# Coarse Grained Parallel Maximum Matching In Convex Bipartite Graphs

P. Bose, A. Chan, F. Dehne, and M. Latzel
School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6
{jit,achan,dehne,mlatzel}@scs.carleton.ca

## Abstract

*We present a coarse grained parallel algorithm for computing a maximum matching in a convex bipartite graph $G = (A, B, E)$. For $p$ processors with $N/p$ memory per processor, $N = |A| + |B|$, $N/p \geq p$, the algorithm requires $O(\log p)$ communication rounds and $O(T_{sequ}(\frac{n}{p}, \frac{m}{p}) + \frac{n}{p} \log p)$ local computation, where $n = |A|$, $m = |B|$ and $T_{sequ}(n, m)$ is the sequential time complexity for the problem. For the BSP model, this implies $O(\log p)$ supersteps with $O(gN + g\frac{n}{p} \log p)$ communication cost and $O(T_{sequ}(\frac{n}{p}, \frac{m}{p}) + \frac{n}{p} \log p)$ local computation.*

## 1 Introduction

### 1.1 The Problem

We study the problem of computing the maximum matching in convex bipartite graphs defined as follows.

**Definition 1** *A Bipartite graph $G = (A, B, E)$ is an undirected graph with vertex set $A \cup B$ where $A$ and $B$ are disjoint and $E \subseteq (A \times B)$.*

For the remainder let $n = |A|$, $m = |B|$ and $N = n + m$.

**Definition 2** *A Convex bipartite graph is a bipartite graph $G = (A, B, E)$ with an ordering $B = (b_1, b_2, \cdots b_n)$ such that for any $a \in A$, if $(a, b_i) \in E$ and $(a, b_j) \in E$ $(i \leq j)$ then $(a, b_k) \in E$ for all $i \leq k \leq j$.*

**Definition 3** *A matching $M$ in a graph $G = (V, E)$ is a subset of $E$ such that no two edges in $M$ are incident to the same vertex.*

**Definition 4** *A maximum matching in a graph $G = (V, E)$ is a matching of $G$ of which the size (number of edges) is maximum.*

The problem of finding a maximum matching in a bipartite graph or a convex bipartite graph is a classical and well studied problem [7, 12, 14, 15]. The case of convex bipartite graphs has several interesting applications as outlined in [7, 15]. A particularly interesting industrial application for matching parts with products was described in [15].

Sequential solutions for maximum matching in bipartite graphs and convex bipartite graphs with time complexities $O(n^{5/2})$ and $O(n + \alpha(m))$ were described in [14] and [15], respectively, where $\alpha(m)$ is a very slowly growing function related to the inverse Ackermann function. A PRAM algorithm for maximum matching in convex bipartite graphs requireing $O(\log^2 n)$ time and $n/2$ processors was presented in [7].

In this paper, we consider the maximum matching problem for the coarse grained parallel and BSP models of computation described in the next subsection. These new parallel models have recently received much attention because they allow the design of parallel algorithms that have much improved practical performance in actual implementations on commercial multiprocessors compared to previous models like the PRAM or fine grained network models (e.g. mesh or hypercube).

Before proceeding to the description of the coarse grained parallel and BSP models of computation, we note that finding a maximum matching in a convex bipartite graph can be transformed into the following problem:

**Definition 5** *Given a set of intervals $I = \{I_1, \cdots, I_n\}$, each of which represents an integer range, i.e. $I_i = (l_i, r_i)$. A maximum interval assignment consists of assigning, for a maximum number of intervals, one integer label to each interval such that the integer for each interval is within the interval's range and no two intervals are assigned the same integer.*

For a given convex bipartite graph $G = (A, B, E)$ let $I(G)$ be the set of $|A|$ intervals containing for each $a \in A$ an interval $I_i = (l_i, r_i)$ where $l_i$ and $r_i$ are the smallest and largest ranks, in $B$, of all element $b \in B$ with $(a, b) \in E$.

**Observation 1** *The maximum matching problem for a convex bipartite graph $G$ can be reduced to finding a maximum interval assignment in $I(G)$.*

### 1.2 The Model

**The Coarse Grained Multicomputer (CGM)**

A $CGM$ (Coarse Grained Multicomputer) is a special case of a BSP (Bulk Synchronous Parallel) multiprocessor [18] where all communication within a superstep is reduced to one $h$-relation. A CGM consists of a collection of $p$ processors with $N/p$ local memory each, connected by a router that can deliver messages in a point to point fashion. A CGM algorithm consists of an alternating sequence of *computation rounds* and *communication rounds* separated by barrier synchronization. A computation round is equivalent to a computation superstep in the BSP model, and the total computation cost $T_{comp}$ is defined analogously. A communication round consists of a single $h$-relation with $h \leq N/p$. The total communication cost,$T_{comm}$, is measured by the number of communication rounds. Consult [3, 4, 5] for more details.

