universität bern
institut für informatik
und angewandte mathematik

Self-Organizing Process Mapping in a Multiprocessor System

Kuno Wyler

wyler@iam.unibe.ch

IAM-93-001

January 1993

Self-Organizing Process Mapping in a Multiprocessor System

Kuno Wyler

In this report we describe a self-organizing map to solve the mapping problem in a multiprocessor system with a distributed memory architecture. Experimental results for 2-dim lattice based processor networks are presented and the generalization of this approach to arbitrary processor topologies is discussed.

CR Categories and Subject Descriptors: D.4.1 [**Operating Systems**]: Process Management–*Concurrency*; G.2.2 [**Discrete Mathematics**]: Graph Theory–*Graph algorithms*; I.2.6 [**Artificial Intelligence**]: Learning–*Connectionism and neural nets.*

General Terms: Algorithm

Additional Key Words and Phrases: Self-Organizing feature map, multiprocessor systems, process mapping, mapping problem

1 Introduction

A multiprocessor system with a distributed memory architecture is a network with a fixed number of processors. Each node in this network is a single processor and its local memory. Every node is connected to other nodes by bidirectional transmission lines. This enables direct communication between connected (neighboring) processors. Furthermore we assume connectedness (there always exists a path between two arbitrary processors). Communication between arbitrary processors can now be realized by a communication line along this path. The connections can be fixed (fixed architectures) or variable (reconfigurable architectures) depending on the type of system . In order to run a program on a distributed system it has to be split into parts which can be executed in parallel. In terms of parallel programming such a part is called a process and a program is formed by a set of communicating sequential processes. By assigning each process to a processor the program will then be executed in parallel. In contrast to pure sequential programs shared data between different processes has now to be communicated between different processors. To exchange data processes are connected by logical communication channels which are physically realized by transmission lines.

High performance of a multiprocessor system emerges from collective computation. But it can only be achieved when the assignment of processes to processors is done efficiently. The goal of the assignment is to minimize the execution time of the

complete program. This goal is equivalent to minimize the idle time of each processor and simultaneously minimizing communication distances between processes. This problem is the so-called *mapping problem* and it is fundamental in distributed computing. Despite the lack of a rigorous proof of NP-completeness of the general mapping problem there is much evidence that it is NP-complete. For some restricted cases it is easy to show the equivalence to problems which are known to be NP-complete [Bok81].

There has been many different approaches suggested to solve the mapping problem: simulated annealing [FKW87, DSS88], solving the Poisson equation in a graph [Boi90, BIK91], cardinality based algorithms [Bok81], heuristic search [BKMW88, She88], evolutionary and genetic algorithms [MGSK87, WBK88, Nor89, MT91, GBW92] and neural networks [FF86, Sch90].

In this paper we present an approach which uses only topological information to find a near optimal process mapping with respect to a specific architecture. The mapping is done by a self-organizing map which imitates the processes of topology preserving self-organization of stimuli found in the cerebral cortex.

2 Formal Definition of the Mapping Problem

In this section we will describe a parallel program and a multiprocessor system as a network of nodes which are processes or processors respectively. Thus a parallel program can be represented as a graph $G = (V_G, E_G)$ where each process is identified by a vertex $p \in V_G$ (with $|V_G| = n$) and each communication channel by an edge $(p, q) \in E_G \subseteq V_G \times V_G$. For each vertex p the map τ denotes the number of the atomic computational steps performed by the process p:

$$\tau : V_G \longrightarrow I\!N, \qquad \tau : p \longmapsto \tau(p) = \tau_p,$$

and for each edge (p, q) the map c^G denotes the communication volume (number of information units) passed from process p to process q:

$$c^G : E_G \longrightarrow I\!N, \qquad c^G : (p, q) \longmapsto c^G(p, q) = c^G_{pq},$$

Similarly a multiprocessor system can be defined as a graph $H = (V_H, E_H)$ where each processor forms a vertex $i \in V_H$ (with $|V_H| = m$) and each transmission line is given by an edge $(i, j) \in E_H \subseteq V_H \times V_H$. For each vertex i the map θ denotes the number of atomic computational steps executed by the processor i per time unit:

$$\theta : V_H \longrightarrow I\!N, \qquad \theta : i \longmapsto \theta(i) = \theta_i,$$

for each edge (i, j) the map c^H denotes the communication capacity of the transmission line between i and j:

$$c^H : E_H \longrightarrow I\!N, \qquad c^H : (i, j) \longmapsto c^H(i, j) = c^H_{ij},$$

and for any two processors i and j the map δ denotes the costs of transferring one information unit between i and j:

$$\delta : V_H \times V_H \longrightarrow I\!N, \qquad \delta : (i,j) \longmapsto \delta(i,j) = \delta_{ij}.$$

Communication costs are mainly determined by the length of the communication path (distance between two processors) and its capacity.

