Salmon J. 1990. *Parallel Hierarchical N-Body Methods*. Ph.D. Thesis, Caltech.

Saltz J., Berryman H. and Wu J. 1991. Multiprocessors and run-time compilation. *Concurrency Practice and Experience*, 3(6), 573-592.

Sears M.P. 1990. Linear algebra for dense matrices on a hypercube. *5th Distributed Memory Computing Conf.*, 317-320.

Shahookar K., and Mazumder P. 1991. VLSI Cell Placement Techniques. *ACM Computing Surveys*, Vol. 23, No. 2, June, 143-220.

Simon H. 1991. Partitioning of unstructured mesh problems for parallel processing. *Conf. Parallel Methods on Large Scale Structural Analysis and Physics Applications*, Permagon Press.

Spiessens P., and Manderick B. 1991. A massively parallel genetic algorithm. *Int. Conf. on Genetic Algorithms*, 279-287.

Syswerda G. 1989. Uniform crossover in genetic algorithms. *Int. Conf. on Genetic Algorithms*, 2-9.

Tanese R. 1989. Distributed genetic algorithms. *Int. Conf. on Genetic Algorithm*s, 434-440.

Vaughn C. 1991. Structural analysis on massively parallel computers. *Conf. Parallel Methods on Large Scale Structural Analysis and Physics Applications*, Permagon Press.

Walker D. 1990. Characterizing the parallel performance of a large-scale, particle-in-cell plasma simulation code. *Concurrency Practice and Experience*, Dec., 257-288.

Whitley D. 1989. The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best. *Int. Conf. on Genetic Algorithms*, 116-123.

Williams R.D. 1986. Minimization by simulated annealing: is detailed balance necessary?. Caltech C3P-354.

Williams R.D. 1991. Performance of dynamic load balancing algorithms for unstructured mesh calculations. *Concurrency Practice and Experience*, 3(5), 457-481.

Wright S. 1943. Isolation by distance. Genetics 28 : 114, March, 114-137.

Wright S. 1977. *Evolution and the Genetics of Populations*. Vol. 3, Univ. of Chicago Press.

Xu J., and Hwang K. 1990. Simulated annealing method for mapping production systems onto multicomputers. *IEEE Conf. AI Applic.*, 350-356.

Koller J. 1989. The MOOS II operating system and dynamic load balancing. *4th Conf. Hypercube Concurrent Computers, and Applications*, 599-602.

Kosak C., Marks J., and Shieber S. 1991. A parallel genetic algorithm for network-diagram layout. *Int. Conf. on Genetic Algorithms*, July, 458-465.

Kramer O., and Muhlenbein H. 1989. Mapping strategies in message-based multiprocessor systems. *Parallel Computing* 9, 213-225.

Laszewski G. 1991. Intelligent structural operators for the k-way graph partitioning problem. *Int. Conf. on Genetic Algorithms*, July, 45-52.

Lee S-Y, and Aggarwal J.K. 1987. A mapping strategy for parallel processing. *IEEE Trans. on Computers*, Vol. C-36, No.4, April, 433-442.

McCurley K., and Plimpton S. 1992. First impressions of our iPSC/860. *MPCRL Connection*, Sandia Laboratory, March.

Morison R., and Otto S. 1987. The scattered decomposition for finite elements*. J. Scientific Computing*, Vol. 2, No. 1, March, 59-76.

Motteler H. 1990. Occam configuration as a task assignment problem. *Transputer Res. Applic. 4*, editor D. L. Fielding, 244-250, IOS Press.

Muhlenbein H., Gorges-Schleuter M., and Kramer O. 1987. New solutions to the mapping problem of parallel systems: the evolution approach. *Parallel Computing* 4, 269-279.

Muhlenbein H. 1989. Parallel genetic algorithms, population genetics, and combinatorial optimization. *Int. Conf. on Genetic Algorithms*, 416-421.

Nolting S. 1991. Nonlinear adaptive finite element systems on distributed memory computers. *European Distributed Memory Computing Conference*, April, 283-293.

Otten R.H.J.M., and van Ginneken L.P.P.P. 1989. *The Annealing Algorithm*. Kluwer Academic.

Pettey C., Leuze M., and Grefenstette J. 1987. A parallel genetic algorithm*, Int. Conf. on Genetic Algorithms*, 155-161.

Ponnusamy R., Saltz J., Das R., Koelbel C., and Choudhary A. 1992. A runtime data mapping scheme for irregular problems. *Scalable High Performance Computing Conference*, May, 216-219.

Pothen A., Simon H., and Liou K-P. 1990. Partitioning sparse matrices with eigenvectors of graphs*. SIAM J. Matrix Anal. Appl*., 11, 3 (July), 430-452.

Ramanathan J., and Ni L. 1989. The mapping problem: in the context of new communication paradigms in multicomputers. *4th Conf. Hypercube Concurrent Computers, and Applications*, 781-784.

Robertson G. 1987. Parallel implementation of genetic algorithms in a classifier system. In *Genetic Algorithms and Simulated Annealing*, editor L. Davis, Morgan Kaufmann, 129-140.

Roussel-Ragot P., Kouicem N., and Dreyfus G. 1991. Error-free parallel implementation of simulated annealing. *Parallel Problem Solving from Nature*, October 1990, editors Schwefel H.P. and Manner R.. Lecture Notes in Computer Science 496, Springer-Verlag.

Sadayappan P., and Ercal F. 1987. Nearest-neighbor mapping of finite element graphs onto processor meshes. *IEEE Trans. on Computers*, vol. C-36, no. 12, Dec., 1408-1424.

Sadayappan P., Ercal F., and Ramanujam J. 1989. Parallel graph partitioning on a hypercube. *4th Conf. Hypercube Concurrent Computers, and Applications*, 67-70.

Fox G.C., and Furmanski W. 1988. Load balancing loosely synchronous problems with a neural network. *3rd Conf. Hypercube Concurrent Computers, and Applications*, 241-278.

Fox G.C., Johnson M., Lyzenga G., Otto S., Salmon J., and Walker D. 1988. *Solving Problems on Concurrent Processors*. Prentice Hall.

Fox G.C., Furmanski W., Koller J., and Simic P. 1989. Physical optimization and load balancing algorithms. *4th Conf. Hypercube Concurrent Computers, and Applications*, 591-594.

Fox G.C., Hiranandani S., Kennedy K., Koelbel C., Kremer U., Tseng C., and Wu M-Y. 1990. Fortran D language specification. Syracuse University. NPAC, SCCS-42.

Fox G.C. 1991a. The architecture of problems and portable parallel software systems. *Supercomputing'91*, also SCCS-78b, NPAC, Syracuse University.

Fox G.C. 1991b. Physical computation. *Concurrency Practice and Experience*, Dec., 627-654.

Fox G.C. 1991c. Achievements and prospects for parallel computing. *Concurrency Practice and Experience*, Dec., 725-740.

Garey M.R., and Johnson D.S. 1979. *Computers and Intractability.* Freeman.

Goldberg D.E. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.

Goldsmith J., and Salmon J. 1987. Automatic creation of object hierarchies for ray traces. *IEEE CG&A*, May, 14-20.

Gorges-Schleuter M. 1989. ASPARAGOS: An asynchronous parallel genetic strategy. *Int. Conf. on Genetic Algorithm*s, 422-427.

Greening D.R. 1990. Parallel simulated annealing techniques. *Physica D* 42, 293-306.

Grefenstette J.J. 1986. Optimization of control parameters for genetic algorithms. *IEEE Trans. on Systems, Man, and Cybernetics*, 16(1), Jan-Feb, 122-128.

Grefenstette J.J. 1987. Incorporating Problem Specific Knowledge into Genetic Algorithms. In *Genetic Algorithms and Simulated Annealing*, editor L. Davis, 42-60, Morgan Kaufmann.

Hartl D.L., and Clark A. 1989. *Principles of Population Genetics*. Sinauer Associates.

Hey A.J.G. 1990. Concurrent supercomputing in Europe. *5th Distributed Memory Computing Conf.*, 639-646.

Holland J.H. 1975. *Adaptation in Natural and Artificial Systems*. Univ. of Michigan Press.

Hopfield J.J., and Tank D.W. 1986. Computing with neural circuits: a model. *Science* 233, 625-639.

Houstis E.N., Rice J.R., Chrisochoides N.P., Karathonases H.C., Papachiou P.N., Samartzis M.K., Vavalis E.A. , Wang K.Y., and Weerawarana S. 1990. //ELLPACK: A numerical simulation programming environment for parallel MIMD machines. *Int. Conf. on Supercomputing*, 3-23, ACM Press.

Hwang K., and Xu J. 1990. Mapping Partitioned Program Modules onto Multicomputer Nodes Using Simulated Annealing. *Int. Conf. Parallel Processing*, Vol. II, 292-293.

Ibaraki T., and Katoh N. 1988. *Resource Allocation Problems*. MIT Press.

Kirkpatrick S., Gelatt C.D., and Vecchi M.P. 1983. Optimization by simulated annealing. *Science* 220, 671-680.

Kernighan B., and Lin S. 1970. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, Vol. 49, No. 2, 291-308.

rithms, 249-256.

Cormen T., Leiserson C., and Rivest R. 1990. *Introduction to Algorithms*. McGraw Hill.

Crow J. F. 1986. *Basic Concepts in Population, Quantitative, and Evolutionary Genetics*. Freeman.

Das R., Ponnusamy R., Saltz J., and Mavripilis D. 1991. Distributed memory compiler methods for irregular problems - data copy reuse and runtime partitioning. ICASE Report No. 91-73.

Das R., Mavripilis D., Saltz J., Gupta S., and Ponnusamy R. 1992. The design and implementation of a parallel unstructured Euler solver using software primitives. *AIAA Aerospace Sciences Meeting*, January.

Davis L. 1989. Adapting operator probabilities on genetic algorithms, *Int. Conf. on Genetic Algorithms*, 61-69.

Davis L. 1990. *Handbook of Genetic Algorithms*. Nostrand Reinhold.

Deb K., and Goldberg D.E. 1989. An Investigation of niche and species formation in genetic function optimization. *Int. Conf. on Genetic Algorithm*s, 42-50.

DeJong K.A. 1975. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. Doctoral Dissertation, University of Michigan.

De Keyser J., and Roose D. 1991. A software tool for load balanced adaptive multiple grids on distributed memory computers. *Sixth Distributed Memory Computing Conference*, April, 122-128.

Dragon K., and Gustafson J. 1989. A low cost hypercube load-balance algorithm. *4th Conf. Hypercube Concurrent Computers, and Applications*, 583-590.

Duncan R. 1990. A survey of parallel computer architectures. *IEEE Computer*, Feb., 5-16.

Eglese R. W. 1990. Simulated annealing: a tool for operational research. *Euro. J. Operational Research*, 46, 271-281.

Ercal F. 1988. *Heuristic Approaches To Task Allocation For Parallel Computing*. Ph.D. thesis, Ohio State University.

Eshelman L.J., Caruana R.A., and Schaffer J.D. 1989. Biases in the crossover landscape. *Int. Conf. on Genetic Algorithms*, 10-19.

Farhat C. 1988. A simple and efficient automatic FEM domain decomposer. *Computers and Structures*. Vol. 28, no. 5, 579-602.

Farhat C. 1989. On the mapping of massively parallel processors onto finite element graphs. *Computers and Structures*. Vol. 32, no. 2, 347-353.

Fiduccia C., and Mattheyses R. 1982. A linear time heuristic for improving network partitions. *19th Design Automation Conf.*, 175-181.

Flower J., Otto S., and Salama M. 1987. A preprocessor for finite element problems. *Symp. Parallel Computations and their Impact on Mechanics*. ASME Winter Meeting (Dec.).

Fox G.C. 1984. Square matrix decomposition- symmetric, local, scattered. Caltech Report C3P-97.

Fox G.C., Kolawa A., and Williams R. 1987. The implementation of a dynamic load balancer. *2nd Conf. Hypercube Multiprocessors,* ed. Heath, 114-121.

Fox G.C. 1988a. A review of automatic load balancing and decomposition methods for the hypercube. In *Numerical Algorithms for Modern Parallel Computers,* ed. M. Schultz, 63-76, Springer-Verlag.

Fox G.C. 1988b. A graphical approach to load balancing and sparse matrix vector multiplication on the hypercube. In *Numerical Algorithms for Modern Parallel Computers,* ed. M. Schultz, Springer-Verlag.

References

Baker J.E. 1985. Adaptive selection methods for genetic algorithms. *Int. Conf. on Genetic Algorithms*, 101-111.

Baker J.E. 1987. Reducing bias and efficiency in the selection algorithm. *Int. Conf. on Genetic Algorithms*, 14-21.

Baiardi F., and Orlando S. 1989. Strategies for a massively parallel implementation of simulated annealing. *Lecture Notes in Computer Science* 366, 273-287, Springer-Verlag.

Banerjee P., Jones M.H., and Sargent J. 1990. Parallel simulated annealing for cell placement on hypercube multiprocessors. *IEEE Trans. on Parallel and Distributed Systems*, Vol. 1, No. 1, Jan., 91-106.

Berger M., and Bokhari S. 1987. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Trans. Computers*, C-36, 5 (May), 570-580.

Berrendorf R., and Helin J. 1992. Evaluating the basic performance of the Intel iPSC/860 parallel computer. *Concurrency: Practice and Experience*, May.

Bokhari S. 1981. On the mapping problem. *IEEE Trans. on Computers*, Vol. C-30, No. 3, March, 207-214.

Bokhari S. 1990a. Communication overhead on the INTEL iPSC-860 hypercube. ICASE Report no. 90-10.

Bokhari S. 1990b. A network flow model for load balancing in circuit-switched multicomputers. ICASE Report no. 90-38.

Bomans L., and Roose D. 1989. Benchmarking the iPSC/2 hypercube multiprocessor. *Concurrency: Practice and Experience*, Sept., 1-18.

Booker L. 1987. Improving search in genetic algorithms. In *Genetic Algorithms and Simulated Annealing*, editor L. Davis, 61-73, Morgan Kaufmann Publishers.

Bozkus Z., Ranka S., and Fox G.C. 1992. Benchmarking the CM-5 multicomputer. To appear in *Frontiers'92*.

Byun H., Kortesis S.K., and Houstis E.N. 1992. A workload partitioning strategy for PDEs by a generalized neural network. Purdue University, Computer Science, Technical Report CSD-TR-92-015.

Choudhary A., Fox G.C., Hiranandani S., Kennedy K., Koelbel C., Ranka S., and Saltz J. 1992. A classification of irregular loosely synchronous problems and their support in scalable parallel software systems. *DARPA Software Technology Conf.*, April, 138-149.

Chrisochoides N.P., Houstis C.E., Houstis E. N., Kortesis S.K., and Rice J.R. 1989. Automatic load balanced partitioning strategies for PDE computations. *Int. Conf. on Supercomputing*, Greece, 99-107, ACM Press.

Chrisochoides N.P., Houstis E.N., and Houstis C.E. 1991a. Geometry based mapping strategies for PDE computations. *Int. Conf. on Supercomputing*, 115-127, ACM Press.

Chrisochoides N.P., Houstis C.E., Houstis E. N., Papachiou P.N., Kortesis S.K., and Rice J.R. 1991b. Domain decomposer. In *Domain Decomposition Methods for Partial Differential Equations*, editors R. Glowinski et al. SIAM Publication.

Cohoon J.P., Hedge S.U., and Martin W.N. 1991. Distributed genetic algorithms for the floorplan problem. *IEEE Trans. CAD*, , Vol. 10, No. 4, April, 483-491.

Collins R.J. and Jefferson D. R. 1991. Selection in massively parallel genetic algorithms. *Int. Conf. on Genetic Algo-*

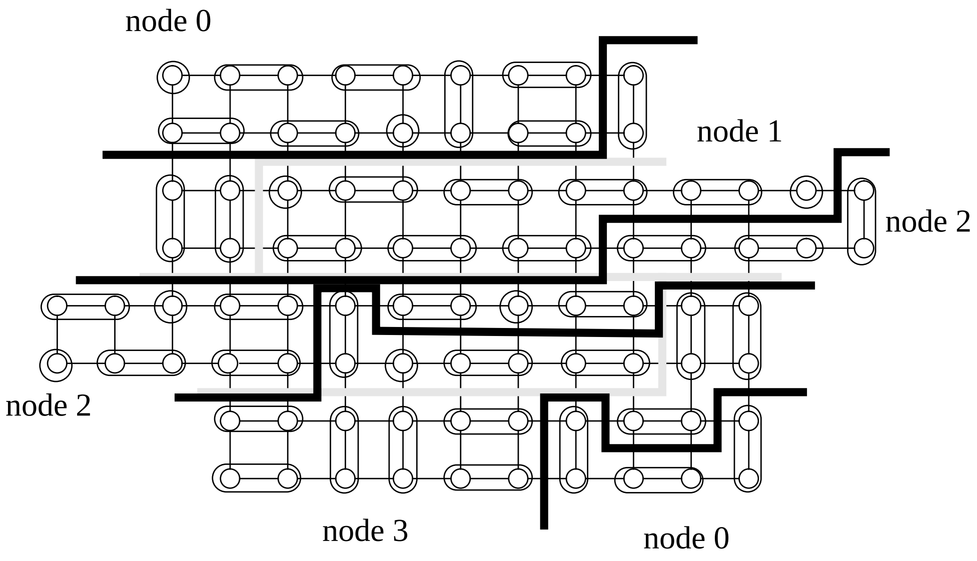


Figure C.2. New node boundaries due to repartitioning in PGC3, with stride 6.

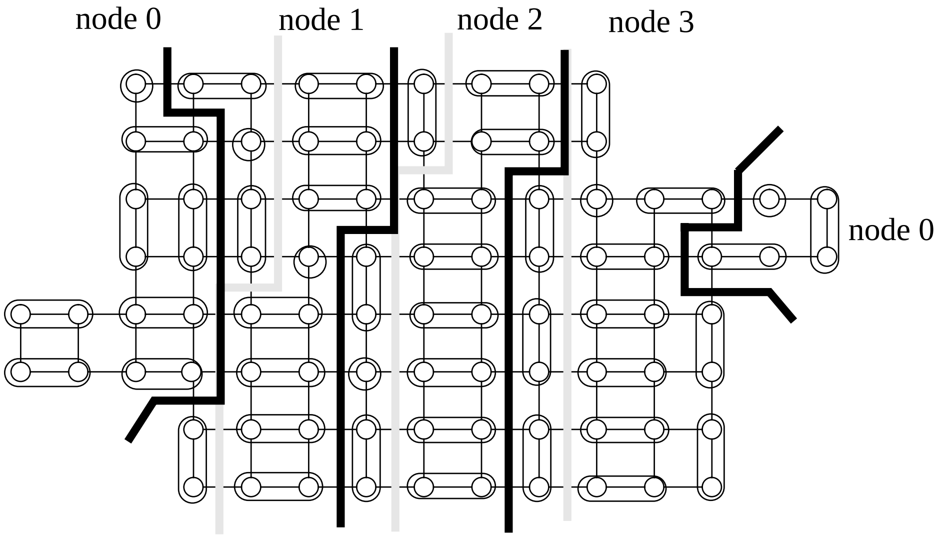


Figure C.3. Repartitioning in PGC4, with stride 6.

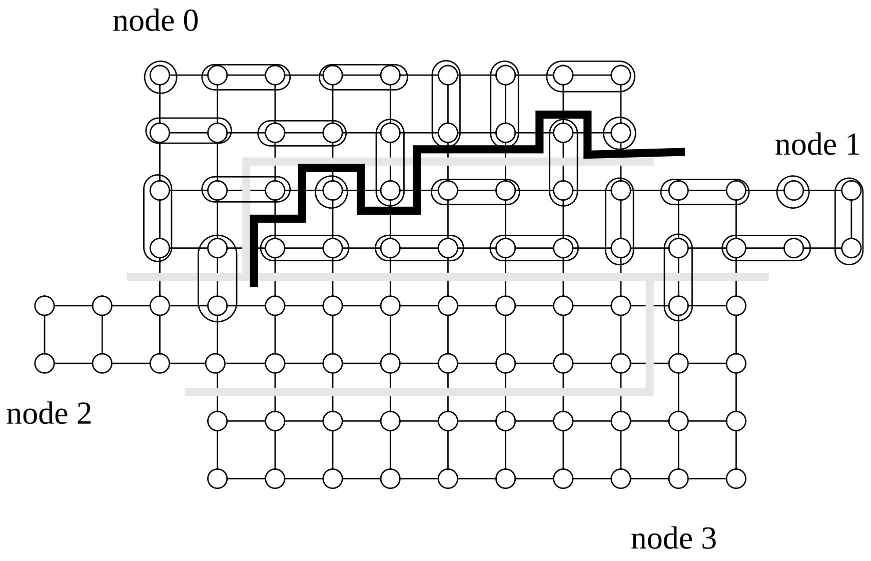


Figure C.1. New node boundaries due to remapping in PGC2.
(only 0-1 boundary shown)

PGC3 performs SGC independently in all nodes without conflicts, similarly to PGC1. Then, it attempts to prevent a portion of the crossedges from being permanently excluded from the contraction pool. This is accomplished by repartitioning the graph after each contraction step in the same “naive” fashion described for PGC1, but now with a certain stride. This amounts to shifting the node boundaries with respect to the graph edges, which creates new crossedges and makes a number of current crossedges local. Consequently, the edges which are excluded from the contraction pool in one iteration become subjected to SGC’s operation in the following iterations. This is illustrated in Figure C.2. PGC3 is an improvement over PGC1. But, it might not yield significant improvements for all graphs.

PGC4 is the same as PGC3 but makes use of the spatial coordinates of the vertices, if available. It assumes that the graph is partitioned such that the vertices in each subgraph lie within a certain range of values of the x-coordinate, for e.g., and that the sizes of the subgraphs are as equal as possible. Then, after each contraction step, repartitioning is done based on the x-coordinates instead of the vertex numbers. This is illustrated in Figure C.3. PGC4 is more expensive than PGC3; however, it would produce better solutions.

```
$\Delta\zeta = \Delta E_b = 0$;
for $i$ = 1 to $\theta(v)$ do
    Save processor, MAP[$v_i$], in list $L_p$ without duplication;
    if MAP[$v_i$] $\neq p1$ then decrement $\Delta E_b$ and $E_b(p1, L_p(i))$;
    if MAP[$v_i$] $\neq p2$ then increment $\Delta E_b$ and $E_b(p1, L_p(i))$;
endfor
for $i$ = 1 to length($L_p$) do
    if there is a change in $E_b(p1, L_p(i))$ from 0 to +ve then (also for $p2$)
        $\Delta\zeta = \Delta\zeta + \sigma + \tau\ H(p1, L_p(i))$; (also for $p2$)
    if there is a change in $E_b(p1, L_p(i))$ from +ve to 0 then (also for $p2$)
        $\Delta\zeta = \Delta\zeta - \sigma - \tau\ H(p1, L_p(i))$; (also for $p2$)
endfor
$\Delta\zeta = 2\ (\Delta\zeta + \rho\ \Delta E_b)$;
```

Figure B.1. Computing $\Delta\zeta$ due to remapping vertex *v* from *p1* to *p2*.

Appendix C

Improved Parallel Graph Contraction Algorithms

In this appendix, we briefly describe the three parallel graph contraction algorithms mentioned at the end of Subsection 9.1.2.

PGC2 allows conflicts over non-local partner vertices and resolves these conflicts at random, for example. Then, it remaps the supervertices that extend across node boundaries so that each of these supervertices lie entirely in one node. An example is given in Figure C.1, showing only nodes 0 and 1. Remapping supervertices and their constituent vertices shifts node boundaries. Hence, it prevents the creation of supervertices that belong to more than one node and avoids the complications associated with expanding non-boundary information. To reduce the number of conflicts in PGC2, a global data structure maintaining "partnering" status information can be updated several times during each contraction step. PGC2 is not straight-forward to implement; but it would produce good quality contracted graphs.

$$= \frac{1}{4\sqrt{|V_C|}\,(\sqrt{|V_M|}-1)} \qquad \text{(A.5)}$$

Also, an estimate for

$$\text{optimal computation term} = 4\,\rho\sqrt{|V_C|}\,(\sqrt{|V_M|}-1) \qquad \text{(A.6)}$$

since *H(p,q)* = 1 for neareset-neighbor communication and $\zeta = C_d$. Equations (A.1), (A.2) and (A.6) can be used to derive μ:

$$\mu = \mu_{user}\,\frac{\lambda^2\sum\theta(v)}{4\rho|V_M|\sqrt{|V_C|}\,(\sqrt{|V_M|}-1)} \qquad \text{(A.7)}$$

where $\mu_{user} < 1$ is a user defined value expressing the importance of communication to computation.

OBJ_{opt} can be obtained by substituting equations (A.2), (A.6) and (A.7) into (A.1). EFF_{opt} can be estimated from the assunptions of G_C and (A.3) as:

$$EFF_{opt} = \frac{\lambda\sum\theta(v)}{\dfrac{\lambda\sum\theta(v)}{|V_M|} + 4\rho\sqrt{|V_C|/|V_M|}}$$

Appendix B

Computing $\Delta\zeta$

To compute $\Delta\zeta$, resulting from remapping vertex *v* from processor *p1* to *p2*, we need to compute ΔB. Since such remapping will be done extensively, the computation of $\Delta\zeta$ must be efficient. Unfortunately, ΔB is not as efficiently computable as desired due to the unsymmetry of the number of boundary vertices, *B(p,q)*. Thus, we approximate *B(p,q)* with crossedges, $E_b(p, q)$, by using a conversion parameter, $\Im_b$, which is the average number of crossedges per boundary vertex in a reasonable mapping configuration. That is, we compute ΔE_b and deduce $\Delta B = \Delta E_b/\Im_b$. For general computation graphs, $\Im_b$ is typically close to $\theta_{av}/2$. However, $\Im_b$ = 1 for rectangular grid based graphs. Figure B.1 outlines how to compute $\Delta\zeta$ for $\zeta = C_p$.

Appendix A

Estimates for μ, Ψ_{max}, OBJ_{opt} and EFF_{opt}

In this appendix, crude estimates are given for the parameters, μ, Ψ_{max}, OBJ_{opt} and EFF_{opt}. These estimates are based on simplifying assumptions about the computation graph and the assumption that an optimal mapping configuration involves communication only between physically adjacent processors. Although the estimates given here are very inaccurate, they serve a purpose in the design of some of the algorithms, where accuracy is not needed.

The main assumption is that an optimal mapping of a computation graph, G_C, to $|V_M|$ processors can be constructed by considering G_C a 2-D rectangular graph to which the classic rectangular decomposition can be applied. That is, G_C can be decomposed into $|V_M|$ subdomains with nearest-neighbor subdomain interfaces. Each subdomain has $|V_C|/|V_M|$ vertices. For large grain sizes, $S_v(p)$ would almost be equal for all p = 0 to $|V_M|$-1. Thus,

$$\text{optimal } S_v(p) = \frac{\sum_{v \in |V_C|} \theta(v)}{|V_M|}.$$

OBJ_{opt} is an estimate of optimal OF_{appr}. That is,

$$OBJ_{opt} = \text{optimal } \lambda^2 \sum_{p \in V_M} S_v^2(p) + \mu \text{ optimal} \sum_{p \in V_M} \zeta(p) \qquad \text{(A.1)}$$

where,

$$\text{optimal computation term} = \frac{\lambda^2}{|V_M|} \sum_{p \in V_M} S_v^2(p)$$

$$= \frac{\lambda^2}{|V_M|} \sum_{v \in V_C} \theta(v) \qquad \text{(A.2)}$$

An estimate for the optimal number of boundary vertices of the subgraphs mapped to processors can be assumed to be

$$l_b = 4 \sqrt{|V_C| / |V_M|} \qquad \text{(A.3)}$$

Thus, the total number of vertices that lie on interprocessor boundaries are

$$L_b = |V_M| \; l_b - 4 \sqrt{|V_C|} \qquad \text{(A.4)}$$

assuming that a rectangular G_C has 4 $\sqrt{|V_C|}$ vertices at its physical boundary. Therefore, an estimate for the maximum degree of clustering is

$$\Psi_{max} = \frac{1}{L_b}$$

ular multiphase computations would be interesting. Multiphase computations would involve different computation graphs in different phases; examples are unstructured multigrid computations and particle-in-cell problems [Choudhary et al. 92]. The mapping time might be more important here than single phase computations for deciding which mapping algorithms to use.

Fifthly, the use of the PO algorithms for dynamically varying irregular computations is challenging. Examples of such computations are adaptive solvers, particle dynamics algorithms and mesh generation. The PO algorithms are promising for adiabatic problems, i.e. slowly varying problems [Fox et al 87; Williams 91]. Some of the issues involved for this class of problems are how acceptable is the ratio of the remapping time to the solver's time and how PNN and ab initio algorithms compare with PGA and PSA, being incremental mapping algorithms, in terms of time and quality.

Sixthly, the application of the PO algorithms to other mapping problems is another research task [Hwang and Xu 90; Motteler 90; Xu and Hwang 90]; interesting examples of such problems are mapping heterogenous problems, mapping large problems to distributed-memory SIMD multiprocessors, such as Maspar and CM-2, and mapping problems to heterogenous multiprocessor systems. Finally, the application of the MIMD and SIMD PGAs to other optimization problems is another area for further research.

differences in execution time tend to decrease as the problem size or the multiprocessor size increases. PGA is consistently the slowest, followed by PSA and then PNN. The ratios of the mapping times of the three algorithms may decrease for larger multiprocessors.

For large problems, the use of the proposed pre-mapping graph contraction algorithms yields remarkable results; the good quality of the mapping solutions of the PO algorithms is maintained while enjoying a considerable decrease in mapping time. This time reduction makes the application of the PO algorithms to large problems feasible and allows the mapping step itself to be an efficient and scalable operation. Therefore, the use of graph contraction is imperative for large problems.

The choice among the three mapping algorithms should be application dependent. With large problems and graph contraction, the differences in mapping quality and time for the three PO algorithms are reduced. Nevertheless, their performance order relative to each other remains the same. Another consideration is scalability; PSA and PNN are scalable, with or without graph contraction. With graph contraction PGA's scalability becomes restrained by the multiprocessor size, and not the data size. Therefore, the criteria for choosing a mapping algorithm are whether we can afford to spend more time on mapping to save the problem solver's time and how large the multiprocessor is. However, the hardware realizability of PNN and its good performance, makes it an appealing choice.

Based on the work presented in this dissertation, a number of research tasks can be pursued. Firstly, the execution time of the parallel PO algorithms can be reduced by hybridizing these algorithms in a similar way to the sequential case described in Chapter 4. The mapping quality is expected to be preserved despite the decrease in the mapping time. Secondly, the parallel graph contraction algorithm described in Chapter 9 needs to be improved to produce higher quality contracted graphs. For this purpose, the algorithms outlined in Appendix C can be explored.

Thirdly, the PO algorithms need to be interfaced with other components of a parallel programming system, such as the Fortran D system. These components should generate computation graphs and link the mapping solution to the problem solver. Components of the PARTI system [Saltz et al. 91] are good candidates. For a portable integration of the PO algorithms into a programming system, they have to fit several multiprocessor architectures. This can be accomplished by incorporating a number of modules into the PO algorithms, where each module is associated with a suitable objective function for standard target multiprocessor topologies. Portability for PNN is not as straight forward as it is for PGA and PSA, since its current formulation is geared to the hypercube topology.

Fourthly, the PO algorithms, in their current form, are suitable for deployment into applications with static single phase irregular computations and high connectivity computation graphs. Examples of such computations are sparse matrix-vector multiplication and sparse conjugate gradient or Euler solvers for unstructured finite-element meshes. The application of the PO algorithms to irreg-

Chapter 10

Conclusions and Further Work

Parallel physical optimization algorithms, based on genetic algorithms, simulated annealing and neural networks, for mapping irregular data sets to distributed-memory multiprocessors have been presented. The performances of the these algorithms have been critically evaluated and compared using test cases that involve different computation graph configurations and a variety of algorithm and machine characteristics. Further, data sets with small, moderate and large sizes have been used. For large problems, sequential and parallel pre-mapping graph contraction algorithms have been proposed and their advantages have been shown. Sequential physical optimization algorithms, which lay the foundation for the parallel algorithms, have also been described and compared.

The three PO algorithms produce high quality mapping solutions and are shown to be competitive with RSB, a good heuristic method. The PO algorithms, especially PGA and PSA, are not restricted by special assumptions about the structure or homogeneity of the computation graph or about the architecture and topology of the multiprocessor. They can adapt to different conditions by modifying the objective function. Further, they are fairly robust with respect to their design parameters. Therefore, they do not have a bias towards particular conditions, and the high quality of their results exhibited for the test cases considered therein is expected to extend to various data and multiprocessor configurations. This property of general applicability makes the PO mapping algorithms suitable for integration into automatic parallelization systems. It is important, however, to ensure that the objective function, guiding the operation of the PO algorithms, has the following properties: its incremental change is efficiently computable, somewhat smooth, has the locality property, and includes appropriate machine and algorithm parameter values.

The advantages of the PO algorithms for the mapping quality are more salient when the communication to computation ratio is not small, due to larger multiprocessor size, a particular algorithm, or higher values of the communication parameters of the multiprocessor. The comparison among the PO algorithms themselves show that PGA yield the best mapping solutions, followed by PSA and then PNN. Evidently, PGA has better adaptability to different conditions. Further, it enjoys easier parallel implementation than PSA and PNN.

The infrequent nearest-neighbor communication scheme in PGA leads to near-perfect speed-up. The adaptive communication schemes in PSA and PNN are useful for limiting the cost of communication. Nevertheless, the PO algorithms are slower than heuristic algorithms. Interestingly, the

9.4. Concluding remarks

All the results discussed above show that SGC and PGC1 are powerful heuristics to accompany data mapping. They lead to considerable reductions in the execution time of the mapping algorithms, while maintaining good sub-optimal mapping qualities. The time reduction is larger for larger problems, because with graph contraction, time is determined by $|V_M|$ and κ, not by $|V_C|$. This time reduction asserts and improves the scalability of the PO mapping algorithms. These findings suggest that the use of graph contraction is imperative for all known mapping algorithms, and definitely for the PO algorithms.

The contraction parameter κ, or χ, is a user-defined value, which determines the resultant mapping quality and time; the user's choice would depend on the specific application. Suitable values for κ are κ>10, for PGA, and κ>20 for PSA and PNN. However, it has been observed that doubling κ, above these values, usually leads to less than 10% improvement in mapping quality and up to 100% increase in mapping time. Hence, the user would double κ if the saving in the application time, due to the few percent improvement in mapping quality, outweighs the mapping time penalty.

PGA shows the best adaptability to the outputs of SGC and PGC1. Thus, it yields the best mapping quality, followed by PSA; PNN is the least adaptable. Also, PGA remains the slowest with graph contraction, followed by PSA, then PNN. Further, PGA's scalability improves with contraction since it becomes effectively restricted by the size of the multiprocessor, $|V_M|$, not that of the data set, $|V_C|$.

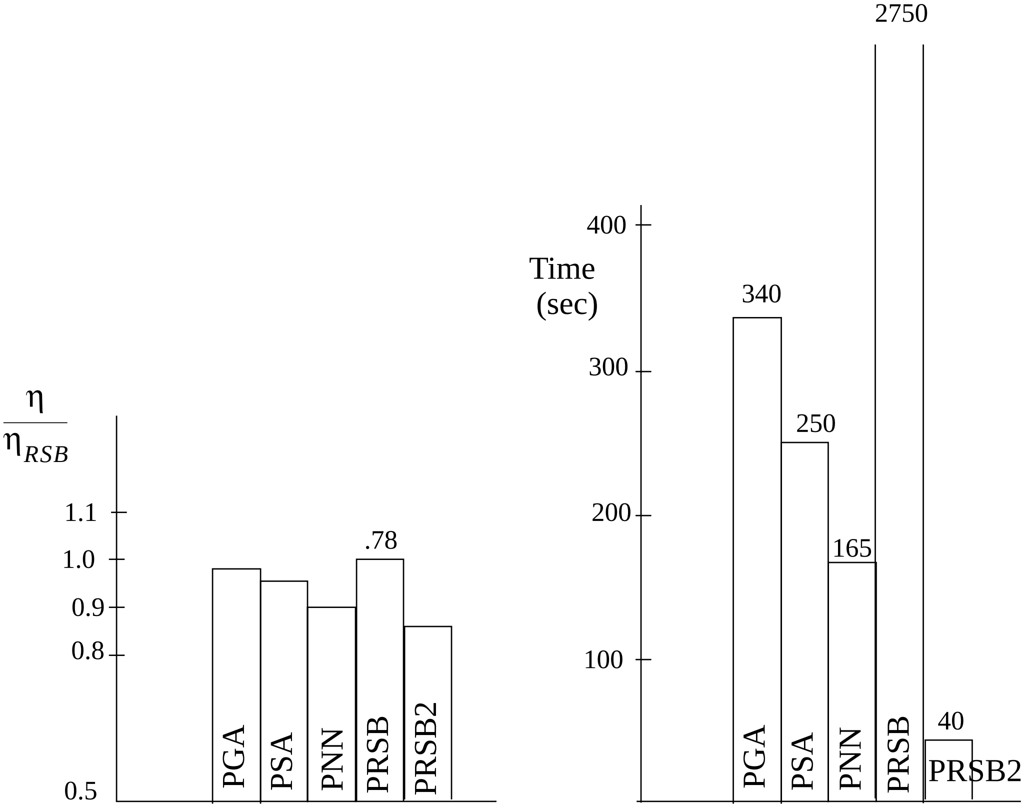


Figure 9.12. Solution quality and time for mapping FEMW(53961) with $|V_M|$=32, using SGC with κ=31, for PGA and κ=60, for PSA, PNN and RSB2.

