List of Figures

1 (a) is a weighted DAG. (b) is a nonlinear clustering and (c) is a linear clustering. . . 4

2 (a) is a Gantt chart. (b) is the scheduled graph derived from the Gantt chart. . . . 5

3 Fork and join sets. 9

4 An optimum clustering algorithm for the join set with $\tau_j + \beta_j \geq \tau_{j+1} + \beta_{j+1}, \quad j = 1 : m-1$. 10

5 The Gantt chart and scheduled DAG for the optimum clustering algorithm of the join set. 11

6 Parallelization vs. sequentialization for the join set. 12

7 Case 1 and 2 of **construct** procedure. 15

8 Case 3 and 4 of **construct** procedure. 16

9 An example of linearization of a nonlinear clustering. 18

10 The kji form for Gauss-Jordan without pivoting and with interior loop task partitioning. 20

11 The Gauss-Jordan DAG. The node computation cost for all nodes is $\tau = n\omega$ and edge communication cost for all edges is $c = \alpha + n\beta$. 21

12 The natural linear clustering for the Gauss-Jordan DAG executed on a clique with $p = n$ processors 22

13 The natural clustering for the Gauss-Jordan DAG on a ring with $p = n$ processors . 24

[18] B. Kruatrachue B. and T. Lewis, *Grain Size Determination for Parallel Processing*, IEEE Software, pp. 23-32, Jan. 1988.

[19] H.T. Kung, *Network-Based Multicomputers: Redefining High Performance Computing in the 1990s*, CMU-CS-89-138, School of Computer Science, Carnegie Mellon University, PA, Jan. 1989.

[20] S.Y. Kung, *VLSI Array Processors*, New Jersey:Prentice Hall, 1988.

[21] R.E. Lord, J.S. Kowalik, and S. P. Kumar, *Solving Linear Algebraic Equations on an MIMD Computer*, Journal of the ACM, vol. 30, pp. 103-117, 1983.

[22] C. McGreary C. and H. Gill H, *Automatic Determination of Grain Size for Efficient Parallel Processing*, Communications of ACM, vol. 32, pp. 1073-1078, Sept., 1989.

[23] U. Manber, *Introduction to Algorithms*, Reading, Mass: Addison Wesley, 1989.

[24] C. Moler, *Matrix Computation on Distributed Memory Multiprocessors*, Hypercube Multiprocessors 1986, SIAM, pp. 181-195.

[25] J.M. Ortega, *Introduction to Parallel and Vector Solution of Linear Systems*, New York:Plenum, 1988.

[26] C. Papadimitriou and M. Yannakakis, *Towards on an Architecture-Independent Analysis of Parallel Algorithms*, SIAM J. Comput., vol. 19, pp. 322-328, 1990.

[27] Y. Robert, B. Tourancheu, and G. Villard, *Data Allocation Strategies for the Gauss and Jordan Algorithms on a Ring of Processors*, Information Processing Letters, vol. 31, pp.21-29, 1989.

[28] Y. Saad, *Gaussian Elimination on Hypercubes*, Parallel Algorithms and Architectures, Cosnard, M. et al. Eds., Elsevier Science Publishers, North-Holland, 1986.

[29] I.C.F. Ipsen, Y. Saad and M. Schultz, *Complexity of Dense Linear System Solution on a Multiprocessor Ring,* Linear Algebra and Appl., vol. 77, pp. 205-239, 1986.

[30] V. Sarkar, *Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors*, The MIT Press, 1989.

[31] H. Stone, *High-Performance Computer Architectures*, Reading, Mass.: Addison-Wesley, 1987.

[32] Min-You Wu and D. Gajski, *A Programming Aid for Hypercube Architectures*, The Journal of Supercomputing, vol. 2, pp. 349-372, 1988.

Journal of Parallel and Distributed Computing, vol. 9, pp. 87-92, 1990.

[2] Ph. Chretienne, *Task Scheduling over Distributed Memory Machines*, Proc. of the International Workshop on Parallel and Distributed Algorithms, (North Holland, Ed.), 1989.

[3] M. Cosnard, M. Marrakchi, Y. Robert, and D. Trystram, *Parallel Gaussian Elimination on an MIMD Computer*, Parallel Computing, vol. 6, pp. 275-296, 1988.

[4] M. Cosnard, B. Tourancheau, and G. Villard, *Gaussian Elimination on Message Passing Architecture*, Lecture Notes in Computer Science 297, Berlin: Springer-Verlag, 1987, pp. 611-628.

[5] G. Davis, *Column LU Factorization with Pivoting on a Hypercube Multiprocessor*, SIAM J. Algebraic and Discrete Methods, vol. 7, pp. 538-550, 1986.

[6] J. J. Dongarra, F.G. Gustavson, and A. Karp, *Implementing Linear Algebra Algorithms for Dense Matrices on a Vector Pipeline Machine*, SIAM Review, vol. 26, pp. 91-112, 1984.

[7] T.H. Dunigan, *Performance of a Second Generation Hypercube*, ORNL/TM-10881, Oak Ridge National Laboratory, Oak Rige, TN, Nov. 1988.

[8] G.A. Geist and M.T. Heath, *Matrix Factorization on a Hypercube Multiprocessor*, Hypercube Multiprocessors, Philadelphia:SIAM, 1986, pp. 161-180.

[9] G.H. Golub and C.F. Van Loan, *Matrix Computations*, Johns Hopkins, 1989.

[10] M.T. Heath and C.H. Romine, *Parallel Solution of Triangular Systems on Distributed Memory Multiprocessors*, SIAM J. Sci. Statist. Comput., vol. 9, pp. 558-588, 1988.

[11] A. Gerasoulis and I. Nelken, *Static Scheduling for Linear Algebra DAGs*, in HCCA4, 1989.

[12] A. Gerasoulis , S. Venugopal, and T. Yang, *Clustering Task Graphs for Message Passing Architectures*, 1990 Proceedings of ACM International Conference on Supercomputing, Amsterdam, pp. 447-456, 1990.

[13] A. Gerasoulis and S. Venugopal, *Linear Clustering of Linear Algebra Task Graphs for Local Memory Systems*, Preliminary Report.

[14] M. Girkar and C. Polychronopoulos, *Partitioning Programs for Parallel Execution*, Proc. of the 1988 ACM International Conference on Supercomputing, St. Malo, France, July 4-8, 1988.

[15] R.L. Graham, *Bounds on Multiprocessing Timing Anomalies*, SIAM J. Appl. Math., vol. 17, pp. 416-429, 1969.

[16] S. J. Kim, *A General Approach to Multiprocessor Scheduling*, TR-88-04, DCS, Univ. of Texas at Austin, 1988.

[17] S.J. Kim and J.C Browne, *A General Approach to Mapping of Parallel Computation upon Multiprocessor Architectures*, Int'l Conf. on Parallel Processing, vol 3, pp. 1-8, 1988.

7 Concluding remarks

In this paper, we considered the question of sequentialization vs. parallelization of task graphs. We have shown that the granularity of the task graph affects the optimum trade off point and if the granularity is greater than one then linear clustering is optimum. The granularity, however, has been defined as a minimum of the task grains over all tasks in the graph. There are cases in which a small subgraph of the task graph determines the granularity of the entire task graph. An interesting question that arises is how good is linear clustering if the granularity is less than one for a small subgraph of the DAG and greater than one for the other part of the DAG. An example of one such case is the Gaussian-Elimination DAG described below.