The mapping problem can now be stated as a topological problem of embedding the graph G into the graph H. Thus the process mapping is a function Φ assigning each process p to a processor i:

$$\Phi : V_G \longrightarrow V_H, \qquad \Phi : p \longmapsto \Phi(p) = i$$

such that

$$\forall (p,q) \in E_G \quad \Longrightarrow \quad (\Phi(p), \Phi(q)) \in E_H$$

holds. This is not always possible but can be forced by inserting dummy vertices (additional routing processes) in G.

To attain the optimization goal (minimization of execution time) the embedding has to satisfy the following two conditions simultaneously:

(1) **Load balancing:** to minimize the idle time of the processors the load caused by the processes should be distributed evenly over all processors.

(2) **Communication minimization:** to minimize communication costs communicating processes should be mapped onto the same or onto neighboring processors (clustering).

The load of a processor i induced by a mapping Φ is given by

$$L_i = \frac{1}{\theta_i} \sum_{p \in \{q : \Phi(q) = i\}} \tau(p).$$

The system is optimally balanced when for each processor the load is equal to the average load $\overline{L}$:

$$\overline{L} = \frac{1}{m} \sum_{i}^{m} L_i.$$

Hence the load variance indicates how well the load of the whole system is balanced. So condition (1) is fulfilled when the load variance is minimized

$$\frac{1}{m-1} \sum_{i}^{m} (L_i - \overline{L})^2 \quad \rightarrow \quad \min.$$

The second condition minimization of communication costs can be expressed by

$$\sum_{p,q} c^G_{pq} \cdot \delta_{\Phi(p)\Phi(q)} \quad \rightarrow \quad \min.$$

A mapping Φ satisfying condition (1) and (2) is called an optimal mapping (it should be noted that condition (1) and (2) may be contradictory). Most approaches are using a linear combination of condition (1) and (2) as an objective function and they consider the mapping problem as a problem of function minimization. In our approach we do not use explicitly an objective function but the minimization is done implicitly by the algorithm (see section 4).

3 Self-Organizing Map

The self-organizing map [Koh88] is a neural network developed by Kohonen and its architecture is inspired by the various maps formed by self-organization in the cerebral cortex that are able to describe topological relations of sensory input signals, e. g. the somatotopic map which contains a topological representation of the body surface. The network performs a mapping from a space $X \subseteq I\!R^n$ into a set of units $\{1, \ldots, m\}$. The units are topologically ordered and the mapping is realized such the topological properties of the points of the input space are preserved. A sample point in X is a n-dimensional vector $\mathbf{x} \in X$ and the density of sample points in X is given by $\rho(\mathbf{x})$. Further for each unit i the function w assigns a reference vector $w(i) = \mathbf{w}_i \in X$. Kohonen's algorithm compares an input vector $\mathbf{x}$ simultaneously with each reference vector $\mathbf{w}_i$ and then adjusts the best matching one to become even more similar to $\mathbf{x}$. In this way the reference vectors $\mathbf{w}_i$ approximate more and more closely the distribution of sample points in X and they can be interpreted as center of mass with respect to the density defined by $\rho(\mathbf{x})$. When the similarity measure is defined by some metric $\mathrm{d}(\cdot, \cdot)$

$$\text{e.g.} \qquad \mathrm{d}(\mathbf{x}, \mathbf{w}) = ||\mathbf{x} - \mathbf{w}||$$

then the most similar reference vectors $\mathbf{w}_i$ is given by

$$\mathrm{d}(\mathbf{x}, \mathbf{w}_i) = \min_{k=1}^{m} \{\mathrm{d}(\mathbf{x}, \mathbf{w}_k)\}.$$

Kohonen's algorithm tends now to minimize the expected approximation error:

$$\sum_{p=1}^{P} \int_X \mathrm{d}(\mathbf{x}_p, \mathbf{w}_i) \rho(\mathbf{x}) d\mathbf{x} \quad \rightarrow \quad \min$$

where $\mathbf{w}_i$ is the most similar reference vector of the pth sample point $\mathbf{x}_p$ (with $1 \leq p \leq P$) and $d\mathbf{x}$ is the volume differential in X.

