ is much smaller than the original input size. *Phase 3:* We compute the connected components of $G$ and label the vertices of $G$ with the numbers of their components. *Phase 4:* Finally the labels of the vertices of $G$ are used as new labels for the components of the subimages represented by these vertices. The labels of the connected component of sub-images, that do not touch the border, remain unchanged.

Clearly this algorithm results in a correct labeling. Phase 1 can be done very efficiently by a breadth-first search algorithm. Phase 3 is described in Subsection 2.3, Phase 2 in Subsection 2.4 and Phase

4 in Subsection 2.5.

The main problem in the algorithm is to compute the connected components of the global connectivity graph $G$. In the 2D case $G$ is a planar graph, and in the 3D case it is a general graph. There are PRAM-algorithms that solve these problems. Hagerup [8] gave a randomized CREW PRAM-algorithm for the planar case. If $n$ is the number of vertices it needs time $O(\log n)$ with $n$ processors. Gazit [7] gave a randomized CRCW PRAM-algorithm for the general case. If $n$ is the number of vertices and $m$ is the number of edges it needs time $O(\log n)$ with $\frac{m+n}{\log n}$ processors.

With a PRAM simulation like that in [13] we could run the above algorithms on the BSP* model. Since the size of $G$ is sufficiently smaller then the original input size the simulation overhead would not affect the asymptotic efficiency of our algorithms. But a PRAM simulation would require $\Omega(\log n)$ communication supersteps and we could not use blockwise communication. Therefore, we present two new CRCW-PRAM algorithms, one for the planar case and the other for the general case: The first algorithm uses ideas of Hagerup [8] and the second is a modification of the algorithm in [7]. The new PRAM algorithms can easily be adapted to the BSP* model such that they perform only $O(\log p)$ communication supersteps and use blockwise communication. In order to do this we need the algorithm *Multi-Phase-Combine* which we describe in Subsection 2.2. With *Multi-Phase-Combine* one can simulate the concurrent accesses of a CRCW-PRAM on the shared memory.

## 2.2: Multi-Phase-Combine

As subroutines we use CRCW-PRAM algorithms adapted to the BSP* model. Therefore we need a method to simulate concurrent accesses to global variables even when all processors want to access the same global variable at the same time. We assume that the global variables are distributed randomly over the local memories of the processors. The method also works for universal hashing instead of a random distribution. A $k$-access is a set of accesses to global variables in which each processor executes at most $k$ *global-read* or *global-write* instructions. Valiant [12] proposed a scheme for $k$-accesses on the BSP model. The algorithm Multi-Phase-Combine($d$) adopts this scheme and choose the parameters such that blockwise communication can be used. The parameter $d$ controls the number of rounds that the scheme in [12] performs.

**Result 1.** Multi-Phase-Combine($d$) executes a $k$-access within $O(d)$ supersteps. It needs computation time $O(d(k+L))$ and communication time $O(gd(k/B+L))$ with probability $1 - e^{-\Theta(k/p^{1/d})}$, if $B \leq k/p^{1/d}$ and $k \geq p^{(1/d)+\epsilon}$ for arbitrary $\epsilon > 0$.

## 2.3: Algorithms for computing Connected Component of Graphs

In this subsection we present the new CRCW-PRAM algorithms for computing the connected components of planar and general graphs. All presented algorithms are randomized. The input, a graph $G = (V, E)$, with $|V| = n$ vertices and $|E| = m$ edges, is stored in the global memory as adjacency list. The PRAM algorithms in [8] and [7] perform $O(\log n)$ *reduction rounds*. In each of these reduction rounds the input graph is reduced by a constant factor. Our PRAM algorithms adopt the ideas in [8] and [7] but they perform only $O(\log p)$ reduction rounds. This property of the new algorithms makes it possible to adapt them to the BSP* model such that they perform only $O(\log p)$ communication supersteps.

**PRAM algorithm for planar graphs:** We give a brief high level description of the algorithm for planar graphs. For a detailed description and proofs see [4], for similar PRAM algorithms for graph problems see JáJá [9]. Note that the algorithm assumes a random distribution of the vertices and edges. This can be constructed out of an adjacency list within the time bounds of the described algorithms, therefore this is no problem.

Our algorithm performs a couple of reduction rounds. At the beginning of such a round each vertex $v$ is represented by a supervertex, which inherits all edges from $v$. A supervertex represents some or all vertices of a component. During a reduction round adjacent supervertices are melted, which one is determined by a randomized selection procedure (see [4] for a detailed description of it). To melt a set of adjacent supervertices $S$ we substitute $S$ by one supervertex which inherits all edges from $S$. The number of supervertices is reduced in each round. One reduction round is done by procedure *Shrink* (see [4] for a detailed description of *Shrink*).

