Optimization of Parallel Programs on Machines with Expensive Communication

Welf Löwe Jörn Eisenbiegler Wolf Zimmermann

Institut für Programmstrukturen und Datenorganisation, Universität Karlsruhe, 76128 Karlsruhe, Germany, E-mail:{loewe|eisen|zimmer}@ipd.info.uni-karlsruhe.de

Abstract. We present a scheduling technique that guarantees asymptotically a performance within a factor of four of the optimum for a subclass of parallel programs even if communication is expensive on the target machine. This class includes programs for FFT and matrix-multiplication for which we give practical results on a Parsytec Power-Xplorer and on a workstation-cluster.

1 Introduction

For most parallel programs the communication behavior only depends on the size of the problem and not on the actual input. Using this property for translation and optimization improves the efficiency of the generated code dramatically [ZL94, LZ95c, LZ95b, ZLG96]. Moreover, programmers may focus on the inherent parallelism of the problems and relax to the properties of the target machine, i.e. they can use a synchronous, shared memory programming model and neither data alignment nor mapping of processes onto processors is explicitly required in the source code[1]. Data and processes are distributed automatically. For the necessary optimization that clusters processes and reduces communications by redundant computations, a cost model is required reflecting latency for point-to-point-communication in the network, overhead of communication on processors themselves, and the network bandwidth. The LogP-machine [CKP+93] models these communication costs with parameters *L*atency, *o*verhead, and *g*ap (which is actually the inverse of the bandwidth). In addition to L, o, and g, parameter P describes the number of processors. These parameters have been determined for the CM-5 [CKP+93] and for the IBM SP1 machine [DMI94]. Both works found all LogP-based predictions on the runtimes confirmed by practice.

With this cost model we can not only perform optimization automatically but also predict the quality of the optimized programs in terms of their runtime on a specific target machine. Our optimization techniques *guarantee* a factor of four of the theoretically optimal performance if the problem size is sufficiently large. We prove this time bound and confirm it by performance measurements of programs for FFT and matrix-multiplication on a Parsytec Power-Xplorer

[1] This programming model is equivalent to the CREW-PRAM machine model, see [KR90].

and a cluster of workstations. With these measurements we also show that the LogP-model is appropriate for these machines, as well.

We structure this paper in the following way: In section 2, we give some basic definitions. In section 3, we define the optimization algorithm and prove the worst case performance of the optimized programs. In section 4, we apply this algorithm on our example programs and measure the runtime of the resulting code. Finally in section 5, we conclude our results and show directions of further work.

2 Basic Definitions

First, we define the notion of communication structure of a parallel program. We assume a PRAM programming model and make explicit the implicit assumption that a PRAM has a global clock. Then parallel programs are classified according to their communication structure.

For defining communication structures, we assume the following: The only composite data structures we use are arrays. This is no restriction since the shared memory may be considered as an array of integers. We allow the introduction of several arrays that may be concatenated to one single array a. The size of a problem n is defined by $|a|$. $P_x(n)$ denotes the maximum number of processors used by a PRAM-program on a problem x of size n. Each sequence of the PRAM-program which has no side effects on the shared memory is a PRAM-step. $D_x(n)$ denotes the maximum number of sequential steps necessary for a problems x of size n.

Definition 1. Processor i *communicates at step* s *with processor* j iff there is a memory cell m which was either written by processor j at step s' or $s' = 0$, no processor writes into m between step s' and step s, and processor i reads at step s from m. We denote this by the predicate $comm(i, s, j, s')$. A *communication structure* of a PRAM-program for an input x for problem size n is a directed acyclic graph $G_x(n) = (V_x(n), E_x(n))$, where $V_x(n) = \{\langle i, s\rangle : 0 \leq i < P_x(n), 0 \leq s \leq D_x(n)\}$ and $E_x(n) = \{(\langle j, s'\rangle, \langle i, s\rangle) : s' < s \wedge comm(i, s, j, s')\}$.

$G_x(n)$ does not always depend on the actual problem, i.e. input x. In most cases of practical relevance it only depends on the problem size n. We call these PRAM-program *oblivious* and denote its communication structures by $G(n)$. In the following, we consider oblivious programs and write G instead of $G(n)$ if n is arbitrary but fixed.