Usually the units of a self-organizing map are arranged in a line or a 2-dimensional lattice. With respect to this topology we can define a topological neighborhood N_i of unit i which includes all units that lie within a certain radius r from i:

$$N_i = \{j : \mathrm{d}(i, j) \leq r\}.$$

The adjustments of the reference vectors are computed during a so-called learning phase. In one learning step a sample point $\mathbf{x}_p$ is randomly chosen and the unit i with the most similar reference vector is determined. Then the reference vectors of this unit and of the units of its neighborhood N_i are slightly adjusted according the following rule:

$$\mathbf{w}_j(t+1) = \begin{cases} \mathbf{w}_j(t) + \alpha(t)[\mathbf{x}_p - \mathbf{w}_j(t)] & : \quad j \in N_i \\ \mathbf{w}_j(t) & : \quad j \notin N_i \end{cases}$$

where $\alpha(t)$ is a slowly decreasing learning rate controlling convergence speed of learning. This procedure is repeated with all sample points $\mathbf{x}_p \in X$ until a satisfying topological order of the reference vectors $\mathbf{w}_i$ in X is achieved.

4 Self-Organizing Process Mapping

In this section we apply the concept of self-organization to the problem of process mapping. The basic idea is to choose for the self-organizing map the same topology H as used for the multiprocessor system and to find an appropriate representation of the processes in a n-dimensional space such they can be used as sample points to train the Kohonen network.

The most difficult problem is to find an appropriate representation of the processes in a n-dimensional space. Each process has to be represented as a point in this space and the topological neighborhoods of the points in this space must be the same as the topological neighborhoods in the graph G. This is again an embedding problem but in some kind an inverse one: we know the topological relations of points and we have to find a representation in a metric space such that the neighborhoods are preserved.

In our approach we use distances between processes in the graph G to define the topological position of a single process. The distance between two processes p and q is defined by the number of edges in a shortest path between p and q. In this way we get for each process p a n-dimensional vector $\mathbf{d}_p = (d_{p1}, \ldots, d_{pq}, \ldots, d_{pn})$ where d_{pq} denotes the distance between p and q. The vector $\mathbf{d}_p$ forms the pth row or column of the distance matrix of G denoted by $D(G) = D$.

If two processes p and q are in a close neighborhood they will have approximately the same distances to all the others processes thus the difference between $\mathbf{d}_p$ and $\mathbf{d}_q$ will be quite small. Conversely when the difference between two distance vectors $\mathbf{d}_p$ and $\mathbf{d}_q$ is small we can conclude that p and q must lie in the same neighborhood as well. So the characterization of the topological position of a process by its distance vector can be used as an appropriate representation in a n-dimensional space. But we are aware that the embedding into the n-dimensional space is not distance preserving. This can be demonstrated by the following example: considering three processes p,

q, s and the Euclidean distance d with

$$\mathrm{d}(\mathbf{d}_p, \mathbf{d}_q) < \mathrm{d}(\mathbf{d}_p, \mathbf{d}_s)$$

does not strictly implies $d_{pq} < d_{ps}$. This occurs because the value of $\mathrm{d}(\mathbf{d}_p, \mathbf{d}_q)$ depends not only from a single distance but from all distances.

We now show how the mapping problem can be solved by a self-organizing map. For simplicity we assume homogeneous processes p and processors i:

$$\tau(p) = 1, \quad \theta(i) = 1, \qquad \forall p, i$$

and also homogeneous communication volumes between communicating processes p and q:

$$c^G(p,q) = \begin{cases} 1 & : \ p \text{ and } q \text{ are communicating} \\ 0 & : \ \text{otherwise} \end{cases}$$

and also homogeneous communication capacities between directly connected processors i and j:

$$c^H(i,j) = \begin{cases} 1 & : \ i \text{ and } j \text{ are connected} \\ 0 & : \ \text{otherwise} \end{cases}$$

By neglecting routing costs we assume communication costs between arbitrary processors to be equal to the length of the shortest path between them. This again corresponds to the distance of two vertices in the graph H. The distance matrix of H is referred as $D(H) = \Delta = [\delta_{ij}]$ where δ_{ij} denominates the distance between processor i and j.