Gaussian-Elimination is similar to Gauss-Jordan except that the $< i = 1 : n$ **and** $i \neq k >$ loop is replaced by $< i = k+1 : n >$ instead. The task size becomes $\tau_k = (n-k+1)w$ for every task in layer k which consists of the tasks $\{T_k^{k+1}, \ldots, T_k^{n+1}\}$, see figure 11. The communication cost of the edges between layers k and $k+1$ is given by $c_k^{k+1} = \alpha + (n-k+1)\beta$. Therefore, the granularity for Gaussian-Elimination may or may not be greater than one depending on the architecture parameters:

$$g(GE) = \frac{(n-k+1)\omega}{\alpha + (n-k+1)\beta} \geq 1 \qquad \Longrightarrow \qquad k \leq n - \frac{\alpha}{\omega - \beta}.$$

For the Ncube/1 parameters the granularity becomes less or equal to one for the subgraph consisting of the layers $k = (n-14) : n$ and greater than one for $k = 1 : (n-15)$. The theorems of the paper remain true for the subgraph consisting of the layers $k = 1 : (n-15)$. Since the subgraph $k = (n-14) : n$ remains constant as n increases, by following the same proof as in theorem 4, we can easily show that the natural linear clustering is asymptotically optimal with respect to n, but not optimal, for Gaussian-Elimination when $p = n$.

In conclusion, the theorems of the paper could be used even in cases where the granularity assumptions are partially true. However, additional investigation is needed to analyze the cases in which the performance of linear clustering is within 50% of the optimum even though the granularity is less than one.

Acknowledgments

We would like to thank Sesh Venugopal for many valuable discussions and suggestions on the initial manuscript. We would also like to thank Srimat Chakradhar for suggesting improvements on presentation.

References

[1] F.D. Anger, J. Hwang, and Y. Chow, *Scheduling with Sufficient Loosely Coupled Processors,*

Proof. Due to linear clustering, the execution ordering within a processor in the ring is defined by the dependence chain of the initial DAG. The scheduled DAG on a ring with $p = n$ processors is given in figure 13. Notice that the edge communication cost of $< T_k^{k+1}, T_{k+1}^j >$ is

$$(j-k)c = (j-k)(\alpha + n\beta), \quad j = k+2 : n+1, \quad k = 1 : n-1$$

where $(j-k)$ is the maximum distance that column k must travel if we use a *store* and *forward* communication protocol, H. T. Kung [19] and Saad [28]. However, even if some of the individual communication edge costs in the scheduled DAG have increased for the ring architecture, the edges $< T_k^{k+1}, T_{k+1}^{k+2} >$ remain unchanged and the critical path length of the scheduled DAG is the same as the clique architecture, which implies

$$PT_{nlc}^{ring} = PT_{opt}^{clique} = n^2\omega \ + \ (n-1)(\alpha + n\beta). \qquad \blacksquare$$

In the following table, we present the parameters of several commercially available message passing architectures for single precision arithmetic, along with the granularity of the Gauss-Jordan algorithm for these architectures. We have taken these parameters from Dunigan [7], tables 3 and 7. The single precision flop is the sum of an addition and multiplication, say for example for the Ncube/1 is 16.6 + 18.5 = 35.1 which has also been confirmed by Heath and Romine [10]. For all architectures the granularity $g(GJ)$ is greater than one for sufficiently large n, for example $n \geq 13$ for the iPSC/1. This implies that the results and theorems of the previous sections are directly applicable to these architectures. The sharpest asymptotic upper bound for the ratio between linear clustering and the optimum clustering is given by the coarsest grain architecture, the iPSC/1.

	Ametek	iPSC/1	iPSC/2	Ncube/1
α (microseconds)	563.5	862.2	697	383.6
β (microseconds/word)	38	7.2	1.6	10.4
ω (microseconds/flop)	103.6	77.5	11.4	35.1
$g(GJ) \geq 1$ when $n \geq$	9	13	72	16
$\lim_{n\to+\infty} g(GJ) = \omega/\beta =$	2.7	10.8	7.1	3.4
$\lim_{n\to+\infty} PT_{lc}/PT_{opt} \leq 1 + \beta/\omega =$	1.37	1.09	1.14	1.29

In conclusion of this section, the last theorem proves that the ring architecture is an optimal architecture for the Gauss-Jordan DAG in the case of $p = n$ processors. The linear array with $p = n$ processors is also an optimal architecture for the Gauss-Jordan DAG. Another conclusion is that the *cut-through* or *wormhole* communication protocols, H. T. Kung [19], in which case data do not have to be stored and forwarded but rather cut-through a processor using a communication co-processor, cannot reduce the parallel time for the Gauss-Jordan DAG when $p = n$, since the store and forward protocol is optimum for this DAG. The cut-through and wormhole protocols have been implemented in the new generation of hypercube architectures such as the Ncube/2 and Intel/2.

embedded in the hypercube. Saad's result is intuitively surprising since one might expect that a richer in communication capabilities architecture should result in a shorter parallel time. We will show that at least in the case of $p = n$ processors the ring architecture is an optimal architecture for the coarse grain Gauss-Jordan DAG. We define an *optimal architecture* for a DAG as the architecture in which the optimum schedule of the DAG has the same parallel time as the optimum schedule on the clique.

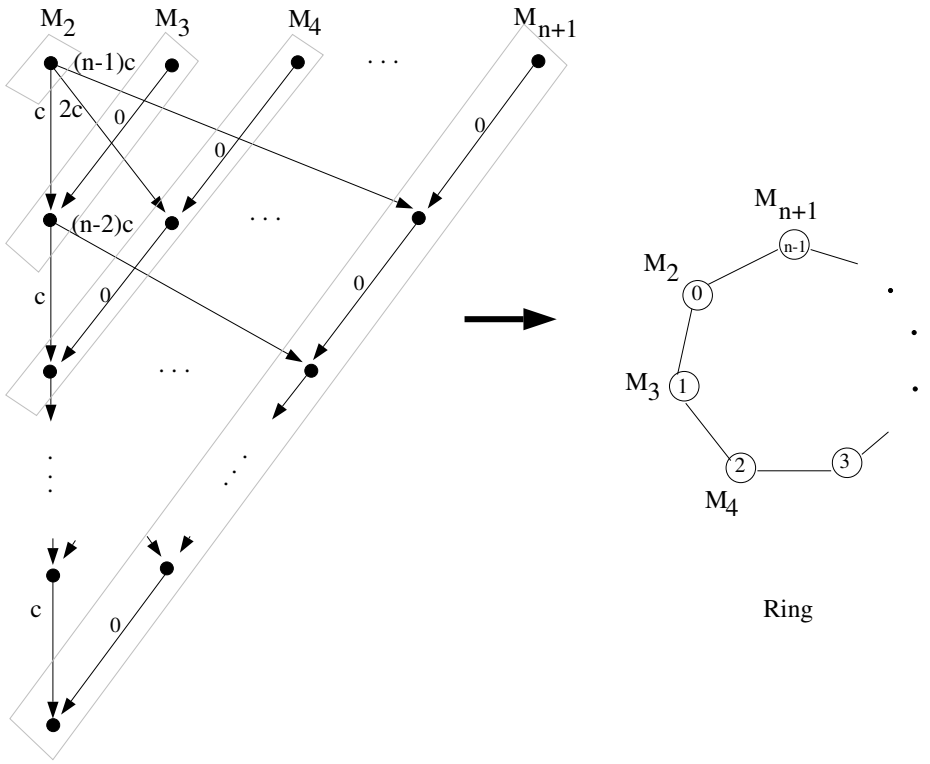


Figure 13: The natural clustering for the Gauss-Jordan DAG on a ring with $p = n$ processors

As mentioned in the introduction, the two step approach to scheduling is: (1) Clustering, (2) Scheduling the clusters. The scheduling problem is equivalent to determining a mapping of the clusters to processors and then determining an ordering of the tasks within each processor. For a general DAG determining the optimum schedule has been shown to be NP-complete. For the first step above and the Gauss-Jordan DAG we have shown that the natural linear clustering yields the optimum parallel time. For the second step above, we will show that a schedule of these clusters on $p = n$ processors of a ring architecture is also optimum.