Each PRAM-step is assigned to exactly one vertex v and $T(v)$ denotes the time for executing this sequence on the target machine. The set of vertices can be partitioned according to the step when the vertices are executed on the PRAM. We call this partitioning a *layering* of the communication structure:

Definition 2 Layer. A *layer* Λ^s of a communication structure $G = (V, E)$ is the set of vertices $\Lambda^s = \{\langle i, s' \in V\rangle : s' = s\}$. The *work* $W^k(n)$ and the *computation time* $T^k(n)$ *of a layer* Λ^k are defined as $W^k(n) = \sum_{i=1}^{P(n)} T(\langle i, k\rangle)$ and

$T^k(n) = \max_{i=1}^{P(n)} T(\langle i,k\rangle)$, respectively[2]. A part of the PRAM-program induced by succeeding layers $\Lambda^j, ..., \Lambda^k, j \leq k$ is called a *super-layer* Λ_j^k. The *work* $W_j^k(n)$ and *computation time* $T_j^k(n)$ of Λ_j^k are defined as $W_j^k(n) = \sum_{i=j}^k W^i(n)$ and $T_j^k(n) = \sum_{i=j}^k T^i(n)$, respectively.

Any oblivious PRAM-program can be transformed into a semantically equivalent program that can be executed asynchronously on a distributed memory machine, i.e. on the LogP-machine. For this transformation, we compute its communication structure. The computations corresponding to each vertex of the communication structure is assigned to a separate process that receives data from the processes corresponding to its predecessor vertices and sending data to the processes corresponding to its succeeding vertices. This transformation leads to a semantically equivalent LogP-program. A formal program transformation and the proof of its correctness is given in [ZL94] for oblivious and in [Löw95] for non-oblivious PRAM-programs. The result of these transformations are LogP-programs or *LogP-schedules*. A definition of the notion of LogP-schedule can be obtained from [ZL94]. Informally, a schedule is a set of sequences of computations, send-, and receive operations and their starting times corresponding to the vertices and edges of the communication structure. For the starting times it must be considered that sending and receiving a message takes time o, that between two sends or receives on one processor, there must be at least time g, that a receive, it must correspond to some send at least $L + o$ time units earlier in order to avoid waiting times, and that computing a process v takes time $T(v)$. Of course, a correct LogP-schedule of a communication structure G must compute all its vertices at least once. The goal is to minimize the execution time of LogP-schedules. We rather refer to the sets of processes computed on the same LogP-processor than to its concrete sequences of computations, send-, and receive-operations. Such a set Cl of processes of a communication structure G is called *cluster* Cl, the set of all clusters of a program is called *clustering* $\mathcal{C}(G)$ of G.

Papadimitriou and Yannakakis showed, that finding an optimal clustering is NP-hard, even if $o = g = 0$ and $P = \infty$, [PY90]. They also showed that under the same assumptions approximations guaranteeing a factor of two of the optimum $TIME_{opt}(G)$ cannot be found in polynomial time, unless P=NP. We can therefore not expect to find an efficient and optimal transformation. If $o = g = 0$ and $P = \infty$, Gerasoulis and Yang [GY93] find solutions guaranteeing $(1+1/\gamma) \times TIME_{opt}(G)$ without redundant computations, where the granularity γ is a constant closely related to the ratio of computation and communication times. For trees they find the optimum if $\gamma \geq 1$ and for a subclass of trees they find the optimum even for small computation times [YG94], both in polynomial time. We showed in [LZ95a] that an optimal solution with redundant computations can be found in polynomial time for general communication structures, if

[2] Note, that these definition of work and time differ form those of the PRAM-work and -time. Ours reflect the actual computational effort on a concrete architecture rather than number of computational steps on some abstract machine.