In a recent overview of different BSP and related models, Goodrich [13] referred to the CGM as the *weak-CREW BSP*. The CGM model aims at designing simple and practical, yet theoretically optimal or efficient, parallel algorithms for coarse grained parallel systems ($N/p >> 1$). Algorithms do usually require a lower bound on $N/p$, e.g. $N/p \geq p$ or $N/p \geq p^{\epsilon}$. The CGM model targets in particular the case where the overall computation speed is considerably larger than the overall communication speed, which is usually the case. Since the message size is maximal, the model also minimizes the message overhead associated with sending a message, which is very important in practice.

**Relationship Between BSP And CGM**

In the remainder of this paper, we will present our algorithms in the CGM model. The relationship to the BSP model is given by the following

**Observation 2** *A CGM algorithm with $\lambda$ rounds and computation cost $T_{comp}$ corresponds to a BSP algorithm with $\lambda$ supersteps, communication cost $O(g\lambda\frac{N}{p})$ and the same computation cost $T_{comp}$.*

### 1.3 The Result

In this paper, we present a coarse grained parallel algorithm for computing a maximum matching of a convex bipartite graph $G = (A, B, E)$. For a $p$ processor CGM with $N/p$ memory per processor, $N/p \geq p$, the algorithm requires $O(\log p)$ communication rounds and $O(T_{sequ}(\frac{n}{p}, \frac{m}{p}) + \frac{n}{p}\log p)$ local computation where $T_{sequ}(n, m)$ is the sequential time complexity for the problem. For the BSP model, this implies $O(\log p)$ supersteps with $O(gN + g\frac{n}{p}\log p)$ communication cost and $O(T_{sequ}(\frac{n}{p}, \frac{m}{p}) + \frac{n}{p}\log p)$ local computation.

## 2 Algorithm Description

In the following Section 2.1, we first present an algorithm that solves a special case: all intervals start at the same left end point. This algorithm will be used in our solution for the general case which is presented in Section 2.2.

### 2.1 Special Case: All Intervals Start At The Same Point

**Algorithm 1** All intervals have the same left end point $l_i = l, i = 1 \cdots n$.
**Input:** A set of $n$ intervals $I = \{I_1, \ldots, I_n\}$ with their associated left and right end points $I_i = (l_i, r_i), i = 1 \ldots n$, distributed over a $p$ processor CGM with $n/p$ intervals per processor. Note that, we assume $\frac{n}{p} \geq p$.

(1) All intervals are sorted by their right end points using CGM parallel integer sort [6, 13].
(2) Each processor $P_i, i = 1 \ldots p-1$, determines the largest right end point, $e_i$, received in Step 1 and sends it to the next processor $P_{i+1}$.
(3) Each processor $P_i, i = 2 \ldots p$, sets $s_i = e_{i-1} + 1$. The first processor sets $s_1 = l$. We call $(s_i, e_i)$ the controlled range of $P_i$. This is the range of integers that the processor can use to label intervals.
(4) Each processor $P_i, i = 1 \ldots p$, temporarily changes the left end points of its intervals to $s_i$ and then solves the modified problem locally using a sequential algorithm (essentially sequential integer sort).
(5) Each processor $P_i, i = 1 \ldots p$, calculates how many labels in the controlled range are left unused ($a_i$) and how many intervals did not yet receive an integer label ($b_i$). These two numbers per processor are broadcast to all processors (in one h-relation). This is possible since $\frac{n}{p} \geq p$.
(6) Based on the data received in the previous step, each processor can now calculate from where it should request labels and to where it should send unused labels. See Algorithm 2 for more details.
(7) Each processor selects unused labels and sends them to the processors that need them.
(8) Upon receiving the needed labels, each processor can now assign them to the respective intervals.