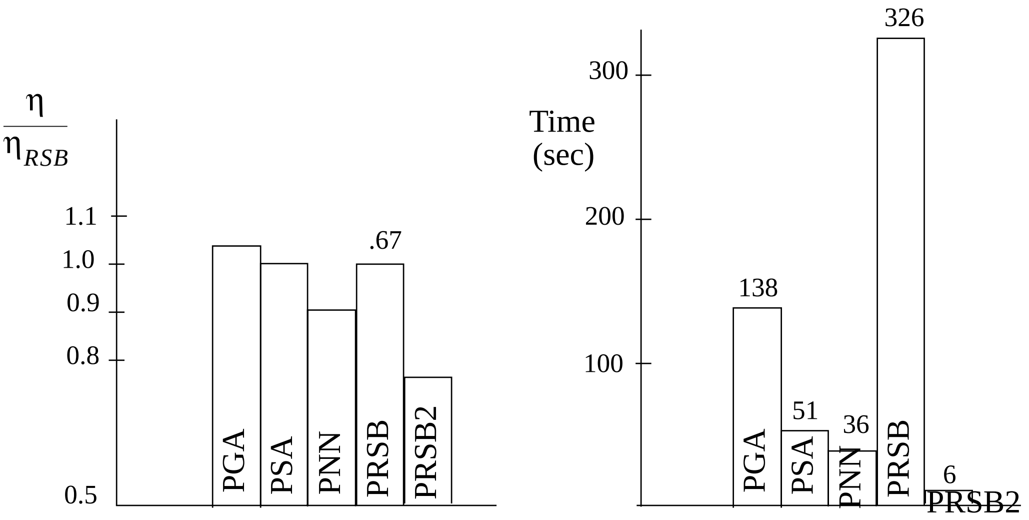


Figure 9.10. Solution quality and time for mapping FEMW(9428) with $|V_M|$=16, using PGC1 with κ=15, for PGA and κ=27, for PSA, PNN and PRSB2.

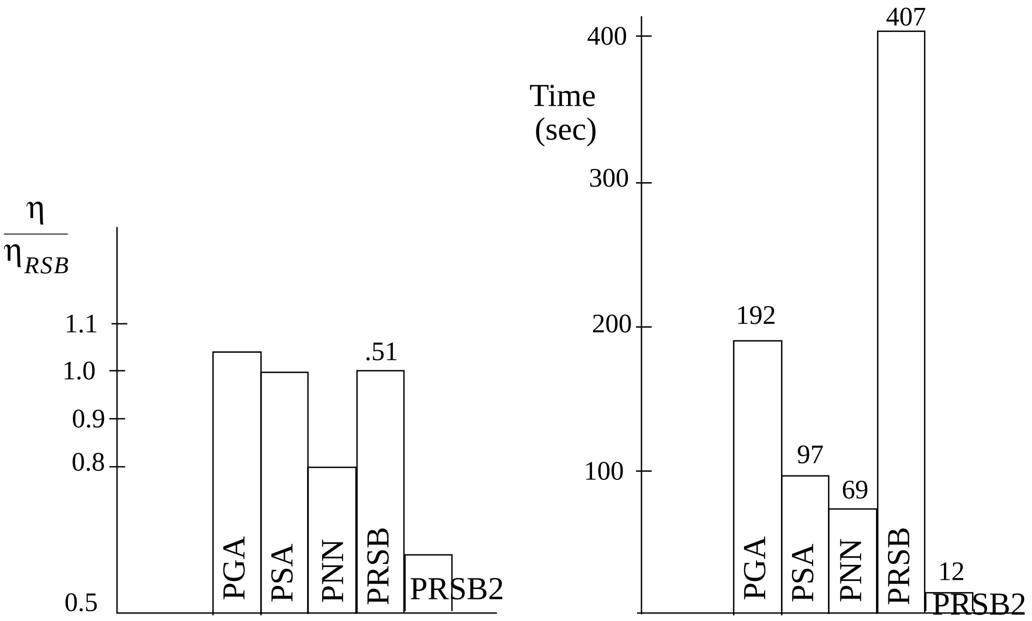


Figure 9.11. Solution quality and time for mapping FEMW(9428) with $|V_M|$=32, using PGC1 with κ=13, for PGA and κ=25, for PSA, PNN and PRSB2.

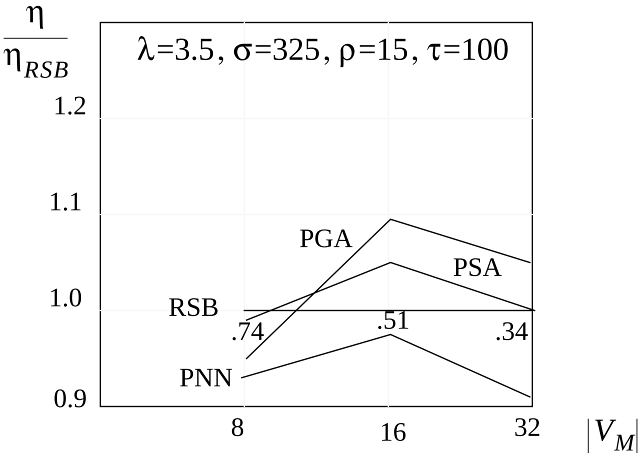


Figure 9.8. Solution quality for mapping FEMW(3681) using PGC1 with κ>20, for PSA and PNN, and κ>10, for PGA. (cf. Figure 8.18)

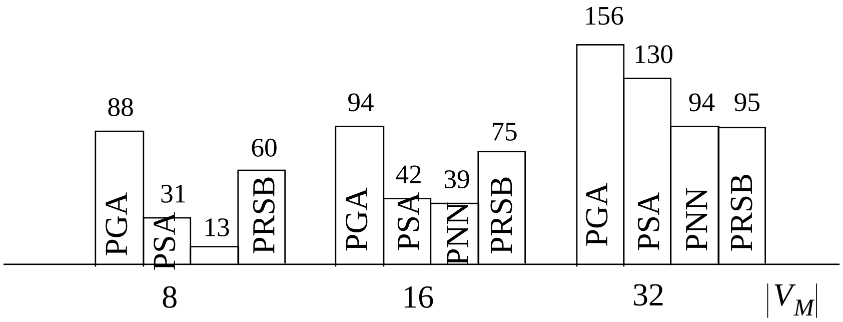


Figure 9.9. Mapping time, in seconds, corresponding to Figure 9.8. (cf. Figure 8.19)

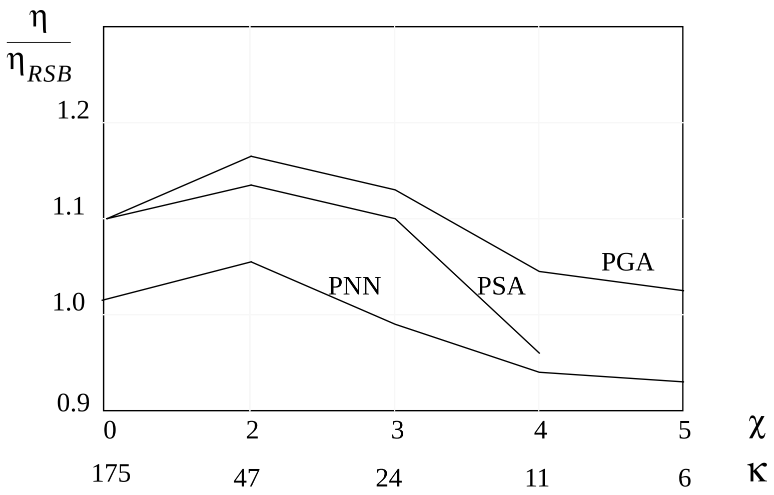


Figure 9.6. Effect of contraction parameters on the Solution quality for mapping FEMW(2800) with $|V_M|$ =16.

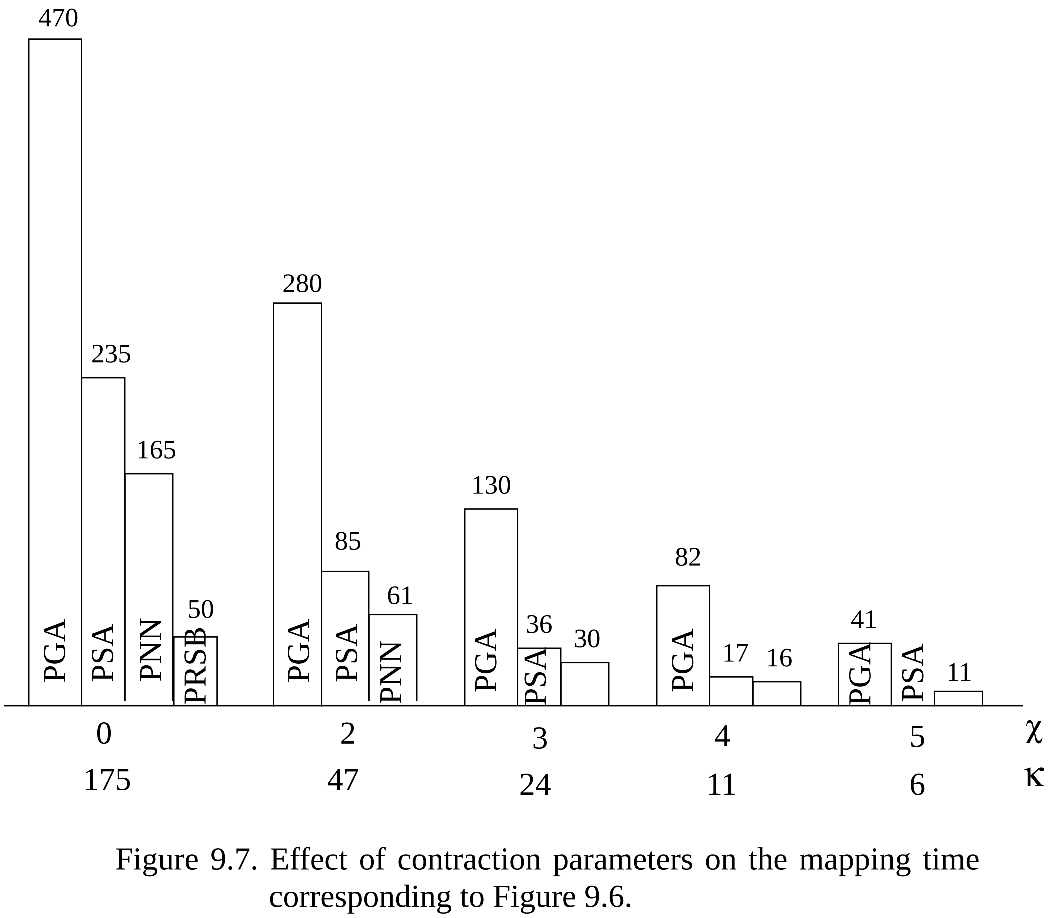


Figure 9.7. Effect of contraction parameters on the mapping time corresponding to Figure 9.6.

less than that of RSB, but not by a great deal, and the mapping time is extremely reduced. In fact, a similar conclusion also holds for PNN, where better contraction quality leads to better mapping quality. However, the results for FEMW(9428) indicate that RSB2, in its present form, would not be reliable in general to produce reasonable results, because its operation is sensitive to the output of the graph contraction algorithm and to the particular problem structure.

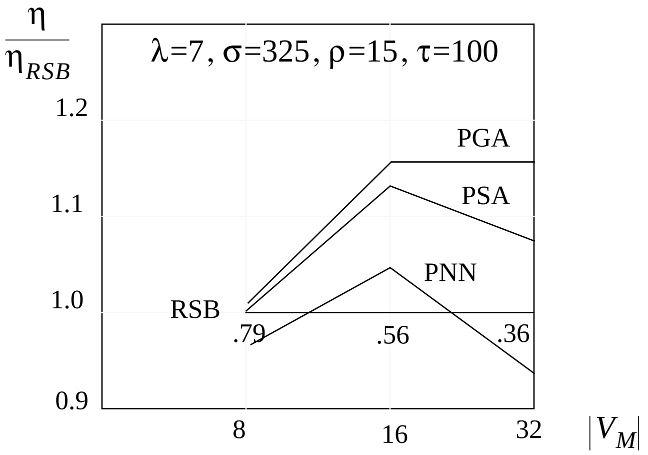


Figure 9.4. Solution quality for mapping FEMW(2800) using PGC1 with $\chi = 2$. (cf. Figure 8.16)

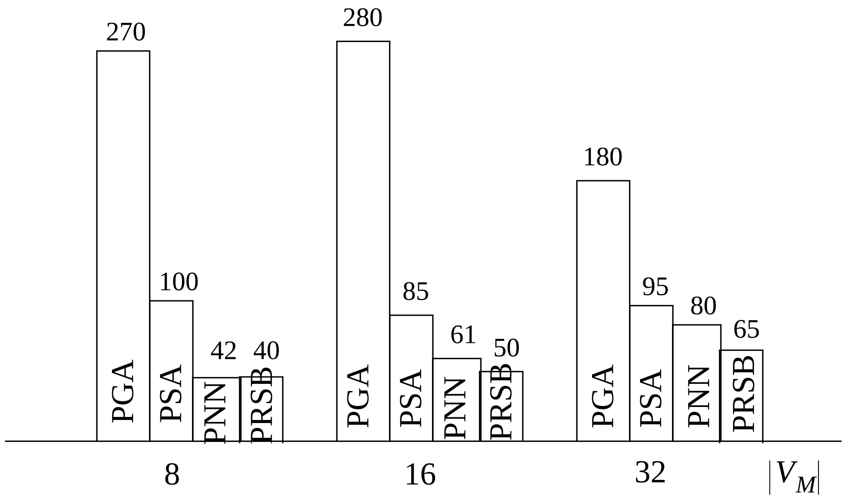


Figure 9.5. Mapping time, in seconds, corresponding to Figure 9.4. (cf. Figure 8.17)

should be κ>10 for PGA and κ>20 for PSA and PNN. Suitable κ values are bounded from below due to the non-optimality of any efficient graph contraction heuristics. For PSA and PNN, another important factor contributing to the lower bound of κ is that κ equals their own parallelization grain size. We have already found, in Chapters 6 and 7, that the grain size should not be small, otherwise PSA and PNN would produce unacceptable mapping solutions. For example, Figures 9.6 and 9.7 do not show a result for PSA for κ=6 because, under such conditions, PSA degenerates and fails to leave the initial random mapping state. On the other hand, κ is bounded from above by the requirement of achieving the biggest reduction possible in mapping time. For PGA, memory space requirements place a formidable upper bound on κ; κ should be chosen such that $(\theta_{av} + \frac{POP}{N_H}) |V_C|_\kappa$ is less than the available node-memory capacity.

Figures 9.8 and 9.9 show the mapping quality and time for FEMW(3681) using PGC1 with suitable κ values, as suggested above. These results support the assessment made about the advantages of using PGC1 for FEMW(2800). They also show that the proposed values for κ are suitable. The contraction time is also less than 2 seconds for FEMW(3681) and is, thus, negligible.

The results in Figures 9.6 through 9.9 indicate that PGA still yields slightly better solution quality and is still the slowest. However, they also show that for reasonable κ values, the difference in the mapping time of PSA and PNN has shrunk in comparison with the uncontracted cases. Further, contrary to the uncontracted case, this mapping time might be smaller than that for RSB, for a small quality decrease.

Figures 9.10, 9.11,and 9.12 show mapping quality and time for large data sets, FEMW(9428) and FEMW(53961), with κ values as prescribed above. PGC1 is used for FEMW(9428). But, sequential SGC is used for FEMW(53961), on the host, because the current implementation of PGC1 required too much memory space on 16 NCUBE nodes. The result for these realistic problems demonstrate the power of using graph contraction as a preprocessor to mapping algorithms, in general, and to the PO mapping algorithms, in particular. The contraction times are 6 seconds, 5 seconds and 129 seconds for the test cases whose results are shown in Figures 9.10, 9.11, and 9.12, respectively. The contraction time is clearly worthwhile incurring, since contraction leads to such an enormous saving in the execution time of the PO algorithms, without causing deterioration in mapping quality. It is clear from Figures 9.10 and 9.11 that PGA still yields slightly better solutions than those of RSB, PSA is comparable, and PNN's solutions are of lower quality, within 20%. But, the figures, also show an impressively smaller mapping time for FEMW(9428) in comparison with RSB. For the larger data set in Figure 9.12, FEMW(53961), the saving in mapping time is enormous for the PO algorithms. This time is substantially smaller than that of RSB, for a small cost in terms of mapping quality.

It can be seen that when the contraction quality is good, such as that produced by SGC for FEMW(53961), the results of RSB2 are surprisingly reasonable; its mapping quality is certainly tion (4.1) is now inaccurately based on “superspins” (associated with supervertices) instead of individual spins.

RSB can also make use of graph contraction to reduce mapping time. For RSB, the contracted graph is reconstructed as a simple graph with $\Theta(v_{ij}) = \theta(v_{ij})$ and $\xi(v_{ij}, v_{nm}) = 1$, i.e. by ignoring vertex and edge weights. After using RSB to map the simple graph the supervertices are unfolded and MAP[] is specified as in the case of PO algorithms. Such a version of RSB is henceforth referred to as RSB2. The operation of RSB2 on the simplified graph misses contraction information. Clearly, the quality of its mapping is sensitive to the contraction technique and the properties of the original computation graph.

9.3. Experimental results and discussion

The experiments described in this section deal with test cases used in Section 8.2, to compare mapping results with and without graph contraction. They also employ new test cases with large sizes, to show the performance of the mapping algorithms for realistic problems. The experimental setting is the same as described in Section 8.2.

Figures 9.4 and 9.5 show the solution quality and execution time of the PO algorithms for mapping FEMW(2800) using PGC1 with χ=2. The contraction time is less than 2 seconds, which is negligible with respect to the mapping time. Comparing these results with their contraction-free counterparts in Figures 8.16 and 8.17, it is clear that the better solution qualities of PGA and PSA are maintained and that the quality of PNN’s solutions decreases only by small amount. Also, a significant reduction in mapping time is evident in Figure 9.5. For the three values of $|V_M|$, the saving in execution time of PSA and PNN ranges from 45% to 65%. PGA does not show a similar saving because the decrease in its memory space demand, due to contraction, allowed the use of standard deme sizes, instead of just the smallest size of two, as used for Figure 8.17. The increase in deme size explains the small improvement in η_{PGA}.

Figures 9.6 and 9.7 show the effect of different contraction parameter values, χ and its reciprocal κ (equation 4.3), on mapping quality and time. As expected, quality decreases for smaller κ, the ratio of the size of the contracted graph being mapped to the multiprocessor size. However, this quality decrease is limited to about 10%, for κ>11, whereas the reduction in mapping time is remarkable, more that 90% for PSA and PNN. Further, for κ=24, for PSA and PNN and κ=11, for PGA, the differences between solution quality and time results for the PO algorithms and RSB become small. Therefore, Figures 9.6 and 9.7 demonstrate quite clearly the advantages of using PGC1 prior to mapping, even for a small data set like FEMW(2800). In addition, they point to suitable values of κ, and consequently χ, for PGC1, with which good mapping qualities are maintained whilst achieving remarkable reductions in mapping time. We estimate that these values

A number of parallel graph contraction strategies that yield better contracted graphs than those of PGC1 can be devised. Three such strategies, which vary in contraction quality and time and in implementation difficulty, lead to the following algorithms: PGC2 which is faithful to SGC to a large extent, PGC3 which is conflict-free but is not guaranteed to offer significant improvements for all graphs, and PGC4 which is the same as PGC3 but uses the spatial coordinates of the vertices for partitioning the graph among the N_H nodes. The three improved algorithms are discussed in Appendix C.

9.2. Mapping using graph contraction

In this section, we illustrate how the three parallel PO algorithms make use of pre-mapping graph contraction and describe some design modification. The operations of the parallel PO algorithms with graph contraction for large problems differ, in some ways, from that described in Chapter 4 for sequential PO algorithms on small problems. In this section, we concentrate on these differences.

All three parallel physical algorithms map the contracted graph first. Then, the mapped graph is decontracted in order to specify MAP[v], for $v = 0$ to $(|V_C| - 1)$. That is, a vertex, v, in the original graph G_C is mapped to the same processor as the supervertex it belongs to in G_C^{χ}. PSA is capable of multiscale operation, as described for SA in Chapter 4. After mapping the coarse-structure contracted graph, it evolves a mapping result by further PSA steps on the restored fine-structure graph. However, the results in Chapter 4 show that the increase in annealing time would be prohibitive relative to the improvement in solution quality. This is why, in our implementation, PSA performs coarse-structure mapping only. The initial temperature used for the coarse-structure system, with parameter χ, is χT_0, where T_0 is the initial temperature computed in a similar way to that for fine-structure mapping.

PGA is not capable of multiscale operation for large problems due to its demanding memory space. Further, it employs crossedges, $E_b(p, q)$ (see Section 2.1), as an approximation to $B(p, q)$ in fitness evaluation for contracted graphs, because it is not possible to compute $B(p, q)$ correctly with merged vertices. Thus, PGA loses some “accuracy” when performing coarse-structure mapping.

PNN does not include a mechanism for multiscale operation due to its ab initio nature. For contracted graphs, PNN uses $\gamma_{min} = 2\Theta_{av(\chi)}$ and $\gamma_{max} = 5\gamma_{min}$ to account for the computational weight embodied in supervertices. Spin reallocation is disabled in our implementation of PNN for contracted graphs, for implementation simplicity and for mimicking PNN’s hardware realization. This leads to an increase in communication overhead for PNN as $|V_M|$ increases. PNN also loses some “accuracy” when performing coarse-structure mapping, because the coupling term in equa-

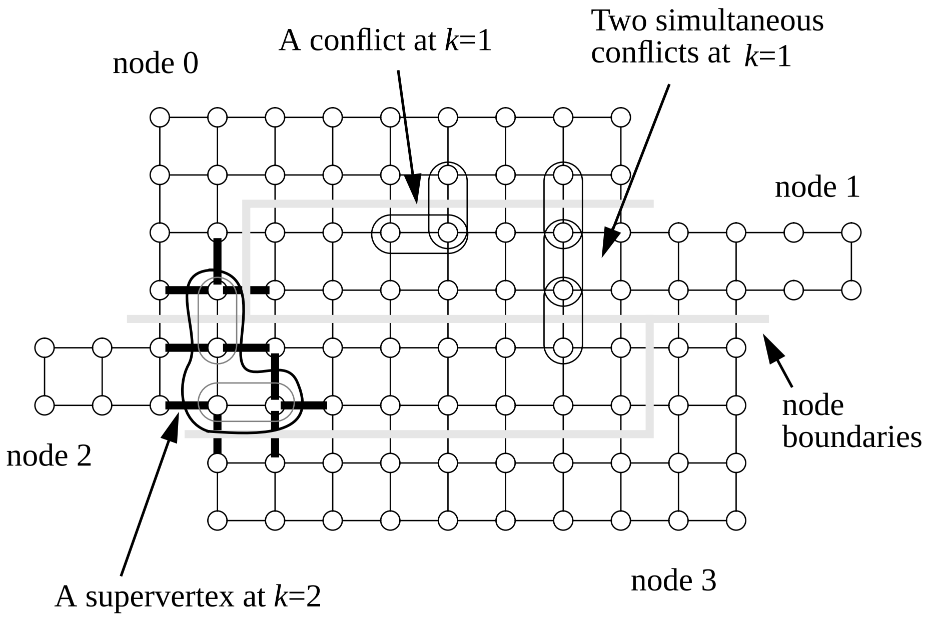


Figure 9.2. Possible conflicts and supervertex produced by a parallel graph contraction strategy faithful to SGC; heavy edges indicate the information needed in iteration k and k+1.

```
Read computation subgraph $(G_C^0/N_H)$ ;
for $k$ = 1 to $\chi$ do
        Perform one iteration of SGC;
        Give global numbers to supervertices in $(V_C^k/N_H)$ ;
        Build contracted subgraph $(G_C^k/N_H)$ ; /* involves communication */
        Update CONTRACT[$v$];
                          /* which initial vertex belongs to current supervertex */
endfor
Output $(G_C^\chi/N_H)$ ;
```

Figure 9.3. PGC1 (node algorithm), $N_H = |V_M|/2$.

the parallel algorithm adheres to the operation of SGC, it would involve conflicts in different nodes over nonlocal partner vertices. Moreover, nonlocal information needed in a node would vary and extend beyond the boundary vertices of adjacent nodes in successive contraction iterations. Figure 9.2 shows possible conflicts between different nodes over partner vertices; it also illustrates how a supervertex formed across node boundaries leads to an expansion in the amount of nonlocal and non-boundary information needed in the nodes. Resolving conflicts and communicating varying non-boundary information lead to large communication overheads and difficult implementation. Since the goal is to efficiently produce reasonable contracted graphs that satisfy the afore-mentioned design requirements, deviating from the operation of SGC is both acceptable and necessary. It is possible to devise a number of strategies for parallel graph contraction that yield acceptable results. In this section, we describe the simplest strategy, which works reasonably well under certain conditions, and then we point to three better strategies.

A simple parallel graph contraction algorithm is given in Figure 9.3 and is henceforth referred to as PGC1. It is based on executing SGC concurrently and independently in N_H NCUBE nodes. The initial graph, G_C^0, is partitioned among the nodes in a naive way: each node is allocated $\left|V_C^0\right| / N_H$ vertices; node n_i is allocated vertices $n_i (\left|V_C^0\right| / N_H)$ to $(n_i+1) (\left|V_C^0\right| / N_H) - 1$. Such subgraphs are denoted as (G_C^0 / N_H). Each PGC1 iteration performs an SGC step on the local subgraph. Local graph contraction is confined only to local vertices and local edges. That is, the crossedges, across node boundaries, connecting vertices allocated to different nodes cannot be contracted and will be carried over to the contracted subgraph regardless of their weight. This is, obviously, a deviation from SGC and has two shortcomings. Firstly, it might lead to large weights on the crossedges and to large supervertex degrees, thus violating the first and second design requirements stated above. Secondly, it might result in locally disconnected supervertices, violating the third design requirement. However, if χ is small, e.g. 3 or less, and the number of crossedges remains a small fraction of the total number of edges, the contracted graphs produced by PGC1 turn out to be reasonable.

PGC1 is adopted in the performance evaluation experiments reported below. To ameliorate the effects of its shortcomings when χ is not small, we execute it on half the available NCUBE nodes. That is, only $N_H = \left|V_M\right| / 2$ nodes are used for PGC1 in order to decrease the number of crossedges. This halves PGC1's efficiency, which is still acceptable because its execution time is considerably smaller than the mapping time, anyway.

$(|V_C|_{k-1} + \Theta_{max(k-1)})$, can be used for sorting vertex weights since the maximum weight is known and is relatively small in every contraction iteration [Cormen et al. 90]. It can be easily shown that the complexity of SGC is of the order of $(\theta_{max}|V_C|)$, which is, obviously, dominated by the first contraction step. It is also clear that SGC's complexity is considerably less than that of any of the PO mapping algorithms.

```
Input: $G_C^0(V_C, E_C)$; $\chi$;
        CONTRACT[$v$] = $v$ for $v$ = 0 to $(|V_C| - 1)$;
        $\Theta_0(v) = \theta(v)$; $\xi_0(v_i, v_j) = 1$; $|V_C|_0 = |V_C|$;

for $k$ = 1 to $\chi$ do
        Counting-Sort($\Theta_{k-1}(v)$);
        repeat (of order of $|V_C|_{k-1}$)
                $v_i$ = unpaired vertex with minimum $\Theta_{k-1}(v_i)$;
                /* find $v_i$'s partner, if exists */
                if $k$ = 1 then
                        $v_j$ = randomly chosen unpaired vertex adjacent to $v_i$, if exists;
                else
                        $v_j$ = unpaired vertex adjacent to $v_j$ with maximum $\xi_k(v_i, v_j)$,
                                                                                if exists;
                end-if-else
                Form supervertex $v_{ij} = \{v_i, v_j\}$;
                CONTRACT[$v_i$] = CONTRACT[$v_j$] = $v_{ij}$;
        until all vertices are paired or considered
        Determine $|V_C|_k$;
        Construct_contracted_graph ($G_C^k$, $\Theta_k(v_{ij})$, $\xi_k(v_{ij}, v_{nm})$);
endfor
Output: $G_C^\chi(V_C^\chi, E_C^\chi)$ with size $|V_C|_\chi$;
```

Figure 9.1. Sequential graph contraction algorithm.

9.1.2. Parallel algorithms

Parallel graph contraction is based on distributing the graph vertices among the NCUBE nodes. If tices and of its superedges are rather identical.

When mapping a contracted graph, the weights of supervertices determine the computational workload of processors, and the edge weights affect the interprocessor communication cost. Hence, for mapping purposes, an optimally contracted graph would be a fairly homogeneous weighted task graph that involves relatively small edge weights. That is, optimal graph contraction is identical to finding an optimal solution to the mapping problem, which is intractable. Therefore, we can only hope for reasonable heterogeneous contracted graphs. The heterogeneity of contraction contributes to placing an upper bound on the contraction parameter, χ, as pointed out in Chapter 4, and shown in the results below. On the other hand, the three PO mapping algorithms have degrees of flexibility and adaptability, which allows them to utilize graph contraction despite non-optimality.

Based on these considerations, the requirements guiding the development of a graph contraction algorithm can be stated as follows. The first requirement is making edges with large weights intra-supervertices edges, ensuring that most of the inter-supervertices edges have relatively small weights. This requirement helps in reducing the communication cost in a mapping configuration. The second requirement is having small average supervertex degree in the contracted graph. Small supervertex degrees are useful for decreasing the number of communicating processors, and hence the communication cost, in a mapping configuration. The third requirement is keeping the Θ_{max} to Θ_{min} ratio as small as possible; smaller variations in the weights of the vertices of a contracted graph reduces heterogeneity, makes the contracted graph less far from optimality, and yields smaller size graphs. This requirement is also necessary to support the second requirement. The fourth, and the most important, requirement is that the graph contraction heuristics must be efficient; its execution time must be smaller than the mapping time.

9.1.1. Sequential algorithm

A sequential graph contraction (SGC) heuristic algorithm is given in Figure 9.1. In each contraction iteration, *k*, pairs of vertices, i.e. partners, are selected from G_C^{k-1}, to be merged. The first vertex, v_i, is that which has the minimum $\Theta(v_i)$. Its partner, v_j, is an unpaired vertex adjacent to v_i with maximum $\xi(v_i, v_j)$. If v_j does not exist, v_i becomes a vertex of G_C^k. The way v_i is selected ensures that vertices with smaller weights are merged before those with larger weights, which limits the differences in the weights of supervertices in G_C^k. It has been observed that this yields a $\Theta_{max(k)}$ to $\Theta_{min(k)}$ ratio in G_C^k that is smaller than or similar to that in G_C^{k-1}, which is a reasonable result, although not optimal, and satisfies the third design requirement mentioned above. A partner vertex, v_j, is selected with maximum $\xi(v_i, v_j)$ to satisfy the first design requirement. Also, both techniques for selecting partner vertices support the second design requirement indirectly.

SGC is an efficient heuristic algorithm. A counting sort algorithm, with complexity of the order of

Chapter 9

Graph Contraction Heuristics for Efficient mapping of large problems

It is evident from the previous results that PO algorithms are very slow in mapping large problems. Their execution time is unacceptable when compared with typical time for solving the problems being mapped. In fact, this is true even for faster algorithms, such as RSB which takes about the same time for mapping FEMW(53961) as the solution time for a fluid dynamics problem using an Euler solver [Das et al. 91]. In addition, PGA has memory space limitation even for moderate size problems. Therefore, no matter which mapping algorithm is chosen, large problems need to be shrunk first, and then the reduced size problem is mapped to a multiprocessor. This fact has been recognized by other researchers. Some researchers have just stated the need for such a pre-mapping step [Fox 88b; De Keyser 91]. Others proposed the formation of blocks of data objects during the process of generating the data set itself for the problems they dealt with [Nolting 91]. This approach is not generalizable and not flexible enough to suit the requirements of different mapping algorithms and different problems.

In Chapter 4, we suggested the use of graph contraction for reducing the size of the search space and demonstrated the significant saving in time it produces. In this chapter, we present simple and efficient sequential and parallel graph contraction heuristic algorithms which are general and can be applied to different computation graph structures. We also describe some design modifications to the PO algorithms which are required when graph contraction premapping is employed. Then, we explore and compare the resultant performances of the mapping algorithms.

9.1. Efficient graph contraction heuristics

Graph contraction, with parameter χ (equation 4.2), consists of χ iterations. The basic step at iteration k involves merging two adjacent vertices, v_i and v_j, to form a supervertex v_{ij} whose computational weight is $\Theta(v_{ij}) = \Theta(v_i) + \Theta(v_j)$; initially $\Theta(v) = \theta(v)$. v_i and v_j are henceforth referred to as partner vertices. Merging two vertices, v_i and v_j, is equivalent to the contraction of the edge connecting them. Also, a superedge connecting supervertices v_{ij} and v_{nm} is assigned a weight $\xi(v_{ij}, v_{nm}) = \sum_{v_x \in v_{ij}, v_y \in v_{nm}} \xi(v_x, v_y)$, where $\xi(v_i, v_j) = 1$ initially. We henceforth refer to a contracted graph as homogeneous if the weights of its superver-

8.3. Concluding remarks

The three parallel PO algorithms exhibit diverse properties which make them suitable for different applications. The mapping solutions produced by the three algorithms are good sub-optimal solutions and do not show a bias towards particular problem configurations. Clearly, the parallel PO algorithms are competitive with good bisection algorithms, specifically RSB, as far as solution quality is concerned; but, they are slower. The improvement in mapping quality produced by PGA and PSA are more pronounced when the ratio of communication to computation is larger. Among the three PO algorithms, PGA produces the best mapping quality, followed by PSA and then PNN. However, PNN is the fastest and PGA the slowest.

All three parallel PO algorithms are somewhat less robust than their sequential counterparts due to additional design parameters. But, the parallel algorithms no longer exhibit the difference in robustness observed for the sequential algorithms, in Chapter 4; the levels of insensitivity to problem and design parameters for the three parallel algorithms are comparable.

There is a significant difference in the memory space requirements between the algorithms. In PGA, a population of structures evolve, and, thus, information is needed in every node (subpopulation) about the whole problem, whereas in PSA and PNN only the local subproblem is considered in a node. For large problems, PGA requires large memory space, which is, generally, not technologically practical. One way to alleviate this restriction is to add a preprocessing graph contraction step to PGA, as advocated in Chapter 4 and elaborated in Chapter 9. With graph contraction, the problem, and consequently the individuals in the population, are reduced in size by a suitable factor, χ (equation 4.2).

PSA and PNN are scalable. The quality of their solutions remains almost constant provided that the grain size, $|V_C|/N_H$, does not become small. Efficiency of both, PSA and PNN, decreases with larger hypercubes, more quickly for PNN. Decreasing efficiency implies smaller decreases in execution time as the size of the hypercube increases. If the memory space restriction is circumvented, as suggested above, PGA would enjoy better scalability than PSA and PNN. With larger hypercubes, its execution time decreases faster. It yields good solutions, even for the smallest deme size (PGA's grain size) of two. Furthermore, larger hypercubes offer the opportunity to increase the total population size and the number of demes for PGA, which is likely to produce yet better solutions. However, even with graph contraction, PGA will be restricted by its memory space requirements for large multiprocessors.