$\gamma \geq 1$, $o = g = 0$, and $P = \infty$. Unfortunately, these results do not generalize for the LogP-machine in a straightforward way. Karp et al. e.g. [KSSS93] give optimal LogP-schedules for fork- and join-trees that significantly differ from the optimal schedules for the same structures proposed in [GY93, YG94, LZ95a]. However, a performance of LogP-programs not worse than a factor of $1 + 1/\gamma$ of the optimal program can be guaranteed if the granularity γ is generalized to the LogP-machine [LZ95c, LZ95b]. It is easy to see that in case of expensive communication this performance guarantee is large. In the next section, we present a scheduling algorithm guaranteeing better results in this latter case. More precisely, we remove the impact of L, o, and g on the performance guarantee for LogP-programs and improve it for programs running on machines with expensive communication.

3 The Algorithm

The basic idea is to save communications by computing some processes redundantly. Therefore, we first cluster the vertices of succeeding layers such that communication within the same cluster is avoided. This construction guarantees that the computation time in each cluster dominates the succeeding communication. Second, we reduce the number of clusters of the same super-layer to P such that the introduced redundancy is minimized. Finally, we assign each cluster to a LogP-processor, and schedule communication operations between clusters of different super-layers.

First of all, we find an upper bound for the all-to-all-communication of m items on LogP-machines, i.e. each processor communicates m items to all other processors.

Lemma 3. *A LogP-machine all-to-all broadcast of m items requires at most:* $\bar{L}(m) = m \cdot (P-1)\max\{2o, g\} + L$.

Proof. To transmit one item per processor to all other processors requires time $(P-1)\max\{2o, g\} + L$ if every processor p sends its item to processor $(p + i) \bmod P, 1 \leq i < P$ at time $(i-1)\max\{2o, g\}$ and receives the item of processor $(p-i) \bmod P$ as soon as possible after time $o+L+(i-1)\max\{2o, g\}$. Note, that send and receive operations interleave on the single processors without gaps[3].

The clustering algorithm is as follows: Start with layer of depth $j = 0$. Consider all layers from depth $k = 0$ to $D(n)$ and check whether $\bar{L}(\lceil |\Lambda^k|/P \rceil) \leq \max\{T_j^k, W_j^k/P\}$ holds. If this is the case, set for all v in Λ^k $Cl(v) = ANC(v) \cap \Lambda_j^k$ and set $j = k + 1$, i.e. layers Λ^j to Λ^k form a super-layer Λ_j^k without internal communication. The set of all $Cl(v)$ together defines a clustering $\mathcal{C}(G)$.

[3] The described schedule for communication is a generalization of [KSSS93]. Note, that the communication between super-layers can be improved for machines where it is more efficient to send one message of size m than m messages of size one.

Lemma 4. *Let $\mathcal{P}$ be a PRAM-program with communication structures $G(n)$. Assume a super-layer-wise (arbitrary) balanced distribution of the $Cl(v)$ to the P processors. If it holds that* (I) *$G(n)$ contains only edges connecting vertices of immediately succeeding layers, then the communication between super-layer Λ_j^k and the succeeding super-layer takes at most $\bar{L}(\lceil|\Lambda^k|/P)\rceil$ Furthermore, the computation of the processes of a super-layer on P processors requires at least time* $\max\{T_j^k, W_j^k/P\}$.

Proof. Because of balanced distribution of clusters, every processor computes at most $\lceil|\Lambda^k|/P\rceil$ clusters. Because of condition (I) there are only $\lceil|\Lambda^k|/P\rceil$ items to transmit between the super-layers. Together with lemma 3 bound for communication is proven. The bound for computation is obvious.

Lemma 5. *Let $\mathcal{P}$ be a PRAM-program. If* (I) *from holds, then for all clusterings $CG(n)$ the depth of each super-layer is bound by a constant.*

Proof. It is obvious that $\bar{L}(\lceil|\Lambda^k|/P\rceil)$ is $O(\max\{T_j^k, W_j^k/P\})$, since L, o, g, and P are constants and already $W^k(n) \geq |\Lambda^k(n)|$. Because of condition (I) only vertices in the last layer of a selected super-layers communicate with succeeding clusters. For each additional layer $\Lambda^i, j \leq i \leq k$ in super-layer Λ_j^k computation time increases by $W^i(n)/P$ without affecting communication costs. Hence, there is only a constant number of layers $\Lambda^i, i < k$, required to make the clustering condition in our algorithm true.