In order to find a process mapping Φ we create a self-organizing map according the topology of H which is trained by the sample points $\mathbf{d}_p$ extracted from G. After learning we get Φ by the relation

$$\Phi(p) = i \quad \Longleftrightarrow \quad \mathrm{d}(\mathbf{d}_p, \mathbf{w}_i) = \min_{k=1}^{m} \{\mathrm{d}(\mathbf{d}_p, \mathbf{w}_k)\}.$$

Thus Kohonen's algorithm tends to preserve neighborhoods the value of $\delta_{\Phi(p)\Phi(q)}$ will be small for all pairs of communicating processes p and q. Hence the communication cost will be minimized by Φ:

$$\sum_{p,q} c^G_{pq} \cdot \delta_{\Phi(p)\Phi(q)} \quad \rightarrow \quad \min.$$

Furthermore a self-organizing map tends to approximate the density of the sample points. Hence the decision regions formed by the reference vectors $\mathbf{w}_k$ and the metric $\mathrm{d}(\cdot,\cdot)$ have approximately the same volume relative to the density $\rho(\mathbf{x})$ and

thus contain approximately the same number of sample points. Therefore we get the following expression for the load of each processor i

$$L_i = |\{p : \Phi(p) = i\}| \approx \frac{m}{n}.$$

The average load is given by $\overline{L} = \frac{m}{n}$. So the load variance of the system will be minimized

$$\frac{1}{m-1}\sum_{i}^{m}(L_i - \overline{L})^2 \approx 0.$$

Thus assuming the applied metric would be strictly correct we have shown that the conditions of section 2 are met by the process mapping induced by Φ hence the optimization goal will be attained.

5 Experiments and Results

In this section we present the results of four experiments. First, the mapping problem for two simple examples is solved for which the optimal solution is known. Both problems consists of 16 processes communicating either in a ring or in a 4×4-lattice topology (fig. 1) and they are mapped onto a 2×2-lattice of processors (ring). The third problem is a nontrivial one consisting of 40 processes which are randomly connected by 110 communication channels and they are mapped onto 4×4-lattice architecture. The fourth problem uses the same number of processes and uses the same architecture but has only 63 communication channels. A projection of the graphs into a plane for the last two problems is depicted in figure 3. In all experiments we used a rectangular neighborhood for training the Kohonen network and we started with a radius equal of half the diameter of H. The radius of the neighborhood was shrunk linearly to 0, e. g. the neighborhood N_i of radius 1 in a lattice contains all direct neighbors of i and all nearest neighbors in the diagonals. The initial value of the learning rate α was 0.4 and it was decreased linearly too.

One problem of our approach is that the number of dimensions of the sample space grows linearly with the number of processes in G. Thus we tried some statistical methods for dimensionality reduction in high dimensional sample space. We used principal component analysis to compute the directions of greatest variance in the sample space. These directions are called the principal components and correspond to the eigenvectors with the largest eigenvalues of the covariance matrix of D. The sample points are then projected onto the principal components. Typically some combination of two of the first four principal components are used to provide a 2-dimensional view of the sample space. Remaining components related to typically smaller eigenvalues show too little variation to be useful.

In our experiments we compared the quality of solution by using raw sample points and dimensionality reduced sample points to train the self-organizing map.

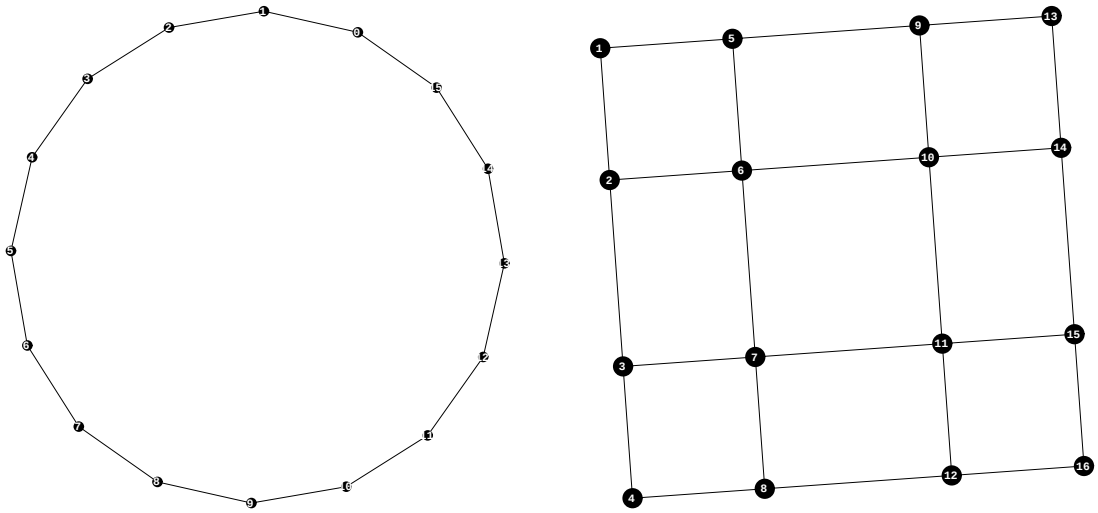

Figure 1: Projection of the program graphs G of the first two problems onto the first two principal components ($\mathbf{d}_p^{(2)}$).