It is clear from the execution time figures of the PO algorithms that such times are long and unacceptable, in practice, especially for large data sets. Moreover, the fraction of improvement produced by the physical algorithms seems to decrease for larger problems and growing grain size, at a big time penalty. This motivates the graph contraction based technique, described in the next chapter, for reducing the execution time significantly. The high memory space demand of PGA for FEMW(3681) also accentuates the need for graph contraction preprocessing.

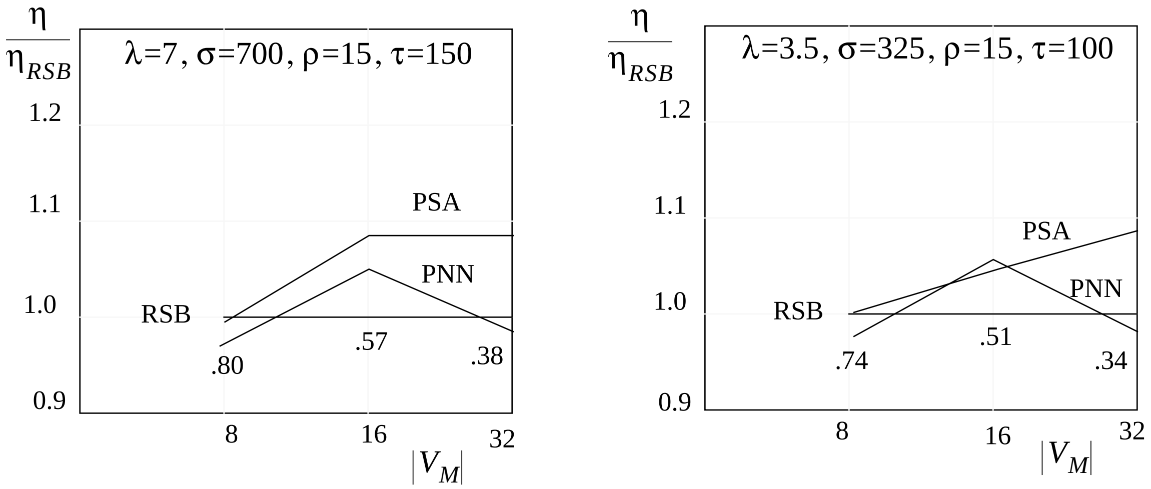


Figure 8.18. Solution quality for mapping FEMW(3681) with some realistic parameter values.

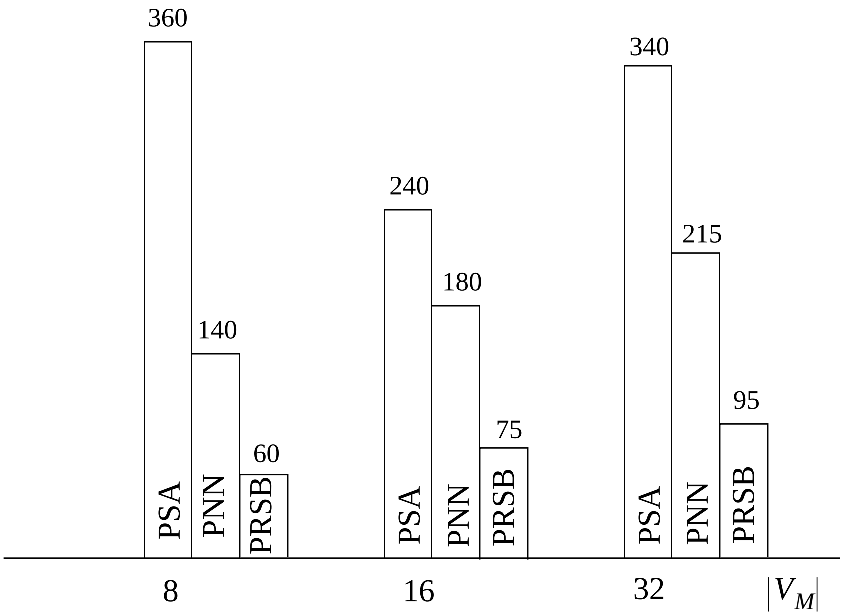


Figure 8.19. Average execution time, in seconds, for mapping FEMW(3681). $(N_H = |V_M|)$

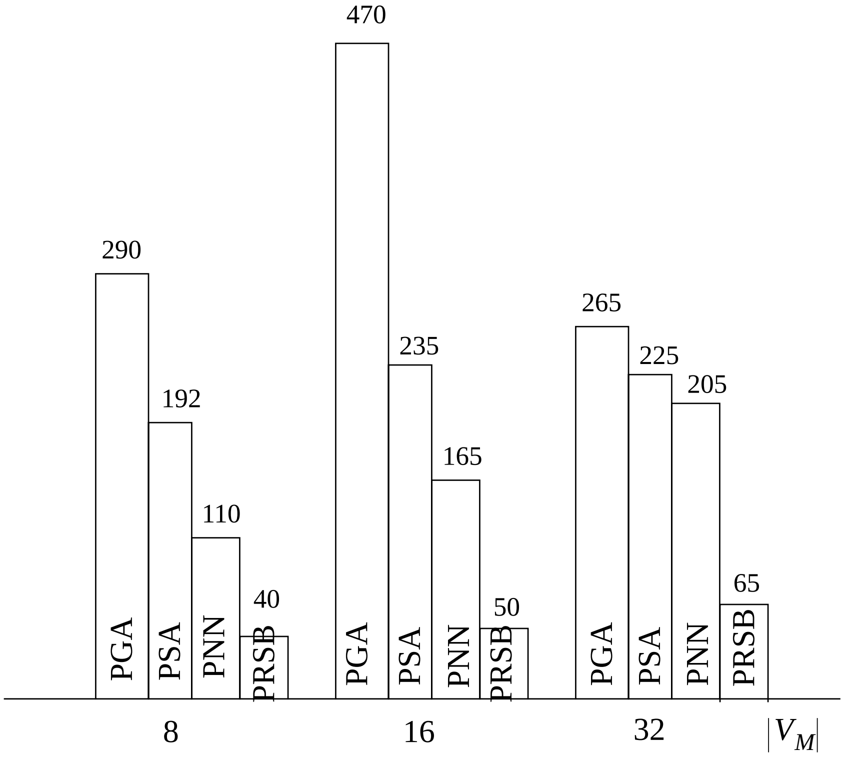


Figure 8.17. Average execution time, in seconds, for mapping FEMW(2800). $(N_H = |V_M|)$

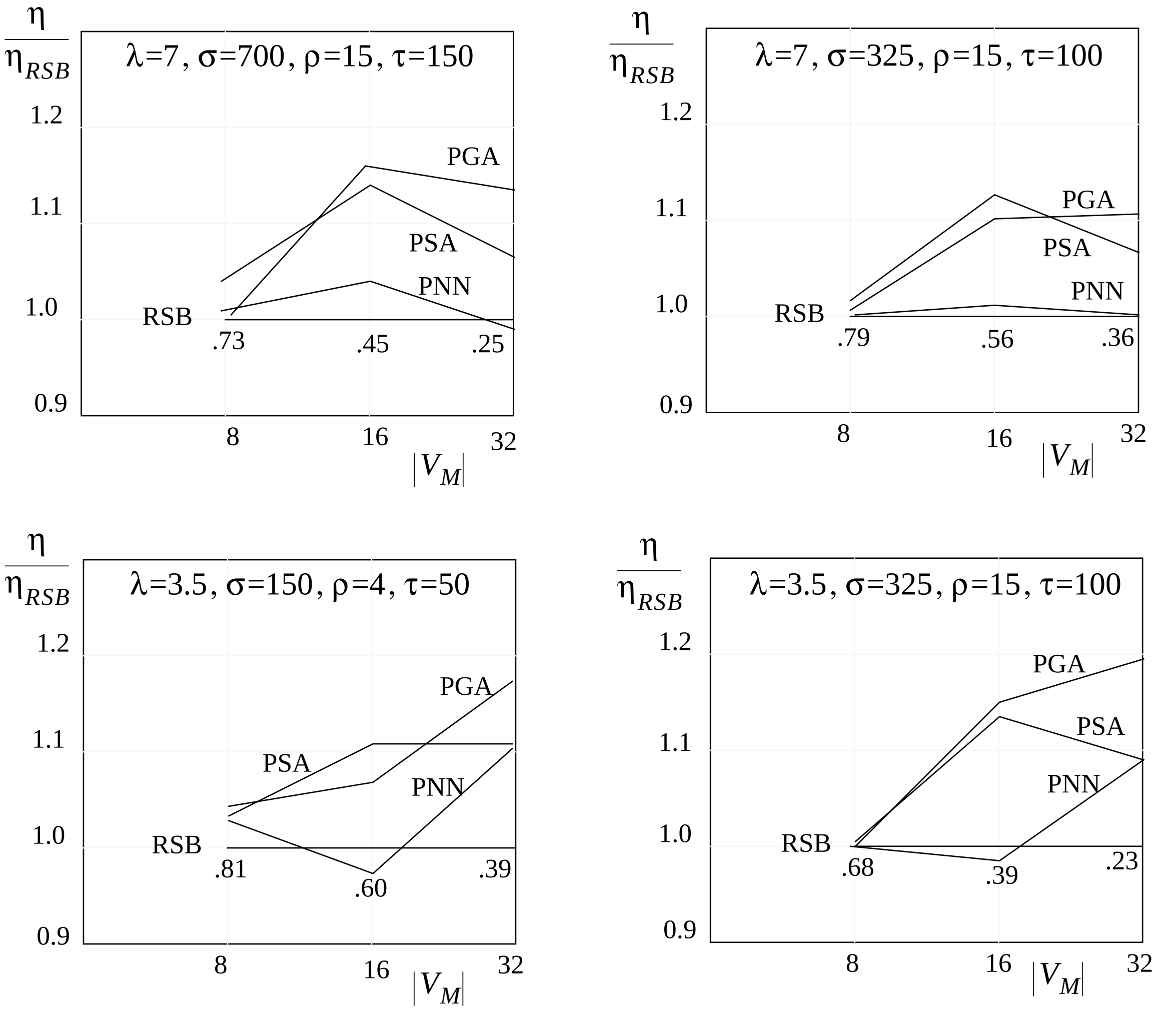


Figure 8.16. Solution quality for mapping FEMW(2800) with some realistic parameter values.

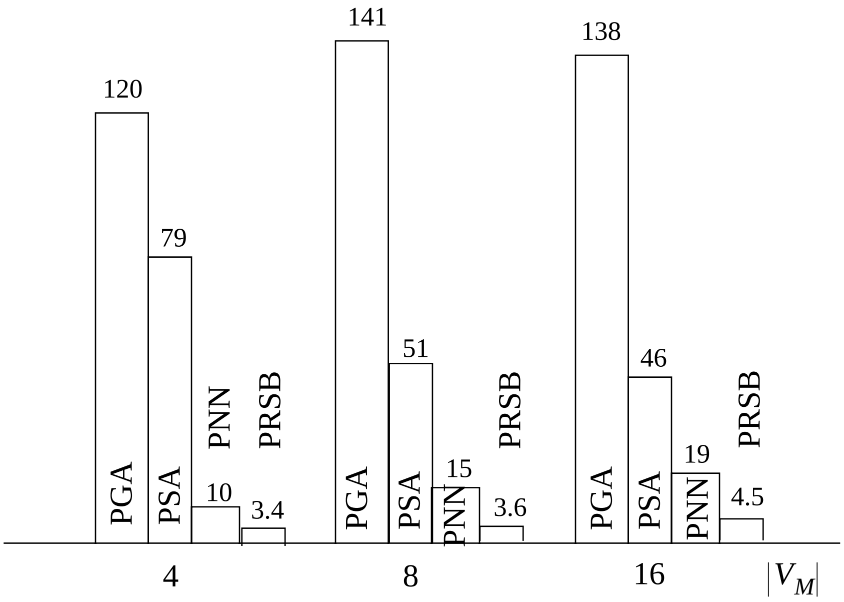


Figure 8.15. Average execution time, in seconds, for mapping FEMW(545). $(N_H = |V_M|)$

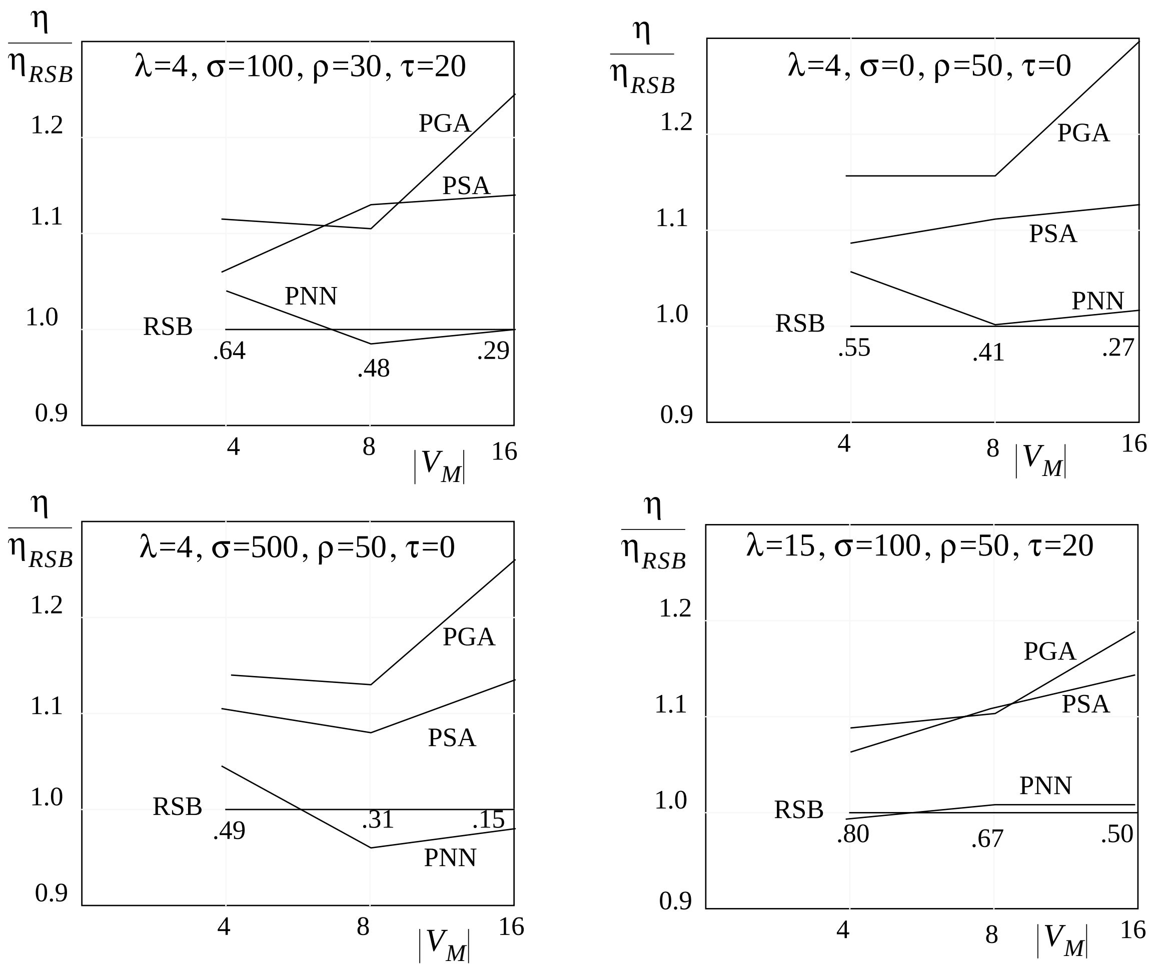


Figure 8.14. Solution quality for mapping FEMW(545) with various parameter values.

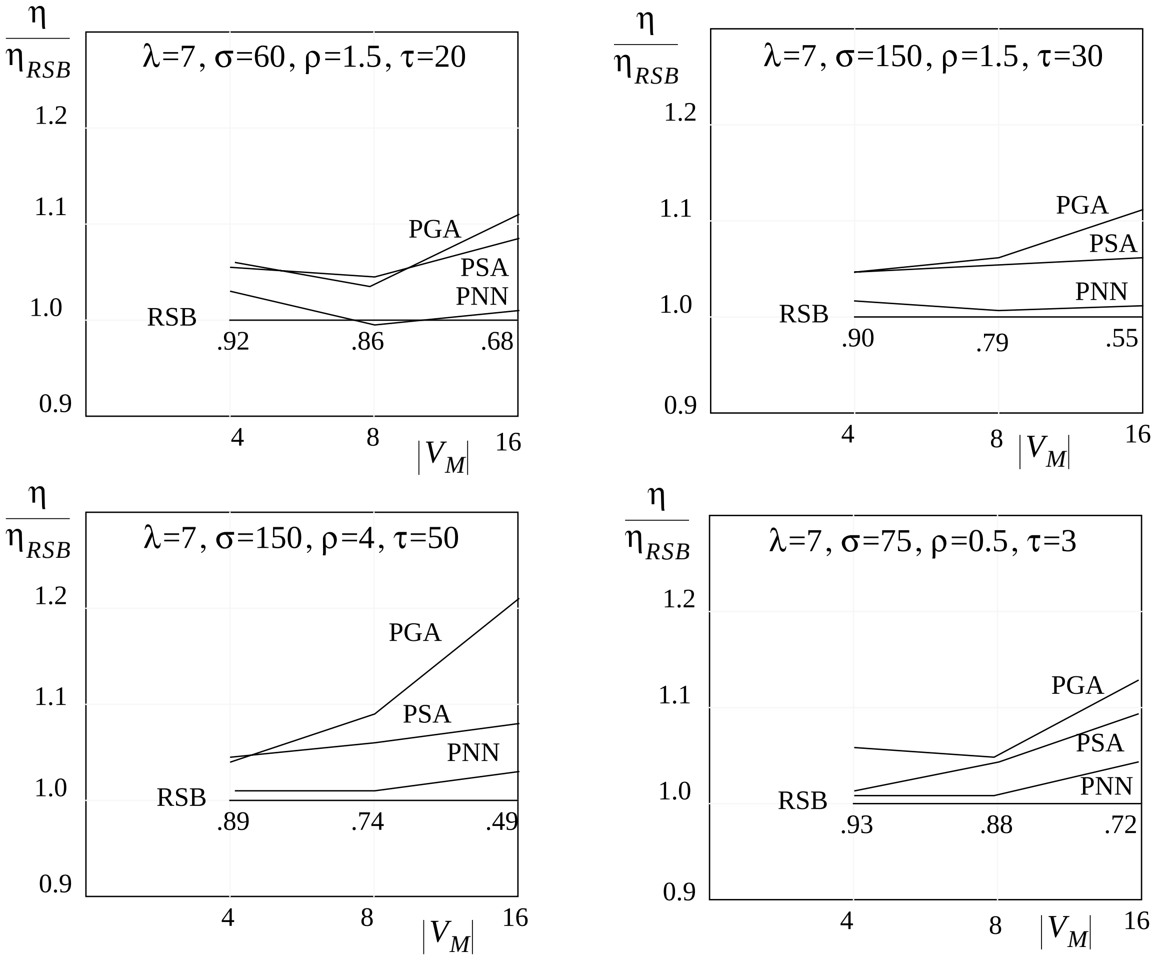


Figure 8.13. Solution quality for mapping FEMW(545) with some realistic parameter values.

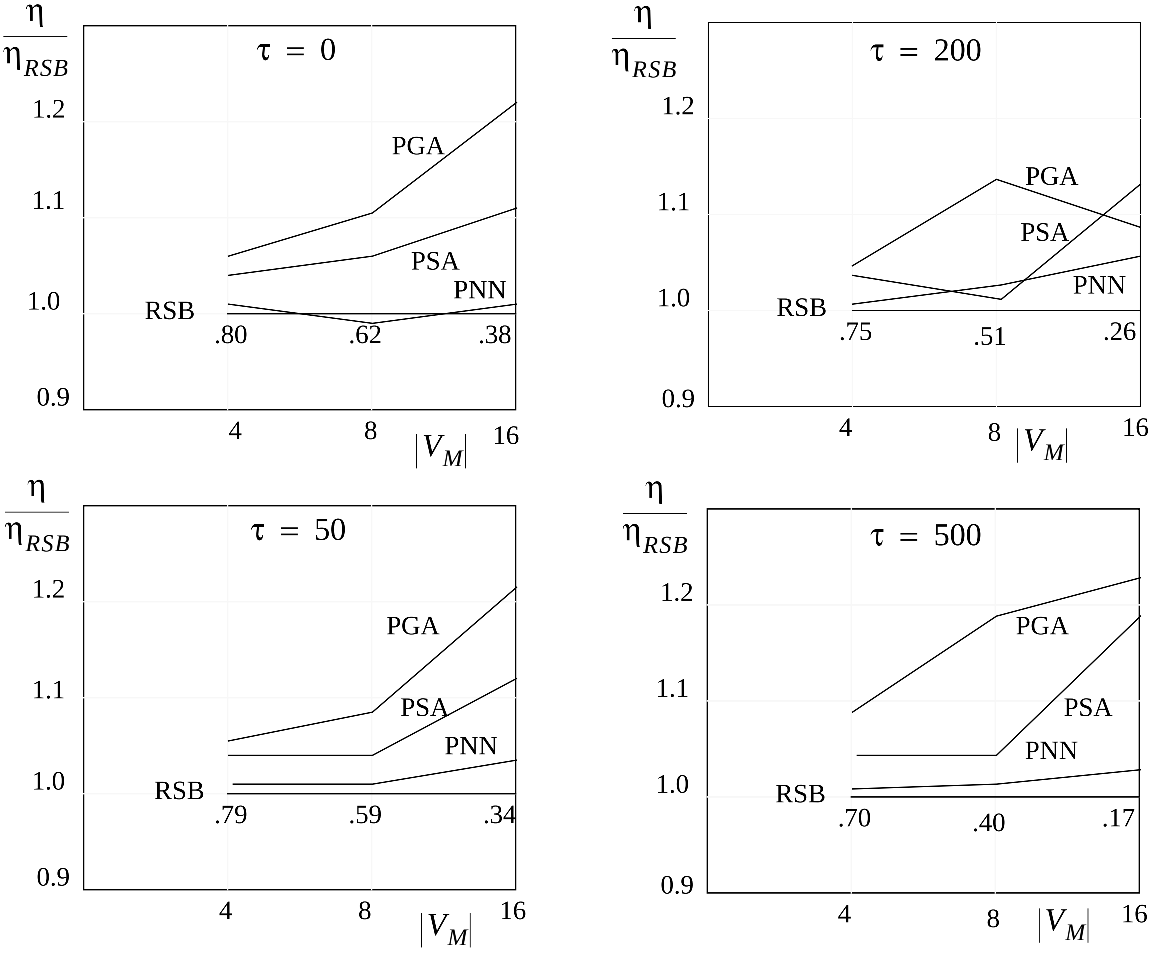


Figure 8.12. Solution quality for mapping FEMW(545) with different τ, $\rho = 15$, $\sigma = 325$ and $\lambda = 7$.

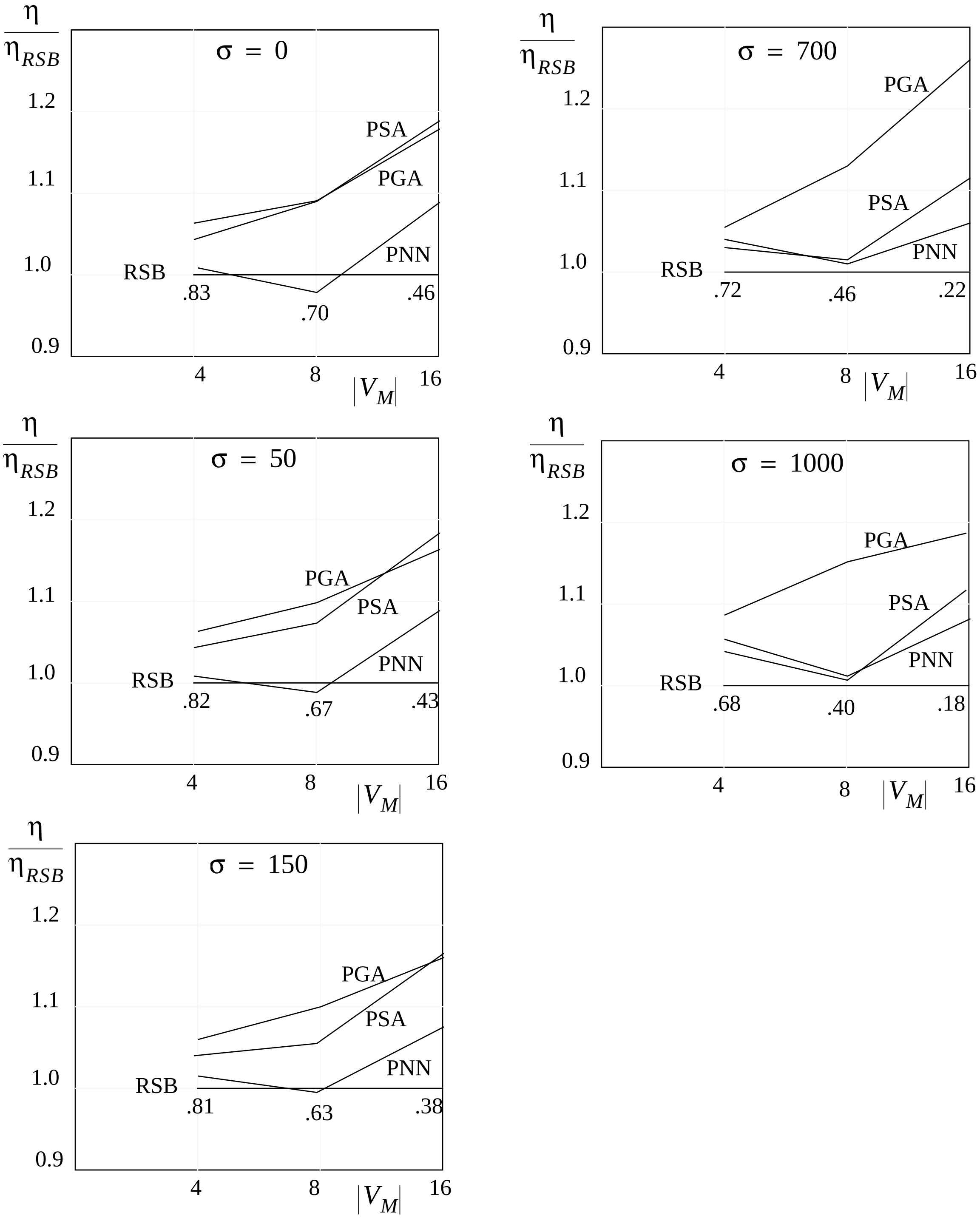


Figure 8.11. Solution quality for mapping FEMW(545) with different σ, $\rho = 15$, $\tau = 100$ and $\lambda = 7$.

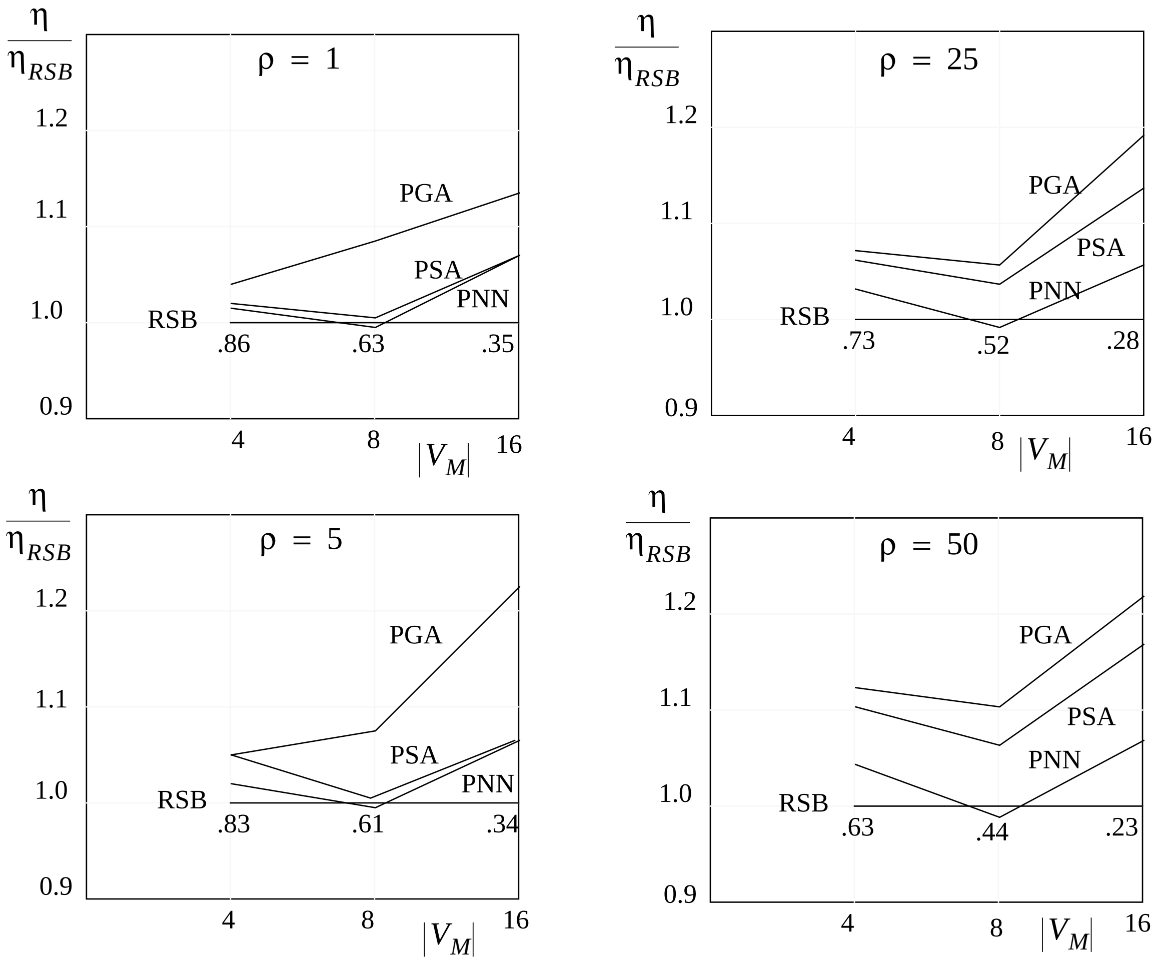


Figure 8.10. Solution quality for mapping FEMW(545) with different ρ, $\sigma = 325$, $\tau = 100$ and $\lambda = 7$.

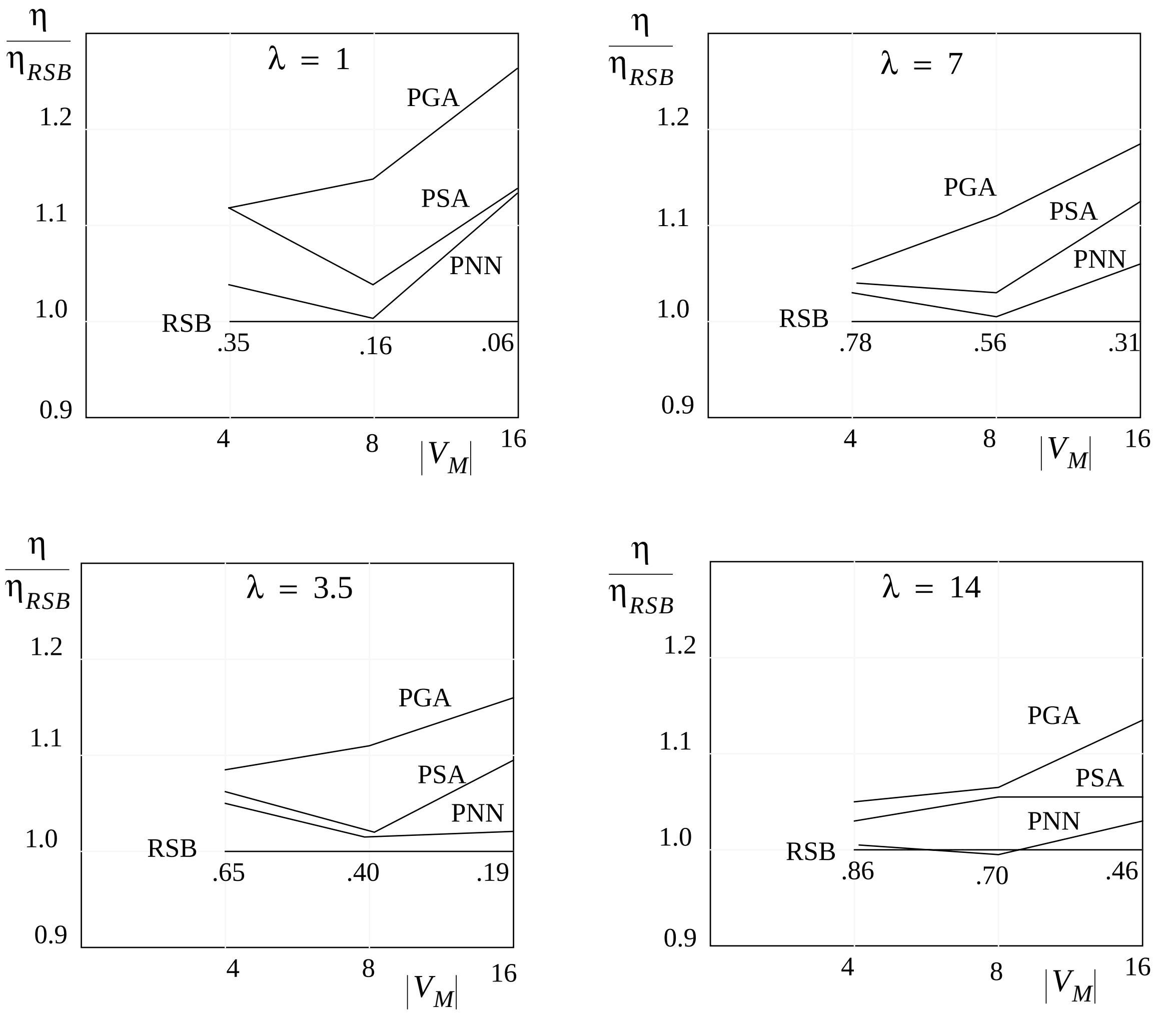


Figure 8.9. Solution quality for mapping FEMW(545) with different λ, $\sigma = 325$, $\rho = 15$ and $\tau = 100$.

three data sets. RSB is faster for these problems. We do not have access to a parallel version of RSB. However, a rough estimate of its time is also shown; it is calculated by scaling reported parallel time [Das et al. 91] using RSB's complexity expression, in Section 4.4, and i860 to NCUBE/2 speed ratio [McCurley and Plimpton 92]. As expected, PGA is the slowest, followed by PSA, and PNN is the fastest. However, the difference in execution time decreases for larger problem size, $|V_C|$, and larger multiprocessor size, $|V_M|$. The reason is, as discussed before, that PNN's efficiency decreases rapidly for larger $|V_M|$, whereas PGA's efficiency, on the other extreme, might not decrease. However, this property of PGA is not clear from the figures due to the use of different deme sizes for different cases. Moreover, as $|V_C|$ and $|V_M|$ increase, the number of generations, GEN, and the number of annealing steps, A, in the complexity estimates of PGA and PSA, do not increase by the same proportion; but, all terms in PNN's complexity estimate scale up with problem and multiprocessor sizes.

8.2. Results for various parameter values

This section includes experimental results for mapping FEMW(545), FEMW(2800) and FEMW(3681), with different parameter values. The goal is to investigate how the performance of the PO algorithms changes for different algorithms and different multiprocessor machines. In this section, $\zeta(p) = C_p(p)$ (expression 2.6) is used for communication cost, which is more general and more realistic than $C_d(p)$ for present day multiprocessors. The parameters considered are: computation workload parameter, λ, and communication parameters, ρ, σ and τ, in addition to granularity, $|V_C| / |V_M|$. We assume λ = 7, ρ = 15, σ = 325, and τ = 100 as reference parameters. The reference value of λ is not small in order to compensate for the small computation grain size of FEMW(545), but not too big to hide the effects of the communication term. The combination of chosen reference values is considered reasonable for an iPSC/860 machine [Berrendorf and Helin 92; Bokhari 90a] and an Euler solver [Das et al. 92] for unstructured meshes of the FEMW type. For clarity, solution quality values, η (equation 2.7), are normalized with respect to those of RSB. All results are averages of three runs.