Now, we consider communication structures where $C(G)$ does not contain redundant computations, e.g. full binary trees, pyramids etc.

Theorem 6. *Let $\mathcal{P}$ be a PRAM-program. If additionally to conditions* (I) *it holds that* (II) *$G(n)$ is a balanced forest*[4]*, and* (III) *there is only a constant number of layers Λ with $|\Lambda| < P$, then any balanced distribution of the clusters of the same super-layer to P processors defines a schedule with an asymptotic execution time of at most:*

$$TIME(G(n)) \leq (4 + o(1)) \cdot TIME_{opt}(G(n)).$$

Proof. Because of condition (II), the sets of ancestors $ANC(v)$ of vertices of the same layer are pairwise disjoint. Hence, the sum of the work of the clusters containing vertices of layers Λ^k to Λ^j is equal to $W_j^k(n)$. The work of each of these clusters is $W_j^k(n)/|\Lambda^j|$ Assume $\forall j \in [0, D(n)] : |\Lambda^j| \geq P$. Because of balanced distribution, for the time to compute the clusters of Λ_j^k on the LogP-machine it holds $TIME_j^k \leq 2 \cdot W_j^k(n)/P$. The time for succeeding communication L_j^k is at most equal to $TIME_j^k$, see lemma 5. Therefore, $TIME_j^k + L_j^k \leq 2 \cdot W_j^k(n)/P + \max\{T_j^k(n), W_j^k(n)/P\}$. Summing up the times on both sides of the inequation over the super-layers leads to $TIME(G) \leq 2 \cdot W(n)/P + T(n) + W(n)/P$, where

[4] A forest is balanced iff all vertices of the same height (same layer) have the same weight (same computation times) and their subtrees are equal.

$W(n)/P$ and $T(n)$ are lower bounds of the optimum execution time. Since, there are only constant many layers containing less than P vertices (III) there is only a constant delay to this bounds for increasing n while $W(n)/P$ and $T(n)$ increase with n. This completes the proof.

Condition (II) excludes a lot of important PRAM-programs. On the other hand, if (II) is not guaranteed our algorithm computes a schedule with a performance depending on the amount of redundant computation. We therefore try to find a weaker conditions such that redundancy can be eliminated while scheduling the clusters to processors. We formalize the problem of reducing the redundancy in the following straightforward way: For all super-layers we construct a redundancy graph $G_R = (V_R, E_R)$. This graph is undirected and there is a one-to-one-correspondence between the clusters $Cl(v)$ and the vertices in V_R. There exists an edge (v', v'') iff $Cl(v')$ and $Cl(v'')$ compute some vertices redundantly. The edges are weighted with the time for computing these redundant vertices sequentially. The general problem is to find a minimum balanced P-cut in G_R[5]. Unfortunately, this problem is NP-hard [SV91], even for the unbalanced version, and the best approximation for the balanced version is within a factor of $|V_R| \cdot (P-1)/P$ of the optimum. Hence, we cannot expect acceptable solutions for all PRAM-programs.

Corollary 7. *Let $\mathcal{P}$ be a PRAM-program. If for all super-layers Λ_j^k and increasing problem size the corresponding redundancy graph $G_R = (V_R, E_R)$ can be divided into $\chi = \omega(1)$ connected components of size $O(P(n)/\chi)$, theorem 6 holds even if its condition* (II) *is neglected.*

Proof. At each super-layer Λ_j^k we compute the redundancy graph $G_R = (V_R, E_R)$. Note, that $|V_R| \leq P(n)$. For each superlayer, we reorder the shared array such that no array cells of different connected components are merged. There are χ sets of clusters with pairwise disjoint sets of vertices. We compute each set on exactly one processor. Hence, we can remove all redundancy. Since their number is growing with the problem size and their sizes are balanced up to a constant factor, we can schedule these χ sets onto P processors such that scheduling is balanced and theorem 6 holds.

The proof is constructive. We can therefore find a correct alignment of the shared memory in these cases. Note, that the induced algorithm performs possibly a realignment at every super-layer. The class of tractable programs include now FFT and matrix-multiplication (with duplicated second matrix), and sorting networks.