Theorem 4 *(**Gauss-Jordan optimum parallel time on a ring**). Assume that the number of processors of a ring architecture is equal to the number of the natural linear clusters and $g(GJ) \geq 1$. Then the following physical mapping of the clusters to the processors is optimal*

$$map(M_j) = j - 2, \qquad j = 2 : n + 1.$$

Moreover, the optimum parallel time on the ring is equal to the optimum parallel time on the clique

$$PT_{nlc}^{ring} = PT_{opt}^{clique} = n^2\omega \ + \ (n-1)(\alpha + n\beta).$$

and PT_{opt}^{clique} must be the parallel time of a linear clustering. We will show that

$$PT_{opt}^{clique} \geq n^2\omega \ + \ (n-1)(\alpha + n\beta).$$

We define a *layer* k of the Gauss-Jordan DAG in figure 11 as the set of tasks $\{T_k^{k+1}, \ldots, T_k^{n+1}\}$. We will prove that the completion time of each task T_k^j at layer k satisfies

$$CT(T_k^j) \geq kn\omega \ + \ (k-1)(\alpha + n\beta), \ j = k+1 : n+1.$$

This is trivial for $k = 1$. Suppose it is true for tasks at layer $k-1$. We examine the completion time for each task T_k^j at layer k. Since each task has two incoming edges from tasks at layer $k-1$, and linear clustering zeros at the most only one edge, task T_k^j has to wait at least $\alpha + n\beta$ time to receive the message from one of its two predecessors, say T_{k-1}^r, at layer $k-1$. Therefore

$$CT(T_k^j) \geq CT(T_{k-1}^r) + \alpha + n\beta + n\omega.$$

From the induction hypothesis we have that

$$CT(T_{k-1}^r) \geq (k-1)n\omega \ + \ (k-2)(\alpha + n\beta)$$

which implies

$$CT(T_k^j) \geq kn\omega \ + \ (k-1)(\alpha + n\beta).$$

Since the the parallel time is the completion time of the last task T_n^{n+1} we have that

$$T_{opt}^{clique} \geq n^2\omega \ + \ (n-1)(\alpha + n\beta). \qquad \blacksquare$$

Even though a clique architecture is the ideal architecture in terms of the connectivity property, which is the number of edges that must be deleted before the architecture graph becomes disconnected, the connectivity becomes an obstacle in building such architectures with a large number of processors. On the other hand, architectures with lower connectivity such as ring or hypercubes can be easily build with thousands of processors. Examples of commercially available existing message passing architectures with more than 1000 processors are the Ncube/1 with a maximum of 1024 processors and the Ncube/2 with a maximum of 8192 processors. Therefore, it is of interest to investigate if the results above can be extended to actual existing architectures.

6.2 Physical mapping of the Gauss-Jordan DAG on a ring architecture

We choose the ring architecture to extend our results of the previous sections. Ring is one of the simplest architectures with low connectivity and can be easily mapped on several existing architectures such as hypercubes by using the gray code mapping, Saad [28]. Saad [28] observed that using the complete broadcast capabilities of the hypercube for the natural linear clustering will result in worst performance, by a $\log p$ factor, than by simply using the gray code ring broadcast

Applying theorem 2 on Gauss-Jordan DAG we can derive the following performance upper bound for the natural linear clustering, assuming that $g(GJ) \geq 1$,

$$\frac{PT_{nlc}^{clique}}{PT_{opt}^{clique}} \leq 1 + \frac{\beta}{\omega} + \frac{\alpha}{n\omega} \asymp 1 + \frac{\beta}{\omega}$$

where PT_{nlc}^{clique} is the parallel time of the natural linear clustering and PT_{opt}^{clique} is the optimum parallel time on the clique architecture. However, theorem 1 could be used to obtain an even stronger result which we present it as a theorem below.

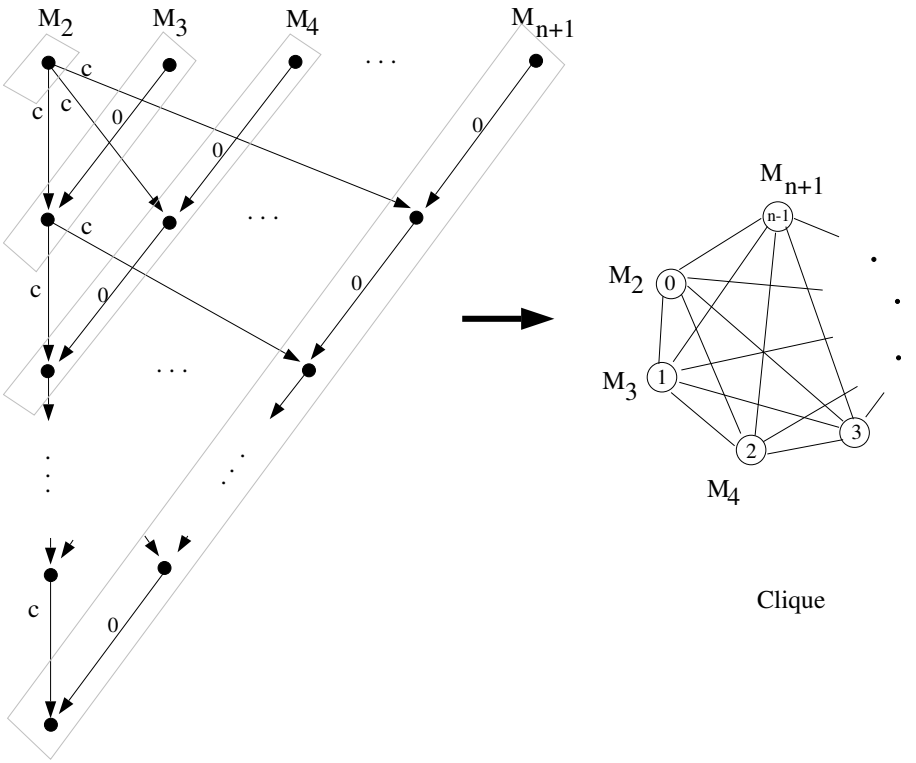


Figure 12: The natural linear clustering for the Gauss-Jordan DAG executed on a clique with $p = n$ processors

Theorem 3 *(***Gauss-Jordan optimum parallel time on a clique***). If the number of processors of the clique is equal to the number of the linear clusters and $g(GJ) \geq 1$, then*

$$PT_{nlc}^{clique} = PT_{opt}^{clique} = n^2\omega \ + \ (n-1)(\alpha + n\beta).$$

Proof. We can easily see that for $p = n$ processors, the parallel time for the natural linear clustering is equal to the length of the critical path $\{T_1^2, T_2^3, \ldots, T_n^{n+1}\}$ of the scheduled Gauss-Jordan DAG, see figures 11 and 12:

$$PT_{nlc}^{clique} = n^2\omega \ + \ (n-1)c = n^2\omega \ + \ (n-1)(\alpha + n\beta).$$

Since we have assumed that the granularity $g(GJ)$ is greater than one, then theorem 1 implies that *the optimum parallel time must be achieved by a linear clustering.* We have that

$$PT_{opt}^{clique} \leq PT_{nlc}^{clique} = n^2\omega \ + \ (n-1)(\alpha + n\beta)$$