To compute the "principal components" of the graph G we compute the principal components of the covariance matrix C of the distance matrix $D = [d_{pq}]$. The covariance matrix C is defined by

$$C = \frac{1}{n-1} B B^T = \frac{1}{n-1} B^2$$

where $B = [d_{pq} - \overline{d}_q]$ with $\overline{d}_q = \frac{1}{n} \sum_p^n d_{pq}$ (column mean). Note that the matrix B is symmetric.

The function $\lambda_\ell(C)$ calculates the ℓth largest eigenvalue λ_ℓ of the matrix C:

$$\lambda_1(C) \geq \lambda_2(C) \geq \ldots \geq \lambda_\ell(C) \geq \ldots \geq \lambda_n(C).$$

The function $x_\ell(C)$ evaluates the corresponding eigenvector $\mathbf{x}_\ell$ of λ_ℓ. Thus we have

$$C\mathbf{x}_\ell = \lambda_\ell \mathbf{x}_\ell.$$

By $\mathbf{d}_p^{(\ell)}$ we denote the ℓ-dimensional reduced sample point obtained by projecting the raw sample point $\mathbf{d}_p$ into the subspace spanned by the first ℓ eigenvectors $\mathbf{x}_1, \ldots, \mathbf{x}_\ell$.

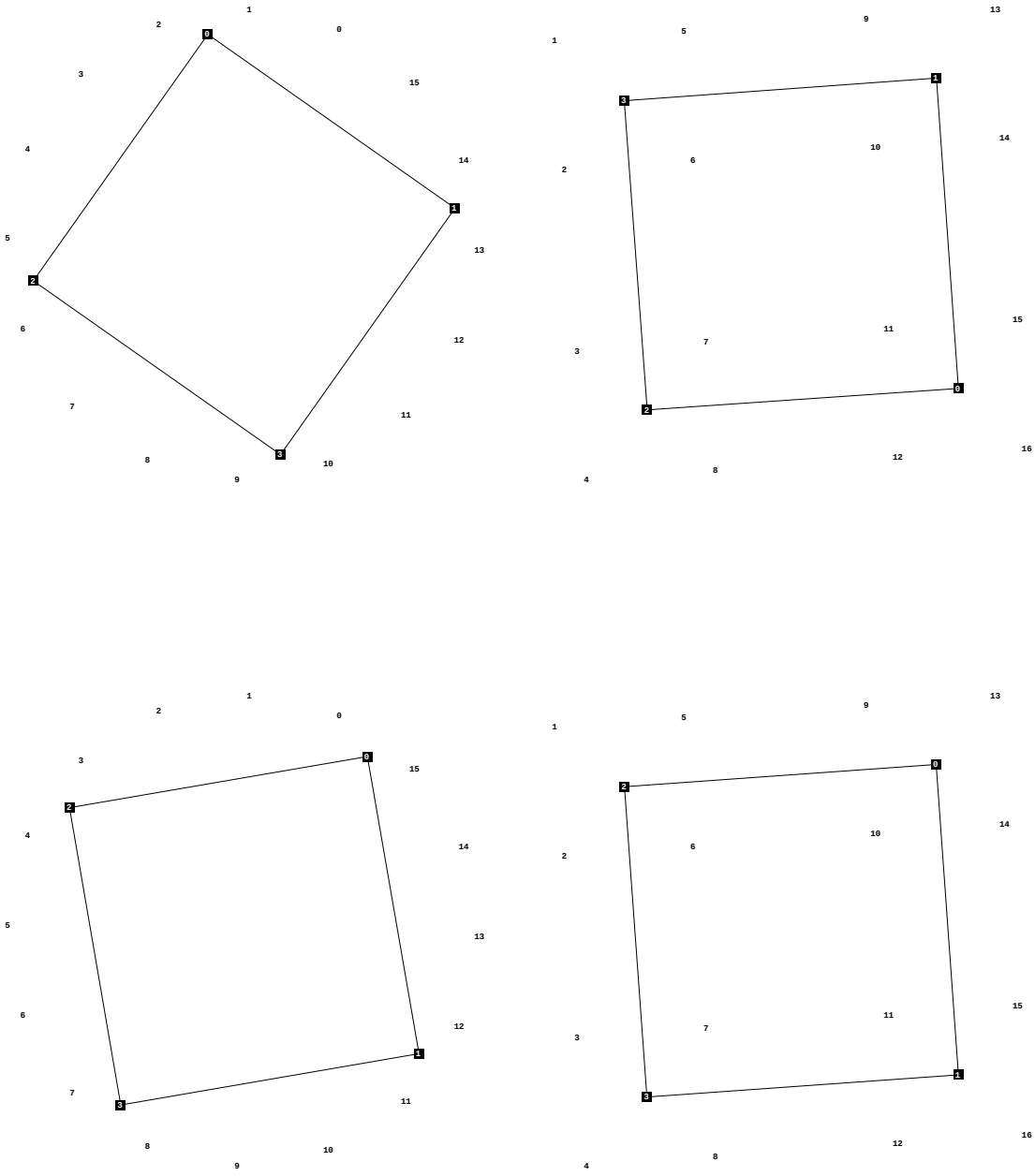

Figure 2: Projection of the learned reference vectors of the self-organizing map with topology H of the first two problems onto the first two principal components ($\mathbf{d}_p^{(2)}$). Directly connected processors (black squares) are connected by lines. For clarity communication channels of processes are omitted only process numbers are drawn. In the upper two pictures $\mathbf{d}_p^{(2)}$ sample points are used to train the self-organizing map and for the lower two raw sample points $\mathbf{d}_p$ are used.