Figures 8.9 through 8.14 consider a variety of combinations of parameter values for the data set, FEMW(545). Figures 8.9-8.12 show solution qualities of the PO algorithms and RSB, varying one parameter at a time. Figure 8.13 shows results for combinations of realistic values, corresponding to multiprocessor machines, such as NCUBE/2, NCUBE/10, CM-5 and iPSC/2 [Bomans and Roose 89; Bozkus et al. 92; Fox 91c; Sears 90]. Figure 8.14 shows results for combinations of values, meant to accentuate the effect of some of the parameters, especially ρ. Figure 8.16 shows η values for FEMW(2800), with different realistic parameter values. Note that PGA's solutions are not the best possible for FEMW(2800); they are constrained by a deme size of only 2, due to memory space limitations. Figure 8.18 shows η values for FEMW(3681); PGA is not included due to insufficient node-memory space.

The results show that the PO algorithms yield good suboptimal solutions, which are better than those of RSB, for a variety of algorithms and parallel computers. As expected, PGA finds the best solutions, 5% to 25% better than RSB's. PSA comes second, producing solutions 3% to 15% better than RSB's. PNN's solutions are close to and, often, only slightly better than RSB's solutions, sometimes a little worse. The improvements in solution quality shown by the physical algorithms are more pronounced for larger communication to computation ratios, particularly for larger $|V_M|$, smaller grain size, smaller λ, or bigger ρ. PGA and PSA exhibit good flexibility and adaptability to various algorithm and multiprocessor characteristics, where they allow appropriate trade-offs between the computation and the communication terms of the of the objective function. Obviously, PGA has better adaptability, since fitness uses OF_{typ}, whereas PSA's energy equals OF_{appr}. PNN lacks such flexibility.

Figures 8.15, 8.17 and 8.19 give the average execution time taken by the PO algorithms for the

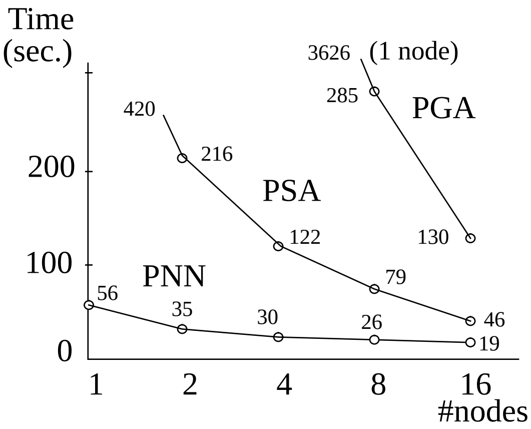


Figure 8.5. Execution time for *TEST2*.

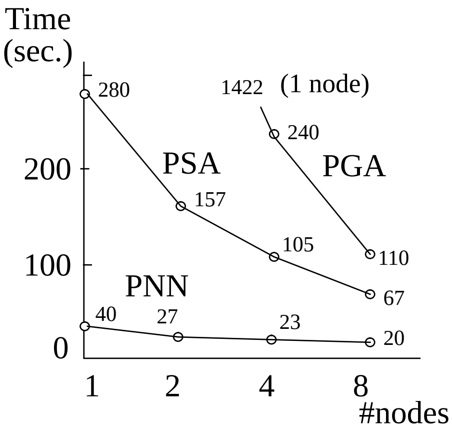


Figure 8.6. Execution time for *TEST3*.

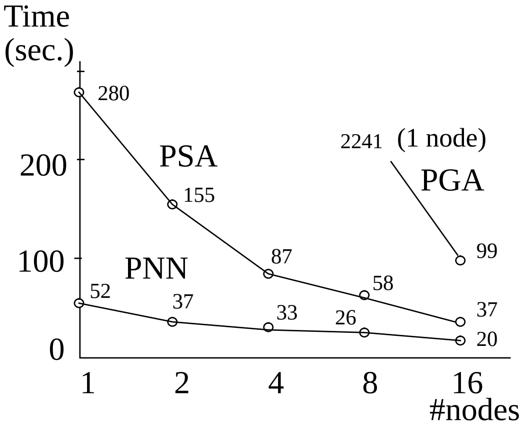


Figure 8.7. Execution time for *TEST4*.

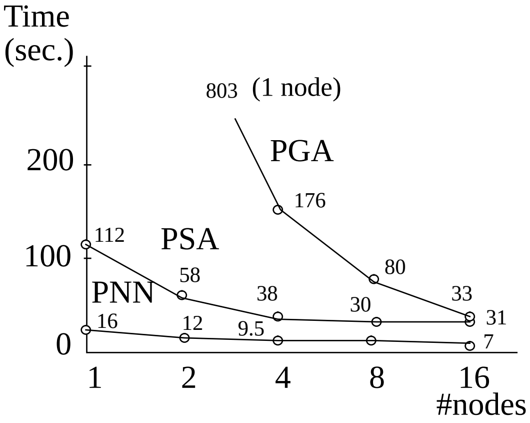


Figure 8.8. Execution time for *TEST5*.

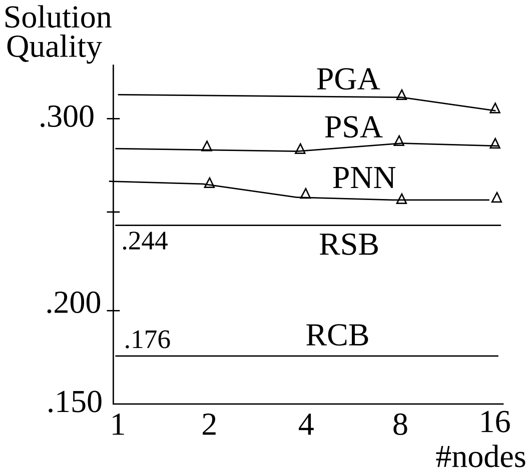


Figure 8.1. Comparison of solution qualities for *TEST2*.

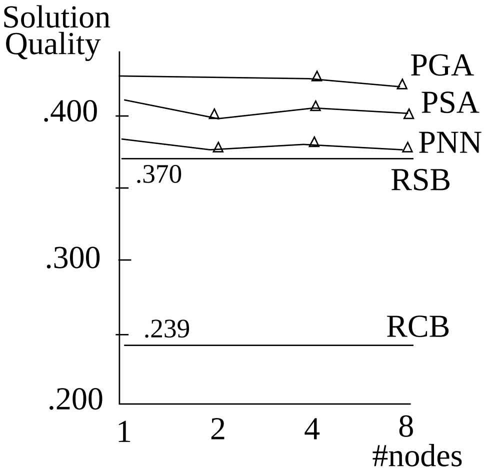


Figure 8.2. Comparison of solution qualities for *TEST3*.

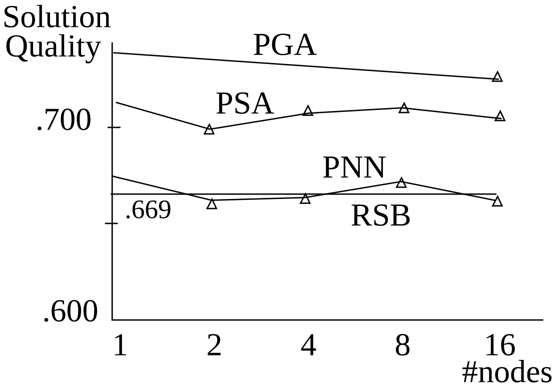


Figure 8.3. Comparison of solution qualities for *TEST4*.

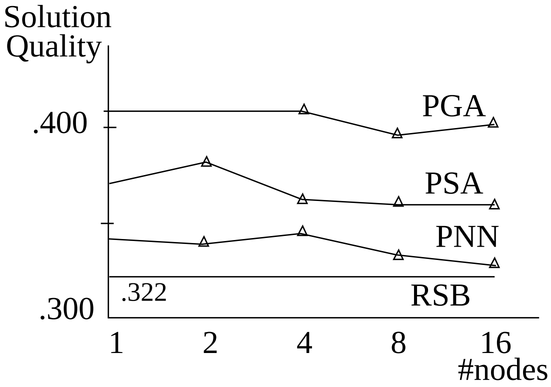


Figure 8.4. Comparison of solution qualities for *TEST5*.

RSB and PNN seem comparable. It should be noted, however, that the discrepancy between the uses of OF_{appr} to guide the operation of the three algorithms and OF_{typ} to evaluate their solutions presents an impediment to the full realization of the capabilities of the physical optimization strategies.

Comparing the three algorithms, it is clear that PGA consistently produces the best solutions, PSA produces the second best, and PNN's solutions come last. For example, for *TEST1* and N_H = 16, PSA's solution is 11% better than PNN's, and PGA's solution is 6% better than PSA's and 18% better than PNN's. This finding is consistent with what is observed for the sequential algorithms (N_H = 1). It is interesting to note that the differences in the solution qualities do not undergo any significant changes with the grain size, i.e. with different N_H values. Further, since the solutions of the parallel algorithms are consistent with those of their sequential counterparts, our earlier conclusion, in Chapter 4, about the applicability of this class of algorithms can be reiterated: the parallel PO algorithms do not exhibit a bias towards particular problem topologies.

The execution times, in seconds, of the three parallel algorithms are summarized in Figures 8.5 to 8.8. It is clear that PGA is the slowest and PNN is the fastest. For example, for *TEST1* and N_H = 16, PSA is 2.4 times slower than PNN, and PGA is 2.8 times slower than PSA. The time order holds for different degrees of parallelism, including the sequential case. However, the gaps separating the time curves shrink with higher degrees of parallelism; PGA's execution time decreases the fastest as N_H increases, followed by PSA. For example, for *TEST3*, the sequential GA time is 29 times that of the sequential NN. But, for 16 nodes, the ratio decreases to only 5. This follows from the result that PGA has the best efficiency and PNN the smallest.

Chapter 8

Comparative Performance Evaluation of PGA, PSA and PNN

Comparative experimental results and discussion of the performances of the parallel PO algorithms are presented in this chapter. The performance measures are solution quality, bias, execution time, robustness, scalability, and memory space requirements. A variety of parameter values are considered for comparing the mapping algorithms. The parameters are computation and communication parameters which characterize parallel algorithms and parallel machines and are included in the objective function. The data sets employed are, again, those involved in *TEST*2 through *TEST*5, explained in Table 5.1, in addition to FEMW(2800) and FEMW(3681).

This chapter consists of two sections; Section 8.1 builds on the results reported in Chapters 5, 6 and 7 for comparing the performances of PGA (i.e. SBPGA), PSA and PNN; Section 8.2 includes results for mapping FEMW(545), FEMW(2800) and FEMW(3681) produced by varying parameter values. In all cases, spectral bisection (RSB) [Pothen et al. 90], a representative of good quality heuristics, is included to give an indication of the quality of the solutions produced by the physical algorithms. We also note that the problems dealt with in this chapter are not large problems. Their computation graphs have a product $|V_C|\theta_{av}$ of a few tens of thousands; this limit is set by PGA's memory space requirements, for the current node-memory capacity of NCUBE/2, since graph contraction is not utilized. All results, in this and following chapters, are for NCUBE/2 implementations.

8.1. Comparison for *TEST*2 through *TEST5*

This section compares the performances of the three PO physical algorithms for *TEST2* through *TEST5*, with the same experimental setting used in the last three chapters.

Figures 8.1 through 8.4 summarize the mapping quality results produced by the PO algorithms, for the four test cases. In addition, results of fast bisection heuristics, recursive coordinate bisection (RCB) and recursive spectral bisection (RSB) [Simon 91], are included. It is clear from the figures that the PO algorithms produce good sub-optimal solutions which outperform the bisection methods for the test cases considered. The only exception is the two-dimensional GRID2 case where

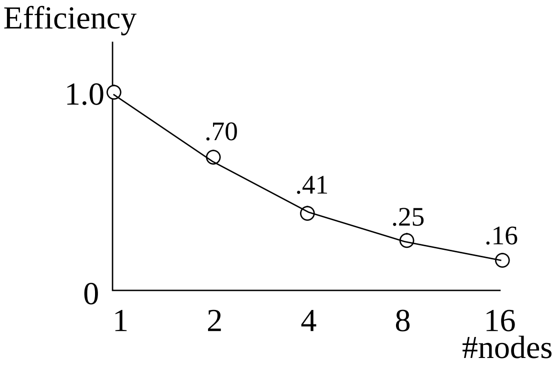


Figure 7.9. PNN efficiency for *TEST4*.

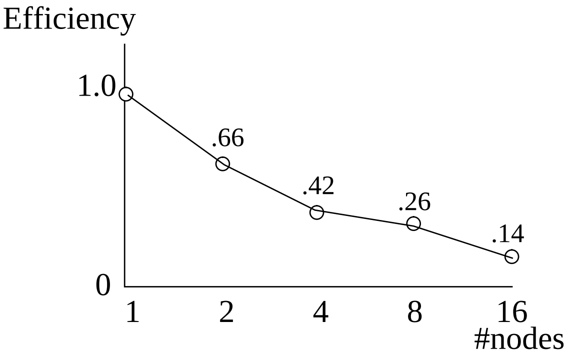


Figure 7.10. PNN efficiency for *TEST5*.

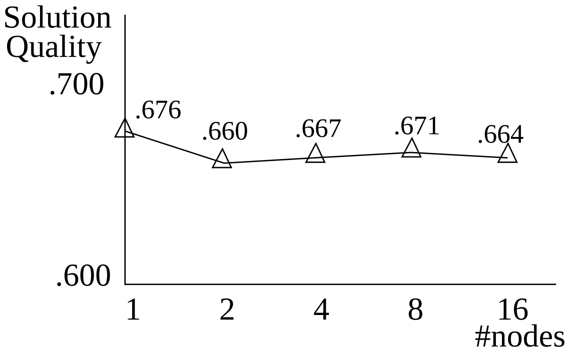


Figure 7.11. PNN solution for *TEST4*.

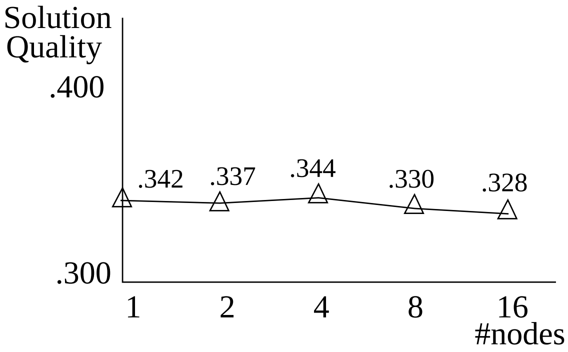


Figure 7.12. PNN solution for *TEST5*.

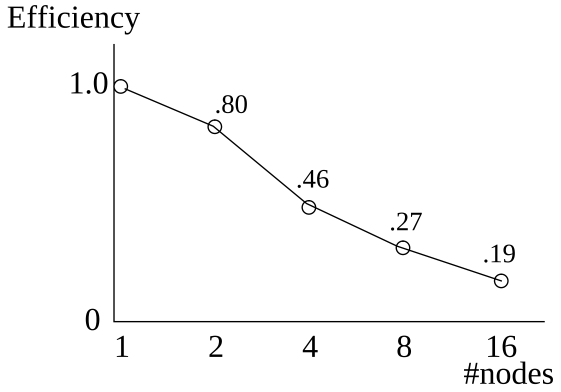


Figure 7.5. PNN efficiency for *TEST2*.

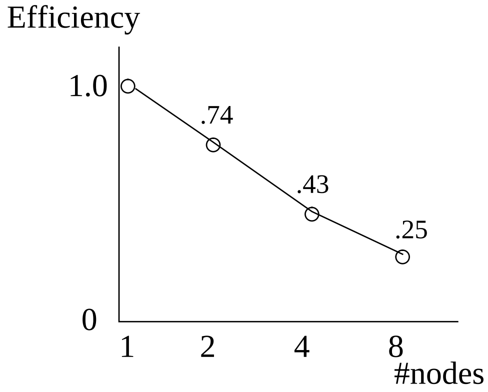


Figure 7.6. PNN efficiency for *TEST3*.

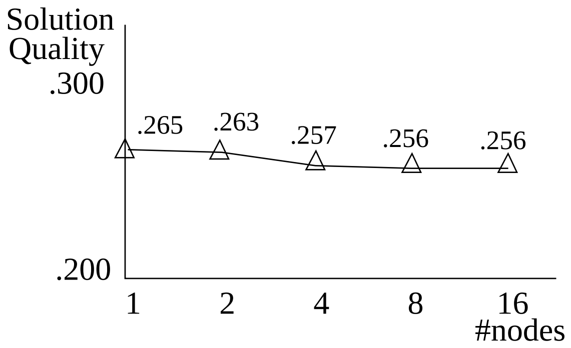


Figure 7.7. PNN solution for *TEST2*.

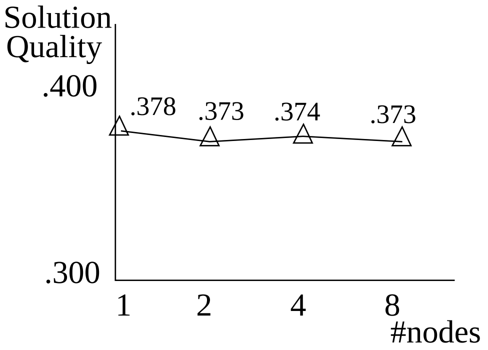


Figure 7.8. PNN solution for *TEST3*.

values of 0.338, 0.446, 0.787 and 0.432, respectively, from Chapter 4. The sharp fall in efficiency reflects the small amount of computation performed by PNN per spin and, thus, the rapid increase in the relative cost of global and semi-global summation operations for $S_s(\Phi_k)$ and in the cost of inter-node communication. However, it is clear that for reasonable grain sizes the efficiency is acceptable.

Figures 7.7, 7.8, 7.11 and 7.12 show the quality of the solutions produced by PNN. It seems that a decrease in granularity leads to some decrease in quality. However, the decrease is small in some cases and negligible in others. The decrease can be attributed to considerable increase in the relative magnitudes of the inconsistencies, in between communication operations, for small grain sizes and can be ignored for reasonable granularities.

In PNN, ν_{S_s} and ν_{bdry} are two additional parameters, to those of sequential NN. Their values affect solutions, needles to mention efficiency. Although the empirically derived frequencies are adequate, experimental experience has shown that the inclusion of these parameters results in somewhat reduced robustness.

7.3. Concluding remarks

A parallel algorithm for data mapping, based on BNN, has been presented. PNN deviates from sequential BNN to achieve acceptable speed-ups. A salient feature of PNN is a communication scheme that exploits some of PNN's own characteristics and is adapted to the characteristics of the mapping problem. PNN also includes spin reallocation after each bisection step for reducing the cost of global communication. The experimental results show that, for reasonable granularity, the adaptive communication scheme and spin reallocation provide an adequate mechanism for limiting the decrease in PNN's speed-up. They also show comparable solution qualities to those of sequential BNN.

```
Determine spin subset and computation subgraph allocated to my_node;
Find inter-node communication info (nodes,boundary spins);
for $i$ = 0 to ($log_2|V_M|$ -1) do
    if ($i$ > 0) then /* after 1st bisection */
        Reallocate_spins() and determine my_subcube;
        Find inter-node communication info (nodes,boundary spins);
    endif
    Generate random spin values, $s(v,i,t)$, $v$=0 to $|V_C|$ -1;
    repeat ( for $N_{swpmax}$ sweeps)
        Determine $\nu_{S_s}$ and $\nu_{bdry}$;
        for all spins 0 to $|V_C|$ -1 do
            Global_add ($S_s(\Phi_k)$,my_subcube) at $\nu_{S_s}$;
            Communicate_boundary(spin values) at $\nu_{bdry}$;
            Pick a spin randomly;
            Compute $s(v,i,t+1)$ in domain $\Phi_k$; /* equation (4.1) */
        end-for
    until (convergence)
    Set bit $i$ in the neurons (0 or 1);
end-for
```

Figure 7.4. PNN node algorithm for data allocation.

7.2. PNN properties

In this subsection, the properties of PNN are investigated. The same experimental setting described for PSA and SBPGA is also employed here, with *TEST2-TEST5*.

Figures 7.5, 7.6, 7.9 and 7.10 show the efficiency of PNN for the four test cases for different N_H. As in PSA, the notion of efficiency is not precise here either. All four figures show a rapid decrease in efficiency with increasing N_H and decreasing granularity. The efficiency values, especially for $|V_M| = N_H$, are, as expected, well below sub-optimal efficiencies for typical iterative algorithms applied to the same data sets. PNN values of 0.19, 0.25, 0.16 and 0.14 compare to the suboptimal bisection pass, two spin domains are generated. The spins in domain 0 can be reallocated to nodes in subcube xx...x0 and those in domain 1 to subcube xx...x1, where x is a don't-care symbol, as shown in Figure 7.3. That is, spins are reallocated so that boundaries of spin domains coincide with node boundaries. In the second bisection pass, each of $S_s(0)$ and $S_s(1)$ is needed in only one subcube. Thus, updating the two values can be carried out within the two subcubes concurrently, which reduces the cost of the global operation to a half of what it is in the first bisection pass. Similarly, after the *i*-th bisection, spins in subdomain *j* are reallocated to the subcube whose node numbers agree in the (*i-1*)-th least-significant bits with those of *j*. The cost of updating $S_s(\Phi_k)$ is therefore halved with each successive bisection pass, which yields significant overhead reduction for large hypercubes and improves PNN's scalability.

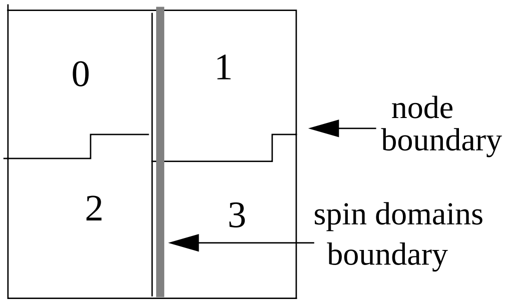


Figure 7.3. Spin reallocation after the first bisection.

The overhead due to reallocation has been, experimentally, found to be reasonable and is, anyway, acceptable since it places the data objects where they should be mapped for the parallel program, ALGO. However, the cost of inter-node communication might increase for some problems because of possible increase in the number communicating nodes, with smaller messages. For example, in Figure 7.3, node 2 has three neighboring nodes instead of two, as in Figure 7.1. The PNN algorithm is summarized in Figure 7.4.

Like PSA, convergence of PNN is detected when no further progress is made. However, progress here is measured in terms of the number of spins that change their direction in one sweep. If this number becomes a small fraction of the total number of spins, then the network is considered converged. It should be noted that the noise term in the network's equation (equation 4.1) causes such small fluctuations even when the effect of the coupling and long-range terms has stabilized.

decreasing ν_{S_s} is, as mentioned above, that after the formation of spin domains, a smaller number of spins change values/directions until convergence. A suitable scheme has been empirically found to be as follows: ν_{S_s} starts with a high value, about 5 global summations in a sweep, and is linearly decreased to 10 summations at $N1_{swp}$; then to a minimum of once every three quarters of a sweep. The variation of the inter-communication period, $(1/\nu_{S_s})$, is depicted in Figure 7.2.

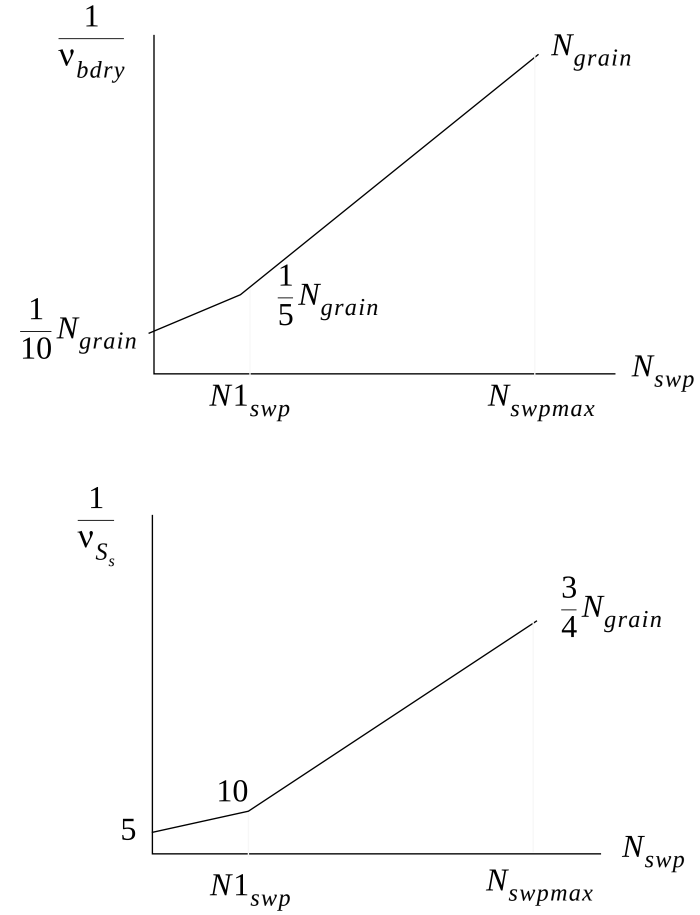


Figure 7.2. Variation of $(1/\nu_{bdry})$ and $(1/\nu_{S_s})$, $N_{grain}=|V_C|/N_H$.

Another way to decrease the cost of updating $S_s(\Phi_k)$ is based on spin reallocation. Given the small amount of computations required for a spin update, the relative cost of global summation, at frequency ν_{S_s}, for updating $S_s(\Phi_k)$ rises rapidly and PNN's speed-up starts to vanish for smaller granularity and large hypercubes. To decrease the summation cost, we reallocate spins to nodes after each bisection pass so that summation will subsequently be needed within smaller subcubes instead of the entire cube. The initial spin allocation remains as described above. After the first after the formation of domains could lie entirely within a node and that spin values at the node boundaries might not change for many updates in a sweep and, therefore, need not be communicated between nodes frequently. The third observation is that some inconsistencies resulting from the use of outdated neighboring spin values can be tolerated since PNN is inherently “erroneous” anyway.

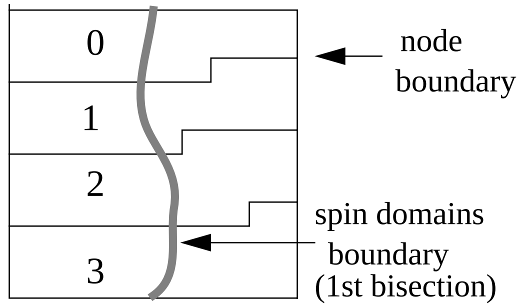


Figure 7.1. Boundaries of nodes and spin domains for the first bisection.

These three observations lead to an internode communication scheme, whereby the frequency of boundary communication, ν_{bdry}, is attenuated as N_{swp} increases, in every bisection iteration. ν_{bdry} starts with a relatively high value, 10 times per sweep, and is linearly decreased to 5 times per sweep at $N1_{swp}$; then to once per sweep at N_{swpmax}. The variation of the inter-communication period, $(1/\nu_{bdry})$, is depicted in Figure 7.2. $N1_{swp}$ is problem dependent. It depends on the dimensionality of the problem and the degree of connectivity of the computation graph. For example, for graphs with large vertex degree, such as that for FEM-W, a suitable $N1_{swp}$ would be $k|V_C|^{1/x}$, with typical values for *x* and *k* being 2 and 1, respectively.

The question of updating spin sums, $S_s(\Phi_k)$, by global operations is similar to that for $S_v(p)$ in PSA. Here also, the frequency of global summation, ν_{S_s}, should be kept to a minimum, whilst not allowing the inconsistency in the local values to grow beyond an acceptable level. We also note that we are interested in net sums of spin values, not in individual values, where inconsistencies contributing to a sum might cancel each other. Further, due to the inherent erroneousness of PNN, some magnitude of inconsistency can be tolerated and, hence, the global sums need not be evaluated every spin update.

Based on these observations, one way to address the question of updating $S_s(\Phi_k)$ is simply to perform global summation with a frequency that decreases with the number of sweeps. The reason for of the neighboring spins (second term) and the most recent value of the sum, $S_s(\Phi_k)$, of the values of all spins, except the spin being updated, that lie in Φ_k. Concurrent spin updating in PNN, therefore, involves three possible sources of inconsistencies. The first source is the local use of outdated nonlocal spin values. The second source of inconsistency is the concurrent updating of spins that belong to the same subdomain, which leads to what we henceforth refer to as inherent "erroneousness" of PNN. The third source of inconsistency is concerned with $S_s(\Phi_k)$, for $k = 0$ to 2^i-1, in different nodes. The design of PNN consists of steps that deal with these inconsistencies by updating boundary information and by global spin summation. Similarly to PSA, an important component of PNN's design is a communication scheme which exploits the characteristics of PNN and is guided by the requirement to maintain both reasonable solution quality and acceptable execution time.

To address the problem of outdated neighboring spin values, we first note that PNN resembles iterative successive relaxation algorithms, such as the Gauss-Seidel algorithm. That is, PNN resembles typical loosely synchronous parallel algorithms whose parallelization is its goal, with an additional burden due to the global sum term, $S_s(\Phi_k)$. This leads to two suggestions. The first suggestion is that the efficiency of a loosely synchronous algorithm, ALGO, on a data set, DATA, represents a loose upper bound for the efficiency that can be attained by PNN when employed for DATA and ALGO. For example, the experimental results of Chapter 4 indicate that a numerical relaxation algorithm with a good suboptimal mapping of FEM-W to a 4-cube can attain an efficiency of 0.338. This means that PNN for FEM-W, $|V_M| = N_H = 16$, and the naive spin allocation scheme described above can never reach an efficiency of 0.338. The second suggestion is that methods used in relaxation algorithms, such as multicoloring, to break up the dependency between connected data objects can be borrowed for PNN. Multicoloring would enable the communication of boundary information between nodes to happen only once every update sweep over the entire data set. However, as in PSA, we have experimentally found that coloring high-connectivity data sets, such as FEM-W, restricts the probabilistic sampling of the spins and leads to degeneration in the solution quality.

Similarly to PSA, the question related to outdated neighboring spin values can be stated as follows: How often should inter-node communication take place for correcting local information about neighboring spin values in other nodes and, yet, keeping the communication frequency as small as possible? Before answering the question, we make three observations. The first observation is that the boundary spins, to be communicated, form only a fraction of the spins allocated to a node. Hence, their values are neither updated nor needed every spin update. The second observation is experimental: after a number of sweeps, $N1_{swp}$, spin domains are formed, although not in their final configuration. The spins in the middle of a domain might become permanently aligned, and only the spins near the boundaries of the domain might change value/direction. Figure 7.1 depicts an example of spin alignment after a number of update sweeps for the first bisection and their naive allocation to a 4-node cube. It shows that a large proportion of spins that keep changing direction

Chapter 7

Parallel Neural Network Algorithm

Although the BNN algorithm is found, in Chapter 4, to be significantly faster than SA and GA, its parallel implementation is also necessary for faster mapping and practical application. BNN consists of $log_2|V_M|$ iterations. In each iteration, *i*, a number of sweeps, N_{swpmax}, over the entire spins are performed. As a result, 2^i spin domains are bisected into 2^{i+1} domains, denoted by Φ_k for *k* = 0 to 2^{i+1}-1, and the corresponding data objects are mapped to 2^{i+1} subcubes. After the last iteration, MAP[] becomes fully specified. It can be seen, from equation (4.1), that a spin update depends on the coupling matrix *G(s,s')* and the weighted sum of spin values $S_s(\Phi_k)$ = $\Sigma_{s \in \Phi_k} s(v', i, t)\theta(v')$. That is, a spin update depends on neighboring spins and the sum of spin values in the same domain.

The coupling dependence and, especially, the intra-domain dependence give rise to communication overhead in a parallel implementation of BNN. This overhead would be large relative to the small amount of computation involved in a spin update. Thus, the reduction of the communication overhead is a challenging task to be addressed in a parallel algorithm. A parallel NN algorithm is described in this chapter and its properties are experimentally investigated. It has been implemented on an N_H-node NCUBE/2.

7.1. Algorithm

The Parallel Neural Network (PNN) algorithm, presented in this section, takes a similar approach to that of PSA. It is based on executing the sequential NN algorithm concurrently and loosely synchronously in the N_H hypercube nodes, where the local memories of the nodes contain disjoint subsets of spins, i.e. data objects, and their associated computation subgraphs. This gives rise to parallelization issues similar to those encountered in PSA. However, the NN algorithm suggests different ways to address these issues. For example, the initial allocation of spins to N_H nodes follows the naive scheme used in PSA for the first bisection step of PNN. But, the results of the bisection steps themselves allow a reallocation of spins which subsequently reduces the communication overhead. A suitable reallocation scheme is discussed below.

PNN also deviates from the sequential operation of NN. According to equation (4.1), an update of a spin value *s(v,i)* in spin domain Φ_k at any time in a sweep depends upon the most recent values

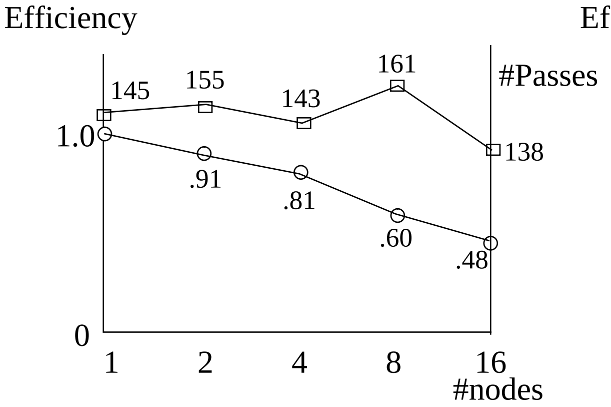


Figure 6.8. PSA efficiency and #passes for *TEST4*.

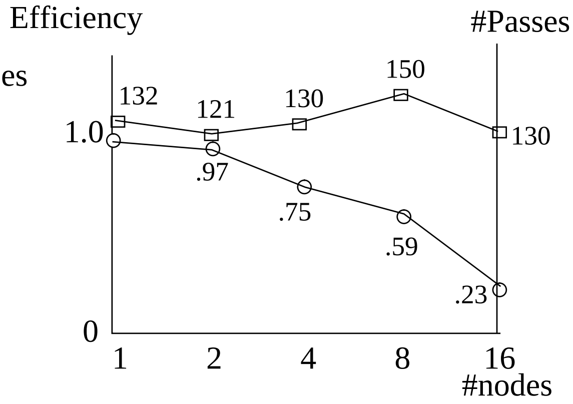


Figure 6.9. PSA efficiency and #passes for *TEST5*.

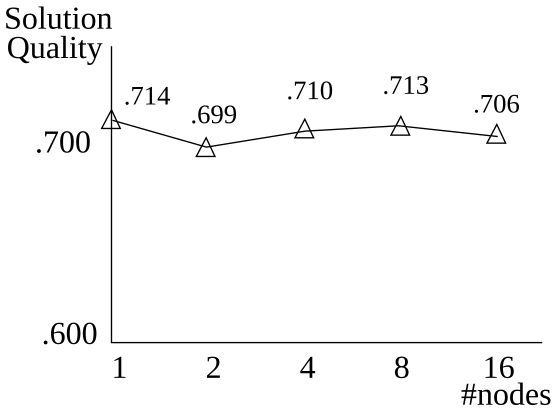


Figure 6.10. PSA solution for *TEST4*.

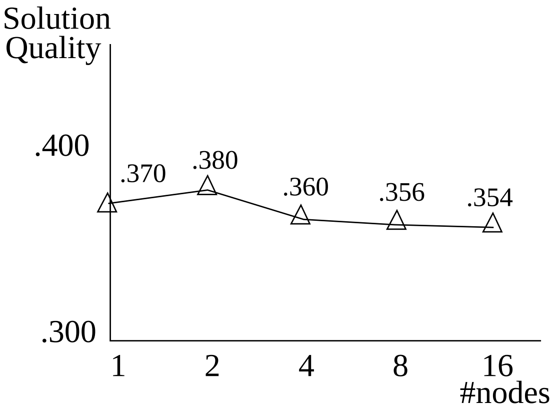


Figure 6.11. PSA solution for *TEST5*.

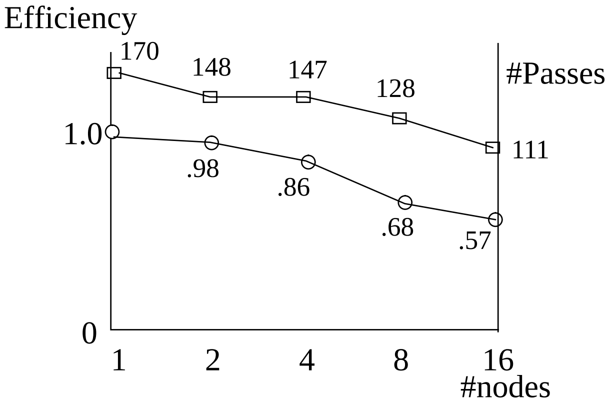


Figure 6.4. PSA efficiency and #passes for *TEST2*.