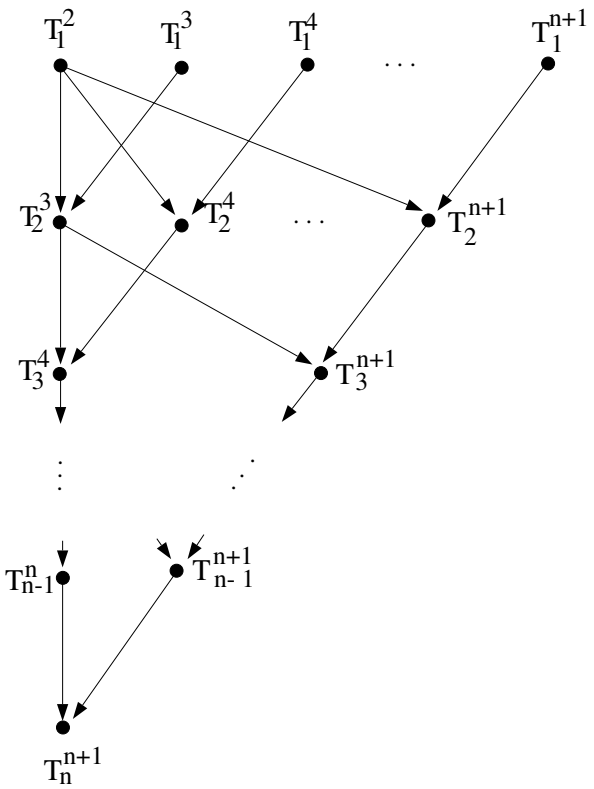


Figure 11: The Gauss-Jordan DAG. The node computation cost for all nodes is $\tau = n\omega$ and edge communication cost for all edges is $c = \alpha + n\beta$.

The granularity for the Gauss-Jordan DAG can be easily computed

$$g(GJ) = \frac{n\omega}{\alpha + n\beta} \asymp \frac{\omega}{\beta}.$$

where the asymptotic limit "$\asymp$" is taken with respect to $n \to +\infty$. The *natural linear clustering*

$$M_j = \{T_k^j \mid k = 1 : j-1\}, \quad j = 2 : n+1$$

shown in figure 12, can be obtained by assuming that each cluster contains all tasks that modify the same column, i.e. the locality assumption in Gerasoulis and Nelken [11]. This linear clustering has been used in the literature to schedule the Gauss-Jordan and Gauss-Elimination DAGs on a ring of processors, e.g. Ortega [25], pp. 88 and 241, Gerasoulis and Nelken [11], Saad [28], Cosnard, Tourancheau and Villard [4], Robert, Tourancheau and Villard [27], Moler [24], Geist and Heath [8], Davis [5] and many others. The natural linear clusters are wrapped around (*wrap mapping*) the ring of processors and then the compute-ahead or naive ordering is used to schedule the tasks, Ortega p. 241. Ipsen, Saad and Schultz [29] have derived lower bounds for the optimum schedule on the ring architecture. Robert, Tourancheau and Villard [27] have shown the asymptotic optimality of the kij form of the Gauss-Jordan algorithm on a ring of processors, under the assumption that communication does not overlap with computation. This assumption, however, is not practical for message passing architectures where communication usually overlaps with computation to increase performance. As far as we know, there has been no direct comparison of the natural linear clustering with the optimum clustering for the kji form.

6 Linear clustering and the Gauss-Jordan algorithm

In this section, we demonstrate the practicality of the previous results in the area of numerical computing where linear clustering has been widely used. The typical examples are the Gaussian-Elimination (GE) and Gauss-Jordan (GJ) algorithms which have been studied in the literature for a variety of parallel architectures, e.g. Ortega [25], pp. 88 and 241, Gerasoulis and Nelken [11], Robert, Tourancheau and Villard [27], Moler [24], Geist and Heath [8], Davis [5], Saad [28], Ipsen, Saad and Schultz [29], Dongarra, Gustavson and Karp [6], S. Y. Kung [20] p. 168 and many others. Here we consider the kji form without pivoting for the Gauss-Jordan algorithm, e.g. Dongarra, Gustavson and Karp [6], Robert, Tourancheau and Villard [27], with interior loop partitioning, see figure 10.

Gauss-Jordan kji form

for $k = 1 : n$
  for $j = k + 1 : n + 1$
$T_k^j : \{ \quad a_{k,j} = a_{k,j}/a_{k,k}$
  for $i = 1 : n$ **and** $i \neq k$
    $a_{i,j} = a_{i,j} - a_{i,k} * a_{k,j}$
  end$\}$
  end
end

Figure 10: The kji form for Gauss-Jordan without pivoting and with interior loop task partitioning.

The task T_k^j defined in figure 10 uses column k to modify column j of the matrix A. The data dependence graph is given in figure 11, e.g. Gerasoulis and Nelken [11], Cosnard, Marrakchi, Robert and Trystram [3], Lord, Kowalik and Kumar [21]. The task T_k^j is functional task with input data being the columns k and j, and output data is the column j. The tasks T_k^{k+1} are broadcasting nodes, broadcasting the same column $k + 1$ to all T_{k+1}^j, $j = k + 2 : n + 1$ tasks.

6.1 Clustering on a clique message passing architecture

Assuming that every task in the Gauss-Jordan DAG has been mapped on a separate processor on a clique, then the computation cost τ for each task and the communication cost c for each edge can be taken

$$\tau = n\omega, \qquad c = \alpha + n\beta$$

where ω is the time for computing either an addition and multiplication or a division. We have chosen the communication cost model between neighbor processors to be the widely accepted linear model, where α is the start up time and $n\beta$ is the transmission time of a vector of n elements, e.g. Dunigan [7], Heath and Romine [10].

where PT_{opt} is the optimum parallel time and PT_{lc} is the parallel time of the linear clustering. Moreover for a coarse grain DAG we have

$$PT_{lc} \leq 2 \times PT_{opt}.$$

Proof: The proof is identical to Gerasoulis and Venugopal [13]. Let L_{cp} be the length of a critical path $cp = \{n_1, n_2, ..., n_k\}$ of G which includes edge communication costs. We have

$$L_{cp} = \sum_{i=1}^{k} \tau_i + \sum_{i=1}^{k-1} c_{i,i+1}$$

and from the definition of $g(G)$

$$\sum_{i=1}^{k-1} c_{i,i+1} \leq \frac{\sum_{i=1}^{k-1} \tau_i}{g(G)}.$$

Consider the final clustered DAG G_m which is the same as G with the exception that the edge weights of the clusters in G have been set equal to zero. Then PT_{lc} is equal to the length of the critical path of G_m. We have that

$$\sum_{i=1}^{k} \tau_i \leq PT_{opt} \leq PT_{lc} \leq L_{cp}.$$

Then

$$PT_{lc} \leq L_{cp} = \sum_{i=1}^{k} \tau_i + \sum_{i=1}^{k-1} c_{i,i+1} \leq \sum_{i=1}^{k} (\tau_i + \frac{\tau_i}{g(G)}) \leq (1 + \frac{1}{g(G)}) PT_{opt}.$$

If $g(G) \geq 1$ then

$$PT_{lc} \leq 2 \times PT_{opt}. \quad \blacksquare$$

This result is reminiscent of the well known result that any list scheduling is within a factor of two of the length of the optimum schedule, Graham [15]. For list scheduling, however, communication edge costs is assumed to be either zero or additive to the task cost, Sarkar [30] p. 60. Here we have assumed the static macro-dataflow model, in which communication may overlap with computation, e.g. Wu and Gajski [32]. Recently Papadimitriou and Yannakakis [26] have proposed a polynomial time algorithm for the static macro-dataflow model whose performance is within a factor of two of the optimum, but their algorithm requires task duplication. The existence of a polynomial time algorithm for scheduling *without duplication* which is within a factor of two of the optimal is posed as an open problem by Papadimitriou and Yannakakis [26].