— End of Algorithm —

**Algorithm 2** Sequential computation for Step 6 in Algorithm 1, executed by processor $P_k$, $1 \leq k \leq p$.

**Input:** An array $a = (a_1, \ldots, a_p)$ representing the numbers of free (unused) labels in the $p$ processors. An array $b = (b_1, \ldots, b_p)$ representing the numbers of intervals so far without labels in the $p$ processors.
**Output:** An array $get = (get_1, \ldots, get_p)$ representing numbers of labels that are to be sent to processor $P_k$ by the other processors. An array $give = (give_i, \ldots, give_p)$ representing numbers of labels that are expected from processor $P_k$ by the other processors.

```
for (i=1; i≤p; i++) { get_i = 0; give_i = 0; }
i = 1;
j = 2;
while ((i<p) and (j≤p))
{
    if (i==j) j++;
    while ((a_i≠0) and (j≤p))
    {
        if (a_i≥b_j)
        {
            if (P_k=i) get_j = b_j;
            if (P_k=j) receive_i = b_j;
            a_i = a_i - b_j;
            b_j = 0;
            j++;
        }
        else
        {
            if (P_k=i) get_j = a_i;
            if (P_k=j) give_i = a_i;
            b_j = b_j - a_i;
            a_i = 0;
        }
    }
    i++;
}
```

— End of Algorithm —

**Lemma 1** *The sequential time complexity of Algorithm 2 is $O(p)$.*

**Proof.** The variable $i$ counts from 1 to $p$-1, and $j$ counts from 2 to $p$. Since both variables are incremented independently, the total time complexity of the algorithm is $O(p)$. □

**Lemma 2** *The integer label assignment produced by Algorithm 1 is maximum.*

**Proof.** It is easy to see that the label assignment based on the order of the right end points is maximum. Let $A_o$ be such an assignment. Let $A_1$ be the label assignment produced by Algorithm 1. Since $A_o$ is maximum, $|A_o| \geq |A_1|$. We compare assignments $A_1$ and $A_o$. We first remove the intervals that are not assigned labels in both assignments. If the remaining intervals all have labels assigned in both assignments, then we have $|A_o| = |A_1|$ and the lemma follows.

If the $k$th interval $(l'_k, r'_k)$ (with respect to the right end points) is assigned a label in $A_o$ but not in $A_1$, then the first $k-1$ intervals must be assigned labels $l, l+1, \ldots, l+k-2$ in $A_o$ and the $k$th interval must be assigned label $l+k-1$. This also means $r'_k \geq l+k-1$.

Now consider $A_1$. If the first $k-1$ intervals are also assigned labels $l, l+1, \ldots, l+k-2$, then $l+k-1$ will be available for the $k$th intervals. If some of the first $k-1$ intervals are not assigned labels in the range $l, l+1, \ldots, l+k-2$, then there will be a "hole" in this range that can be used for the $k$th interval.

Hence, for every interval with a label assigned in $A_o$, there is a label assigned to that interval in $A_1$. Therefore, $|A_1| \geq |A_o|$. Thus, the label assignment obtained from Algorithm 1 is maximum. □

**Theorem 1** *Algorithm 1 finds a maximum label assignment for $n$ intervals with the same left endpoints stored on a $p$ processor CGM with $n/p$ local memory per processor, $\frac{n}{p} \geq p$, in $O(1)$ communication rounds with $O(\frac{n}{p})$ local computation.*

**Proof.** The correctness of the algorithm is shown in Lemma 2. The communication rounds required in each step of Algorithm 1 is $O(1)$. Hence the total number of communication rounds needed is $O(1)$. In each round, the total message size per processor is $O(\frac{n}{p})$. Step 1 uses CGM integer sort [6, 13] which requires $O(1)$ communication rounds and $O(\frac{n}{p})$ local computation. The local computation of Step 6 is $O(p)$ (from Lemma 1). For all other steps, it is either $O(1)$ or $O(\frac{n}{p})$. Therefore, the total local computation per processor is $O(\frac{n}{p}) + O(p) = O(\frac{n}{p})$ since $\frac{n}{p} \geq p$. □

## 2.2 The General Case

In order to solve the maximum interval assignment problem for arbitrary intervals, we now combine Algorithm 1 with a merge/unmerge scheme presented in [7].

**Algorithm 3** General case where the intervals can have different left end points.
**Input:** A set of $n$ intervals $I = \{I_1, \ldots, I_n\}$ with their associated left and right end points $I_i = (l_i, r_i), i = 1 \cdots n$, distributed over a $p$ processor CGM with $n/p$ intervals per processor. Note that, we assume that $\frac{n}{p} \geq p$.

(1) All intervals are sorted by their left end points using CGM parallel integer sort [6, 13]. (In case of a tie, intervals are compared by their right end points.)

(2) Intervals with the same left end points are combined into groups. All groups that are stored completely within a processor are merged into a single group. Let $\gamma$ be the number of groups. Note that $\gamma$ is at most $2p+1$.