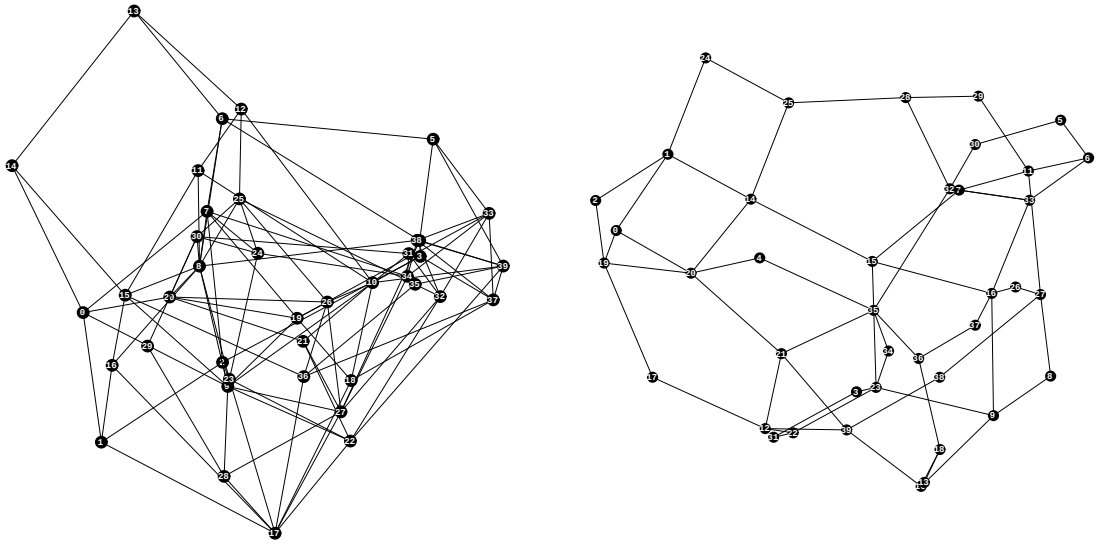

Figure 3: Projection of the program graphs G of the problems three and four onto the first two principal components ($\mathbf{d}_p^{(2)}$).

The projection of sample points onto their first two principal components is also used for plotting the graph G and visualizing the mapping Φ by projecting the reference vectors $\mathbf{w}_i$ of the self-organizing map onto the same principal components.

For the first two problems we used $\mathbf{d}_p^{(2)}$ and $\mathbf{d}_p$ respectively to train the Kohonen networks. Typically each sample point was presented 500 times in a random order. In both cases the solutions found were optimal and the results are depicted in figur 2. Another interesting fact is that for this simple planar graphs only the first two eigenvalues are significantly different from zero. This fact could eventually be useful for investigating the planarity of a graph hence it could be a hint for the topology of the optimal architecture by taking the number of significant eigenvalues as an upper bound for the embedding dimension.

For the third problem we found eight significant eigenvalues so we used $\mathbf{d}_p^{(2)}$, $\mathbf{d}_p^{(8)}$ and $\mathbf{d}_p$ to train the Kohonen network. Each sample point was presented 2000 times in a random order. The solutions for the case $\mathbf{d}_p^{(2)}$ and $\mathbf{d}_p$ are depicted on the left side in figure 4.