We conclude this section with the remark that we finally apply heuristics which keep the communications between the clustered layers small: The clusters and communication edges between these clusters define a directed acyclic graph.

[5] A balanced k-cut of a undirected graph is a set of edges that, if removed, divides G into k connected components of equal size. A minimum k-cut of a weighted undirected graph is a k-cut with minimum weight of removed edges.

Recursively, we assign the clusters on the maximum weighted path in this graph to the same processor and remove them, until the graph is empty. This sequentialization of clusters possibly decreases the overall computation time since it saves some communication.

4 Practical Results

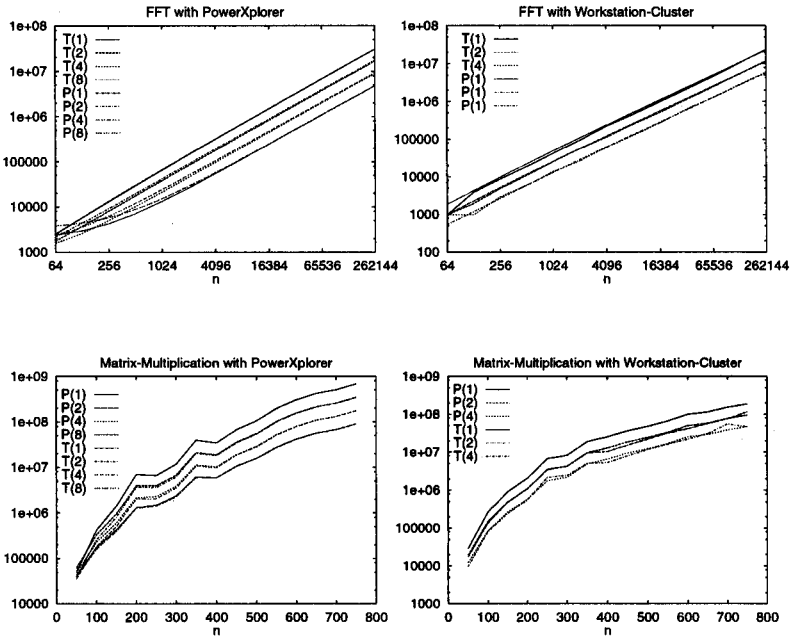

Fig. 1. Predicted and measured performance for matrix-multiplication and FFT on a Power-Xplorer and a Workstation-Clusters. T(x) is the measured and P(x) the predicted running time for x processors.

We applied this scheduling algorithm for a PRAM-implementation of the Fast-Fourier-Transform (FFT) and matrix-multiplication on a Parsytec Power-Xplorer with eight processors and on a cluster of four workstations with a 360 MHz DEC-21064 Processor connected via the ParaStation Network[6]. Comparing the predicted runtime of these programs with practical times we found that the LogP-model is adequate for these machines (see figure 1). With the range of input-sizes shown in figure 1 we obtained the maximal speed-ups shown in figure 2. We measured the maximum speed-up for FFT with an input size of 2^{18} and for matrix-multiplication of 750×750-matrices.

[6] `http://wwwipd.ira.uka.de/~psuser`

	Parsytec Power-Xplorer			Workstation-Cluster	
	P=2	P=4	P=8	P=2	P=4
FFT	1.83	3.46	6.35	1.92	3.72
Matrix-Multiplication	1.98	3.92	7.60	1.60	3.90

Fig. 2. Measured speed-up for matrix-multiplication and FFT on a Power-Xplorer and on a network of four DEC-Workstations.

Note, that the speed-ups are close to the theoretical optimum. It shows that our scheduling algorithm may in practice lead to significantly better programs than proved above. However, the upper time bound holds and it should be possible to construct odd programs where this bound is tight.

5 Conclusion

The main result of this article showed that it is possible to get both: machine independent programming and efficient execution of parallel programs, at least for the class of oblivious programs. This class includes programs for matrix-multiplications, FFT, summations, sorting networks etc. It can be extended to programs for solving partial differential equations with finite element methods or methods of finite differences and others[7].