(3) Each group is assigned a *controlled range* $(l_i, r_i), i = 1 \ldots \gamma$ where $l_i$ is equal to the smallest left end point of that group and $r_i = l_{i+1} - 1, i = 1 \ldots \gamma - 1$. Let $r_\gamma$ be the largest right end point of the intervals.

(4) Using a sequential algorithm, each processor solves the problem for interval groups that are completely within the processor. Remove the label of all those intervals that received a label outside their group's controlled range. Classify the intervals using one of the following three types. Matchable ($M$): all labeled intervals. To-be-determined ($T$): all non labeled intervals that extend beyond the rightmost label given by that processor. Unmatchable ($U$): all remaining intervals. Remove all unmatchable intervals.

(5) Using Algorithm 1, solve the problem for interval groups that cross a processor boundary (for each such group in isolation). Remove the label of all intervals that received a label outside their group's controlled range. Classify the intervals using one of the following three types. Matchable ($M$): all labeled intervals. To-be-determined ($T$): all nonlabeled intervals that extend beyond the rightmost label given by that processor. Unmatchable ($U$): all remaining intervals. Remove all unmatchable intervals.

(6) Merge the intervals in adjacent groups. Let $M_L$, $T_L$ be the intervals from the left group, let $M_R$, $T_R$ be the intervals from the right group, and let $(l_L, r_L)$ and $(l_R, r_R)$ be the controlled ranges of the left and right groups, respectively. Using Algorithm 1, solve the problem for $T_L \cup M_R$ over the range $(l_R, r_R)$. Let the resulting sets of machable and to-be-determined intervals be $M_n$, and $T_n$. For the combined group, set $M = M_L \cup M_n$ and $T = T_n \cup T_R$. The controlled range of the combined group is $(l_L, r_R)$.

(7) Repeat Step 6 $\mathrm{O}(\log \gamma)$ times until a single group is obtained.

(8) Using a reverse process of the previous steps, using $\mathrm{O}(\log \gamma)$ iterations, redistribute the intervals back into the $\gamma$ groups obtained at the end of Step 3. In each splitting phase, let $M$ be the group we are splitting and let $M_L$ and $M_R$ be those two components. Let $V$ be the set of intervals that have a left end point smaller than the starting value of the controlled range of $M$. Note that, $V$ can be empty. Let $W$ be the union of $V$ and $M_L$. $W$ should be distributed to $M_L$ and the rest to $M_R$. However, the controlled range of $M_L$ may not allow the entire $W$ to be assigned to $M_L$. Hence, we assign only the first $\min(|W|, lR-L|)$ intervals to $M_L$ and the remainder to $M_R$.

(9) Step 8 is repeated until $\gamma$ groups are obtained.

(10) Using a sequential algorithm, each processor solves the problem for its interval groups over each group's controlled range. For groups that span over more than one processor, adjust the left end points to the starting values of the controlled range and then apply Algorithm 1.

— End of Algorithm —

**Theorem 2** *Algorithm 3 solves the label assignment problem in $O(\log p)$ communication rounds and $O(T_{sequ}(\frac{n}{p}, \frac{m}{p}) + \frac{n}{p} \log p)$ local computation.*

**Proof.** The correctness of the split/merge scheme follows from [7]. Step 6 is executed $\mathrm{O}(\log \gamma)$ times. In Step 6, we invoke Algorithm 1 which requires $\mathrm{O}(1)$ communication rounds. Since $\gamma \leq 2p+1$, the total number of communication rounds is $\mathrm{O}(\log p)$. The local computation time is dominated by the $O(1)$ executions of the sequential algorithm on each processor and the linear local time in each of the $\mathrm{O}(\log \gamma)$ iterations. □

As described in Section 1.1, this theorem implies an algorithm for computing the maximum matching of a convex bipartite graph with the same number of communication rounds and local computation.

## 3 Conclusion

In this paper, we have presented a coarse grained parallel algorithm for computing a maximum matching of a convex bipartite graph $G = (A, B, E)$. For $p$ processors with $N/p$ memory per processor, $N/p \geq p$, the algorithm requires $\mathrm{O}(\log p)$ communication rounds and $\mathrm{O}(T_{sequ}(\frac{n}{p}, \frac{m}{p}) + \frac{n}{p} \log p)$ local computation. For the BSP model, this implies $\mathrm{O}(\log p)$ supersteps with $\mathrm{O}(gN + g\frac{n}{p} \log p)$ communication cost and $\mathrm{O}(T_{sequ}(\frac{n}{p}, \frac{m}{p}) + \frac{n}{p} \log p)$ local computation.