The mapping produced by $\mathbf{d}_p^{(8)}$ does not significantly differ from the one formed by $\mathbf{d}_p$. Thus the additional 32 dimensions of the raw sample points do not contain

Figure 4: Projection of the learned reference vectors of the self-organizing map with topology H of problem three (left side) and four (right side) onto the first two principal components ($\mathbf{d}_p^{(2)}$). Directly connected processors (black squares) are connected by lines. For clarity communication channels of processes are omitted only process numbers are drawn. In the upper two pictures $\mathbf{d}_p^{(2)}$ sample points are used to train the self-organizing map and for the lower two raw sample points $\mathbf{d}_p$ are used.

more information to improve the mapping. But it is obvious and the experiment demonstrated that $\mathbf{d}_p^{(2)}$ contains not enough information to produce a near optimal mapping (the load is poorly balanced). By comparing both solutions we can see that the projections of the reference vectors do not coincide but the twists of the second solution indicates that the embedding is done by respecting more than two principal components. The optimal solution for this problem is not known therefore we compare our solutions with solutions obtained by another mapping tool (MARC) developed by Boillat *et al.* [BIK91]. We compare communication costs (comm), number of on-chip communications (on-chip), number of direct communications (short), number of routed communications (long) and standard-deviation of load (load) induced by the mapping:

	$\mathbf{d}_p^{(2)}$	$\mathbf{d}_p^{(8)}$	$\mathbf{d}_p$	MARC
comm	156	177	178	192
on-chip	20	29	25	10
short	45	25	36	41
long	45	56	49	59
load	1.75	0.89	1.09	0.63

The solutions seems to be of the same quality where the self-organizing map is slightly better in terms of communication costs and the MARC system in terms of load balancing respectively. This effect can be explained by the fact that the process mapping performed by the MARC system is mainly driven by load-balancing.

For the last problem we obtained a similar result. Here we got seven significant eigenvalues thus we used $\mathbf{d}_p^{(2)}$, $\mathbf{d}_p^{(7)}$ and $\mathbf{d}_p$ to train the Kohonen network. Again each sample point was presented 2000 times in a random order. The solutions for the case $\mathbf{d}_p^{(2)}$ and $\mathbf{d}_p$ are depicted on the right side in figure 4. The mapping formed in the planar region of the graph G is quite similar for both solution. But again in the non-planar region the topology of G is better approximated by the second solution.

	$\mathbf{d}_p^{(2)}$	$\mathbf{d}_p^{(7)}$	$\mathbf{d}_p$	MARC
comm	55	61	58	86
on-chip	22	25	25	10
short	31	22	24	36
long	10	16	14	17
load	1.15	0.89	0.63	0.56

6 Generalization

In this section we sketch some possibilities to generalize this approach for solving a more general mapping problem.

- **inhomogeneous processes:** An inhomogeneous process p can be simulated by randomly generating a number of additional sample points proportional to $\tau(p)$ around the original sample point $\mathbf{d}_p$.
- **arbitrary processor topologies:** In our experiments we restricted ourself to lattice based topologies for the self-organizing map only for simplicity. But it is obvious that we can realize the process mapping for arbitrary processor topologies simply by changing the topology of the self-organizing map.
- **restrictions:** For some reasons a process p must be mapped onto a specific processor i. This fact forms a restriction of the mapping. A restriction can be simulated by randomly generating a number of additional sample points proportional to $\tau(p)$ around the reference vector $\mathbf{w}_i$.
- **migration strategies:** Migration strategies are needed when partial system failures occurs. Our approach also delivers some kind of migration strategy. When a processor i fails the reference vector $\mathbf{w}_i$ is removed. Then immediately the decision region formed by $\mathbf{w}_i$ is split and reassigned to the nearest neighbors of i by the metric $\mathrm{d}(\cdot,\cdot)$. By this splitting we get the direction of migration for each process. This migration process causes locally unbalanced system load. So we have to readjust slightly the reference vectors respecting the modified topology caused by the failure.
- **optimal architecture:** The optimality of an architecture can be described by the preservation and violation of neighborhood relations. By intersecting and comparing the formed neighborhoods of a self-organizing map a violation of topology preserving can be detected. Recently published methods for optimal embedding of chaotic attractors [LPS91] can be used to measure the preservation of topology formed by a self-organizing map [BP92]. By applying this measure it should be possible to get hints for the topology of the optimal architecture.