Anger, Hwang and Chow [1] assume: (S2) an unbounded number of processors, (S3) short communication delays, $c_{i,j} \leq min_{i=1:v}\{\tau_i\}$ for all communication edges, (S5) a fully connected architecture. Then they use their bound for their Earliest Task First (ETF) heuristic algorithm

$$PT_{ETF} \leq (2 - \frac{1}{p}) PT_{opt} + MaxChainComm$$

where PT_{opt} is the optimum parallel time and $MaxChainComm$ is the the longest communication chain in the DAG, to show that

$$PT_{ETF} \leq 3 \times PT_{opt}$$

since $MaxChainComm \leq PT_{opt}$ because of assumption (S3). However, assumption (S3) implies that $g(G) \geq 1$ and theorems 1 and 2 provide a sharper upper bound for this case.

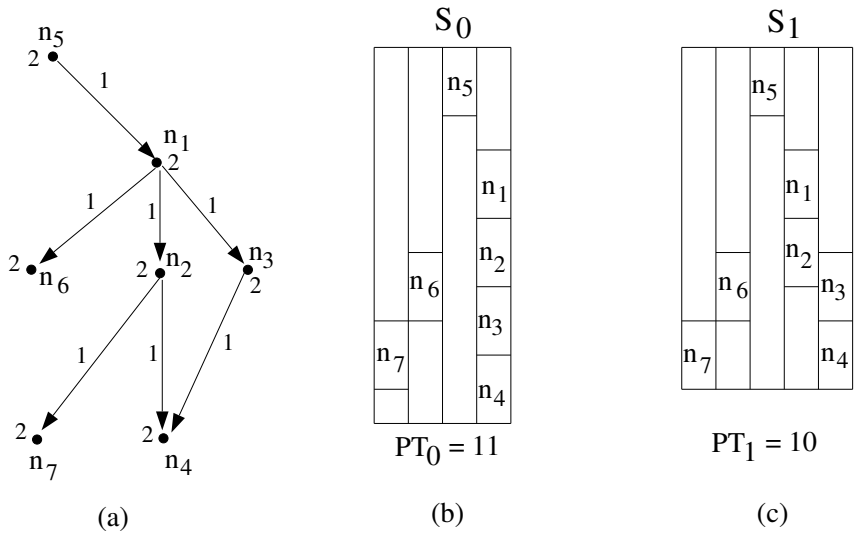


Figure 9: An example of linearization of a nonlinear clustering.

We now present an example to demonstrate how the above procedure works. Assume that the following nonlinear clustering of the DAG in figure 9 is given,

$$CLUSTER(0) = \{\ M_1 = \{n_7\},\quad M_2 = \{n_6\},\quad M_3 = \{n_5\},\quad M_4 = \{n_1, n_2, n_3, n_4\}\ \}.$$

The optimum schedule S_0 with parallel time $PT_0 = 11$ is shown in Figure 9(b). The NLC cluster is M_4. The node n_4 is extracted first and $pred_0(n_4) = (n_2, n_3)$ and $LC = \{n_4\}$. Because of case 2, the node n_3 is extracted next and $pred_0(n_3) = (n_1)$ and $LC = \{n_3, n_4\}$. Since $n_1 \preceq_i n_2 \preceq_i n_3$ the extraction stops because of case 3. We rename LC to be M_5, and the linear clustering is,

$$CLUSTER(1) = \{\ M_1 = \{n_7\},\quad M_2 = \{n_6\},\quad M_3 = \{n_5\},\quad M_4 = \{n_1, n_2\},\quad M_5 = \{n_3, n_4\}\ \}.$$

In figure 9(b) S_0 is the schedule at the beginning of the first step of the linearization procedure. At the end of that step the linear cluster M_5 is extracted. A schedule S_1 is constructed by following **Construct** procedure and shown in figure 9(c). This schedule satisfies $ST_l(n_t) \geq ST_{l+1}(n_t)$ for all nodes n_t. Notice also that the execution order of S_1 is the same as S_0, i.e. $n_3 \preceq_i n_4$ in S_0 and S_1, and that the parallel time of S_1 is less than the parallel time of S_0.

5 Bounds for linear clustering for coarse grain DAGs

Theorem 1 proves that for coarse grain DAGs the optimum clustering is a linear clustering. This theorem, however, does not give performance bounds for linear clustering algorithms. The next theorem shows that the linear clustering performance bound result of Gerasoulis and Venugopal [13] still holds for our granularity. Thus, theorem 2 is true for both definitions of granularity, but theorem 1 is true only for our fork/join granularity definition.

Theorem 2 *For any linear clustering algorithm we have*

$$PT_{opt} \ \leq\ PT_{lc} \ \leq\ (1 + \frac{1}{g(G)})PT_{opt}$$

will not affect its ready time since the processor architecture is a clique. For case 3 of extraction, $pred_l(n_{c_1}) = (n_{k_1}, ..., n_{k_q})$ and there exists a task n_{s_q} in NLC satisfying $n_{k_q} \preceq_i n_{s_q} \preceq_i n_{c_1}$ in S_l. Therefore,

$$ST_l(n_{c_1}) \geq \max\{P_l(n_{c_1}, pred_l(n_{c_1})),\ Q_l(n_{c_1}, R_l(n_{c_1})),\ ST_l(n_{s_q}) + \tau_{s_q}\}$$

$$RT_{l+1}(n_{c_1}) = Q_{l+1}(n_{c_1}, PRED(n_{c_1})) = \max\{Q_{l+1}(n_{c_1}, R_l(n_{c_1})), Q_{l+1}(n_{c_1}, pred_l(n_{c_1}))\}$$

The induction hypothesis and because $n_{k_q} \preceq_i n_{s_q} \preceq_i n_{c_1}$ in S_l, imply

$$ST_l(n_{s_q}) \geq P_l(n_{c_1}, pred_l(n_{c_1})) \geq P_{l+1}(n_{c_1}, pred_l(n_{c_1}))$$

$$ST_l(n_{s_q}) + \tau_{s_q} \geq P_l(n_{c_1}, pred_l(n_{c_1})), \qquad Q_{l+1}(n_{c_1}, R_l(n_{c_1})) \leq Q_l(n_{c_1}, R_l(n_{c_1})).$$

Because the graph is coarse then $\tau_{s_q} \geq \max_{1 \leq t \leq q}\{c_{k_t,c_1}\}$ and

$$ST_l(n_{s_q}) + \tau_{s_q} \geq P_{l+1}(n_{c_1}, pred_l(n_{c_1})) + \max_{1 \leq t \leq q}\{c_{k_t,c_1}\} \geq Q_{l+1}(n_{c_1}, pred_l(n_{c_1})).$$

Consequently,

$$RT_{l+1}(n_{c_1}) \leq ST_l(n_{c_1}).$$

Since processor j contains a linear cluster, we can set

$$ST_{l+1}(n_{c_1}) = RT_{l+1}(n_{c_1}) \quad \Longrightarrow \quad ST_{l+1}(n_{c_1}) \leq ST_l(n_{c_1}).$$

4. **n_t is the k-th node n_{c_k} in LC**

Refer to figure 8(4). We examine the relationship between $n_{c_{k-1}}$ and n_{c_k} in case 2 of the extraction procedure. Since $pred_l(n_{c_k}) = (n_{k_1}, ..., n_{k_q})$, and $k_q = c_{k-1}$ and there is no successor $n_{s_{k-1}}$ of $n_{c_{k-1}}$ satisfying $n_{c_{k-1}} \preceq_i n_{s_{k-1}} \preceq_i n_{c_k}$ in S_l,

$$ST_l(n_{c_k}) \geq \max\{P_l(n_{c_k}, pred_l(n_{c_k})),\ Q_l(n_{c_k}, R_l(n_{c_k}))\} \geq \max\{ST_l(n_{c_{k-1}}) + \tau_{c_{k-1}},\ Q_l(n_{c_k}, R_l(n_{c_k}))\}$$

$$RT_{l+1}(n_{c_k}) = \max\{ST_{l+1}(n_{c_{k-1}}) + \tau_{c_{k-1}}, Q_{l+1}(n_{c_k}, PRED(n_{c_k}) - \{n_{c_{k-1}}\})\}$$

$$Q_{l+1}(n_{c_k}, PRED(n_{c_k}) - \{n_{c_{k-1}}\}) = \max\{Q_{l+1}(n_{c_k}, R_l(n_{c_k})), Q_{l+1}(n_{c_k}, pred_l(n_{c_k}) - \{n_{c_{k-1}}\})\}.$$