We call the vertices (supervertices) and edges still in the graph *live* vertices (supervertices) and edges, respectively.

A global merge phase follows the reduction phase. For that we need the following definitions. Each connected component $C = \{v_{j_1}, \ldots v_{j_k}\} \subseteq V$ of a graph $G = (V, E)$ can be represented by a graph $G(C) = (C, E)$, where $E = \{\{v_{j_i}, v_{j_{i+1}}\} | \, i \in \{1, \ldots k-1\}\}$. Let $K(G) = \{G(C) | \text{ C is a connected component of G}\}$. Let $G_i = (V, E_i), i \in \{1, 2\}$. Merging $K(G_1)$ and $K(G_2)$ means to compute $K(G = (V, E_1 \cup E_2))$, this can be done with a sequential connected component algorithm in time $O(|V|)$. Next we give the algorithm:

1. Each processor is responsible for $O(n/p)$ vertices and edges, this assignment remains the same for this and the next phase. The processors execute $O(\log\log p)$ times the procedure *Shrink* until the graph has $O(n/\log p)$ supervertices and edges. During each iteration duplicate edges are deleted, i.e. are marked as not live.
2. The processors perform O($\log p$) times the procedure *Shrink* until the graph has $O(n/p \log p)$ vertices but without reducing duplicate edges.
3. The processors number all live supervertices and store a copy of each one in the global memory at consecutive addresses. Denote these vertex set $V_L$. Afterwards the processors subdivide the live edges into $p$ disjoint sets $E_i$, so that $|E_i| = O(n/p \log p)$. For each edge $e = \{v, w\} \in E_i$ the processor $P_i$ changes the numbering of $v$ and $w$ according to the numbering of $V_L$. Then processor $P_i$ computes the connected components of $G_i = (V_L, E_i)$ with a sequential connected component algorithm. This yields $p$ different graphs $K(G_i)$ all with vertex set $V_L$.
4. The processors compute graph $K(G_R := \cup G_i)$ by merging the $p$ graphs $K(G_i)$ in a tree-like computation. The computation proceeds in $O(\log p)$ rounds. In the $i$-th round $p/2^i$ processors are active, each active processor merges two graphs. Finally $K(G_R)$ is stored in $P_0$. $P_0$ labels the components of graph $K(G_R)$.
5. The processors compute the labels of all removed supervertices in $O(\log p + \log\log p)$ iterations. During iteration $i$

each processor relabels the vertices removed during iteration $r - i$, where $r$ is the total number of iterations of Step 1 and 2. Thus we perform the reduction rounds of Step 1 and 2 in reverse order.

**Result 2.** The PRAM algorithm for planar graphs needs time $O(n/p)$ and $O(\log p)$ reduction rounds with probability $1 - \frac{1}{n^\epsilon}$ and $p \leq n/\log^{1+\epsilon} n$ processors.

**PRAM algorithm for general graphs:** The algorithm of Gazit [7] performs $O(\log n)$ reduction rounds on the input graph. We modify this algorithm such that it needs only $\log p$ reduction rounds. For a description of this modification and a proof of the following result see [4].

**Result 3.** The PRAM algorithm for general graphs needs time $O(n/p + \log p)$ and $O(\log p)$ reduction rounds with probability $1 - \frac{1}{n^\epsilon}$, $\epsilon > 1$ and $p \leq (n+m)/(\log n)^{1+\epsilon}$ processors.

**BSP* algorithms for planar and general graphs:** With the help of the algorithm *Multi-Phase-Combine(d)* the CRCW-PRAM algorithms can easily be adapted to the BSP* model. Since the PRAM algorithms that we use need only $O(\log p)$ reduction rounds we get BSP* algorithms that need only $O(\log p)$ communication supersteps.

**Theorem 2.1.** For every integer $d > 0$ there is a BSP* version of the PRAM algorithm for planar graphs that runs in computation time $O(dn/p + Ld \log p)$ and communication time $O(gdn/Bp + Ld \log p)$ with probability $1 - e^{-\Theta(n/p^{1+(1/d)} \log p)}$ for $B \leq \frac{n}{p^{1+1/d}}$ and $n/p \geq p^{(1/d)+\epsilon}$, $\epsilon > 0$.