Our optimization algorithm guarantees asymptotically a factor of four of the theoretical optimum even on parallel machines with expensive communication costs. This optimization considers the limited number of available processors, the time bound does not depend on communication times. It is therefore a generalization of [ZL94, LZ95c, LZ95b, ZLG96]. We applied the algorithm on FFT and matrix-multiplication programs. Performance measurements of the optimized programs on a Parsytec-Xplorer and on a workstation-cluster confirmed the practical relevance of our main result. The measurements also show, that the LogP-model yields precise runtime predictions and is therefore appropriate for these machines.

The next step is to integrate the optimization methods into a compiler. If this is done, then programming languages on a PRAM-level can be implemented and efficient code can be guaranteed. The main result of this article shows, that there is no serious slowdown in spite of the abstract PRAM-model, even if communication is expensive. Such a language would simplify parallel programming a lot. The programmers could focus on the parallelization of their problem at hand. All other tasks like data distribution, introducing and optimizing communication, and mapping parallel processes and processors will be done by the compiler.

[7] It is omitted because of the strict page limit and can be taken form the (also reviewed) extended version of this paper `http://i44www.inof.uni-karlsruhe.de/~loewe/parallel.html`

References

[CKP+93] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a realistic model of parallel computation. In *4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP 93)*, pages 1–12, 1993. published in: SIGPLAN Notices (28) 7.

[DMI94] B. Di Martino and G. Ianello. Parallelization of non-simultaneous iterative methods for systems of linear equations. In *LNCS 854, Parallel Processing: CONPAR'94–VAPP VI*, pages 254–264. Springer, 1994.

[GY93] A. Gerasoulis and T. Yang. On the granularity and clustering of directed acyclic task graphs. *IEEE Transactions on Parallel and Distributed Systems*, 4:686–701, june 1993.

[KR90] R. M. Karp and V. Ramachandran. Parallel algorithms for shared memory machines. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science Vol. A*, pages 871–941. MIT-Press, 1990.

[KSSS93] R. M. Karp, A. Sahay, E. E. Santos, and K. E. Schauser. Optimal broadcast and summation in the LogP model. In *5th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 142–153. ACM, 1993.

[Löw95] W. Löwe. Optimization of pram-programs with input-dependent memory access. In S. Haridi and K. Ali, editors, *EUROPAR' 95. Parallel Processing*, volume 966 of *Lecture Notes in Computer Science*, pages 243–254, 1995.

[LZ95a] W. Löwe and W. Zimmermann. On finding optimal clusterings in task graphs. In N. Mirenkov, editor, *Parallel Algorithms/Architecture Synthesis pAs'95*, pages 241–247. IEEE, 1995.

[LZ95b] W. Löwe and W. Zimmermann. Programming data-parallel – executing process parallel. In P. Fritzson and L. Finmo, editors, *Parallel Programming and Applications*, pages 50–64. IOS Press, 1995.

[LZ95c] W. Löwe and W. Zimmermann. Upper time bounds for executing pram-programs on the logp-machine. In M. Wolfe, editor, *Proceedings of the 9th ACM International Conference on Supercomputing*, pages 41–50. ACM, 1995.

[PY90] C.H. Papadimitriou and M. Yannakakis. Towards an architecture-independent analysis of parallel algorithms. *SIAM Journal on Computing*, 19(2):322 – 328, 1990.

[SV91] H. Saran and V. Vazirani. Finding k-cuts within twice the optimal. In *Proceedings of the 32 Ann. IEEE Symp. on Foundations of Computer Science*, pages 743–751. IEEE Compute Society, 1991.

[YG94] T. Yang and A. Gerasoulis. Dsc: Scheduling parallel tasks on an unbounded number of processors. *IEEE Transactions on Parallel and Distributed Systems*, 5(9):951–967, 1994.

[ZL94] W. Zimmermann and W. Löwe. An approach to machine-independent parallel programming. In *Parallel Processing: CONPAR 94 – VAPP VI*, volume 854 of *Lecture Notes in Computer Science*, pages 277–288. Springer, 1994.

[ZLG96] W. Zimmermann, W. Löwe, and J. Gottlieb. On the design and implementation of parallel algorithms for solving inverse problems. In *Workshop on Parameter Identification and Inverse Problems in Hydrology, Geology, and Ecology*, 1996. to appear.