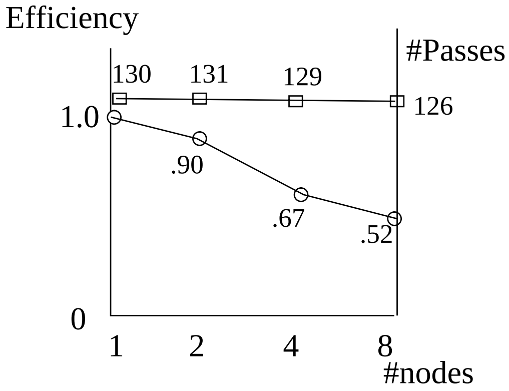


Figure 6.5. PSA efficiency and #passes for *TEST3*.

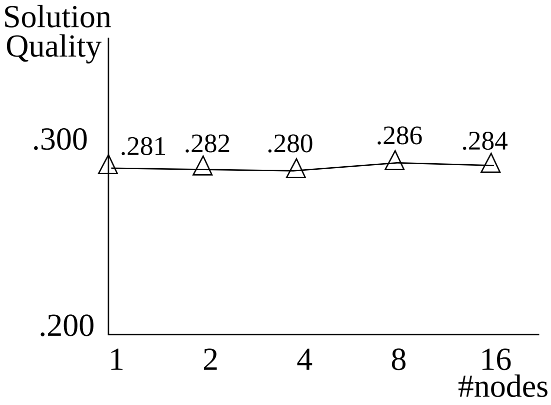


Figure 6.6. PSA solution for *TEST2*.

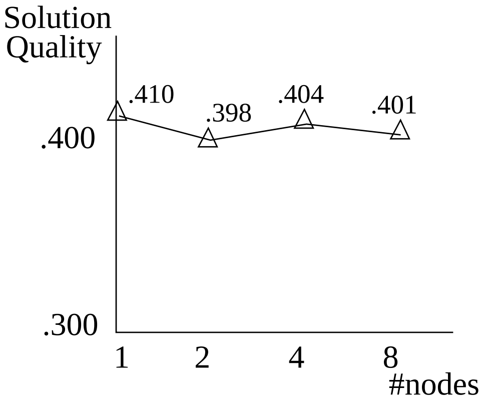


Figure 6.7. PSA solution for *TEST3*.

tion to possible differences in performance between PSA and SA due to the inconsistencies allowed in PSA. All results are averages of ten runs.

Figures 6.4, 6.5, 6.8 and 6.9 show the efficiency and the number of passes (temperatures) of PSA for different number of nodes, N_H, for the four test cases. The notion of efficiency, or speed-up, is not precise here since the parallel algorithm deviates from the sequential one. However, it is still used as a measure of the parallelizability of the annealing algorithm. All figures show a decrease in efficiency when N_H increases and granularity decreases. Efficiency drops due to an increase in the relative cost of global summation operations and inter-node communication. It decreases more rapidly for FEM-2 because it has the smallest granularity. The curves of the number of passes do not show a uniform and consistent behavior. However, it can be seen in the four cases that, with the adaptive communication frequencies, there is no significant increase in the number of passes for larger N_H. Hence, the inconsistencies allowed in PSA do not lead to significant delays in the progress towards the final solution. But, the communication cost per pass increases with N_H.

PSA's solutions are given in Figures 6.6, 6.7, 6.10 and 6.11. Clearly, PSA's solutions are very close to those of sequential SA (N_H=1) in all cases except for the small granularity cases of FEM-2, which is still within a small fraction. Thus, the deviation of PSA from its sequential counterpart does not result in premature convergence and degradation in solution quality, as long as the grain size is not too small. Consequently, the solutions and the efficiency figures show that our scheme of annealed ν_{mvs} and adaptive ν_{S_v} and ν_{bdry} leads to both, preservation of SA's solution quality and acceptable efficiency values.

In addition to the parameters needed in sequential SA, PSA includes additional parameters, namely the communication frequencies. Although the adaptive communication scheme described above is adequate, these parameters make PSA somewhat less robust than sequential SA.

6.3. Concluding remarks

A parallel simulated annealing algorithm for data mapping has been presented. PSA deviates from sequential SA in order to achieve reasonable speed-ups, for reasonable grain sizes. This deviation leads to inconsistencies, which are corrected by communicating global and boundary information. An adaptive communication scheme, which makes use of the characteristics of PSA and the mapping problem, has been proposed for reducing communication cost while maintaining good quality solutions close to those of sequential SA. The experimental results support this assessment of the communication scheme.

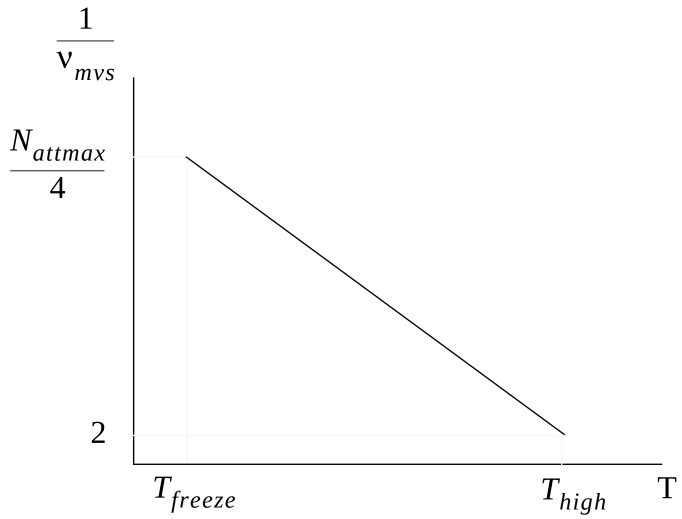


Figure 6.3. Variation of $(1/\nu_{mvs})$.

Another alternative for reducing inter-node communication cost would be the use of multicoloring, where nonadjacent vertices of the computation graph are assigned the same color to break up the computational dependencies among them. This allows ν_{bdry} to be one every $(|V_C|/N_H)$ attempted moves per node. However, it has been experimentally found that multicoloring produces solutions of lower quality for computation graphs with nonsmall vertex degrees because it restricts the probabilistic sampling of the data objects. Further, multicoloring did not show much improvement to PSA's speed-up for two reasons. The first reason is that the dominant communication cost for larger N_H is that of the global summation operations. The second reason is that at low temperatures adaptive ν_{bdry} values, described above, become smaller than what multicoloring offers.

Convergence of PSA is detected when no further progress is made. Progress refers to improvement in the concurrent efficiency of the best-so-far mapping configuration. Determining the best-so-far according to OF_{typ} involves a global operation. This is why it is only done at low temperatures. If the best-so-far does not improve for a number of annealing steps, say 10, then it is assumed that convergence has been reached.

6.2. PSA properties

In this section, the properties of PSA are experimentally investigated; the quality of its solutions, PSA's efficiency, and robustness are experimentally examined. Test cases *TEST2* to *TEST5* and the same experimental setting as that in SBPGA's experiments is used here. We give particular atten-

An outline of PSA is given in Figure 6.2. After the naive mapping step, the algorithm includes boundary communication and global summation steps, in addition to the local sequential SA procedure. The global summation of the number of attempted and accepted perturbations in all nodes is required to detect thermal equilibrium. At high temperatures, the number of accepted moves at one temperature is high and, thus, the number of elements that change in MAP[] at each attempt might be large. Therefore, the magnitudes of the three inconsistency types, described above, would grow rapidly if local SA steps proceed without correction. Corrections can be accomplished in two ways: a global sum operation to unify $S_v(p)$, for $p = 0$ to $|V_M|$-1, in all nodes, correcting the second inconsistency type, and inter-node communication of boundary information, correcting the third inconsistency. The first inconsistency type occurs randomly and contributes to an inherent "erroneousness" of PSA, mainly at high temperatures.

Obviously, it would be disastrous for PSA's speed-up to make the above-mentioned corrections at every attempted move or at a high frequency. On the other hand, low-frequency corrections would lead to degeneration. However, as temperature decreases the number of accepted moves, N_{acc}, decreases and, thus, the likelihood of inconsistencies also decreases; at low temperatures PSA approaches SA. This observation points to a remedy for the speed-solution dilemma, which is the use of an adaptive correction scheme. In this scheme, the frequency of global summation of the numbers of attempted and accepted moves, ν_{mvs}, is annealed, i.e. attenuated with temperature; the frequencies of updating $S_v(p)$, ν_{S_v}, and of inter-node communication, ν_{bdry}, are made adaptive to the number of accepted moves. Specifically, ν_{mvs} is decreased linearly from one communication every two attempts, at initial temperature, to a few times, e.g. four times, per N_{attmax} per node, at freezing temperature. The variation of the inter-communication period, $(1/\nu_{mvs})$, is depicted in Figure 6.3. ν_{S_v} equals one every few, e.g. two, accepted moves per node. That is,

$$\nu_{S_v} = \frac{1}{\Delta N_{acc}}, \text{ where } \Delta N_{acc} = 2N_H.$$

ν_{bdry} equals one communication every several, e.g. eight, accepted moves per node. That is,

$$\nu_{bdry} = \frac{1}{\Delta N_{acc}}, \text{ where } \Delta N_{acc} = 8N_H.$$

This value of ν_{bdry} makes use of the fact that the number of boundary objects, needed in other nodes, are only a fraction of the local grain size.

We emphasize here that these empirically derived estimates for ν_{bdry} and ν_{S_v} assume a reasonable grain size, $|V_C|/N_H$. We also note that these communication frequencies are decreased with temperature due to their dependence on the decreasing number of accepted moves. The experimental results below show that while such frequencies maintain reasonable speed-ups by allowing inconsistencies to occur in between corrections, these inconsistencies are corrected so frequently that the final solution quality is not degraded.

The design of PSA, discussed in this section, consists of steps, which address the potential sources of inconsistencies, for producing solutions comparable to those of sequential SA in acceptable execution times. The design strategy is based on frequently unifying the local views of the global state in order to prevent degeneration. Unifying involves inter-node communication, which reduces the speed-up of PSA. However, an adaptive scheme is devised below for reducing the communication cost.

The first question that arises for PSA is how to map MAP[] and the associated computation graph to N_H hypercube nodes, which is the same problem that PSA aims for solving in the first place. We have chosen a negligible-time naive mapping scheme, where MAP[] is split into contiguous segments, MS_0, MS_1, ... that are as equal as possible. The elements in MS_i and the corresponding computation subgraph are mapped to node *i*. Clearly, this mapping scheme is far from optimal and the speed-up for the PSA algorithm is sensitive to the numbering order of the data objects because it determines the amount of inter-node communication. For problems with large $|V_M|$, more 'intelligent' schemes whose complexity is of the order of ($|V_C|$) can be used, such as Farhat's domain decomposer [Farhat 88], for reducing communication cost and boosting PSA's speed-up.

```
Determine segment of MAP[] and computation subgraph mapped to my_node;
Determine inter-node communication information (nodes, boundary objects);
Generate random MAP[] segment;
Determine Initial temperature, T(0) (1 global comm.);
Determine Freezing temperature;
Communicate boundary information;
Global summation for $S_v(p)$ ;
while ( T(i) > Tfreeze and NOT converged ) do
    Determine $\nu_{mvs}$;
    while (not equilibrium) do
        Local SA step;
        Update #attempted and #accepted perturbations at $\nu_{mvs}$;
        Communicate boundary info at $\nu_{bdry}$;
        Update $S_v(p)$ at $\nu_{S_v}$ (global summation);
    end_while
    if ($N_{acc}$ < THRESH) then save best-so-far according to $OF_{typ}$;
    T(i) = $k$ * T(i-1)
end_while (end 1 pass)
```

Figure 6.2. PSA node algorithm for data mapping.

In this chapter, the PSA algorithm for data mapping is described, and its properties are experimentally explored. PSA has been implemented on an N_H-node NCUBE/2 hypercube.

6.1. Algorithm

PSA is based on performing sequential SA concurrently in different nodes of the N_H-node hypercube. Clearly, this algorithm is not faithful to the sequential SA because perturbations can occur concurrently and not successively. Hence, the local node (in N_H-node cube) view of ΔOF_{appr} due to a remapping (i.e. a perturbation), in the local MAP[] segment, of a data object from processor *p1* to *p2* is not always consistent with the global view. Figure 6.1 helps illustrate this discrepancy. It shows how data objects are mapped at some point in the annealing process, with $|V_M| = 2$ and $N_H = 2$.

Since ΔOF_{appr} depends on the sums, $S_v(p1)$ and $S_v(p2)$, and on the change in communication costs, $\Delta\zeta$, three types of inconsistencies can be identified in PSA. The first type occurs if two (or more) concurrent perturbations, in different nodes in the N_H-node cube, involve data objects *v1* and *v2* (or more) with MAP[*v1*] = MAP[*v2*]. The second inconsistency type is concerned with $S_v(p)$, for $p = 0$ to $|V_M|$-1, in different nodes. The third inconsistency type occurs due to local use of outdated information about nonlocal elements of MAP[], across node boundaries (refer to Figure 6.1), that are involved in $\Delta\zeta$. We emphasize, again, that these inconsistencies are due to the deviation from the sequential algorithm. If they are allowed to accumulate, they lead to degeneration. Inconsistency accumulation leads to either a convergence to a bad minimum or an increase in the number of passes needed to maintain reasonable solutions, causing a decrease in speed-up in the latter case.

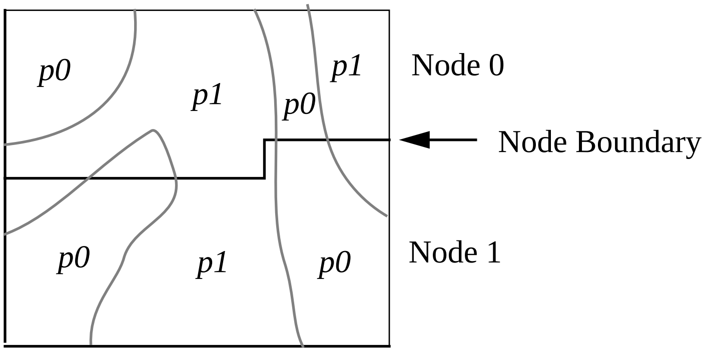


Figure 6.1. A mapping configuration at some annealing step; $|V_M| = 2$ and $N_H = 2$.

Chapter 6

Parallel Simulated Annealing Algorithm

The comparative results of Chapter 4 show that simulated annealing yields good quality mapping solutions. In this chapter, a parallel simulated annealing (PSA) algorithm is described. PSA is needed for faster mapping, especially for problems with realistic sizes. The PSA algorithm, presented in this chapter, uses OF_{appr} as energy function and, thus, ΔOF_{appr} for energy change. That is, PSA is based on the sequential SA1, and not SA2, since SA1 is considerably faster and because ΔOF_{typ} is more expensive to parallelize anyway.

As explained in Section 4.1, simulated annealing starts with some configuration represented by the data mapping vector MAP[], at a high temperature, and the goal is to find a configuration that minimizes OF_{appr}. SA is based on successive perturbations to the configuration. A perturbation, or move, is accomplished by a random change to MAP[*v*] in the range of processor numbers, 0 to $|V_M|$-1, where *v* is a randomly chosen computation graph vertex. Acceptance, or rejection, of a perturbation depends on temperature and ΔOF_{appr}, which involves $S_v(p)$ and $\Delta\zeta$.

The SA algorithm is very sequential, since the acceptance of a perturbation depends on the outcome of the previous ones. A number of strategies have been suggested for its parallelization with acceptable speed-ups [Baiardi and Orlando 89; Eglese 90; Greening 90; Roussel-Ragot et al. 91; Williams 86]. The strategy adopted in this dissertation is based on executing sequential SA concurrently and loosely synchronously in all processors of a multiprocessor, where the processors contain disjoint segments of MAP[] and the associated computation subgraphs. This strategy is called asynchronous in [Greening 90] and error SA in [Eglese 90]; we henceforth refer to it simply as parallel simulated annealing (PSA). It is adopted in this work because it is faster and more scalable than the other known strategies, at least for our application. This strategy has been applied to a VLSI problem in [Banerjee et al. 90] and to dynamic load balancing in [Williams 91]. Our design of PSA and its application to data mapping is an extension to the work of Williams [Williams 91]; we present a flexible and reasonable-cost adaptive communication scheme and include explicit discussion of the design choices made. In addition, we use a different convergence criterion for a reasonable quality-time trade-off. Also, diverse test cases, including 3-dimensional unstructured tetrahedral meshes, are employed for performance evaluations. Further, we do not use clustered perturbations. Instead, our choice for faster annealing is the exploitation of graph contraction, mentioned in Chapter 4 and elaborated in Chapter 9.

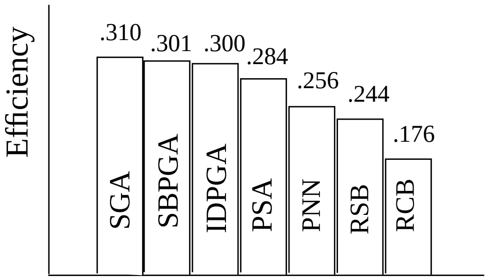


Figure 5.16. Solutions for *TEST2*.

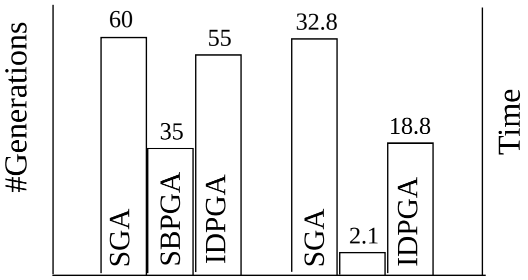


Figure 5.17. Results for *TEST2*.

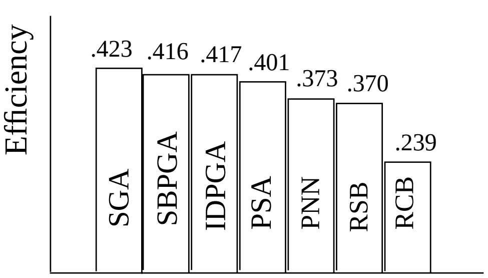


Figure 5.18. Solutions for *TEST3*.

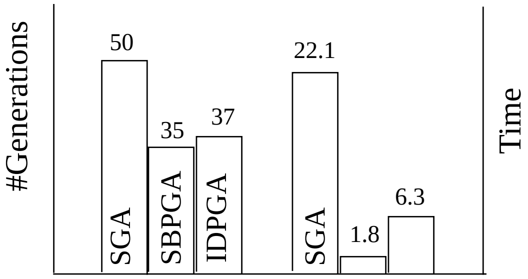


Figure 5.19. Results for *TEST3*.

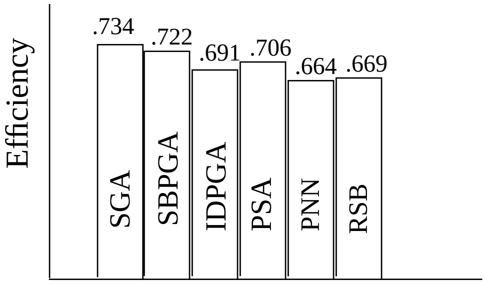


Figure 5.20. Solutions for *TEST4*.

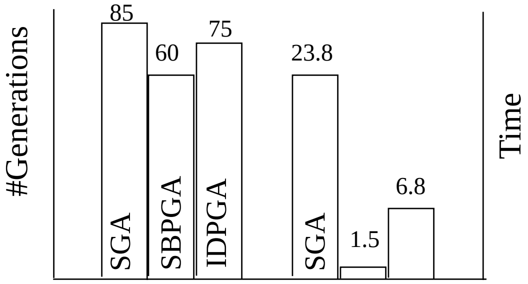


Figure 5.21. Results for *TEST4*.

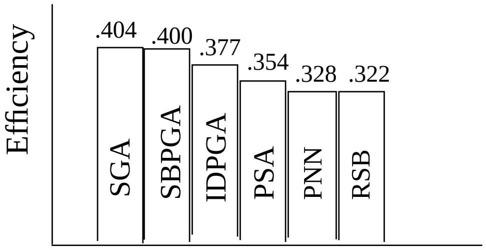


Figure 5.22. Solutions for *TEST5*.

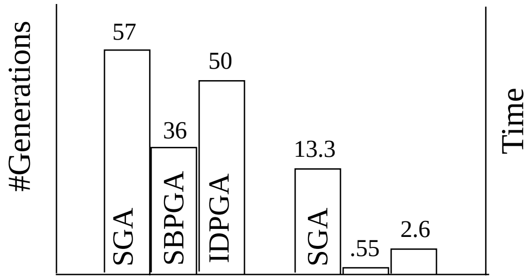


Figure 5.23. Results for *TEST5*.

SBPGA and IDPGA are fairly robust. Experimental experience has shown that their performance does not show much sensitivity to design and problem parameters. The intrinsic parallelism in the underlying evolution models provides a natural way for controlling the convergence of the evolving structures. Thus, the probability that the genetic search gets trapped in bad local optima is minimized, and the need for additional measures and parameters such as those used in SGA is obviated. However, the limit on their robustness stems from the finite range of choices for the design parameters, such as the length of the drift phase in SBPGA, and the neighborhood size in IDPGA.

Further work can be done to improve SBPGA and IDPGA. This would include the exploration of other CM configurations for IDPGA, corresponding to different geographic population distributions. Also, the chromosomes can be allocated to the CM columns of processors in a way that exploits the physical hardware for reducing communication cost. For example, chromosomes can be allocated to the 16-processor chips that form the nodes of the CM's cube, or contiguous segments of the chromosome can be allocated to the same physical processor. For SBPGA, fully asynchronous operation seems appealing. It would involve variable values for M and the use of different values for D in different demes accounted for by polling in every generation.

5.5. Concluding remarks

Two genetic algorithms based on natural population models and suitable for parallel implementation have been proposed. These are a new coarse grain MIMD PGA based on the shifting balance theory and a new fine grain SIMD implementation of a PGA based on the isolation by distance model. The two PGAs enjoy the property of intrinsic parallelism which leads to superlinear speedups, in comparison with SGA. Their application to the data mapping problem shows that they produce good sub-optimal solutions that are comparable to those of SGA, although IDPGA exhibits some decrease in solution quality due to the inconsistencies it involves in the hill climbing step. The two PGAs also show comparable robustness to that of SGA.

Since the focus of this work is on mapping data to MIMD multiprocessors, only SBPGA will be used for further investigation and comparison in the remainder of this dissertation; it is henceforth referred to simply as PGA.

rable solutions.

(ii) Neighborhoods of sizes different than 25 have been experimented with; results are averages of 10 runs. For a neighborhood size of 13, the efficiency is 0.951 found in 69 generations. For a larger size of 49, the efficiency becomes 0.933 found in 53 generations. These results show that a range of sizes are suitable for IDPGA; an appropriate neighborhood size would be in the range of 5 to 10 percent of the population size.

5.4. Further experimental results and discussion

In this section, more experimental results are presented for the two PGAs, followed by a comparative discussion of their performances which also involves SGA. *TEST2-TEST5* are considered. The performance measures are efficiency, for solution quality, and number of generations, for evolution time. SGA has been implemented on a SPARC 1+, SBPGA on NCUBE/2, and IDPGA on 16K processors CM-2. All results are averages of ten runs.

Figures 5.16, 5.18, 5.20 and 5.22 show the efficiency values produced by the three algorithms for the four test cases. For comparison, the results of recursive spectral bisection (RSB) and recursive coordinate bisection (RCB), from Chapter 4, are shown. Also shown are results of parallel simulated annealing (PSA) and parallel neural network (PNN) algorithms, which are described in the next two chapters. Clearly, the three genetic algorithms yield good suboptimal solutions which are superior to those of other methods. The only exception is for *TEST3*, in Figure 5.20, where IDPGA's efficiency is somewhat less than that of the annealing algorithm. SGA and SBPGA show comparable solutions. The solutions of IDPGA for *TEST1* and *TEST2* are also close to those of SGA and SBPGA; but, those for *TEST3* and *TEST4* are of lower quality. This is due to the inconsistencies resulting from the concurrent hill climbing implementation in IDPGA. That is, the quality decrease does not stem from the basic IDPGA evolution model. Instead, it is due to a component of the algorithm related to the specific application, i.e. data mapping.

Figures 5.17, 5.19, 5.21 and 5.23 show the numbers of generations and the actual execution times, in minutes, for the three GAs. It is clear that the parallel algorithms take a smaller number of generations than the sequential one, for evolving comparable solutions; SBPGA takes the least number of generations. In this sense, the two PGAs exhibit superlinear speedups; the intrinsic parallelism of the distributed population evolutionary models has the potential to evolve fit genotypes faster than the classic panmictic GA model. The actual execution times do not reflect the real speedup of the PGAs since they refer to different computers and are technology dependent; they are included as examples of real performance figures. It is worth noting here that the GAs are slower than the other lower quality mapping methods mentioned above, as is shown in later chapters, and for large problems, graph contraction still needs to be employed.

work, a neighborhood of individual *i* is formed of individuals within a certain distance in the X-Y plane, with individual *i* being at the center. An example of two overlapping neighborhoods, each with 13 members, whose centers are 3 unit distances apart is shown in Figure 5.15. Obviously, this PGA also enjoys intrinsic parallelism. A small neighborhood size enhances intrinsic parallelism and local differentiation and, thus, minimizes the possibility of premature convergence. Another advantage of small neighborhoods is smaller communication cost. However, a neighborhood should not be too small otherwise it might take a long time for the search to converge to reasonable solutions.

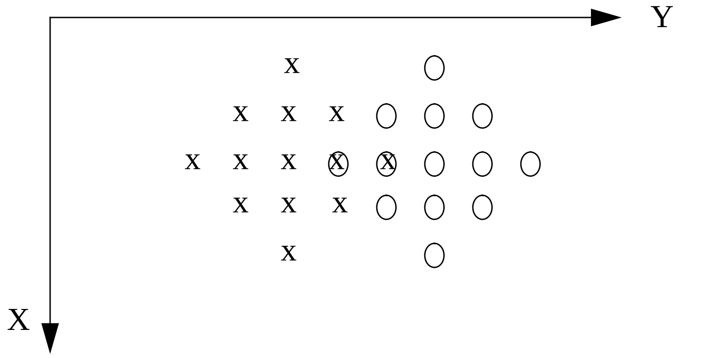


Figure 5.15. An example of two overlapping neighborhoods, each with 13 virtual processor members.

The choice of the fittest in the neighborhood as the second parent, half the time, in the reproduction step is a modification to the original isolation by distance model. Since the diffusion process is slow, this modification is justified for increasing the selection pressure locally and, hence, for speeding up the evolution of a solution. The better convergence caused by this modification is not expected to sacrifice the quality of the solution because of the small size of the neighborhoods, within which the fittest is sought, relative to the population size.

5.3.2. IDPGA properties

IDPGA has been implemented on a 16 K-processor CM-2. Its operation is tested here only for *TEST1*. More results are reported in the next section. The population size used is the same as for SBPGA, and the results are also normalized. The neighborhood size is chosen to be 25 unless stated otherwise. It is formed of the individuals that are within a distance of three units in the X-Y plane. The crossover and mutation rates are 1.0 and 0.004, respectively.

(i) The best mapping for *TEST1* corresponds to an efficiency of 0.967 found in 66 generations. The averages of 10 runs are 0.946 efficiency and 60 generations. IDPGA, also, exhibits superlinear speed-up with respect to SGA, since it takes a smaller number of generations for evolving compa-

In the reproduction step, the mating pool of each individual is restricted to a small local subpopulation, referred to as neighborhood. Each individual selects a mate from its neighborhood that is either a random member or the fittest, with equal probability. Selection is carried out in the (X,Y,0) plane and the result is, then, spread in the Z-direction. The local individual undergoes one-point crossover with the selected mate, where the crossover point is also determined in the (X,Y,0) plane. The second genetic operator, mutation at 0.004 probability, is a completely local operation for the random mutant genes. Then, hill climbing and fitness evaluation are carried out for the new individuals. Fitness is the total sum, in the Z-direction, of gene contributions. With the genes allocation described above, hill climbing can no longer be performed sequentially as in SGA. Instead, it is carried out concurrently, where boundary data objects concurrently attempt to be remapped according to their local view of ΔOF_{appr}. Since ΔOF_{appr} involves the global terms, $S_v(p)$, determined by the mapping of other data objects in the individual, concurrent hill climbing involves possible inconsistencies. Concurrent hill climbing has, sometimes, been found to cause a decrease in solution quality. However, this decrease is small, as will be seen below, and can a tolerable penalty for the increase in speed offered by the concurrent operation. Interestingly, this problem of inconsistency due to concurrency is similar to that arising from concurrent perturbations in parallel simulated annealing, described in the next chapter. After fitness evaluation, an elitism step is performed; the new offspring is forced to compete, 70% of the time, with its parent, and the fitter of the two survives. An important feature of IDPGA is that most steps involve CM communication operations. This is the price paid for exploiting the massive parallelism of the CM.

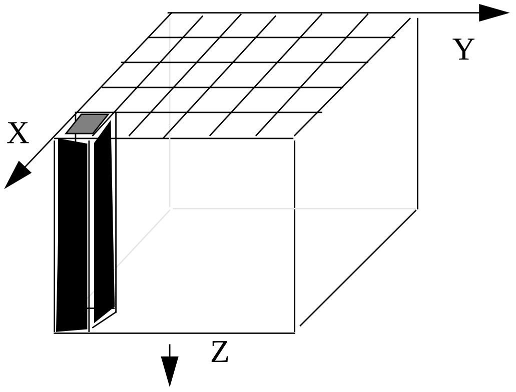


Figure 5.14. CM configuration for IDPGA.
(Individual = dark column)

Clearly, subpopulations overlap in IDPGA and, thus, the fitter genotypes spread throughout the population by diffusion. The neighborhood of an individual can be defined in several ways. In this for a graphics problem [Kosak et al. 91]. Its selection scheme is similar to ours; but, it does not fully exploit the massive parallelism of the CM. The CM has also been used to verify the superiority of local selection to panmictic selection [Collins and Jefferson 91]. The isolation by distance model has been employed in [Muhlenbein 89], [Gorges-Schleuter 89], [Laszewski 91] and [Spiessens and Manderick 91] for solving quadratic assignment, traveling salesperson, graph partitioning, and GA-deceptive problems, respectively. These PGAs have been implemented on Transputer based systems and on a mesh-connected DAP. In [Muhlenbein 89], the global fittest individual is included in all local subpopulations during selection in order to increase the convergence speed.

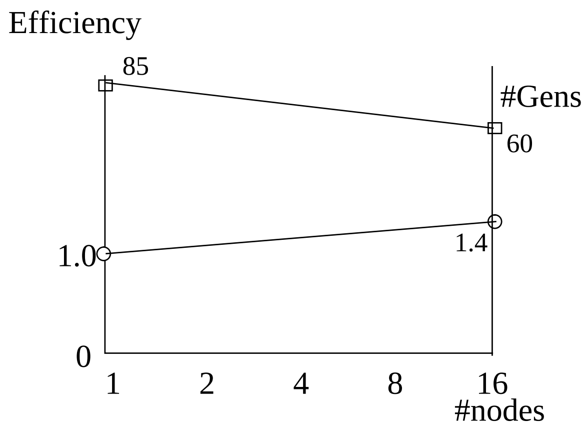
In this subsection, we describe an algorithm based on the isolation by distance model which exploits the massive parallelism of the Connection Machine and does not resort to global information. The algorithm is henceforth referred to as IDPGA.

5.3.1. Isolation by distance based PGA

An outline of IDPGA is shown in Figure 5.13. The CM is configured as a 3-dimensional shape, as shown in Figure 5.14, and the population is distributed as follows: the number of virtual processors in the X-Y plane equals the population size, and each chromosome is distributed over a column of virtual processors in the Z-direction, one gene per processor. Each new generation is created in a distributed fashion by having each column of processors replace a parent by its offspring.

```
Configure CM as a 3-dimensional shape;
Random generation of initial population (1 gene/proc);
Read input gene information (from computation graph);
Evaluate fitness;
for (gen=1 to maxgen) OR until convergence do
    Set $\mu$;
    Select from neighborhood {fittest OR random} in X-Y;
    SPREAD in Z-direction from (x,y,0) the location of selected mate
                                                            & xover point;
    Crossover (with mutation) the local individual with the selected mate;
    Hill climbing by offsprings (involves REDUCE comm.);
    Evaluate fitness (involves REDUCE comm.);
    Retain better of {offspring , parent} at 0.7;
endfor
Solution = Fittest.
```

Figure 5.13. Outline of IDPGA (CM-2 implementation).



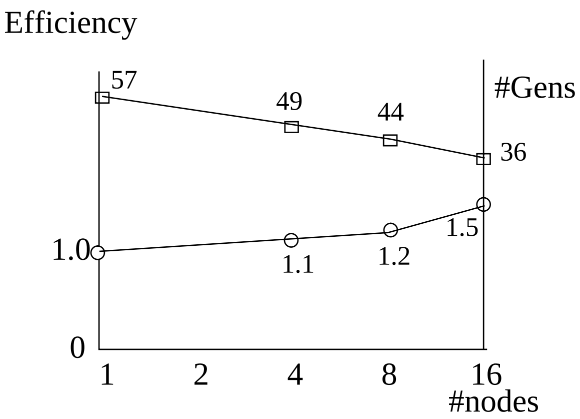
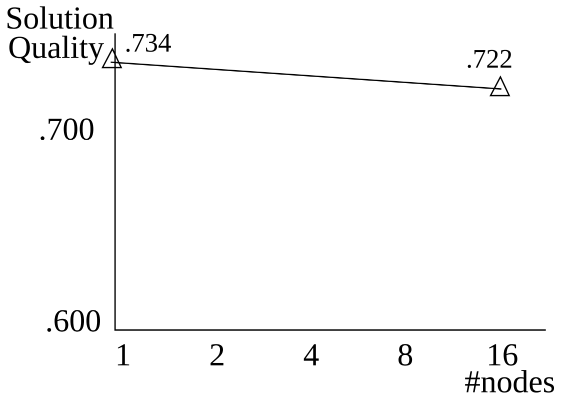


Figure 5.9. SBPGA efficiency and #generations for *TEST4*.

Figure 5.10. SBPGA efficiency and #generations for *TEST5*.



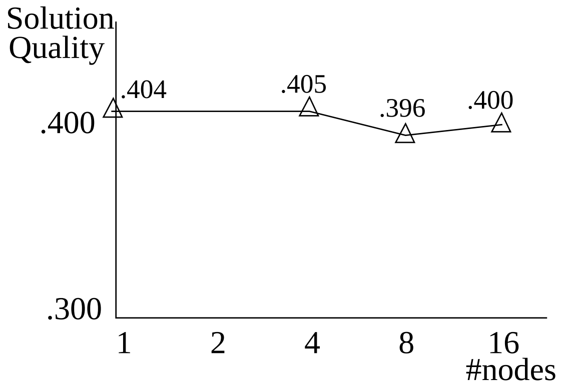


Figure 5.11. SBPGA solution for *TEST4*.

Figure 5.12. SBPGA solution for *TEST5*.

5.3. SIMD PGA

The fine-grain SIMD PGA described here is based on the isolation by distance model, where the population has a continuous and uniform distribution over a large area. It provides another computational model suitable for different computers and different applications, especially for mapping problems for fine grain SIMD distributed memory multiprocessors. This model lends itself to fine grain parallelism, which makes the Connection Machine, CM-2, an attractive choice for its simulation. A CM implementation of a PGA has appeared in [Robertson 87] for a classifier system. However, Robertson's work is based on panmictic selection and makes heavy use of the communication mechanisms of the CM. Another CM implementation has been independently developed

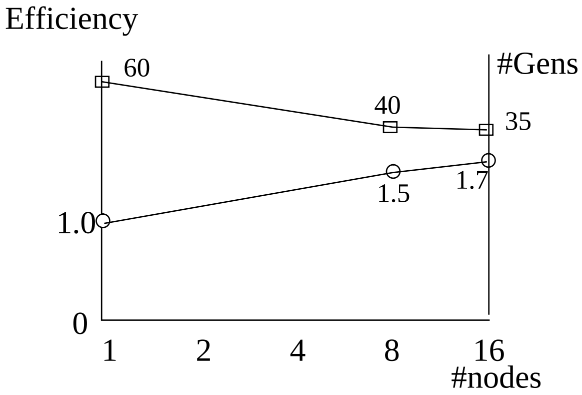


Figure 5.5. SBPGA efficiency and #generations for *TEST2*.