7 Conclusions

In this report we presented a new approach to solve the mapping problem in a multiprocessor system by a self-organizing map. Despite we used a metric that was not strictly correct we obtained satisfying results in all of our experiments.

Future research is directed to improve the representation of processes and to find a better metric. We will also implement a more general system according to section 6 and do more systematical experiments. In addition we will investigate the influence of parameter-settings on the quality of solutions.

Acknowledgements

This work was realized as a project of a postgraduate course on artificial and biological neural networks organized by Professor J.-N. Nicoud and Dr. F. Blayo at the Ecole Polytechnique Fédérale de Lausanne.

I would like to thank Dr. Lorenz Müller, Dr. Jacques Boillat, Philipp Schmid and Jürg Stiefenhofer for many helpful discussions and for their comments on earlier versions of this report.

References

[BIK91] J. E. Boillat, N. Iselin, and P. G. Kropf. MARC: A Tool for Automatic Configuration of Parallel Programs. In P. Welch et al., editors, *TRANSPUTING 91*, Amsterdam, 1991. IOS.

[BKMW88] J. E. Boillat, P. G. Kropf, D. Chr. Meier, and A. Wespi. An Analysis and Reconfiguration Tool for Mapping Paralle Programs onto Transputer Networks. In T. Muntean, editor, *OUG-7: Parallel Programming of Transputer based Machines*, Amsterdam, 1988. IOS.

[Boi90] J. E. Boillat. Load Balancing and Poisson Equation in a Graph. *Concurrency: Practice and Experience*, 2(4), 1990.

[Bok81] S. H. Bokhari. On the Mapping Problem. *IEEE Transaction on Computers*, C-30(3):207–214, 1981.

[BP92] H.-U. Bauer and K. Pawelzik. Quantifying the Neighborhood Preservation of Self-Organizing Feature Maps. *IEEE Transactions on Neural Networks*, 3(4):570–579, 1992.

[DSS88] J. G. Donnet, M. Starkey, and D. B. Skillicorn. Effective algorithms for partitioning distributed programs. In *Proceedings of the 7th Ann. Int. Phoenix Conf. on Computers and Communications*. IEE, 1988.

[FF86] G. C. Fox and W. Furmansky. Load Balancing by a Neural Network. Caltech Report C^3P363, California Institute of Technology, 1986.

[FKW87] G. Fox, A. Kolawa, and R. Williams. The Implementation of a Dynamic Load Balancer. In M. T. Heath, editor, *Proceedings of 2nd Conf. on Hypercube Multiprocessors*, page 114, Philadelphia, 1987. SIAM.

[GBW92] M. Günter, J. E. Boillat, and K. Wyler. Prozess-Mapping in rekonfigurierbaren Multiprozessorsystemen mit genetischen Algorithmen. Technical Report IAM-PR-91416, Institut für Informatik, Universität Bern, 1992.

[Koh88] T. Kohonen. *Self-Organization and Associative Memory*, volume 8 of *Springer Series in Information Sciences*. Springer Verlag, Berlin, 2nd edition, 1988.

[LPS91] W. Liebert, K. Pawelzik, and H. G. Schuster. Optimal Embedding of Chaotic Attractors from Topological Considerations. *Europhysics Letter*, 14(6):521–526, 1991.

[MGSK87] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer. New Solution to the Mapping Problem of Parallel Systems: the Evolution Approach. *Parallel Computing*, 4:269–279, 1987.

[MT91] T. Muntean and E.-G. Talbi. A New Approach to the Mapping Problem: a Parallel Genetic Algorithm. In *Proceedings of the Symposium on High Performance Computing*, Montpellier, October 1991.

[Nor89] M. Norman. A Genetic Approach to Topology Optimization for Multiprozessor Architectures. University of Edinburgh, Physics Dept., preliminary version, 1989.

[Sch90] P. Schmid. The Mapping Problem: A Neural Network Approach. In *Proceedings of the International Neural Network Conference INNC 90, Paris, July 9–13, 1990*, volume 1, pages 274–277, Dordrecht, 1990. INNS, Kluwer Academic Publ.

[She88] H. Shen. Self-Adjusting Mapping: A Heuristic Mapping Algorithm for Mapping Parallel Programs onto Transputer Networks. In J. Wexler, editor, *Developing Transputer Applications*, pages 89–98, Amsterdam, 1988. IOS.

[WBK88] K. Wyler, J. E. Boillat, and P. G. Kropf. Evolutionäre Algorithmen. Technical Report IAM-PR-87205, Institut für Informatik, Universität Bern, 1988.