Thus from the induction assumption we have that

$$Q_{l+1}(n_{c_k}, R_l(n_{c_k})) \leq Q_l(n_{c_k}, R_l(n_{c_k})).$$

Since the graph is coarse then $\tau_{k_q} \geq \max_{1 \leq b \leq q-1}\{c_{k_b,c_k}\}$ and since $n_{k_1} \preceq_i, ..., \preceq_i n_{k_q}$, we have that

$$ST_l(n_{c_{k-1}}) + \tau_{c_{k-1}} \geq P_{l+1}(n_{c_k}, pred_l(n_{c_k}) - \{n_{c_{k-1}}\}) + \max_{1 \leq b \leq q-1}\{c_{k_b,c_k}\} \geq Q_{l+1}(n_{c_k}, pred_l(n_{c_k}) - \{n_{c_{k-1}}\}).$$

Therefore, we have shown that

$$RT_{l+1}(n_{c_k}) \leq ST_l(n_{c_k})$$

and since processor j contains a linear cluster, we can set

$$ST_{l+1}(n_{c_1}) = RT_{l+1}(n_{c_1}), \quad \text{and thus } ST_{l+1}(n_{c_1}) \leq ST_l(n_{c_1}) \qquad \blacksquare$$

According to the extraction procedure, another immediate successor of n_{c_k}, say $n_{c_{k+1}}$, must be in LC and they must satisfy $n_{c_k} \preceq_i n_{c_{k+1}} \preceq_i n_t$. Therefore

$$ST_l(n_t) \geq \max\{P_l(n_t, pred_l(n_t)),\ Q_l(n_t, R_l(n_t)),\ ST_l(n_{c_{k+1}}) + \tau_{c_{k+1}}\}$$

$$RT_{l+1}(n_t) = \max\{P_{l+1}(n_t, pred_{l+1}(n_t)),\ Q_{l+1}(n_t, R_{l+1}(n_t))\}.$$

Because n_{c_k} is on a different processor at schedule S_{l+1} we have that

$$pred_{l+1}(n_t) = pred_l(n_t) - \{n_{c_k}\}, \quad R_{l+1}(n_t) = R_l(n_t) \cup \{n_{c_k}\}.$$

We can easily see from the definition of P and Q and the induction hypothesis

$$P_{l+1}(n_t, pred_l(n_t) - \{n_{c_k}\}) \leq P_l(n_t, pred_l(n_t)), \quad Q_{l+1}(n_t, R_l(n_t)) \leq Q_l(n_t, R_l(n_t))$$

$$Q_{l+1}(n_t, R_l(n_t) \cup \{n_{c_k}\}) = \max\{Q_{l+1}(n_t,\ R_l(n_t)),\ ST_{l+1}(n_{c_k}) + \tau_{c_k} + c_{c_k,t}\}.$$

We also have that

$$ST_l(n_{c_{k+1}}) \geq ST_l(n_{c_k}) + \tau_{c_k}.$$

Since the graph is coarse, then $\tau_{c_{k+1}} \geq c_{c_k,t}$ and

$$ST_l(n_{c_{k+1}}) + \tau_{c_{k+1}} \geq ST_l(n_{c_k}) + \tau_{c_k} + c_{c_k,t} \geq ST_{l+1}(n_{c_k}) + \tau_{c_k} + c_{c_k,t}.$$

The above imply that

$$ST_l(n_t) \geq RT_{l+1}(n_t)$$

and thus we can set

$$ST_{l+1}(n_t) \leq ST_l(n_t).$$

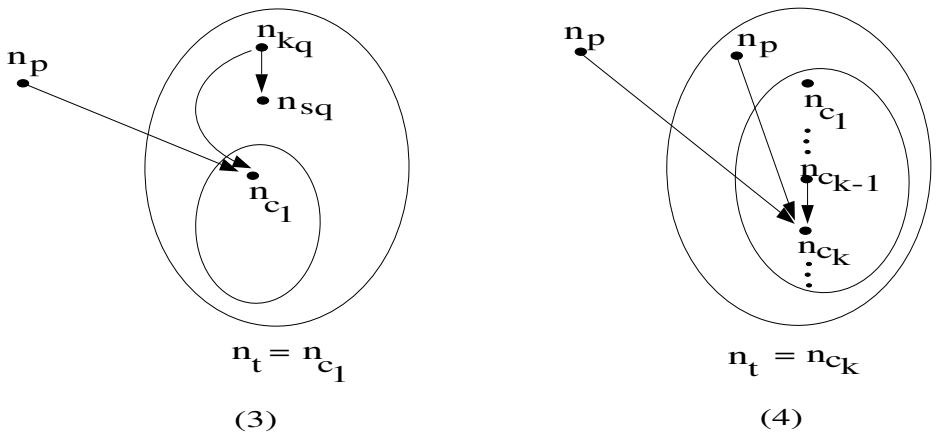


Figure 8: Case 3 and 4 of **construct** procedure.

3. n_t **is the first task** n_{c_1} **in** LC.

Refer to figure 8(3). We check the termination condition of **Extract** to determine the start time of n_{c_1}. For case 1 of extraction, n_{c_1} has no predecessor in NLC; thus moving it into processor j

1. n_t **is not in** NLC or LC.

Refer to figure 7(1). The node n_t is in some processor $r \neq i$. We have that

$$RT_l(n_t) = \max\{P_l(n_t, pred_l(n_t)),\ Q_l(n_t, R_l(n_t))\}$$

$$RT_{l+1}(n_t) = \max\{P_{l+1}(n_t, pred_{l+1}(n_t)),\ Q_l(n_t, R_{l+1}(n_t))\}.$$

This is because each $n_p \in PRED(n_t)$ sends the *data in parallel* immediately after completion of its execution and since the architecture is a *clique* the arrival time of the data for n_t is not affected if LC becomes a new processor j; see the assumptions on parallel send and clique architecture in section 2.1.

Because the predecessors of n_t in the same processor are not changed by the extraction we have that

$$pred_{l+1}(n_t) = pred_l(n_t), \qquad R_{l+1}(n_t) = R_l(n_t).$$

The above combined with the induction hypothesis $ST_l(n_p) \geq ST_{l+1}(n_p)$ for all $n_p \in PRED(n_t)$ imply

$$P_{l+1}(n_t, pred_{l+1}(n_t)) \leq P_l(n_t, pred_l(n_t)), \quad Q_{l+1}(n_t, R_{l+1}(n_t)) \leq Q_l(n_t, R_l(n_t)).$$

Therefore

$$RT_{l+1}(n_t) \leq RT_l(n_t) \leq ST_l(n_t)$$

and because the execution order in S_{l+1} is chosen to be the same as in S_l we can always set

$$ST_l(n_t) \geq ST_{l+1}(n_t).$$

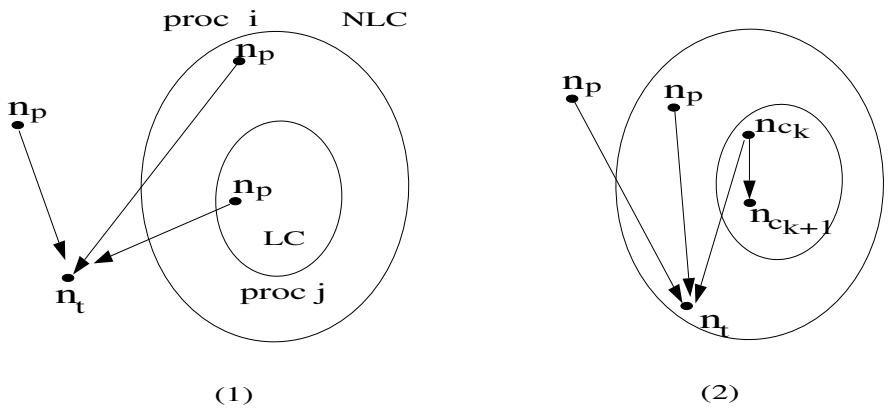


Figure 7: Case 1 and 2 of **construct** procedure.