**Theorem 2.2.** For every integer $d > 0$ there is a BSP* version of the PRAM algorithm for general graphs that runs in computation time $O(dn/p + Ld \log p)$ and communication time $O(gdn/Bp + Ld \log p)$, with probability $1 - e^{-\Theta(n/p^{1+(1/d)} \log p)}$ for $B \leq \frac{n}{p^{1+1/d}}$ and $n/p \geq p^{(1/d)+\epsilon}$, $\epsilon > 0$.

## 2.4: Construction of *G*

The input consists of an $N$ x $N$ or an $N$ x $N$ x $N$ image, which is subdivided into $p$ sub-images and distributed as described in Subsection 2.1. Moreover each processor has already computed the components for its sub-image. The output consists of the edges and vertices of the global connectivity graph $G$, distributed randomly among the destination processors, so that each one holds $O(n/p)$ vertices and edges. The random distribution can be done with the help of *Multi-Phase-Combine(d)*. For details see [4].

**Theorem 2.3.** For every integer $d > 0$ the global connectivity graph $G$ of a $N$ x $N$ colour image can be constructed on the BSP* model within computation time $O(dN/p^{1/2} + dL)$ and communication time $O(gdN/Bp^{1/2} + dL)$ with probability $1 - e^{-\Theta(N/p^{0.5+1/d})}$, for $B \leq N/p^{0.5+1/d}$ and $N \geq p^{0.5+1/d+\epsilon}$, $\epsilon > 0$.

**Theorem 2.4.** For every integer $d > 0$ the global connectivity graph $G$ of a $N$ x $N$ x $N$ colour image can be constructed on the BSP* model within computation time $O(dN^2/p^{2/3} + dL)$ and communication time $O(gdN^2/Bp^{2/3} + dL)$ with probability $1 - e^{-\Theta(N^2/p^{(2/3)+1/d})}$, for $B \leq N^2/p^{(2/3)+1/d}$ and $N^2 \geq p^{2/3+1/d+\epsilon}$, $\epsilon > 0$.

| $N$ | 512 | 512 | 1024 | 1024 |
|---|---|---|---|---|
| $Eff_{Pict1}(N)$ | 45.0 | 54.7 | 52.0 | 61.0 |
| $Eff_{Pict2}(N)$ | 48.0 | 53.5 | 53.0 | 57.2 |
| $Eff_{Pict3}(N)$ | 49.0 | 53.0 | 53.0 | 57.0 |

Figure 2: #Procs $\hat{=}$ 4. The first column of each pair shows the efficiency of our algorithm, the second one the efficiency of Bader's algorithm.

## 2.5: How to relabel the pixels

The input consists of the vertices of $G$ randomly distributed among the destination processors. Further we know for each vertex its new globally unique label. The output is a unique labeling of all pixels of the image according to the labeling of $G$. This can easily be done with the help of of *Multi-Phase-Combine(d)*. For details see [4]. Let $T_2$ and $T_3$ be the runtimes for the best sequential component labeling algorithms for the 2D and 3D case, respectively. Then we get the following:

**Theorem 2.5.** For every integer $d > 0$ there is a relabeling algorithm for the 2D case on the BSP* model that needs computation time $T_2/p + O(dN/p^{1/2} + dL)$ and communication time $O(gdN/Bp^{1/2} + dL)$ with probability $1 - e^{-\Theta(N/p^{0.5+1/d})}$, for $B \leq N/p^{0.5+1/d}$, $N \geq p^{0.5+1/d+\epsilon}$, $\epsilon > 0$.

**Theorem 2.6.** For every integer $d > 0$ there is a relabeling algorithm for the 3D case on the BSP* model that needs computation time $T_3/p + O(dN^2/p^{2/3} + dL)$ and communication time $O(gdN^2/Bp^{2/3} + dL)$ with probability $1 - e^{-\Theta(N^2/p^{2/3+1/d})}$, for $B \leq N^2/p^{2/3+1/d}$, $N^2 \geq p^{2/3+1/d+\epsilon}$, $\epsilon > 0$.

Assembling the single results we get Theorems 1.1 and 1.2.

## 3: Environment, Implementation and Experimental Results

We have implemented our algorithm for the 2D case and the algorithm of Bader et al. [2] on the Supercluster SC 320 from Parsytec. The SC 320 consists of 320 INMOS T800 Transputers, each running at 25Mhz. The processors are connected via a Clos network of crossbar switches. We have implemented a BSP library which supports the BSP style of programming, and contains some basic routines like broadcast and parallel prefix. All described algorithms use this BSP library.