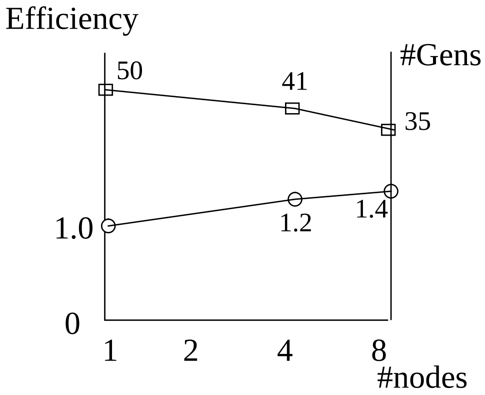


Figure 5.6. SBPGA efficiency and #generations for *TEST3*.

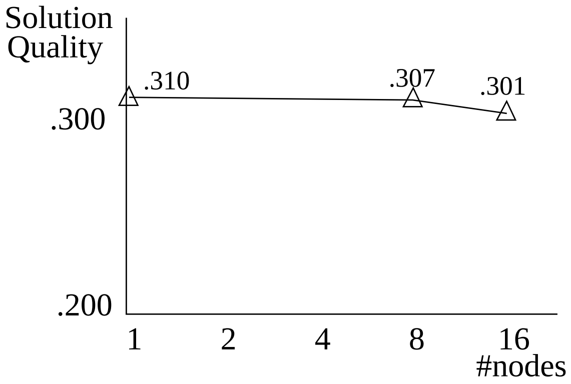


Figure 5.7. SBPGA solution for *TEST2*.

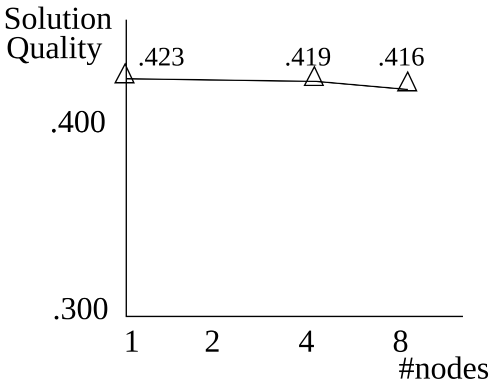


Figure 5.8. SBPGA solution for *TEST3*.

increase with N_H. This indicates that intrinsic parallelism tends to evolve good solutions in a shorter time, which is evident in the number of generations taken. Superlinear speed-up is clearly a property of SBPGA. As expected, inter-processor communication was found responsible for less than 3% of the time in the worst case attempted.

(iv) The quality of SBPGA's solutions for *TEST2* through *TEST5* are shown in Figures 5.7, 5.8, 5.11 and 5.12 as a function of N_H. They are close to the sequential solutions, although they show a small decrease in quality for the largest N_H. This decrease does not contradict the result of part **(i)**, because the design parameters were not tuned for the four test cases. We chose to favor general setting of parameters and faster execution to small improvements in solution quality.

(v) SBPGA has reduced sensitivity with respect to some parameters, such as operator frequencies, since it embodies another mechanism for controlling premature convergence. But D, M and the global convergence detection parameter are additional parameters that affect PGA's performance for different problems. The overall result is some decrease in PGA's robustness with respect to that of SGA.

is greater than or equal to 15, the search becomes slow. Therefore, D should be in the range 5 to 15 or, equivalently, 0.25 to 0.75 of the deme size. In Figure 5.4, the results of M=5% are surprisingly good but are generally unreliable. If M is greater than 40%, the selection pressure becomes too high, whereas a value less than 15% might not provide a sufficient shifting of control. It is concluded that 20% to 40% of the deme size is a suitable range of values for M. Moreover, it is intuitive that as D decreases, M should also drop to balance out the increase in the selection pressure.

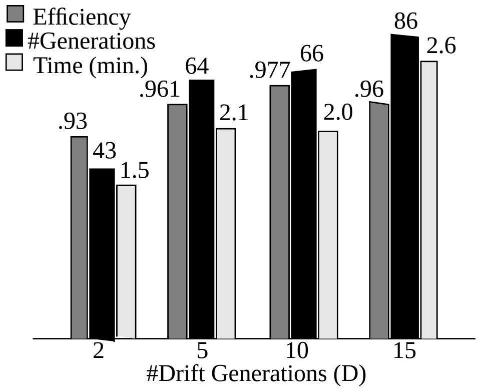


Figure 5.3. *TEST1* Results for different D (M=30%).

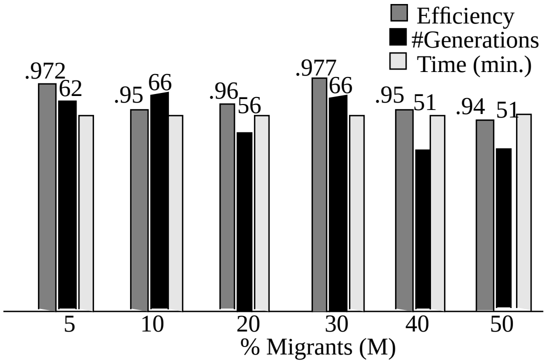


Figure 5.4. *TEST1* Results for different M (D=10, time=2.0).

(iii) For *TEST2-TEST5*: Figures 5.5, 5.6, 5.9 and 5.10 show the speed-up and the number of generations for different number of NCUBE/2 nodes, N_H and POP = 96. The notion of Speed-up is imprecise here, since the parallel algorithm deviates from the sequential one. However, it still serves as a measure of parallelizability of SBPGA. All figures show superlinear speed-ups, which

Table 5.1. Test cases of data sets to be mapped to hypercubes.

	Data Set ($\|V_C\|$)	$\|V_M\|$
TEST1	GRID1(301)	8
TEST2	FEMW(545)	16
TEST3	FEMW(545)	8
TEST4	GRID2(551)	16
TEST5	FEM2(198)	16

(i) For *TEST1* and deme size equal 20, the evolution of the solution in one deme is depicted in Figure 5.2. The step improvement in the efficiency value after the arrival of migrants from the fittest neighbor is clearly manifested at some points, such as generations 11 and 21. SBPGA finds a solution of 0.977 efficiency in only 66 generations. The averages of 10 runs are 0.96 efficiency and 65 generations. The time taken by the interdeme selection phase has been found to be 1.3% of that for the drift-mass selection phase, which shows that the communication overhead is almost negligible. For comparison purposes, the best solution found by SGA is also shown in Figure 5.2. It shows that although both algorithms evolve comparable solutions, SBPGA takes a smaller number of generations and enjoys superlinear speedup with respect to SGA.

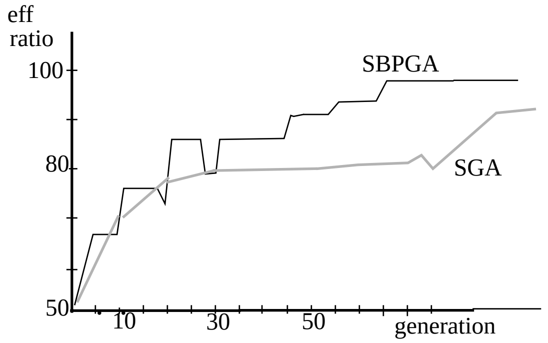


Figure 5.2. Evolution of the efficiency of mapping GRID1 in one deme.

(ii) For *TEST1*, the efficiency of the mapping, the number of generations required for evolving the mapping, and the time taken are illustrated in Figures 5.3 and 5.4 for different lengths of the drift phase and migration percentages, respectively. The values shown are the best of five runs. From Figure 5.3, it can be seen that if D is less than 5, the evolution model approaches that of a single mating unit and migrants increase the selection pressure, leading to premature convergence. If D and nodes become associated with each other. For example, the neighborhood of a deme is defined as the demes allocated to neighboring nodes, with direct physical connection. A sequential GA, such as SGA, is performed in each node for D generations as a simulation of the drift and mass selection phases of the deme's evolution. For this purpose, SGA can be simplified by removing features which are no longer necessary for maintaining diversity. These features include inversion and variable operator rates. Also, 1-point crossover and any acceptable selection scheme can be used instead of 2-point ring-like crossover and ranking. After a drift phase of D generations, one-way migration is carried out by allowing the demes with the higher adaptive peaks within their neighborhood to expand. Expansion is accomplished by sending copies of the M% (of the deme size) best individuals to the neighboring demes with lower peaks. It is assumed that limited resources are available for each deme and, thus, the M% least fit individuals in the receiving deme are replaced by the immigrants. Then, the drift-migrate cycle is repeated.

The assumption of limited resources prevents any growth in the deme size and, thus, averts an increase in the implementation complexity and time. The length of the genetic drift phase, D, is dependent upon the deme size and the parameters of the sequential GA that affect convergence, such as the rates of the genetic operators. A good choice for D has been empirically estimated to be about half the deme size. The number of migrants should be big enough to force the shifting of control to higher adaptive peaks; but not too big that it swamps fit genotypes in the receiving deme. Further, M should increase for longer drift phases. An empirical estimate of 20 to 40 per cent of the deme size seems to be adequate.

5.2.2. SBPGA properties

SBPGA has been implemented on iPSC/2 and NCUBE/2 hypercubes, for exploring some of its properties, namely its operation, parameters, solution qualities and speed-up. The mapping test cases considered are given in Table 5.1, where the data sets are shown in Figure 4.4 and are explained in Section 4.7. Again, we note that these test cases provide different geometry, granularity, spatial dimensionality, and graph connectivity. The solution quality is the concurrent efficiency. All results are averages of ten runs, unless stated otherwise. Fixed rates for the genetic operators are used. The number of drift generations, D, and the fraction of migrants, M, are half the deme size and 30%, respectively, unless stated otherwise. However, in the tuning stage of the search, D is halved to allow faster spreading of the genotypes produced by decreasing μ in ΔOF_{appr}. The results in parts **(i)** and **(ii)** are for iPSC/2 implementation. They use OF_{appr} for fitness and are normalized as in SGA's experiments, in Chapter 3, for comparing the two sets of results. The remaining results, in parts (iii) to (iv) are for NCUBE/2 implementation.

For these test cases, the solution quality, η (equation 2.7), uses $\zeta(p) = C_d(p)$, $\lambda = 12/\theta_{av}$ and $\rho = 5/\theta_{av}$.

a data mapping solution is sought, the bias towards better candidate solutions in the adaptive topography is appealing. Therefore, we conjecture that although the shifting balance model may not be the most general model for natural evolution, it has advantages for artificial evolution and that it is faster than PGAs based on other models. However, the implementation of SBPGA deviates from the theory of shifting balance because natural evolution is slow and aims at continuously producing fitter individuals. In artificial evolution, the objective is convergence to as good a solution as possible in a reasonable time. Hence, in our application, it is undesirable to have a long drift phase followed by interdeme selection in each shifting balance iteration in order to allow the fitter genotypes to spread throughout the population. Instead, the coverage of the whole population is accomplished over a number of shorter iterations.

```
Random generation of initial deme;
Evaluate fitness of this deme;
repeat
    /* Drift and local selection phases */
    for (D drift generations) do
        Perform Sequential GA;
        If Tuning Stage, set D = D_tuning;
    endfor
    /* 1-way migration phase (interdeme selection) */
    Find the highest fitness peak in the immediate neighborhood (incl. this deme);
    Exchange with neighbors the pair: (mynode, highest peak in my neighborhood);
    Save received pairs in nodelist[] , requestedlist[];
    if (mynode is in requestedlist[]) then
        Nonblocking send copies of M% migrants to requesting demes in nodelist[];
    endif
    if (mynode not contain highest peak in neighborhood) then
        Blocking receive M% migrants from the fittest (requested) neighbor;
        Replace M% weakest individuals with migrants;
    endif
until convergence
Solution = Fittest.
```

Figure 5.1. Outline of SBPGA (hypercube node algorithm).

An outline of SBPGA is presented in Figure 5.1 as a hypercube node algorithm. It assumes that the total population is evenly distributed as demes allocated to the nodes of a hypercube. Hence, demes

In contrast with the above-mentioned models, Wright considered a model where the population is distributed uniformly over a large area, but interbreeding is restricted to small distances [Wright 43]. Genetic divergence within the population takes place merely due to isolation by distance. Each individual has its origin at a particular place and its parents are drawn at random from a small neighborhood. Fitter genotypes are spread throughout the population by diffusion rather than migration. The size of the neighborhood and the shape of the habitat play an important role in the analysis of the model. For example, it has been shown that there is more local differentiation in linear than in two-dimensional habitats. Also, local differentiation increases with smaller neighborhood size.

5.2. MIMD PGA

The coarse-grain MIMD parallel genetic algorithm presented here is based on the shifting balance theory of evolution and is henceforth referred to as SBPGA. Previous Coarse-grain PGAs are based on other models of population structure. In [Pettey et al. 87], a PGA is presented for optimizing DeJong's functions. In this PGA, the population is split into subpopulations, and neighboring subpopulations exchange and insert into their local population the fittest individual in every generation. The distributed GAs in [Tanese 89] and [Cohoon et al. 91] share significant aspects with the stepping stone models. In these algorithms, subpopulations are assigned to the processors of a hypercube, and migration occurs periodically every epoch of generations. Migrants are exchanged among all neighboring processors. During a migration generation, subpopulations grow in size, and migrants are selected randomly in the originating subpopulations. After receiving the incoming individuals, the local population is reduced back to its original size. In [Tanese 89], the PGA is used for optimizing Walsh functions, whereas in [Cohoon et al. 91] it is applied to a VLSI problem.

5.2.1. Shifting balance based PGA

SBPGA inherits the favorable aspects of the shifting balance model of evolution. An important aspect is its intrinsic parallelism, which provides a suitable mechanism for controlling population diversity and convergence. Hence, it allows SBPGA to dispense with those features included in SGA for this purpose. The shifting balance model lends itself to an embarrassingly parallel decomposition, which makes it attractive for multiprocessor (e.g. hypercube) implementation; demes can be allocated to the processors of the multiprocessor and interdeme selection can be accomplished by migration between immediate neighbors. Another important consideration is that the time required for the drift and mass selection phases, associated with local computations, is much greater than that for the interdeme selection phase, associated with interprocessor communication, which makes the communication overhead small. Hence, the shifting balance model is quite suitable for multiprocessor implementation. Furthermore, the shifting balance model has been adopted in this work because it supports a constant drive towards higher fitness peaks. Since a rapid evolution of

Wright’s island model of population structure [Crow 86] assumes that the population is large and is split into semi-isolated subpopulations, called demes, dispersed geographically like islands, each breeding at random within itself. Each generation, a deme exchanges a fraction of its members for migrants drawn at random from the rest of the population. If their number is not too small, the migrants can be considered representative of the subpopulations in terms of allele frequency, and incoming alleles can be assumed to be independent. The mathematical analysis for this model has shown that the coefficient of genetic differentiation is predominantly determined by the amount of migration and is independent of the mutation rate and the total number of alleles [Crow 86; Hartl and Clark 89]. The analysis assumes that the mutation rate is much smaller than the proportion of migrants and that the population size is sufficiently large.

The island model is not likely to be realized in nature since the immigrants usually come from adjacent demes and, thus, are not a random sample of the species. Kimura’s stepping-stone models are based on the adjacency observation. These models assume certain geometrical patterns for the deme locations, such as linear arrays, rectangular grids and torroidal patterns [Hartl and Clark 89]. Migration is allowed only between immediate neighbors. These models are mathematically intractable. However, asymptotic solutions and numerical studies have shown that they share the property of genetic differentiation coefficient of the island model.

The shifting balance theory of evolution [Wright 77] presents another model of discontinuous population structure. The shifting balance process iterates through three phases. The first phase is the random genetic drift phase, in which the allele frequencies drift to some extent and, thus, the demes explore their adaptive topography. The second phase is for mass selection which permits the favorable gene combinations created in the first phase to rapidly become incorporated into the genome of the subpopulation by means of natural selection. Different demes now contain sets of allele frequencies that are likely to correspond, by chance, to various adaptive peaks with different heights. The third phase is for interdeme selection, where demes with higher fitness increase in size and shift the allele frequencies of adjacent demes by one-way migration until they come under the control of the higher fitness peak. The favorable genotypes become spread throughout the population in ever-widening concentric circles. In this fashion, larger parts of the adaptive topography can be explored, and a continual shifting of control from one adaptive peak to a higher one takes place. In Wright’s view, this model offers a good chance for the population to avoid being hung up on a low adaptive peak and to evolve novel types of gene interactions.

In contrast, Fisher argued against the shifting balance theory by suggesting that the adaptive peaks in multidimensional fitness landscapes are not very high and that they are connected by fairly high ridges, always shifting because of environmental changes [Crow 86]. Thus, the landscape is more analogous to waves and troughs in an ocean than to a static one. The alleles are selected because of their average effects and the population is likely to improve continuously.

problems such as the optimization of Walsh functions, the traveling salesperson problem...etc. These PGAs differ in the models they adopt for the population structure, in the mechanisms used for implementing some features of the models, and in the applications they are adapted to.

In this chapter, a new coarse-grain MIMD PGA and a new implementation of a fine-grain SIMD PGA are presented. The coarse-grain PGA is based upon a model of discontinuous population structure and the theory of shifting balance of evolution [Wright 77]; it has been implemented on a hypercube. The fine-grain PGA is based on the isolation by distance model of populations with continuous distribution [Wright 43]; it has been implemented on the Connection Machine, CM-2. The coarse-grain PGA offers faster convergence than do other coarse-grain PGAs, which is advantageous for data mapping and other applications. The fine-grain PGA provides a model that exploits the massive parallelism of the CM-2. The two PGAs incorporate SGA in a simplified form. They dispense with features included in SGA for precluding premature convergence. Such features are substituted for by the advantages of local reproduction and intrinsic parallelism in the PGAs. SGA here refers to GA2 (see Table 4.1), where OF_{typ} is used for fitness evaluation and ΔOF_{appr} for hill climbing. We note that although the two PGAs are applied to data mapping in this dissertation, they represent general GA models and are not restricted to only the mapping problem.

A brief summary of important models of natural populations is presented in section 5.1. In the following sections, the two PGAs are described, and their application to data mapping is experimentally demonstrated, compared and discussed.

5.1. Models of natural population structure

Populations of natural species are usually spread over a large area. Hence, they do not constitute single random mating units, as viewed by the classic GA [Holland 75], because the distance of individual movement would be much smaller than the entire distribution area of the population. The mating pool for selection is restricted to a certain range of distances and distant individuals would lie in different pools giving rise to some form of subpopulations. Associated with genetic drift, such population distribution leads to local differentiation in allele frequencies and to genetic divergence among subpopulations. Such geographic population structures can have profound effects on the evolution of species. In contrast with the case where the population is a single mating unit, variability across the populations persists and the problem of premature convergence is not encountered. Several models for population structures have been devised in population genetics. They involve various views for the subdivision of population and various schemes for intergroup selection and for genetic exchange or migration among the groups, i.e. subpopulations. Important and relevant models are summarized here. These models can be broadly divided into two categories according to whether the distribution of population is continuous or discontinuous.

Chapter 5

Parallel Genetic Algorithms

The time results in Chapter 4 show that SGA is very slow for practical problems even with hybridization or multiscale application. Hence, its parallelization is necessary, especially that the multiprocessor on which the parallel algorithm, ALGO, is to run is available anyway. However, SGA and classic GAs involve global dependences at both the population and chromosomes levels. The single mating unit model of reproduction, i.e. panmictic reproduction, leads to dependence among all chromosomes in the population. Also, the genetic operators and hill climbing lead to dependence among all genes in a chromosome. These vertical, among individuals, and horizontal, among genes, dependences make a straight parallelization of SGA inefficient, since interprocessor communication overhead would be substantial.

Fortunately, models of natural evolution offer a suitable solution for parallelization. Populations of natural species are normally distributed in various ways that confine reproduction to subpopulations, with interaction among subpopulations. Distributed population and local reproduction ameliorate the vertical dependency and confines the horizontal dependency to local individuals. Consequently, in addition to being more relevant for species in nature, these properties of natural evolution models makes them attractive and suitable for parallel simulation. Furthermore, it turns out that distributed population with local reproduction is a natural way for circumventing the problem of premature convergence, encountered in the implementation of classic GAs due to panmictic reproduction; panmictic reproduction can allow the exploitation aspect of the genetic search to dominate. Distributed population models enjoy intrinsic parallelism, which refers to concurrent and independent exploration by the subpopulations of many different regions in the adaptive topography. Intrinsic parallelism reduces the likelihood of premature convergence and, also, leads to faster evolution, in comparison with panmictic GAs.

Parallel genetic algorithms (PGAs) based on distributed population structures are easy to implement. Subpopulations can be allocated to the processors of a multiprocessor, and interactions among subpopulations can occur via the interconnection network. Different population structures can be modeled by different PGAs that are suitable for different parallel computers. A number of models for distributed natural population structures have been proposed in the population genetics literature [Crow 86; Hartl and Clark 89; Wright 43, 77]; important models are summarized in Section 5.1. Previous PGAs [Cohoon et al. 91; Laszewski 91; Muhlenbein 89; Pettey et al. 87; Tanese 89] share features with some of these models. Their operation has been demonstrated by solving

The annealing and genetic algorithms share the property of unpredictable convergence and, thus, execution time. Nevertheless, their execution times increase with the size of the problem and the multicomputer. The complexity expressions, mentioned above, serve only as indicators of the factors that determine the execution time. Although the bold neural network involves a probabilistic component, it has deterministic convergence.

The three PO methods, with their parameters chosen as described above, can be considered to be fairly robust, where robustness, in this dissertation, refers to insensitivity to design and problem parameters. Their robustness is enhanced by making some important parameters adaptive; these are the cooling schedule in SA, the operator frequencies in GA, and γ in BNN. In our implementation, they vary within a range of acceptable values. BNN is the most robust among the three methods. Interestingly, SA and GA have analogous sensitivities to their design parameters. Both the cooling schedule for SA and the frequencies of the genetic operators for GA affect the convergence speed and have been made adaptive in our implementation. The number of attempted perturbations at a particular temperature for SA and the population size in each generation for GA determine how many points in the solution space can be sampled. Both parameters have been empirically determined. However, GA has been observed to be somewhat less robust than SA.

4.9. Concluding remarks

Sequential versions of a Genetic Algorithm, a Simulated Annealing Algorithm and a Bold Neural Network for data mapping have been described. Their performances have been evaluated and compared for examples of various geometric shapes, dimensions and sizes. The solutions produced by these PO methods are good sub-optimal solutions. The PO methods are clearly competitive with recursive bisection methods, especially for 3-dimensional irregular and unstructured problems. However, they have diverse properties. SA2 produces the best solution quality, followed by GA2, NN2, SA1, and NN1 in the order of decreasing quality, for general problems. The order of decreasing execution time is the same as that for solution quality with NN2 and SA1 swapped in several cases. The PO algorithms are slower than recursive bisection. However, the execution times of NN1 and RSB do not differ greatly.

The annealing and genetic algorithms have the ability to start from partial information about the solution. This property results in a reduction in the overall execution time; the reduction is the biggest for GA. BNN and recursive bisection do not share this property. The applicability of the PO methods to realistic applications has been explored by using multiscale mapping. The results show that this strategy is advantageous for large problems because it leads to a significant reduction in execution time without sacrificing solution quality. It has been found that SA and GA make better use of graph contraction than does BNN. Concerning the robustness of the physical methods, BNN comes first, followed by SA; GA is the least robust.

Tables 4.5 and 4.6 and Figure 4.6 show results of multiscale mapping. These results show a remarkable reduction in time for all algorithms, the reduction for GA2 and SA1 being the greatest. Table 4.5 includes results for different values of κ. The efficiency values for CONT-GA2, CONT-SA2 and CONT-SA1 are consistent with those in Table 4.3 and κ can be as small as 2, leading to the greatest decrease in time without degrading the solution quality. However, for lower quality graph contraction, we suggest that κ should be greater. For CONT-NN1 and CONT-NN2, κ should be greater than or equal to 16 to maintain reasonable efficiency values. In this case, κ should be greater because NN1 only maps the contracted graph and does not share, with GA and SA, the flexibility of operating on the restored original graph. Also, when the solution quality of NN1 is low, local optimization in NN2 gets trapped in high local minima. CONT-SA2 still yields the best efficiency values followed by GA2, but the time difference has become more pronounced in favor of CONT-GA2. Further, unlike the case of the uncontracted graph, CONT-SA1 is, for most cases, comparable to CONT-NN2 in terms of time and efficiency values.

Table 4.7 includes results for other examples, with κ=16 for CONT-NN and κ=2 for CONT-SA and CONT-GA2. These results clearly support the assessments made above, based on Tables 4.3, 4.5 and 4.6. We note, however, that for GRID2 and FEMW, in Table 4.7, RSB yields better solutions than CONT-NN1 with comparable execution time. This is partly due to the use of contraction and because RSB performs well on 2-dimensional graphs. In the remaining paragraphs, overall evaluations are presented.

The solutions evolved by GA2, SA2 and NN2 are very good sub-optimal solutions. They are consistently better than those of recursive bisection. SA1 and NN1 also generate better solutions than recursive bisection for general, unstructured and 3-dimensional problems. For FEMW, for example, the improvements over RSB's solutions by NN1, SA1, GA2 and SA2 are 7%, 11%, 28% and 32%, respectively. The results for the various topologies and sizes indicate that the annealing and genetic algorithms are not biased towards particular problem topologies. Recursive bisection methods might favor 2-dimensional problems. The neural network performs better for 2-dimensional geometrical shapes, such as GRID1, than for 3-dimensional irregular structures, such as FEM2. But, it does not show a strong bias. Therefore, the PO methods seem promising for a variety of problems with different topologies and complexities. Interestingly, comparative studies of algorithms for another NP-complete optimization problem, VLSI placement, have given similar conclusions about the better solution qualities of annealing and genetic algorithms [Shahookar and Mazumder 91].

The better solutions of the PO algorithms come at a price; they are slower than bisection algorithms. For FEMW, the ratios of the execution times of NN1, SA1, GA2 and SA2 to that of RSB are 2, 16, 100 and 120, respectively. These ratios decrease to 1, 3, 20 and 60 when contraction is employed. SA2 is generally the slowest and NN1 the fastest. It is worth noting that although NN1 and RSB have identical complexity, NN1 is slower by a small factor.

4.8. Discussion

This section starts with a discussion of the individual Tables, 4.2 to 4.7, leading into overall evaluations of the results. The measures considered for assessing and comparing the performance of the algorithms are solution quality and execution time. In addition, bias and robustness are qualitatively discussed.

Table 4.2 shows the results of the versions of the PO algorithms that use the approximate objective function, OF_{appr}. Table 4.3 shows the results of the versions involving OF_{typ}. The results of RCB and RSB are also included. Figure 4.5 illustrates efficiency and time comparisons for FEMW. From the two tables and the figure, the following observations can be made. When OF_{appr} only is used, GA1 yields the best solutions, but at a high cost in terms of execution time. For GRID1, all the solutions are good. For FEMW, the quality of the solutions of the physical algorithms is clearly better than those of RCB and RSB, for a longer execution time. Nevertheless, the pronounced difference between the solutions of GA1, SA1 and NN1 themselves justifies the exploration of SA2 and NN2. Table 4.3 and Figure 4.5 show a clear improvement in the efficiency values with a substantial increase in time for SA2 and NN2. GA2 is more favorable than GA1. Also, due to its lower insensitivity to design parameters, GA1 is not pursued further. For FEMW, SA2 yields the best efficiency and is the slowest. NN2 produces smaller efficiency values than those of SA2 and GA2, but is the fastest of the three. It should be emphasized here that the difference in solution quality and execution time of SA2 and GA2 for loosely synchronous computations is mostly a result of the way OF_{typ} is used, explicitly by SA2 and only partially by GA2. This difference is not sufficient to evaluate the generic methods themselves. However, it highlights the importance of the formulation of the objective function for both solution quality and time.

In Table 4.3, results for FEM3 and FEM4 are given. Due to its symmetry and convenient number of points, FEM3 turns out to be an easy problem. The three PO algorithms find what seems to be an optimum. For FEM4, GA2 finds the best solution for the longest time.

The long time taken, especially by GA2 and SA2 in Table 4.3, justifies the exploration of the other versions, as in Tables 4.4, 4.5 and 4.6. Table 4.4 and Figure 4.6 give the results for the hybrid methods, which start with NN1 and continue with SA1, SA2, or GA2. It can be seen that starting from partial information about the solution leads to a reduction in time for SA1 and SA2 without degrading the final solution. The reduction in GA2’s time is more pronounced, at a small price in terms of solution quality due to restricting randomness. In comparison with Table 4.3 and Figure 4.5, SA2 is still the slowest, with the best efficiency; the quality of NN1-GA2’s solutions is only a little better than that of NN2 while still being slower. The ability of SA2 and GA2 to start from partial information about the solution is, nevertheless, an advantage over ab initio methods such as NN2, RCB and RSB. However, the times taken by NN1-SA2 and NN1-GA2 are still long and their decrease, as illustrated next, is of interest.

Table 4.4. Results of NN1-M hybrid versions (N=$|V_M|$).

Test case	Best Eff	NN1-GA2		NN1-SA1		NN1-SA2	
		%Eff	time	%Eff	time	%Eff	time
FEMW N = 16	0.338	90	15.96	82	4.82	96	28.38
GRID1 N = 8	0.85	94	0.91	90	0.60	95	0.91

Table 4.5. Results of CONT-M versions for FEMW & $|V_M|$=16.

	Best Eff	CONT-GA2		CONT-SA1		CONT-SA2		CONT-NN1		CONT-NN2	
		%Eff	time	%Eff	time	%Eff	time	%Eff	time	%Eff	time
K = 2	0.338	91	5.89	82	0.84	95	15.05	44	0.01	54	1.12
K = 4		92	9.17	85	1.10	97	17.63	47	0.02	63	1.44
K = 8		93	12.7	85	1.65	97	19.58	62	0.06	83	1.96
K = 16								73	0.18	86	2.21

Table 4.6. Results of CONT-M versions for GRID1 & $|V_M|$=8.

	Best Eff	CONT-GA2		CONT-SA1		CONT-SA2		CONT-NN1		CONT-NN2	
		%Eff	time	%Eff	time	%Eff	time	%Eff	time	%Eff	time
K = 2	0.85	93	0.18	91	0.21	96	0.36				
K = 16	0.85							74	0.03	92	0.10

Table 4.7. Results of CONT-M versions, κ=2 for GA & SA, κ=16 for NN (N=$|V_M|$).

Test case	Best Eff	CONT-GA2		CONT-SA1		CONT-SA2		CONT-NN1		CONT-NN2		RCB		RSB	
		%Eff	time	%Eff	time	%Eff	time	%Eff	time	%Eff	time	%Eff	time	%Eff	time
FEMW N = 8	0.452	92	2.92	88	1.61	96	7.15	76	0.05	90	1.50	53	0.04	81	0.10
FEM1 N = 8	0.569	93	0.27	86	0.06	93	0.24	83	0.09	85	0.10			84	0.03
FEM2 N = 8	0.578	92	0.26	86	0.06	97	0.43	81	0.05	89	0.10			76	0.04
FEM2 N = 16	0.432	92	0.81	80	0.18	96	0.85	78	0.12	83	0.13			74	0.07
GRID2 N = 16	0.787	92	1.42	77	0.28	94	1.94	73	0.20	85	0.30			84	0.16

Table 4.1. Summary of algorithms.

Version	Description
GA1, SA1, NN1	Basic algorithms using only OF_{appr}.
GA2	OF_{typ} for fitness, OF_{appr} for hill climbing.
SA2	SA1, then uses OF_{typ} and neighbor-perturbation.
NN2	NN1, then local optimization based on OF_{typ}.
NN1-M hybrids	NN1, followed by physical algorithm M.
CONT-M	PO algorithm M preceded by graph contraction for multiscale mapping (contracted graph size = $\kappa \lvert V_M \rvert$)

Table 4.2. Results of versions using OF_{appr} (N=$\lvert V_M \rvert$).

Test case	Best Eff	GA1		SA1		NN1		RCB		RSB	
		%Eff	time	%Eff	time	%Eff	time	%Eff	time	%Eff	time
FEMW N = 16	0.338	89	40.2	80	5.26	77	0.58	52	0.06+	72	0.16+
GRID1 N = 8	0.85	95	2.11	89	0.66	88	0.17	88	0.03+	90	0.06+

Table 4.3. Results of versions involving OF_{typ} (N=$\lvert V_M \rvert$).

Test case	Best Eff	GA2		SA2		NN2		RCB		RSB	
		%Eff	time	%Eff	time	%Eff	time	%Eff	time	%Eff	time
FEMW N = 16	0.338	92	32.81	95	36.54	89	3.64	52	0.06+	72	0.16+
GRID1 N = 8	0.85	96	2.41	94	1.03	93	0.21	88	0.03+	90	0.06+
FEM3 N = 8	0.490	100	1.42	100	0.51	100	0.07			96	0.03+
FEM4 N = 16	0.495	96	13.40	94	9.01	85	0.41			79	0.07+

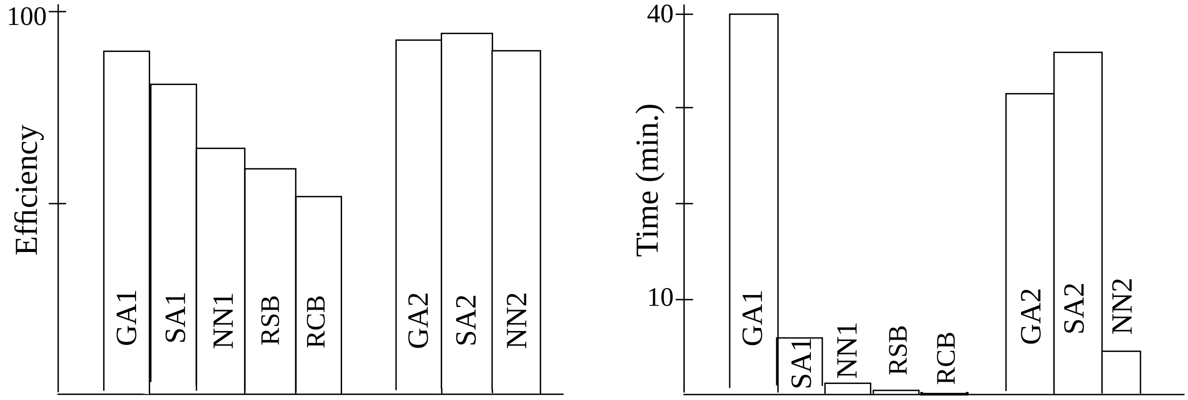


Figure 4.5. Results of versions involving OF_{typ} and OF_{appr} for FEMW and $|V_M|$=16.

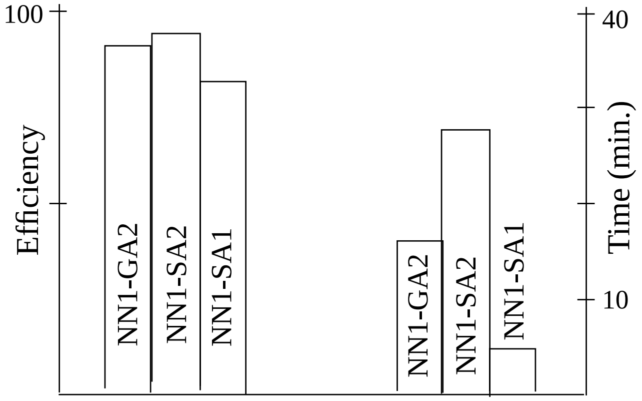


Figure 4.6. Results of NN1-M hybrid versions for FEMW and $|V_M|$=16.

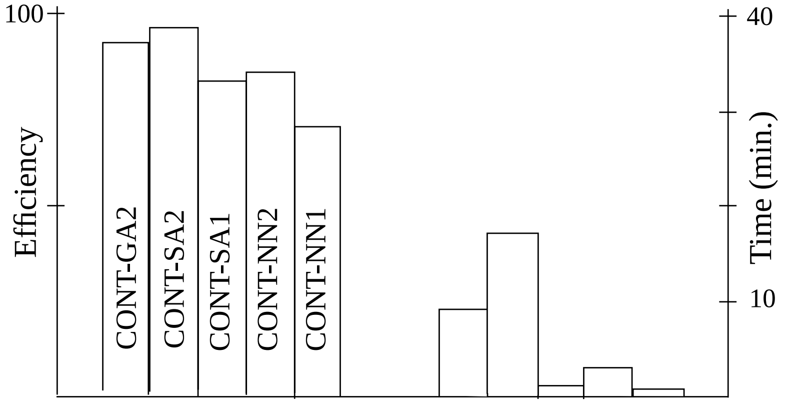
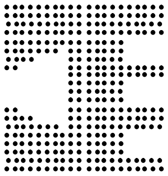


Figure 4.7. Results of CONT-M versions for FEMW, $|V_M|$=16.
κ = 4 for GA and SA, and κ =16 for NN.