2. n_t **is in** $NLC - LC$

Refer to figure 7(2). Since tasks in LC are extracted from processor i, the ready time of n_t is possibly delayed when one of its immediate predecessors, say n_{c_k}, is in LC. This is because n_{c_k} is moved from processor i to processor j and the communication between n_{c_k} and n_t becomes nonzero.

that $NLC = \{n_1, n_2, \ldots, n_m\}$ is the nonlinear cluster in processor i and that the execution order imposed by S_l is $\{\preceq_i\} = (n_1, n_2, \ldots, n_m)$.

Extract a linear cluster from a nonlinear cluster:

- Extract n_m first, by removing the node from NLC.
- Assume n_k has been extracted. Then, we show how to extract the next node. There are three cases for performing the extraction:

  Case 1: If $pred_l(n_k) = \emptyset$ then stop the extraction. The node n_k is the first node in LC.

  Case 2: If $pred_l(n_k) = (n_{k_1}, \ldots, n_{k_q})$ and there exists **no** successor n_{s_q} of n_{k_q} inserted between n_{k_q} and n_k in $\{\preceq_i\}$, then extract n_{k_q} and assign it into LC.

  Case 3: If $pred_l(n_k) = (n_{k_1}, \ldots, n_{k_q})$ and there exists a successor n_{s_q} of n_{k_q} such that $n_{k_q} \preceq_i n_{s_q} \preceq_i n_k$, then stop the extraction. The node n_k is the first node of LC.

Notice that the while loop assures that all linear clusters have been extracted from NLC. Also the extracting continues even if NLC becomes linear since checking of linearity of NLC is done after the extraction stops.

Define $ST_l(n_p)$ the *starting time* and $RT_l(n_p)$ the *ready time* of node n_p in S_l. The ready time is the time when n_p has received all data from its immediate predecessors and is ready for execution. We have that

$$ST_l(n_p) \geq RT_l(n_p).$$

We also need to define two functions

$$P_l(n_t, X) = \max_{n_p \in X}\{ST_l(n_p) + \tau_p\}, \qquad Q_l(n_t, X) = \max_{n_p \in X}\{ST_l(n_p) + \tau_p + c_{p,t}\}$$

for schedule S_l where X is a subset of $PRED(n_t)$.

Next we construct a schedule S_{l+1} from S_l with the same execution order, i.e. if $n_j \preceq_k n_p$ in S_l then the order will remain the same in S_{l+1} if the nodes are in the same processor. We construct S_{l+1} by topologically examining each task in G and showing that $ST_l(n_p) \geq RT_{l+1}(n_p)$. Consequently we can define S_{l+1} such that $ST_l(n_p) \geq ST_{l+1}(n_p)$ for all $n_p \in G$, which imply that the parallel times satisfy $PT_{l+1} \leq PT_l$. Assume that for schedule S_l the extracted linear cluster from $NLC = \{n_1, n_2, \ldots, n_m\}$ is $LC = \{n_{c_1}, n_{c_2} \ldots, n_{c_h}\}$ and $n_{c_h} = n_m$. We present the construction inductively:

Construct a schedule:

- First set $ST_{l+1}(n_p) = ST_l(n_p)$ for all entry nodes n_p which have no predecessors.
- Assume for a node n_t that $ST_l(n_p) \geq ST_{l+1}(n_p)$ for all $n_p \in PRED(n_t)$. We only consider schedules S_{l+1} with the same execution order as S_l. There are four distinct cases to consider.

This indicates that the precedence constraint $< n_1, n_k >$ is automatically satisfied. ■

Theorem 1 *For any nonlinear clustering of a coarse grain DAG, there exists a linear clustering with less or equal parallel time.*

Proof : According to Lemma 1, we can assume that the given DAG has no transitive edges. There are two parts in the proof. The first part describes a procedure that extracts linear clusters from nonlinear clusters. The second part shows how to construct a schedule at each step of the extraction whose parallel time is less or equal to the schedule in the previous step.

Let us assume that $CLUSTER(0)$ is the set of the initial nonlinear clustering and S is its optimum schedule under the assumption that each cluster is executed on a separate processor of a clique architecture. The set $CLUSTER(0)$ consists of nonlinear NLC and linear LC clusters and there exists at least one $NLC \in CLUSTER(0)$. The optimum schedule defines execution sequence, $\{\preceq_i\}$, of the tasks for each processor i. The linearization procedure is given below:

INPUT: $CLUSTER(0)(nonlinear)$; S;
OUTPUT: $CLUSTER(l)(linear)$; S_l;
$l = 0$; $S_0 = S$;
FOR each processor i that contains a nonlinear cluster NLC **DO**
    WHILE NLC is not linear **DO**
        Extract a linear cluster from NLC and assign LC to a new processor j;
        $NLC = NLC - LC$;
        $CLUSTER(l+1) = CLUSTER(l) \cup \{LC\}$;
        Construct a schedule S_{l+1} from S_l with smaller parallel time;
        $l = l + 1$;
    ENDWHILE
ENDFOR

Define

- $PRED(n_k)$ to be the set of all immediate predecessors of n_k in G.
- $pred_l(n_k) = (n_{k_1}, \ldots, n_{k_t}, \ldots, n_{k_q})$ to be the sequence of all immediate predecessors of n_k assigned to the same processor as n_k *for schedule* S_l. This sequence is derived from the order imposed by S_l.
- $R_l(n_t) = PRED(n_t) - pred_l(n_t)$, the set of all immediate predecessors of n_t that are not in the same processor as n_t for schedule S_l. The "$-$" symbol is the set difference operation.

We first show how to extract a linear cluster from a nonlinear cluster using a bottom-up traversal of the graph and then show that it is possible to construct a schedule for the new clustering with less or equal parallel time. We show both the extraction and the construction inductively. Assume

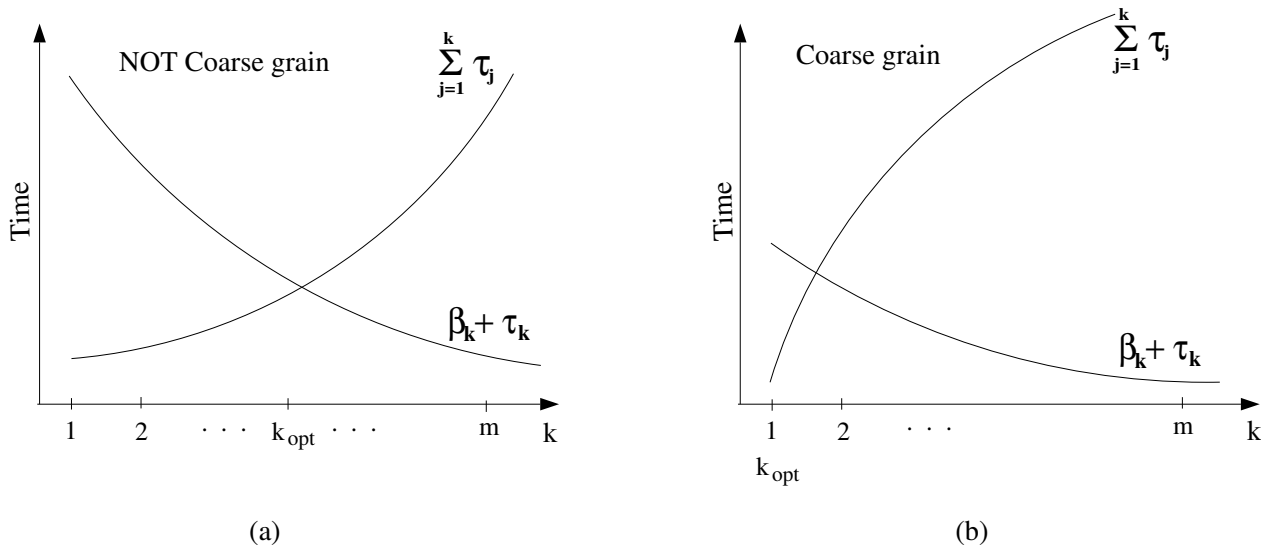


Figure 6: Parallelization vs. sequentialization for the join set.

clustering is a linear clustering. The question that remains to be investigated is that if this result can be extended to general DAGs using our granularity definition. We will do this in the next section.