From a practical point of view, i.e. for implementing this method on commercial multiprocessors, it is very important that the problem size is not a parameter for the number of communication rounds. The result obtained, i.e. $\mathrm{O}(\log p)$ communication rounds, is a function of $p$ only. Since $p$ is usually fixed or grows on very slowly in practice, and $\log p$ is a slowly growing function, the number of communication rounds is essentially a fixed constant for most practical arrangements. This is important because empirical studies show that the number of communication rounds is the most important parameter influencing the observed running time.

From a theoretical point of view it is an important open problem to study whether it is possible to find an algorithm with even fewer communication rounds (is $\mathrm{O}(1)$ possible?) and/or determine lower bounds for the number of communication rounds.

## References


[1] R.J. Anderson and L. Snyder, "A Comparison of Shared and Nonshared Memory Models of Computation." *Proc. of IEEE*, vol 79(4),pp.480-487.

[2] G.E. Blelloch, C.E. Leiserson, B.M. Maggs, and C.G. Plaxton, "A Comparison of Sorting Algorithms for the Connection Machine CM-2.," *Proc. ACM Symp. on Parallel Algorithms and Architectures*, 1991, pp. 3-16.

[3] E. Caceres, F. Dehne, A. Ferreira, P. Flocchini, I. Rieping, A. Roncato, N. Santoro, and S. W. Song, "Efficient parallel graph algorithms for coarse grained multicomputers and BSP," in Proc. 24th International Colloquium on Automata, Languages and Programming (ICALP'97), 1997, Springer Verlag Lecture Notes in Computer Science, Vol. 1256, pp. 390–400.

[4] F. Dehne, A. Fabri, and A. Rau-Chaplin, "Scalable Parallel Geometric Algorithms for Coarse Grained Multicomputers," *Proc. ACM 9th Annual Computational Geometry*, pages 298–307, 1993

[5] F. Dehne, X. Deng, P. Dymond, A. Fabri, and A. A. Kokhar, "A randomized parallel 3D convex hull algorithm for coarse grained multicomputers," in Proc. *ACM Symposium on Parallel Algorithms and Architectures* (SPAA'95), pp. 27–33, 1995.

[6] F. Dehne, A.Chan, "A note on coarse grained parallel integer sorting," Technical Report, School of Computer Science, Carleton University, 1998, http://www.scs.carleton.ca.

[7] E. Dekel and S. Sahni, "A Parallel Matching Algorithm for Convex Bipartite Graphs," International Conference on Parallel Processing, 1982, pp. 178–184.

[8] X. Deng, "A Convex Hull Algorithm for Coarse Grained Multiprocessors." *Proc. 5th International Symposium on Algorithms and Computation*, 1994.

[9] X. Deng and P. Dymond. "Efficient Routing and Message Bounds for Optimal Parallel Algorithms." *to appear in IPPS 1995*, Santa Barbara, April, 1995.

[10] X. Deng and N. Gu, "Good Programming Style on Multiprocessors." *Proceedings of IEEE Symposium on Parallel and Distributed Processing*, Dallas, October, 1994, pp.538-543,

[11] A.V. Gerbessiotis and L.G. Valiant, "Direct Bulk-Synchronous Parallel Algorithms," *Proc. 3rd Scandinavian Workshop on Algorithm Theory*, Lecture Notes in Computer Science, Vol. 621, 1992, pp. 1-18.

[12] F. Glover, "Maximum matching in convex bipartite graphs," Naval Res. Logist. Quarterly, 14, 1967, pp. 313–316.

[13] M.T. Goodrich, "Communication efficient parallel sorting," ACM Symposium on Theory of Computing (STOC), 1996.

[14] J.E. Hopkroft, R.M. Karp, "An $n^{5/2}$ algorithm for maximum matching in bipartite graphs," SIAM J. Comput., 2, 1973, pp. 225–231.

[15] W. Lipski and F.P. Preparata, "Efficient algorithms for finding maximum matchings in convex bibartite graphs and related problems," Acta Informatica, 15, 1981, pp. 329–346.

[16] Hui Li, and K.C. Sevcik, "Parallel Sorting by Overpartitioning." *SPAA94*, pp.46-56, 1994.

[17] L. Snyder, "Type architectures, shared memory and the corollary of modest potential." *Annu. Rev. Comput. Sci. 1*, pp.289-317, 1986.

[18] L.G. Valiant, "A Bridging Model for Parallel Computation." *Communications of the ACM*, Vol. 33, pages 103–111, 1990.