In order to compare our algorithm with that of Bader et al. [2] we hve done the following experiments. As test images we use concentric circles (Picture 1), diagonal bars (Picture 2) and a helix (Picture 3). We varied the size of the pictures form 512 x 512 pixels over 1024 x 1024 pixels to 2048 x 2048 pixels. We made the experiments on 4 and 16 processors. See Figure 2 and 3 for the results. The entries in the tables are efficiency values, i.e, the ratio of the sequential time divided by $p$ to the runtime of the parallel algorithm.

As we expected, our algorithm is only better for large images consisting of many components, whereas the algorithm of Bader et al. achieves better results for smaller input sizes. This is due to the fact that our algorithm has smaller communication complexity whereas the algorithm of Bader et al. profits from smaller constant runtime factors, if the image is not too big.

| $N$ | 512 | 512 | 1024 | 1024 | 2048 | 2048 |
|---|---|---|---|---|---|---|
| $\mathit{Eff}_{Pict1}(N)$ | 22.0 | 27.8 | 37.0 | 43.6 | 48.0 | 50.7 |
| $\mathit{Eff}_{Pict2}(N)$ | 26.0 | 29.6 | 42.0 | 46.4 | 52.0 | 54.1 |
| $\mathit{Eff}_{Pict3}(N)$ | 26.0 | 26.3 | 42.0 | 44.6 | 51.0 | 36.4 |

Figure 3: #Procs $\hat{=}$ 16. The first column of each pair shows the efficiency of our algorithm, the second one the efficiency of Bader's algorithm.

| $k$ | 60 | 70 | 80 | 90 | 100 |
|---|---|---|---|---|---|
| $\mathit{Eff}_B(k)$ | 50.7 | 49.0 | 47.0 | 45.0 | 43.0 |
| $\mathit{Eff}(k)$ | 48.0 | 48.0 | 47.0 | 47.0 | 46.0 |

Figure 4: #Procs = 16. The first row shows the efficiency of Bader's algorithm for a 2048X2048 picture of concentric circles, if $k$ is the number of black circules. The second row shows the efficiencies of our algorithm for the same picture.

Figure 4 shows the result of the following experiment. As test image, we used a 2048 x 2048 pixel image with concentric circles. We varied the number of circles from 60 over 70, 80 and 90 to 100. The more circles we have the more complex the image becomes and the more both algorithms have to communicate. Our algorithm achieves better results compared to the Bader algorithm with growing circle numbers becouse the problem becomes more and more communication intensive. See [4] for further experiments also on other machines.

## References

1. J. Apostolakis, P. Coddington and E. Marinari. New SIMD algorithms for cluster labeling on parallel computers. Int. J. Mod. Phys. C., 1993.
2. D.A. Bader and J. JáJá. Parallel Algorithms for Image Histogramming and Connected Components with an Experimental Study. 5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 1995.
3. A. Bäumker, W. Dittrich, and F. Meyer auf der Heide. Truly Efficient Parallel Algorithms: c-Optimal Multisearch for an Extension of the BSP model. Proc. of European Symposium on Algorithms, 1995.
4. A. Bäumker, W. Dittrich. Parallel algorithms for image processing: practical algorithms with experiments. To appear as: Technical Report, University of Paderborn, 1996.
5. R. Cypher, J.L.C. Sanz and L. Snyder. Hypercube and Shuffle-Exchange Algorithms for Image Component Labeling. Journal of Algorithms 10, 1989, pp. 140–150.
6. R. Cypher, J.L.C. Sanz and L. Snyder. Algorithms for Image Component Labeling on SIMD Mesh-Connected Computers. IEEE Transactions on Computers 39, 1990.
7. H. Gazit. An Optimal Randomized Parallel Algorithm for Finding Connected Components in a Graph. Proc. of 27th FOCS, 1986, pp. 492–501.
8. T. Hagerup. Optimal Parallel Algorithms on Planar Graphs. Information and Computation, Volume 84, 1990.
9. J.F. JáJá. Introduction to Parallel Algorithms. Addison-Wesley, 1992.
10. J.F. JáJá and K.W. Ryu. The Block Distributed Memory Model for Shared Memory Multiprocessors. In Proceedings of the 8th Int. Parallel Processing Symposium, Cancún, Mexico, April 1994, pp. 752-756.
11. A. Rosenfeld. A report on the DARPA image understanding architecture workshop. Proc. of the Image Understanding Workshop, 1987, pp. 298–302.
12. L.G. Valiant. A Combining Mechanism for Parallel Computers. Proc. of 1st Heinz Nixdorf Symposium, LNCS 678, 1992, pp. 1–10.
13. A.V. Gerbessiotis and L. Valiant. Direct Bulk-Synchronous Parallel Algorithms. Journal of Parallel and Distributed Computing, 1994.