GRID1

GRID2

FEM1

FEM3

FEM2

FEM4

FEMW

Figure 4.4. Data sets.

a reasonable solution at a low temperature. For GA's, the smaller contracted graph allows a smaller population size, POP, which can be kept the same in the subsequent generations after the restoration of the original graph.

Graph contraction itself is addressed in Chapter 9. For the experimental results reported next, NN1 is used for contraction.

4.7. Experimental results

The performance of the different versions is presented in this section. Data sets of different geometric shapes, dimensions, sizes and granularities are mapped to hypercube multiprocessors. The performance measures are efficiency (equation 2.7), with $\zeta(p)=C_d(p)$, and execution time. We assume that $\rho = 5/\theta_{av}$ and $\lambda = 12/\theta_{av}$; that is, the computation workload is not large and communication is not inexpensive.

The data sets used for testing are shown in Figure 4.4. GRID1 and GRID2 are 2-dimensional and yield computation graphs with a maximum vertex degree, θ_{max}= 4. GRID1 is a 301-point uniform, symmetric and irregular structure. GRID2 is a 551-point discretization of a broken plate, having a large variation in the spatial density of its points. FEM1 and FEM2 are 160-point and 198-point finite-element meshes, respectively, with θ_{max}= 12. FEM1 is 2-dimensional and nonuniform. FEM2 is 3-dimensional. FEM3 and FEM4 are 160-point and 340-point 3-dimensional structures, respectively, with θ_{max}= 8. FEMW is the most realistic of the seven examples; it is an irregular and unstructured tetrahedral finite-element discretization of an aircraft wing with θ_{max}= 16. Its size in these experiments is 545 points. We concentrate on FEMW and GRID1 because of their interesting and distinct properties.

Table 4.1 provides a summary of the algorithms considered. Tables 4.2 through 4.6 show results for these algorithms. The results for the most interesting case, mapping FEMW to a 16-node hypercube, are also given in graphical form, in Figures 4.5, 4.6 and 4.7. Each result of a physical algorithm is the average of ten runs. Each entry in the best efficiency column is the best result obtained from all runs carried out. The time given, in minutes, is for a SPARC 1+ workstation. For clarity, the efficiency figures are shown as percentages of the best efficiency which itself is kept as an absolute number. Since the optimum is not known, the best efficiency column serves as an indicator of how good the individual average figures are. To further illustrate the performance of the PO methods and to broaden the scope of comparison, Table 4.7 presents results for more test cases. Each result in Table 4.7 is the average of five runs. RCB's results are given only for FEMW and GRID1 whose coordinates were available. The execution time shown for RCB and RSB does not include the second mapping step because the annealing algorithm aimed for the best mapping and was not optimized for time. The "+" sign refers to this additional time.

that SA and GA take longer time to evolve solutions of the same quality as those of NN1 or the bisection methods. Therefore, hybrid methods which start with NN1, for example, and continue with GA or SA can be faster than pure GA or SA. In this work, NN1-GA and NN1-SA are explored. SA picks up at a low temperature which accepts uphill moves with probability 15%. GA creates its initial population by randomizing the boundary regions of the solution provided by NN1.

4.6. Multiscale mapping

Multiscale mapping also aims for reducing the execution time of the PO algorithms, especially for large problems. A multiscale mapping strategy consists of two phases: coarse-graph mapping and fine-graph mapping. Coarse mapping refers to mapping a contracted form of the computation graph. Then, the contracted mapping configuration is interpolated, producing a coarse mapping configuration of the original graph. In the fine mapping phase, the coarse mapping solution is evolved further by applying the PO mapping algorithms, GA and SA, again. BNN can also make use of multiscale mapping, with NN2's local optimization applied in the second phase. Methods involving multiscale mapping are henceforth referred to as CONT-M, where M is the method itself.

Graph contraction, utilized for multiscale mapping, is a preprocessing step in which edges in the computation graph are contracted and vertices are merged to form a multigraph whose super-vertices are weighted by the sum of the computational weight, $\theta(v)$, of the merged vertices and whose edges are weighted by the sum of the edges in the original graph. The level of contraction is determined by the parameter

$$\chi = log_2\langle\frac{|V_C|}{|V_C|_\chi}\rangle \tag{4.2}$$

where $|V_C|_\chi$ is the size of the contracted graph and $\langle X\rangle$ is the nearest higher power of 2 integer to X, or by the parameter

$$\kappa = \frac{|V_C|_\chi}{|V_M|} \tag{4.3}$$

the ratio of the sizes of the contracted graph and the multiprocessor. Following the mapping of the contracted multigraph to the processors, the original problem graph can be restored and more SA or GA iterations can be carried out for improving the quality of the solution. Contraction can also speed up BNN in the same way, However, κ should not be small, as discussed below.

Graph contraction, with parameter κ leads to big reduction in the search space of data mapping from $|V_M|^{|V_C|}$ to $|V_M|^{\kappa|V_M|}$, where $\kappa|V_M|$ is the size of the contracted graph and can be considerably smaller than the original size, $|V_C|$. The assignment of the contracted graph to the processors becomes a fast step. Subsequent SA iterations on the restored original graph, therefore, start with

4.3. Genetic algorithms

Two versions of SGA are explored in this chapter. Version GA1 employs OF_{appr} in both fitness evaluation and hill climbing. It also uses problem dependent user-defined parameters to invoke the tuning stage. A second version, GA2, uses OF_{typ} for fitness and OF_{appr} for hill climbing. It also includes automatic invocation, based on OF_{typ}, of the tuning stage, which increases robustness at the expense of a small fraction of solution quality and/or execution time for some problems.

4.4. Recursive bisection

Two recursive bisection methods are considered here to give some indication of the performance of the physical algorithms. These are orthogonal recursive coordinate bisection (RCB) [Berger and Bokhari 87; Walker 90] and recursive spectral bisection [Pothen et al. 90; Simon 91].

The operation of both methods is not guided by an objective function. Instead, RCB utilizes the physical coordinates of the vertices (data objects) of the computation graph to recursively bisect the graph into two subgraphs with equal sizes. In each bisection step, a direction (x or y) is chosen as a separator and directions alternate in successive steps. Data objects are sorted by coordinates in the selected separator direction and each half of the objects is assigned to a subgraph. The recursive process continues until the number of subgraphs equals $|V_M|$. The complexity of the RCB is of the order of $(|V_C| * log_2|V_M| (log_2|V_C| - log_2|V_M|))$.

RSB utilizes the properties of the Laplacian matrix associated with the computation graph. Briefly, each bisection step consists of computing the eigenvector corresponding to the second largest eigenvalue of the Laplacian matrix. The components of this vector provide distance information about the vertices of the graph. Then, the vertices are sorted according to the size of the eigenvector's components and split into two subgraphs accordingly. The complexity of this algorithm is of the order of $(\theta_{av} * |V_C| \sqrt{|V_C|} * log_2|V_M|)$ [Pothen et al. 90].

For a consistent comparison of the bisection methods with the PO algorithms, we have added a second step to map the subgraphs produced to the processors. The mapping step is carried out by a simulated annealing algorithm that minimizes OF_{typ}.

4.5. Hybrid algorithms

Hybrid algorithms aim for reducing the execution time of GA and SA. Hybridization is based on two observations. The first observation is that methods such as NN1, RSB or RCB yield solutions of lower quality than those of SA and GA, but are considerably faster. The second observation is cally equal to 2 [Fox and Furmanski 88].

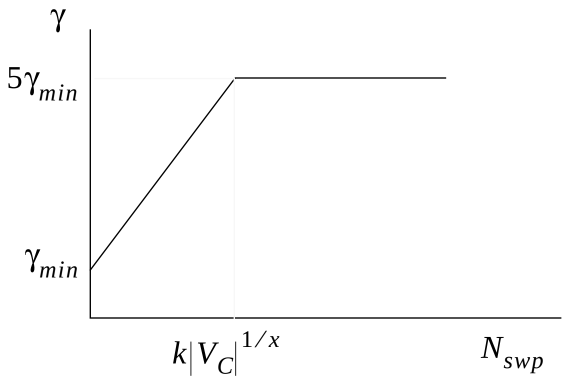


Figure 4.2. Variation of γ with the number of sweeps.

```
for i = 0 to ($log_2|V_M|$ -1) do
    Generate random spins s(v,i,t) over whole domain;
    repeat (for $N_{swpmax}$)
        Determine $\gamma$;
        for all spins in the domain
            pick a spin randomly;
            Compute s(v,i,t+1); /* equation (4.1) */
        end-for
    until (convergence)
    Determine bit i in the neurons;
end-for
```

Figure 4.3. Bold neural network algorithm for data mapping.

The BNN algorithm is summarized in Figure 4.3. Its complexity is of the order of ($\theta_{av}*|V_C|*|V_C|^{1/x}*log_2|V_M|$). This algorithm is henceforth called NN1. NN2 is a second version which includes a local optimization step for adjusting the boundaries of the mapping configuration produced by NN1. In this step, boundary data objects are transferred to neighboring processors only if OF_{typ} decreases. In most cases, the execution time of NN1 is much smaller than that of NN2. However, NN2 can, sometimes, improve the quality of NN1's solutions significantly and is included here for comparison with the genetic and annealing algorithms.

computation graph vertices (data) are mapped. That is, in each iteration, *i*, the current 2^{i} subgraphs are bisected and mapped to 2^{i+1} subcubes. The subsets of spins corresponding to the subgraphs are referred to as spin domains and are denoted by Φ_k, where *k* = 0 to 2^i-1. After the last iteration, the computation graph will be partitioned into $|V_M|$ subgraphs mapped to the processors. It is noteworthy that the neural representation used for BNN provides a natural way for removing ambiguities, such as placing the same object in two subdomains. Hence, it dispenses with the redundant synaptic connections that would have been required, in a generic Hopfield network, to enforce the problem constraints.

The fixed point of the network is associated with a minimum of the energy function, OF_{appr}. To determine the network equations, the neural variables are replaced by spin variables, *s*(*v*,*i*,*t*) = -1 or +1, and the energy expression is rewritten in terms of spin variables. Then, a standard mean field approximation technique, from physics, is used to derive the BNN formula for a spin update in domain Φ_k. A spin update equation from [Fox and Furmanski 88], which has been modified by including the computation weights $\theta(v)$, is:

$$s(v,i,t+1) = tanh\ \{-\alpha\ s(v,i,t)\ \theta(v) + \beta \sum_{s'} G(s,s')\ \theta(v') - \frac{\gamma}{D_{\mathrm{i-1}}} \sum_{s\ \in\ \Phi_k} s(v',i,t)\ \theta(v')\}, \tag{4.1}$$

where α, β and γ are appropriate scaling factors*; G* is the coupling matrix given by the computation graph; $D_{\mathrm{i-1}} = \Sigma_{v \in \Phi_k} \theta(v)$ is the size of the current computation subgraph (to be further bisected) to which *v* belongs, weighted by the degrees of the vertices; $v' \neq v$ in the third term refers to vertices only in the current subgraph, which corresponds to domain Φ_k.

The BNN formula can be interpreted in the light of magnetic properties of materials. At a critical temperature (Curie point), spontaneous magnetization domains of nearly-equal number of spins are formed in solids in such a way that spins within each domain are lined up, but have opposite direction to those in the other domain. In equation (4.1), the second term can be interpreted as the ferromagnetic interaction that aligns neighboring spins. The third term can be interpreted as the long-range paramagnetic force responsible for the global up/down spin balance. The first term is inserted in the BNN equation as a noise term that tries to flip the current spin and, thus, helps the system avoid local minima. The scaling factors have the following effects: α determines how stable a solution can be after a number of iterations, β determines the speed of the formation of the spin domain structure, and γ controls the spin balance in the configuration. In our implementation, $\alpha = \beta = 2$; β plays the role of inverse temperature, and its value is chosen to ensure that the system is near the critical point. γ is gradually increased, as shown in Figure 4.2, from γ_{min} = (the integer part of) $2\theta_{av}$ to $\gamma_{max} = 5\gamma_{min}$ when the number of sweeps, N_{swp}, equals $k|V_C|^{1/x}$ at every bisection level, where *k* and *x* are typically 1 and 2, respectively. With these values, it has been shown that the number of iterations required for the network convergence is a small number times the (1/*x*)-th root of the problem size, where *x* is the dimensionality of the problem structure, typiprobabilistically allowed. At each temperature, equilibrium is reached when the number of attempted, N_{att}, or accepted, N_{acc}, perturbations equals a predetermined maximum number. The maximum number of attempts, N_{attmax}, allowed is the larger of $|V_M|$ and θ_{av}, whereas the maximum number of accepted moves, N_{accmax}, is the smaller of the two. These choices secure a sufficient number of moves for thermal equilibrium while not spending too much time at high temperatures. The cooling schedule determines the next temperature as a fraction, *k*, of the present one. In our implementation, this fraction varies within the range 0.91 to 0.99 in a way that counteracts quenching (fast cooling) and speeds up cooling when possible; *k* increases if the number of accepted moves decreases, and vice versa. Since it is possible that SA finds a good configuration and then departs to a different region in the energy landscape, the best-so-far is always saved below a certain threshold for N_{acc}. SA is considered converged if $N_{acc} = 0$ or if no further progress is made for a number of annealing steps.

The complexity of the SA algorithm, or SA1, is of the order of (θ_{av}*$max\{\theta_{av},|V_M|\}$ *$|V_C|$*A), where *A* is the number of annealing steps. For adaptive schedules, *A* is problem-dependent, although of the order of *log*(initial temperature/ freezing temperature).

Two versions of SA are explored below. The first version, SA1, uses OF_{appr} for the energy until freezing. The second version, SA2, is identical to SA1 until the number of accepted perturbations reaches a fraction, THRESH2, of the initial number of accepts. Then, at low temperatures, OF_{typ} is used for the energy. Also, random perturbation is replaced by neighbor perturbation. That is, only the remapping of boundary data objects to neighboring processors is attempted, so that time is not wasted in random remapping. The computation of ΔOF_{typ} makes SA2 much slower than SA1.

4.2. Bold neural network algorithms

The Bold Neural Network (BNN), proposed in [Fox and Furmanski 88], is an improvement to the Hopfield and Tank model [Hopfield and Tank 86] applied to data mapping. It aims at quickly finding low minima for the system energy, OF_{appr}. It is built from $|V_C|$ rows and $log_2|V_M|$ columns of neurons. Each row corresponds to a vertex in G_C, and the number of neurons per row equals the number of bits for a processor label in a multiprocessor (hypercube). Each neuron has a neural variable, *n(v,i,t)* = 0 or 1, associated with it. The neuron's label *(v, i,t)* corresponds to vertex *v* and bit *i* of the processor label. Note that the label of a hypercube processor is given by $\Sigma(n(i)\cdot 2^i)$, where the summation is over *i* = 0 to ($log_2|V_M|$ -1). The neural variables represent the amount of local information about the solution at time *t*.

The network starts with random neural values and converges to a fixed point, after a number of sweeps, N_{swpmax}, over the entire computation graph. The BNN repeats this procedure $log_2|V_M|$ times, each time determining the bits in column *i* in the network and, hence, the subcube to which the perturbation is accepted and the new lower energy configuration becomes the starting point for the next perturbation. Zero change is also accepted. If the energy change is positive, corresponding to an uphill move, the proposed perturbation may be accepted with a temperature-dependent probability. The main advantage of this Monte Carlo algorithm is that the controlled uphill movements can prevent the system from being prematurely trapped in a bad local minimum-energy state.

```
Determine initial temp. T(0);
Initial configuration = Random data mapping MAP[];
/* Annealing - SA1 and SA2 */
while (T>THRESH1 and $N_{acc}$>THRESH2) do
    T = T(i);
    repeat
        Perturb(configuration);
        E = $OF_{appr}$;
        if (dE <= 0) then Accept; Update configuration;
        else rnd = random number (0,1);
            if (rnd < exp(-dE/T) then Accept and Update;
            else Reject;
    until (Equilibrium);
    if ($N_{acc}$<THRESH3) then save best-so-far;
    Determine $k$;
    T(i+1) = $k$ * T(i); /* cooling schedule */
end-while
/* Annealing at low temperatures - SA2 only */
repeat
    Anneal with Neighbor-Perturb(configuration) & E = $OF_{typ}$;
until (freezing or convergence)
```

Figure 4.1. A simulated annealing algorithm for data mapping.

An outline of a simulated annealing algorithm for data mapping is given in Figure 4.1, referred to as SA1. The initial data mapping is random. The energy of the system is given by the objective function used. The initial temperature is determined such that the probability of accepting uphill moves is initially 0.8. The freezing point is the temperature at which the probability of accepting a minimum energy increase, resulting from remapping a data object from an underloaded processor to an overloaded one, is very small, e.g. 2^{-30}. A perturbation, or a move, is accomplished by a random remapping of a randomly chosen data object. As explained above, a remapping that leads to a lower or identical system energy is always accepted, whereas an increase in the energy is only

Chapter 4

Comparison of Versions of Sequential PO Algorithms

Simulated annealing [Kirkpatrick et al. 83] and neural networks [Hopfield and Tank 86] have been adapted to the data mapping problem. Simulated annealing (SA) has been applied to several cases [Flower et al. 87; Fox et al. 88; Williams 91] and neural networks (NN) have been applied to some illustrative examples [Fox and Furmanski 88; Byun et al. 92]. However, SA continues to be of interest because of the design choices it involves, and applying NN algorithms to a variety of problems is useful for evaluating their performances. Further, there is a lack of comparative studies of SA, NN, GA and mapping heuristics.

In this chapter, we present annealing and neural algorithms for data mapping and explain our design choices. These sequential algorithms are the basis for the parallel algorithms proposed in Chapters 6 and 7. Also, the performances of sequential algorithms based on SA, NN and SGA are experimentally compared and discussed. Several versions of the three PO algorithms are involved in the comparison. These versions are based on the following modifications: making some parameters adaptive, modifying some steps to reduce execution time or improve robustness, employing different objective functions, adding postprocessing tuning steps, using hybrid techniques, and performing multiscale mapping. The goal of these modifications is to explore the applicability of the resultant versions to realistic examples and their suitability for problems with different requirements. To broaden the scope of evaluation, the results of two recursive bisection methods are also included in the comparison.

4.1. Simulated annealing algorithms

The simulated annealing approach is based on ideas from statistical mechanics and is motivated by an analogy to the physical annealing of a solid [Kirkpatrick et al. 83]. To coerce some material into a low-energy state, it is heated and then cooled very slowly, allowing it to come to thermal equilibrium at each temperature. The behavior of the system at each fixed temperature in the cooling schedule can be simulated by the Metropolis algorithm. An iteration of the Metropolis algorithm starts with proposing a random perturbation and evaluating the resultant change in the energy of the system. If the change is negative, corresponding to a downhill move in the energy landscape,

3.3. Discussion

The complexity of SGA and some design issues are discussed in this section.

It can easily be seen, from Figure 3.1, that the complexity of SGA is of the order of $(\mathrm{GEN} \cdot \mathrm{POP} \cdot |V_C| \cdot \theta_{av})$, where GEN is the number of generations and POP is the population size. $|V_C|\theta_{av}$ is a loose upper bound for hill climbing which considers boundary objects only. GEN is unpredictable, leading to unpredictable evolution time. POP is user-defined. It determines the number of building blocks in the population in different generations and, hence, influences the quality of the evolving solutions. It also impacts the evolution time. Therefore, a large POP may lead to better solutions; but, it is also likely to increase evolution time. It has been empirically found that POP can be made dependent on $|V_C|$ and $|V_M|$; a suitable range of values is 0.2*$|V_C|$ to 0.6*$|V_C|$, with the larger values corresponding to larger $|V_M|$.

Convergence detection and termination are, in general, nontrivial issues for GAs. In our application, they are resolved rather easily. Convergence coincides with the tuning stage of evolution whose invocation is determined by two quantities: the average Ψ in the population, as explained in subsection 3.1.2, and a threshold for the number of generations during which no improvement in the fittest individual is produced. Although these quantities are only approximate estimates of the state of the population, experimental work has indicated that their use provides an adequate compromise between improving solution quality and limiting the evolution time.

The trade-off between the solution quality and the evolution time is worth emphasizing. The genetic search can be made faster by resorting to measures such as, for example, increasing the selection pressure by some proportion as in SGA3. But, in such cases the solution quality is likely to be sacrificed, although at a smaller proportion. The range of values of 1.2 to 2.0 for the maximum rank in the selection scheme allows the user to choose the desired compromise between solution quality and execution time.

3.4. Concluding remarks

We have presented a genetic algorithm whose design constituents provide a good balance between exploration and exploitation forces for the data mapping problem. The design choices include elitist ranking selection, variable rates for the genetic operators, and a hybridizing hill climbing heuristic. These choices minimize the likelihood of premature convergence and improve the efficiency of the genetic search. Moreover, the genetic algorithm makes use of problem-specific information to evade some computational costs and to reinforce favorable aspects of the genetic search.

in only 66 generations, which is 61% of the time required by SGA to find a solution of the same quality. However, the large percentage of individuals (up to 20%) that die every generation, makes a maximum rank of 2.0 too high to be reliable, in general, for producing good solutions. This highlights the trade-off that exists between the solution quality and the search efficiency.

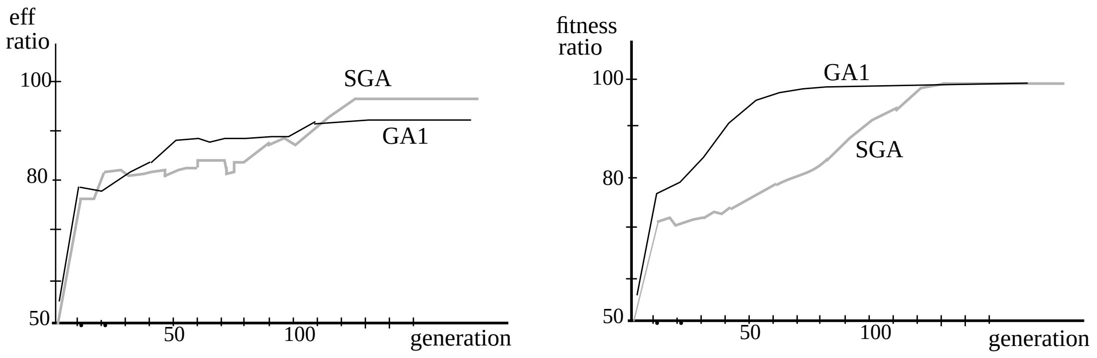


Figure 3.8. Comparison of SGA and GA1.

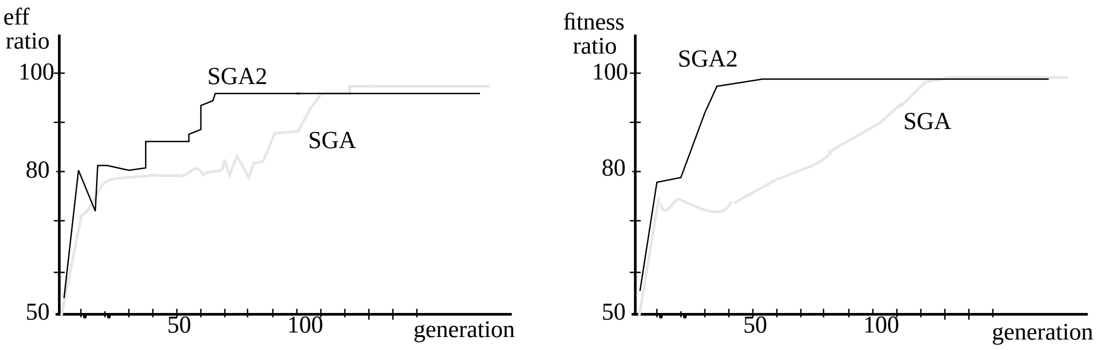


Figure 3.9. Comparison of SGA and SGA2.

(iv) Without hill climbing, the search efficiency deteriorates tremendously; SGA becomes more than ten times slower for GRID1.

(v) The amount of improvement in the solution quality acquired in the tuning stage of the evolution has been found somewhat sensitive to the parameter that triggers this stage. If tuning is triggered too early, the time allowed for the first two stages of the evolution might be insufficient for producing near-optimal building blocks. If the tuning stage is invoked too late, convergence to a local optimum might have already prevailed in the population as a result of the first two stages.

The three evolution stages can be identified in the fitness and load curves in Figure 3.7. Roughly, their overlapping points are generations 50 and 100. It can be seen that in the first stage, the communication load drops steadily regardless of the computation load, which happens to increase. In the second stage, both loads decrease, and the fitness rises. Decreasing μ in the tuning stage enhances SGA's tendency to reduce the computation load. If μ had not been decreased at this advanced stage, the efficiency would have been trapped at 0.89.

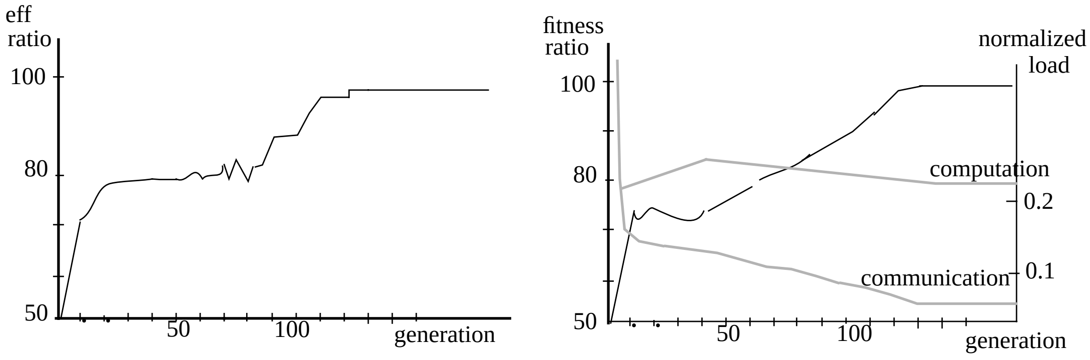


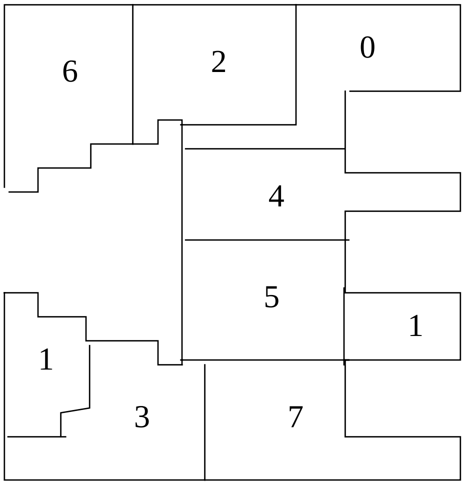
Figure 3.7. Efficiency and fitness for GRID1.

(ii) In Figure 3.8, SGA is compared with GA1 = (RSIS, 1-point, no, fixed, 300, -), where RSIS is Remainder Stochastic Independent Sampling fitness proportionate selection scheme implemented here with prescaling. RSIS allocates one reproduction trial for above-average individuals and treats the fractional part of the fitness to average ratio as a probability for allocation of trials. RSIS has been chosen due to its favorable properties of low bias and minimum spread [Baker 87]. Prescaling refers to stretching clustered fitness values and pulling together values that are far apart. The components of GA1 are those of a classic GA; however, it still includes hill climbing, for speed, and the problem-specific features in the tuning stage, for improving the final solution. The results are averages of 20 runs. Figure 3.8 shows that GA1 converges before generation 70 to an efficiency of 0.89. The efficiency is later improved to 0.925 under the effect of mutation and tuning. SGA takes 55 more generations to converge to 0.96 efficiency in generation 125. The best solutions found in the 20 runs were 0.942 and 0.972 by GA1 and SGA, respectively. The worst was 0.904 and 0.935 for GA1 and SGA, respectively. The mean square deviations were 1.18 for SGA and 1.1 for GA1. Clearly, GA1, without expensive sharing functions or crowding factors, results in higher selection pressure and lacks flexibility in controlling convergence. This explains the lower quality of its solutions and highlights the advantages of the combination of choices adopted in SGA.

(iii) The effect of increasing the selection pressure is also explored by increasing MRANK in SGA2 = (ranking, 2-point, yes, var, 300, 2.0). The best of five runs is shown in Figure 3.9. Clearly, SGA2 undergoes early convergence. Surprisingly, it finds a good solution, of 0.96 efficiency ratio, A genetic algorithm is considered a 6-tuple of variables: (REP, XOV, INV, OPRATE, POP, MRANK), where REP and XOV refer to the reproduction scheme and the crossover operator, respectively, INV indicates whether inversion is included, OPRATE indicates either variable or fixed rates for the genetic operators, POP is the population size, and MRANK is the maximum rank for the ranking-based reproduction scheme. Fixed rates for the genetic operators [Grefenstette 86] are 0.6 for crossover, 0.002 for mutation and 0.02 for inversion.

In all experiments, a solution obtained at a certain point in the evolution refers to the fittest individual in the respective generation. The performance measures are the efficiency, defined in equation (2.7), and the average fitness of the population. Both measures are plotted with respect to the number of generations, which, in turn, is used to assess the evolution efficiency. For clarity, the results are given as ratios; they are normalized with respect to OBJ_{opt} and EFF_{opt}. EFF_{opt} is a geometric estimate of optimal concurrent efficiency, explained in Appendix A. The results presented below are averages of 5 runs on a SPARC 1+, except for part (ii) which involves 20 runs.

(i) SGA = (ranking, 2-point, yes, var, 300, 1.2) finds a solution shown in Figure 3.6. The efficiency and fitness are shown in Figure 3.7, where the relative average loads of computation and communication are also depicted. After generation 118, the search converges to a solution with normalized efficiency 0.97. Figure 3.6 also includes processor loads: number of local computation graph edges and number of boundary vertices. Processor loads show that SGA does not strictly insist on assigning equal number of computations to processors. In contrast with classic bisection methods, it emphasizes the balancing of the combined computation and communication load, as required by the computational model.



Proc.	Comp.	Comm.
0	111	17
1	108	11
2	108	16
3	111	15
4	114	14
5	105	18
6	126	06
7	120	11

Figure 3.6. Mapping GRID1 to 3-cube and processor loads.

As pointed out in Chapter 2, the choice of μ in OF_{appr} and ΔOF_{appr} is of particular interest. Its value should be chosen in accordance with the properties of the SGA search. μ should be so large that the communication term in ΔOF_{appr} acquires sufficient importance in the clustering stage. But, it should not be too large, otherwise it will swamp the effect of the computation term in the later stages. In other words, μ should be chosen to favor the fitness of the individuals whose structure involves nearest-neighbor interprocessor communication in the clustering stage. In the later phases of the search, the value of μ should allow the emphasis to shift to the computation term in the fitness. A value that satisfies these requirements can be determined from the ratio of the computation and communication terms of OBJ_{opt}, which is derived in Appendix A.

In the tuning stage, hill climbing plays a distinctive role, where it fine-tunes the structures by adjusting the boundaries of the clusters assigned to the processors. In this advanced stage, the basic pattern of interprocessor communication can not be significantly changed, and the evolution ceases to offer significant gains. For these reasons, the emphasis upon balancing the computation load should be artificially increased for the purpose of facilitating boundary adjustment. This is achieved by decreasing the value of μ gradually from the fixed value used in the first two stages of the evolution to a small suitable value determined from the ΔOF_{appr} expression. The smallest useful value for μ is one that makes ΔOF_{appr} zero or negative when the following two conditions are both true: an overloaded processor has two objects more than the underloaded processor, and that the remapping of an object does not increase $\Delta\zeta$ much.

3.2. SGA properties

In this section the design choices of SGA are experimentally studied. The test case used is mapping GRID1, a 301-point grid discretization of an irregular structure shown in Figure 3.5, to an 8-processor hypercube. We assume $\zeta(p)=C_d(p)$, λ=12/θ_{av} and ρ=5/θ_{av}.

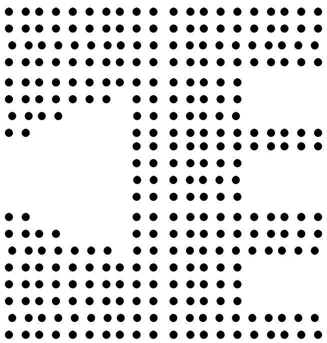


Figure 3.5. GRID1.

Inversion is used in the standard biological way, where a contiguous segment of the chromosome is inverted. In SGA, the chromosome is considered as a ring and the boundaries of the segment, to be inverted, are determined randomly. Inversion, at a low rate, helps in introducing new building blocks into the population for data mapping chromosomes, where nonadjacent alleles do interact.

3.1.7. Operator rates

Variable operator rates are useful for maintaining diversity in the population, and hence, for alleviating the premature convergence problem [Booker 87; Davis 89]. Rates are varied in the direction that counteracts the drop in diversity. Several measures have been suggested for the detection of diversity, such as lost alleles, entropy, percent involvement, and others [Baker 85; Booker 87; Goldberg 89]. The evaluation of these measures of diversity requires costly computations. In SGA, this cost is not incurred. Instead, the degree of clustering of objects is used to determine the variation in the rates. This design decision is based upon the observation that diversity is reduced in the population as Ψ increases. The current implementation uses a simple linear change in the rates. The rates ranges are: 0.5 to 0.85 for crossover, 0.004 to 0.01 for mutation, and 0.03 to 0.0 for inversion, where the first and last rates are associated with Ψ of the first generation and Ψ_{max} estimate, respectively.

3.1.8. Hill climbing

Individuals carry out a simple problem-specific hill climbing procedure that can increase their fitness. The procedure is greedy, and its inclusion improves the efficiency of the evolution significantly.

Hill climbing for an individual is performed by considering only the boundary data objects mapped to the processors. A boundary object is remapped from processor *p1* to *p2*, if and only if, the objective function (fitness) decreases (increases) or stays the same. To keep the computational cost of such a large number of incremental changes low, ΔOF_{appr} (equation 2.9) is used to decide about remapping, accentuating the importance of the locality property of OF_{appr}. From ΔOF_{appr}, it can easily be seen that remapping of data objects can only take place from overloaded processors to underloaded processors, considering combined computation-communication load.

The hill-climbing procedure enables qualified individuals to rapidly climb peaks in the adaptive landscape, which speeds up the evolution. This improvement in the efficiency of the search may seem to cause the exploitation feature to gain an upper hand over exploration, contributing to premature convergence. However, although hill-climbing does fuel the exploitation aspect of the genetic search, the experimental results do not reveal any negative effects. Hill-climbing enables exploration to be carried out in the space of genotypes representing local fitness optima.

3.1.6. Genetic operators

The Genetic operators employed in SGA are crossover, mutation and inversion. They are illustrated in Figure 3.4.

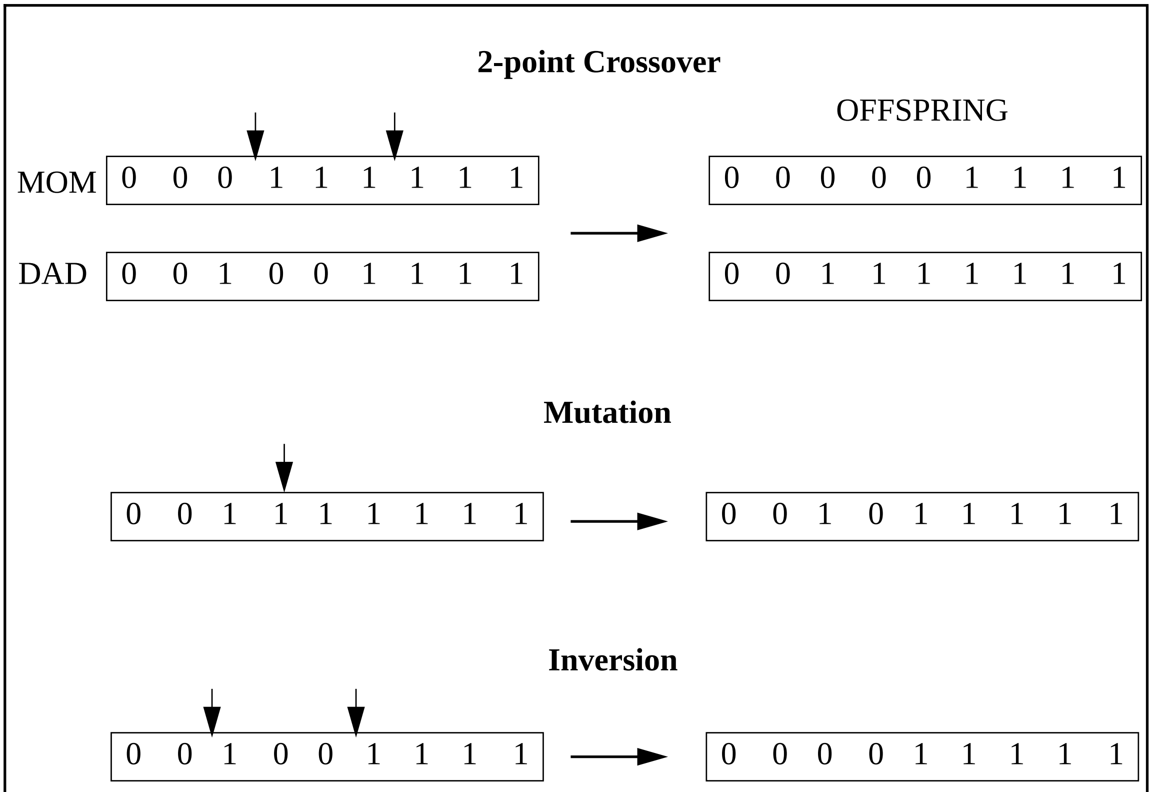


Figure 3.4. Genetic operators.

Two-point ring-like crossover is performed on a pair of individuals by swapping contiguous segments of genes. The segment boundaries are randomly selected and are the same in both parents. Two-point crossover is used because it offers less positional bias than the standard one-point crossover [Eshelman et al. 89]. Other more complex and, presumably, higher-performance crossover operators, such as uniform crossover [Syswerda 89], are not used in this work in order to avoid excessive computations.

The standard mutation operator is employed. It refers to randomly remapping a data object from processor *p1* to a random *p2*. For the reason explained in subsection 3.1.2, mutation becomes directed in the tuning stage of evolution, where the selection of *p2* is restricted to those processors which are already in the neighborhood of *p1*.

are first assigned single copies. Then, the fractional part of their ranks and the ranks of the lower half of individuals are treated as probabilities for assignment of copies. Figure 3.3 illustrates ranking selection in a population of 4 individuals, referring to the computation graph in Figure 3.2 and using equations (2.8) and (2.4) with $\lambda=\mu=\rho=1$ for fitness evaluation.

	Individ.	**Fitness**	**Rank**	**#copies**
0 0 0 1 1 1 1 1 1	0	0.019	0.93	1
0 0 1 0 0 1 1 1 1	1	0.022	1.2	2
0 0 0 0 0 0 0 0 0	2	0.012	0.8	0
1 1 1 1 1 1 1 0 0	3	0.017	1.07	1

Parents selected randomly: 0 and 1
3 and 1

Figure 3.3. Illustration of ranking based selection scheme.

Ranking based selection with maximum rank of 1.2 has been found to produce a survival percentage of 92% to 98% in different generations. It offers a suitable way for controlling the selective pressure, and hence, the inversely related population diversity [Whitley 89]. This results in the control of premature convergence, which is the main reason for using ranking-based reproduction in SGA. The advantage of controlling premature convergence by ranking outweighs the disadvantage due to ignoring knowledge about the relative fitness. Furthermore, ranking dispenses with prescaling, which is usually necessary for fitness proportionate reproduction schemes. From an efficiency point of view and for GAs based on the single mating unit model, ranking provides a computationally cheap method for controlling population diversity in comparison with expensive methods needed with fitness proportionate selection, such as sharing functions or DeJong’s crowding schemes [Goldberg 89].

Elitism in the reproduction scheme refers to the preservation of the fittest individual. In SGA, the fittest-so-far is inserted into the population, replacing the least fit individual, if it is better than the current fittest. The purpose of elitism and its current implementation is the exploitation of good building blocks and ensuring that good candidate solutions are saved if the search is to be truncated at any point.

3.1.3. Chromosomal representation

A data mapping configuration is encoded by a chromosome of $|V_C|$ genes. The value assigned to each gene, i.e. allele value, is an integer representing the processor to which a data object is mapped. The object is, therefore, the index (locus) of the respective processor (gene). An example is shown in Figure 3.2 for a computation graph of 9 vertices and 2-processor multiprocessor. The genotype (0,0,1,0,0,1,1,1,1) indicates that objects 0,1,3 and 4 are mapped to processor 0 and objects 2, 5, 6, 7 and 8 to processor 1.

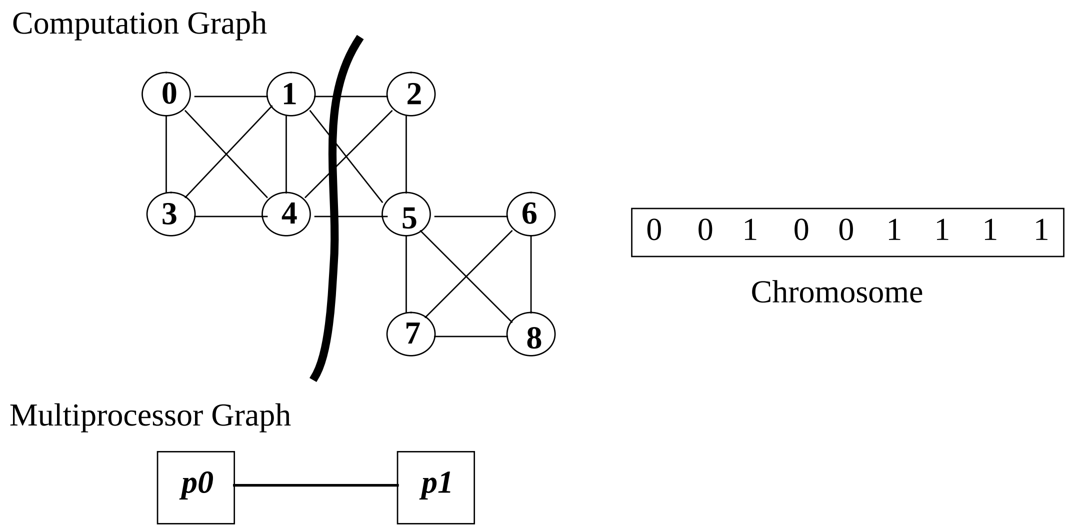


Figure 3.2. Example of chromosome encoding for data mapping.

3.1.4. Fitness evaluation

The fitness of an individual is evaluated as the reciprocal of OF_{appr}, so that maximum fitness corresponds to optimal data mapping.

3.1.5. Reproduction scheme

In SGA, the whole population is considered a single reproduction unit within which random selection, in such a case called panmictic, is performed. The reproduction scheme consists of elitist ranking followed by random selection of mates from the list of reproduction trials, or copies, assigned to the ranked individuals. In ranking [Baker 85], the individuals are sorted by their fitness values and are assigned a number of copies according to a predetermined scale of equidistant values for the population, not according to their relative fitness. In SGA, the ranks assigned to the fittest and the least fit individuals are 1.2 and 0.8, respectively. Individuals with ranks bigger than 1 verse of the total number of units of information that need to be exchanged by the processors, $1/(\Sigma_p\Sigma_q B(p, q))$. A smaller value of the average amount of communication is likely to imply better data mapping and higher Ψ. Thus, the maximum degree of clustering, Ψ_{max}, would correspond to good mapping configurations, assuming reasonable distribution of vertices among the processors. An estimate of Ψ_{max} is derived in Appendix A and is based on a simplifying geometric argument. It involves only the sizes of the computation graph and the multiprocessor.

The second parameter is an estimate of the value of an optimal objective function, OBJ_{opt}. It is based on the same geometric assumptions as Ψ_{max} and is explained in Appendix A.

3.1.2. Three stages of evolution

In the beginning of the evolution, the mapping of data objects to processors is random, and thus, the communication among processors is heavy and far from optimal, regardless of the distribution of the number of data objects. In the successive generations, clusters of objects mapped to the same processor grow gradually, the degree of clustering, Ψ, increases, the cost of interprocessor communication is constantly reduced, and fitness is increased. Then, at some point in the evolution, the balancing of the computational load becomes more significant for increasing the fitness. Therefore, two stages of evolution can be distinguished. The first stage is the clustering stage, which lays down the foundation for the pattern of interprocessor communication. The second stage is the computation-balancing stage. Obviously, the two successive stages overlap.

A third stage in the evolution can also be identified when the population is near convergence. In this advanced stage, the average Ψ of the population approaches Ψ_{max} and the clusters of objects crystallize. If these clusters are broken, the fitness of the respective individual would drop significantly, and its survival becomes less likely. At this point, crossover becomes less useful for introducing new building blocks, mutation of objects in the middle of the clusters is undesirable, and a fruitful search is that which concentrates on the adjustment of the boundaries of the clusters in the processors. This stage will henceforth be referred to as the tuning stage. Boundary adjustment can be accomplished mainly by the hill climbing of individuals, which is explained below, aided by mutation of boundary objects. The main responsibility of crossover becomes the propagation and the inheritance of high-performance building blocks and the maintenance of the drive towards convergence for the sake of search efficiency. For hill climbing and boundary mutation to take on their role in this stage, it is necessary to increase the relative weight of the computation term in the fitness function. This point is elaborated below with the description of hill climbing. To reduce the time taken by the tuning stage, the population size can be reduced by gradually eliminating some copies of identical individuals. The elimination of redundant individuals from the converging population also alleviates the excessive selection pressure of the dominant individuals and might help improve the final outcome of the evolution.

to the reproduction scheme and genetic operators. SGA is also hybridized by including a simple problem-specific hill climbing procedure. The objectives guiding these design choices are minimizing the likelihood of premature convergence for producing good quality solutions, reducing evolution time, and utilizing domain knowledge for satisfying the first two objectives. Further, SGA makes use of domain knowledge and problem parameters, such as μ and Ψ, to evade some computational costs and to reinforce some favorable aspects of the genetic search. In this chapter, SGA is described and its properties are experimentally explored using one of the test cases.

3.1. SGA design

SGA is outlined in Figure 1, and its components are described in the following subsections. In the first subsection, some design parameters are introduced. In Subsection 3.1.2, some observations about the evolution of data mapping configurations are made, as a prelude to the description of some of SGA's design choices.

```
Random generation of initial population, size POP;
Evaluate fitness of individuals;
repeat (for GEN generations)
        Set μ and rates of genetic operators;
        Rank individuals & allocate reproduction trials;
        for i = 1 to POP step 2 do
            Randomly select 2 parents from list of reproduction trials;
            Apply crossover, mutation, inversion;
            Hill climbing by offsprings;
        endfor
        Evaluate fitness of offsprings;
        Preserve the fittest-so-far (Elitism);
until (convergence)
Solution = Fittest.
```

Figure 3.1. Sequential genetic algorithm for data mapping.

3.1.1. Design parameters related to the objective functions

SGA makes use of two parameters related to the objective function. The first parameter is the degree of clustering, Ψ, of the computation graph vertices in a mapping configuration. Ψ is the in-

Chapter 3

Sequential Hybrid Genetic Algorithm

Genetic algorithms (GAs) are based on the mechanics of natural evolution [Holland 75; Goldberg 89]. In natural evolution, species search for beneficial adaptations to a changing environment. In GAs, artificial evolution takes place over successive, usually discontinuous, generations for solving a problem. Each generation consists of a population of chromosomes, also called individuals. Each individual represents a possible solution. The initial generation consists of randomly created individuals. Each consecutive generation is created by the individuals concurrently searching the adaptive topography. Firstly, individuals reproduce according to their fitness. Then, mates are selected and genetic operators are applied to create offsprings, which replace the parents. In this process, high-performance building blocks are propagated and combined to find fitter structures leading to optimal or near-optimal solutions. The parameters of this search strategy should be designed so that a balance between the exploitation of fitter structures and the exploration of the search space is secured for a sufficient number of generations.

GAs are powerful paradigms for solving optimization problems, such as data mapping. However, the implementation of GAs often encounters the problem of premature convergence to local optima; otherwise, a long time may be required for the evolution to reach near-optimal solutions. Techniques for overcoming the two problems of premature convergence and inefficiency are usually conflicting, and a compromise is required for applications like data mapping. This compromise amounts to balancing the exploration and the exploitation forces of the genetic search. A number of techniques, often dealing with only single design issues, have been proposed in the literature. Examples are: selection schemes for reducing the stochastic sampling errors [Baker 87], controlling the level of competition among individuals by prescaling, ranking, sharing functions or crowding factors [Baker 85; Deb and Goldberg 89], reduced-surrogate crossover for enhancing exploration [Booker 87], adaptive rates for the genetic operators [Booker 87; Davis 89], and incorporating problem-specific knowledge for directing the blind genetic search to the fruitful regions of the adaptive topography and improving the efficiency [Grefenstette 87; Davis 90]. The advantages of these techniques have been demonstrated by comparing the resultant performance with that of the classical GA [Holland 75]. Their performance verification has been carried out for DeJong's testbed of functions [DeJong 75] or for other specific applications, such as the traveling salesperson problem.

The sequential GA (SGA), described in this chapter, combines a number of design choices related

$$OF_{appr} = \lambda^2 \sum_{p \in V_M} S_v^2(p) + \mu \sum_{p \in V_M} \zeta(p) \qquad (2.8)$$

where μ is a scaling factor expressing the relative importance of the communication term with respect to the computation term. Values for μ are chosen according to a user defined ratio between the communication and the computation terms of a presumably good mapping solution. The derivation of such values is given in Appendix A. OF_{appr} does not take into account the concurrency in performing communication among processors. But, it still represents a good approximation to the cost of a mapping configuration and its minimization leads to a fairly balanced combined load distribution among the processors. Clearly, the first term is quadratic in the deviation of computation loads from the optimum, $\lambda\Sigma_p S_v(p)/|V_M|$, and is minimal when all deviations are near zero. A minimum of the second term means that the sum of all interprocessor communication costs is minimized.

The main advantages of the quadratic objective function, OF_{appr}, are its smoothness, its locality property, and that it is cheaper to parallelize. Smoothness makes it more suitable for optimization methods. Locality means that a change in the cost due to a change in the mapping of data objects to processors is determined by the remapped objects and the relevant processors only. Specifically, the change in OF_{appr} due to remapping of object *v* from processor *p1* to *p2* is given by

$$\Delta OF_{appr} = 2\lambda^2\theta(v)\,[\theta(v) + S_v(p2) - S_v(p1)] + \mu\,[\Delta\zeta] \qquad (2.9)$$

where $\Delta\zeta = \Sigma_p \Delta\zeta(p)$ is the net change in the communication term in expression (2.8). An algorithm is given in Appendix B for computing $\Delta\zeta$. Its computation using *B(p,q)* is not cheap due to the unsymmetry of *B(p,q)*. Since it is important to perform such a computation efficiently, boundary edges, also called crossedges and denoted by $E_b(p, q)$, can be used instead of boundary vertices. $E_b(p, q)$ is symmetric; together with a graph-dependent conversion parameter, $\Im_b$, it provides a good approximation to $B(p, q)$. This approximation is also explained in Appendix B. The locality property of OF_{appr} is very important for simulated annealing and genetic algorithms, since they extensively employ incremental remapping of objects. Also, the fact that ΔOF_{appr} is less expensive to parallelize than ΔOF_{typ}, a change in OF_{typ}, is important for our parallel algorithms.

Although OF_{typ} and OF_{appr} are used in this dissertation, we emphasize that the choice of an exact objective function depends on the computation model and the architecture and software of multiprocessor machines [Bokhari 90b; Ramanathan and Ni 89]. The choices and assumptions of the two objective functions are considered reasonable for loosely synchronous algorithms and typical multiprocessors. However, these choices are by no means restrictive and can be moulded according to the particular setting. In fact, an important property of the PO algorithms, especially the genetic and annealing algorithms, is their flexibility and adaptability to various classes of problems, algorithms and machines.

the weight *b* is the number of values per vertex to be communicated; *H(p,q)* is the physical (e.g. Hamming) distance between *p* and *q*; ρ is the machine time for communicating one word divided by t_{float}. $C_d(p)$ is a classic representation of communication cost and is relevant for early hypercube multiprocessors.

The second expression, $C_p(p)$, includes the effects of message latency and the number of processors that *p* has to communicate with, which makes it more reasonable for modern multiprocessors [Bokhari 90a; Hey 90]:

$$C_p(p) = \sum_{q \in V_M} [\rho B(p, q) + \sigma + \tau H(p, q)] \operatorname{sgn}(B(p, q)) \qquad (2.6)$$

where

$$\operatorname{sgn}(x) = \begin{cases} 1 & x > 0 \\ 0 & x = 0; \end{cases}$$

σ is the message start-up time divided by t_{float}; τ is the communication time per unit distance divided by t_{float}. We note that ρ, σ and τ are determined by the particular multiprocessor used.

Again, $C_d(p)$ and $C_p(p)$ are by no means unique or precise. For example, neither of them includes the effect of link contention, synchronization delays, or communication-computation overlap. However, they are considered to be reasonable choices for representing communication cost and are popular in the mapping literature. The two communication expressions are used in this work for exploring the properties of the mapping algorithms and their performance evaluation.

OF_{typ} is the basis for evaluating and comparing the solution qualities of the mapping algorithms. The solution quality is the concurrent efficiency corresponding to a mapping configuration, which is defined as

$$\eta = \frac{\sum_{p \in V_M} W(p)}{|V_M| \cdot OF_{typ}}$$

$$= \frac{\lambda \sum_{p \in V_M} S_v(p)}{|V_M| \cdot max_p \{\lambda S_v(p) + \zeta(p)\}} \qquad (2.7)$$

However, OF_{typ} is not a smooth function. Also, minimizing OF_{typ} gives rise to a minimax criterion which is computationally expensive, because the calculation of a new OF_{typ} caused by an incremental change in the mapping of data objects to processors may require the calculation of the loads of all processors. To avoid these two shortcomings, a quadratic objective function, OF_{appr}, can be used as an approximate cost for a mapping configuration:

$$OF_{typ} = max_{p \in V_M} \{ W(p) + C(p) \} \tag{2.1}$$

The minimization of OF_{typ} is difficult, because the optimization of *W(p)* and *C(p)* corresponds to conflicting requirements. Using physical analogy, computation graph vertices can be viewed as interacting particles. Minimizing computation workload, commonly referred to as load balancing, corresponds to a short-range repulsive force on the particles, causing them to spread throughout the multiprocessor. Minimizing communication cost corresponds to a long-range attractive force between interacting particles, causing them to coalesce and remain close to one another.

The computation workload, *W(p)*, for a processor, *p*, is given by

$$\begin{aligned} W(p) &= \sum_{v \in V_C} w(v) \cdot \delta(v, p) \\ &= t_{float} \cdot \lambda \sum_{v \in V_C} \theta(v) \cdot \delta(v, p) \\ &= t_{float} \cdot \lambda \cdot S_v(p) \end{aligned} \tag{2.2}$$

where $w(v)$ is the computation time per vertex v, $S_v(p)$ is the number of local computation graph edges in p, t_{float} is the machine time for an arithmetic operation, λ is the number of computation operations per computation graph edge per iteration, $\theta(v)$ is the degree of vertex v in the computation graph, and $\delta(v, p)$ equals 1 if vertex v is mapped to processor p and equals 0 otherwise. Both λ and $\theta(v)$ are determined by the particular algorithm used. λ is a constant expressing the number of values updated for the data objects in an iteration. $\theta(v)$ represents the number of computation operations required for updating a value for vertex v. θ_{av} and θ_{max} are used to denote the average and maximum vertex degree in G_C, respectively.

The amount of communication for a processor, *p*, is difficult to express accurately. It depends on several hardware and software components of a multiprocessor, which vary from one machine to another. Further, some of these components might be impossible to quantify. Let

$$C(p) = t_{float} \zeta(p) \tag{2.3}$$

so that all parameters can be normalized with respect to t_{float}. In this dissertation, we use two expressions for $\zeta(p)$. One expression, $C_d(p)$, is based on the physical distance between processors and the message size:

$$C_d(p) = \sum_{q \in V_M} \rho B(p, q) H(p, q) \tag{2.4}$$

where *B(p,q)* is the weighted number of vertices mapped to *p* and are boundary with *q*; that is,

$$B(p,q) = b \sum_{\substack{v, u \in V_C \\ <v,u> \in E_C}} \delta(v, p) \cdot \delta(u, q); \tag{2.5}$$

Chapter 2

Data Mapping Problem

Mapping data to multiprocessors aims for the minimization of the execution time of the associated parallel algorithm, ALGO. The execution time depends on the characteristics of the algorithm, the data set, the computation model and the multiprocessor machine. In this chapter, the characteristics assumed in this dissertation are presented and utilized in the definition of the mapping problem and in the formulation of appropriate objective functions.

Let $G_C = (V_C,E_C)$ and $G_M = (V_M,E_M)$ represent the problem graph and the multiprocessor graph, respectively. The vertex set, V_C, represents the set of data objects on which computations are to be performed. The edge set, E_C, represents the computation dependences among the data objects specified by the particular algorithm, ALGO, used. G_C can either be supplied by the user prior to program execution or derived automatically at runtime [Ponnusamy et al. 92]. It is henceforth referred to as the computation graph, and the two terms, data objects and computation graph vertices, will be used interchangeably. When performing computations in parallel, G_C also contains information about interprocessor communication. The vertices of the multiprocessor graph, V_M, refer to the processors, and the edges, E_M, refer to the physical interconnections.

The data mapping problem is an optimization problem that refers to determining an onto (many-to-one) function,

$$MAP : V_C \to V_M,$$

such that an objective function, associated with the execution time, of ALGO, is minimized. A solution that satisfies the minimization criterion is an optimal mapping. Mapping results in partitioning the computation graph into subgraphs allocated to the processors of the multiprocessor. The array MAP[*v*], for *v* = 0 to $|V_C|$ -1, is henceforth used to represent a mapping configuration, where MAP[*v*] = *MAP*(*v*) is the processor number, in the range 0 to $|V_M|$ -1, to which vertex *v* is mapped. To formulate objective functions for the mapping problem, a loosely synchronous data parallel computation model is assumed, where processors perform computations on their allocated subgraphs and then communicate with other processors to exchange boundary vertex information, in each compute-communicate iteration. The total parallel execution time is determined by the slowest processor. Thus, a typical objective function, OF_{typ}, representing parallel execution time is equal to the maximum combined workload of computation, *W(p)*, and communication, *C(p)*, for a processor, *p*, in a loosely synchronous iteration. That is, to be an efficient step for large-scale problems.

This dissertation is organized as follows. Chapter 2 describes the problem formulation and objective functions which guide the operation of the PO algorithms. Chapter 3 presents a sequential genetic algorithm for mapping and investigates its properties. This algorithm forms the basis for the parallel genetic algorithms. Chapter 4 contains comparative experimental evaluations of several sequential versions of the three PO algorithms and two recursive bisection heuristics. The aim is to explore the properties of these sequential versions and their applicability to different problem topologies. Chapter 4 includes a brief review of annealing and neural algorithms for mapping. These reviews and the results of the comparisons provide a basis for design choices made for the parallel PO algorithms and the mapping scheme used for large problems in the next chapters. Chapter 5 presents coarse grain and fine grain parallel genetic algorithms and explores their properties. The two parallel genetic algorithms suit MIMD and SIMD computational models. Chapters 6 and 7 describe an improved parallel simulated annealing algorithm and a parallel neural network algorithm, respectively, and discuss their properties. Chapter 8 provides a comparative experimental evaluation of the performances of the three parallel genetic (MIMD), annealing and neural algorithms for small to moderate problem sizes. The evaluations are conducted for a variety of algorithm and machine parameter values to investigate bias and applicability of the PO algorithms. In all the tests, recursive spectral bisection is used as a reference. Chapter 9 presents efficient parallel pre-mapping graph contraction algorithms for large-scale problems. The results in Chapter 9 show remarkable saving in mapping time, for large problems. Chapter 10 contains conclusions and suggestions for further research.

to the data mapping problem. PO algorithms are more promising for general applicability than heuristic algorithms and their parallelization is important for fast execution. The performances of the proposed PO algorithms are critically evaluated and compared using several test cases of different geometric shapes, dimensionality, sizes and granularities. Their performances are also compared to recursive spectral bisection, which is a representative of good quality heuristics. The main performance measures are mapping quality and mapping time. In addition, the properties of bias, robustness, and scalability are investigated. Furthermore, we propose graph contraction algorithms to allow the application of mapping methods to large problems. Graph contraction leads to significant reduction in the mapping time. Experimental results show clearly that the PO algorithms produce good sub-optimal mapping solutions. Such good quality solutions are maintained by the PO algorithms with graph contraction, although the mapping time is substantially decreased.

This work concentrates on mapping loosely synchronous computations with irregular data sets. The distributed-memory multiprocessors used for mapping are assumed to have a hypercube topology with MIMD message-passing operation [Duncan 90]. Nevertheless, the PO algorithms have more general applicability and are not restricted to these conditions. Different conditions can be accounted for only by modifying the objective function guiding the operation of the PO algorithms. This work constitutes a part of a broader automatic parallelization effort, the Fortran D programming system [Fox et al. 90]. In the Fortran D system, we are interested in including a number of data mapping schemes that suit a variety of problem and multiprocessor topologies. High quality data mapping is needed for irregular problems. Also, both, ab initio mapping and adaptive refinement of existing mappings, need to be addressed in the Fortran D system. Further, MIMD and SIMD parallel machines with a variety of topologies and communication mechanisms will be targeted.

The main contributions of this dissertation can be summarized as follows:

(a) Adaptation, for the first time, of genetic algorithms to the data mapping problem. Sequential as well as MIMD and SIMD parallel genetic algorithms are presented. The parallel genetic algorithms also serve as general paradigms for solving other optimization problems.

(b) Development of an improved parallel simulated annealing algorithm for data mapping.

(c) Development of a parallel neural network algorithm for data mapping.

(d) Comparative performance evaluation of the sequential and parallel PO algorithms.

(e) Development of efficient pre-mapping graph contraction algorithms that not only make the application of the PO algorithms to large problems practical, but also allow mapping ment steps to balance the computational weights. A greedy technique for clustering starts with a data object and keeps adding other neighboring objects to the cluster until the required size is reached [Farhat 88]. Other geometry-based heuristics use versions of the above-mentioned methods for partitioning the data set, followed by iterative improvement algorithms or distance-based heuristics for mapping partitions to processors [Chrisochoides et al. 89, 91a, 91b; Farhat 89; Houstis et al 90]. Clearly, most of these heuristics divide the mapping process into two steps: data partitioning or clustering, then assignment of data clusters to processors.

Heuristic algorithms are fast. But, many of them do not offer a balanced emphasis on the computation and communication components of the processor workload or tend to be biased towards particular problem structures and multiprocessor topologies. The second class of methods, physical optimization (PO) methods, do not make assumptions about the problem considered; but, they require greater execution time. Physical computation employs techniques from natural sciences and has been advocated for describing, simulating, and solving complex systems, especially intractable optimization problems [Fox 91b]. The operation of physical optimization methods is guided by an objective function and, usually, combines the two steps of data partitioning and processor assignment. Simulated annealing [Kirkpatrick et al. 83; Otten and van Ginneken 89], from statistical physics, views optimization as finding the ground state of a system in a heat bath. Biologically motivated neural networks [Hopfield and Tank 86] are based on a mean field theory derivation, from physics, to quickly find good minima for an energy function. These two paradigms have been adapted to the data mapping problem [Flower et al. 87; Koller 89; Fox and Furmanski 88; Fox et al. 89; Byun 92; Williams 91] and have demonstrated good potential to produce better mapping quality than that produced by heuristics based on spatial or graph-theoretic information. However, it should be emphasized that all these techniques, heuristic and physical, aim for producing good sub-optimal mapping solutions, and not necessarily optimal solutions.

So far, most parallel applications have used heuristic algorithms for data mapping, the most popular being recursive bisection algorithms. Based on this observation and the brief survey of methods given above, a number of gaps in the data mapping work can be identified. Firstly, several heuristic algorithms lack general applicability. Secondly, comparative performance evaluations of various mapping algorithms are lacking. In particular, evaluations of the performances of the PO algorithms have been limited. Thirdly, most of the previous mapping methods have not been parallelized for distributed-memory multiprocessors; sequential mapping is slow and unsuitable for realistic applications. This is particularly serious for the PO algorithms. Fourthly, not much has been reported about the applicability of various mapping algorithms to large problems, where it seems that recursive coordinate bisection is commonly used.

In this dissertation, we present parallel physical optimization algorithms for data mapping. These algorithms are based on genetic algorithms, simulated annealing and neural networks. Genetic algorithms [Holland 75] are inspired by evolutionary biology and are adapted in this dissertation accomplished by inspection or by simple techniques. However, good mapping of general and irregular data sets to various multiprocessor topologies is difficult to achieve. In fact, the data mapping problem is NP-hard, whether it is reduced to a graph partitioning problem [Garey and Johnson 79] or viewed as a resource allocation problem [Ibaraki and Katoh 88]. Examples of irregular problems are solving fluid dynamics equations on unstructured discretization of 3-dimensional geometric structures, and simulating particle dynamics systems with finite range forces, on hypercube or mesh-connected multiprocessors.

Several methods have been proposed for finding good suboptimal mapping solutions. They include greedy algorithms, nearest neighbor mapping, clustering techniques, mincut-based heuristics, recursive coordinate bisection, recursive graph bisection, recursive spectral bisection, geometry-based mapping, scattered decomposition, simulated annealing, neural networks, and evolutionary algorithms based on the replicator equation [Berger and Bokhari 87; Chrisochoides et al 91a; Ercal 88; Farhat 88; Fox 88a; Fox and Furmanski 88; Fox et al 88; Kramer and Muhlenbein 89; Muhlenbein 87; Pothen et al 90; Simon 91; Williams 91]. These methods can be divided into two classes: heuristics and physical optimization.

Heuristic methods can be based on either spatial information, e.g. the coordinates of data objects, or information extracted from the problem graph, e.g. graph-theoretic distance. Recursive coordinate bisection uses the physical coordinates of the data objects alternately in a number of bisection iterations [Berger and Bokhari 87; Dragon and Gustafson 89; Salmon 90; Walker 90] . In each iteration, the center of mass of the data objects, in each problem subdomain, forms a separator field that determines how to split the current subdomain into two. Recursive spectral bisection uses the second eigenvector of the Laplacian matrix, associated with the problem graph, for determining separator fields [Pothen et al 90; Simon 91]. Orthogonal recursive graph bisection employs an extremal peripheral vertex, one of two vertices separated by a distance equal to the diameter of the problem graph, to label other vertices in a subdomain according to a level structure [Fox 88b]. The separator field becomes the middle level.

Mincut-based heuristics [Ercal 88; Sadayappan et al. 89; Vaughn 91] use the Kernighan-Lin algorithm for graph partitioning, or modified versions [Kernighan and Lin 70; Fiduccia and Mattheyses 82]. A classic work of Bokhari [Bokhari 81] proposes sequences of pairwise exchanges alternating with probabilistic jumps. Scattered decomposition refers to superimposing a layer of contiguous regions over the whole problem domain [Fox 84; Goldsmith and Salmon 87; Morison and Otto 87]. Each region is formed of patches, where a patch corresponds to a mapping of the data objects that lie within it to a processor. The number of patches in a region equals the number of processors, and the topology of the region is identical to the multiprocessor topology. Nearest-neighbor techniques partition a 2-dimensional problem domain into clusters and map clusters to processors such that adjacent data objects are mapped to neighboring processors [Ercal 88; Lee and Aggarwal 87; Sadayappan and Ercal 87]. This mapping is followed by boundary refinement or iterative improve-

Chapter 1

Introduction

Given an algorithm, ALGO, for solving a problem with an underlying data set, DATA, the data mapping problem refers to mapping disjoint subsets of DATA to the processors of a multiprocessor such that the execution time of ALGO, on the multiprocessor, is minimized. The development of data mapping algorithms that minimize execution time is important for cost-effective utilization of the computational resources offered by current and future multiprocessors. Equally important is the development of mapping algorithms that have general applicability, that is, algorithms that can be applied to general problem configurations and multiprocessor topologies. General applicability of mapping algorithms allows their inclusion in automatic parallelization programming systems. This dissertation presents general mapping algorithms for minimizing the execution time of data-parallel algorithms on distributed-memory multiprocessors.

Data-parallelism is based on distributing data and associated computations among the processors. With the single program multiple data programming model, processors execute the same program independently on the data subsets distributed to them and communicate when nonlocal information is needed. Thus, the minimization of the execution time of data-parallel programs requires equal distribution of the workload associated with the data objects and the minimization of concurrency overheads, such as overheads due to communication, synchronization and other hardware and software factors.

From a programming point of view, processor workload consists of local computations and interprocessor communication. The combination of the two components, which comprises the total workload, depends mainly on the computation model. In this work, we assume a loosely synchronous data-parallel model [Fox et al. 88; Fox 91a] in which processors run the same algorithm, repeating compute-communicate cycles. In every cycle, the processors perform computations on their disjoint data subsets and then communicate to exchange boundary information. For this model, equal distribution of processor workload and, hence, minimal execution time and optimal data mapping requires the minimization of the time taken by the slowest processor. The loosely synchronous model is commonly used for science and engineering problems [Choudhary et al. 92; Fox et al 88], such as solving partial differential equations using iterative methods.

Data mapping is, obviously, an optimization problem. For applications with regular and uniform data sets and certain multiprocessor architectures, optimal or near-optimal data mapping can be

Acknowledgements

I would like to thank Professor Geoffrey Fox, my advisor. I am indebted to him not only for his valuable and inspiring comments and encouragement, but also for providing such an intellectually rich research environment.

I wish to thank J. Saltz and R. Das for providing interesting data sets and for useful conversations, H. Simon for supplying the RSB code, T. Starmer for helping me learn some evolutionary biology, W. Furmanski and P. Coddington for helpful discussions, S. Ranka for miscellaneous help, A. Goel for his support and friendship, Y. Chung for data sets, Z. Bozkus for programming help, and L. Wenderholm for her comments on some chapters. I am grateful to all NPAC staff for providing a responsive and friendly work atmosphere and to all researchers of the Syracuse Center for Computational Science for their friendship. I am very grateful to President R. Nassar and Dean R. Hajjar, of BUC, for their personal support.

Throughout my stay at Syracuse University, I have received generous financial support. I have been supported by a Fulbright Scholarship from CIES and a Research Assistantship from the NY Center for Computer Applications and Software Engineering. My dissertation work has been funded by the Center for Research on Parallel Computation, the Joint Tactical Fusion Program Office, and the ASAS agency. The implementation work was carried out using the computational resources of the Northeast Parallel Architectures Center at Syracuse University, California Institute of Technology, Cornell University, Sandia Laboratory, and ICASE-NASA Langley.

Special thanks go to my family and friends, for their encouraging love, and to all those from whom I learnt hard work and endurance.

CONTENTS

Chapter

To all Humanists,
on any planet

Abstract

We present three parallel physical optimization algorithms for mapping data to distributed-memory multiprocessors, concentrating on irregular loosely synchronous problems. We also present a technique for efficient mapping of large data sets. The algorithms include a parallel genetic algorithm (PGA), a parallel neural network algorithm (PNN) and a parallel simulated annealing algorithm (PSA). An important feature of these algorithms is that they deviate from the operation of their sequential counterparts in order to achieve reasonable speed-ups and, yet, they maintain similar solution qualities. PGA has excellent speed-ups by virtue of the natural evolution model on which it is based. PSA and PNN include communication schemes adapted to the properties of the mapping problem and of the algorithms themselves for reducing the communication overhead. The performances of the three physical optimization algorithms are evaluated and compared, among themselves and with previous good algorithms, for a variety of test cases. They are found to produce high quality mapping solutions and do not show a bias towards particular problem configurations. However, they are slower than previous algorithms. Further, the comparison results show that the three algorithms are suitable for different requirements of mapping time and quality. PGA produces the best solutions, followed by PSA and then PNN. But, PNN is the fastest and PGA is the slowest. The technique proposed for large problems is based on a pre-mapping graph contraction heuristic algorithm, which results in a smaller search space. Graph contraction leads to remarkable reductions in mapping time, while maintaining good mapping qualities. It allows large-scale mapping to become efficient, especially when the physical optimization algorithms are used.

Physical Optimization Algorithms for Mapping Data to Distributed-Memory Multiprocessors

by

Nashat Mansour

B.E., University of New South Wales, 1980
M.Eng.Sc., University of New South Wales, 1983
M.S., Syracuse University, 1990

DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer and Information Science
in the Graduate School of Syracuse University
August 1992