4 Parallelization vs. sequentialization for a general DAG

The theorem below generalizes the result, observed of the example in the previous section, that linear clustering is the best choice for coarse grain DAGs. This result provides a theoretical justification for using linear clustering in numerical computing and VLSI processor design, S.Y. Kung [20]. We first need to show the following lemma.

Lemma 1 *Given a coarse grain DAG. Then any schedule of the reduced graph, derived by deleting all transitive edges of the DAG, still satisfies all precedence constraints of the given DAG.*

Proof: Suppose that there is a path $n_1 \rightarrow n_2 \rightarrow ... \rightarrow n_k$ where $k > 2$, and that there is also a transitive edge from n_1 to n_k. Next assume that the transitive edge $< n_1, n_k >$ is deleted. The task n_k still can not be ready for execution for any schedule until the completion of all tasks n_1, n_2, ..., and n_{k-1}, that is,

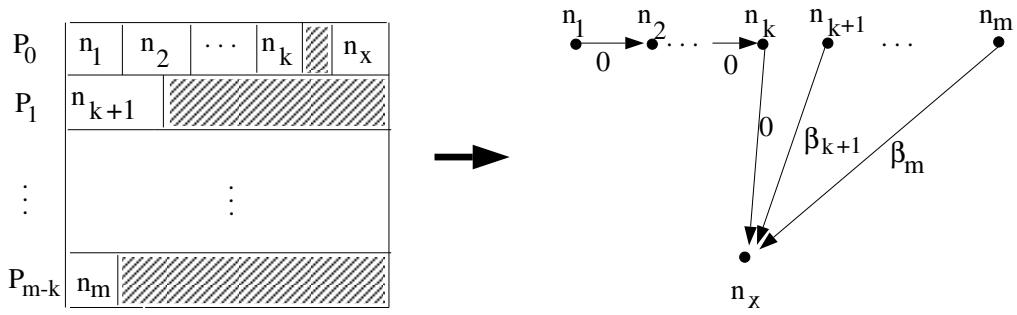
$$ST(n_k) \geq CT(n_1) + \sum_{i=2}^{k-1} \tau_i.$$

Since the DAG is coarse grain then

$$c_{1,k} \leq \tau_2 \leq \sum_{i=2}^{k-1} \tau_i$$

which implies that

$$ST(n_k) \geq CT(n_1) + c_{1,k}.$$



(c) Gantt chart (d) Scheduled DAG

Figure 5: The Gantt chart and scheduled DAG for the optimum clustering algorithm of the join set.

$k = 1;$
WHILE $(\sum_{j=1}^{k} \tau_j \leq \tau_k + \beta_k)$ **DO**
$\beta_k = 0;$
$k = k + 1;$
ENDWHILE
$k = k - 1;$

Initially all tasks are mapped on a separate processor in the clique, see figure 4 (a). At each step k of the algorithm, the edge $< n_k, n_x >$ is visited and if the parallel time reduces by zeroing that edge then this edge is zeroed, figure 4 (b). This implies that all tasks connected with a zero edge must be sequentialized, see figures 4 (b) and 5 (d). Therefore, the task n_j, $j = 1 : k$ will be mapped in the same cluster as long as the execution time for this cluster $\tau_x + \sum_{j=1}^{k} \tau_j$ remains less than $\tau_x + \beta_k + \tau_k$. Each clustering step k trades off parallelism with sequentialization depending on whether or not the the while loop condition is satisfied. The question is, what role does the granularity play in the sequentialization process? Notice that the algorithm will stop at some step k if the condition in the while loop becomes untrue, implying the following inequalities

$$\sum_{j=1}^{k} \tau_j \leq \beta_k + \tau_k, \qquad \sum_{j=1}^{k+1} \tau_j \geq \beta_{k+1} + \tau_{k+1}.$$

These inequalities determine the optimum trade off point k_{opt} between sequentialization and parallelization. In figure 6 (a) and (b) we show two cases of the graphs for $\sum_{j=1}^{k} \tau_j$ and $\beta_k + \tau_k$ along with their trade off points k_{opt}. If the DAG is not coarse grain then $k_{opt} \geq 1$, see figure 6 (a), while for the coarse grain case $k_{opt} = 1$, see figure 6 (b).

To explain this result, rewrite the first inequality above as follows

$$(k-1) \min_{l=1:m}\{\tau_l\} \;\leq\; \sum_{j=1}^{k-1} \tau_j \;\leq\; \beta_k \;\leq\; \max_{l=1:m}\{\beta_l\} \quad \Longrightarrow \quad k \leq\; 1 + \frac{1}{g(G)}.$$

The last inequality intuitively explains why we have included the adjacent nodes of n_x in our granularity definition. The interesting conclusion is that if $g(G) > 1$ then the last inequality implies that $k_{opt} = 1$, since $k < 2$ and k is an integer greater or equal to one, which means that the optimum

We introduce the *grain* of a task as

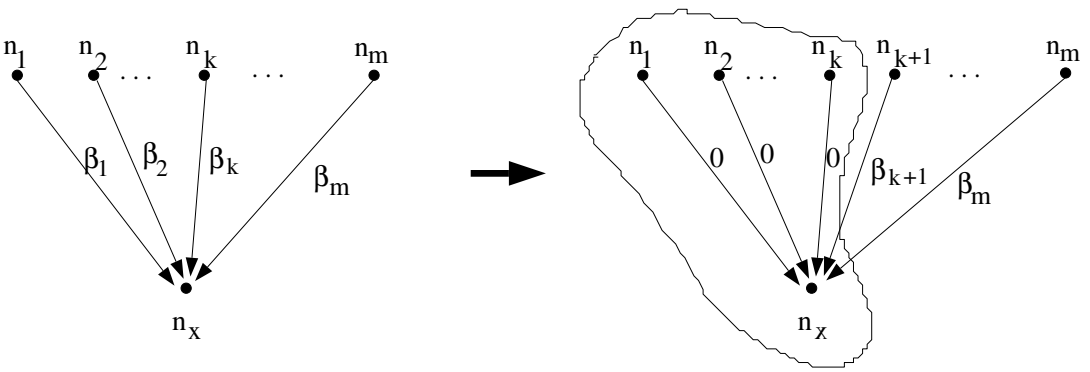
$$g_x = \min\{g(F_x), g(J_x)\}$$

and the *granularity* of a DAG as

$$g(G) = \min_{x=1:v}\{g_x\}.$$

We call a DAG *coarse grain* if $g(G) \geq 1$. If $\tau_k = R$ and $c_{i,k} = C$ then the grain of every task and the granularity of the DAG reduces to the ratio R/C which is the same as Stone's definition. For coarse grain DAGs each task receives or sends a small amount of communication compared to that the computation of its adjacent tasks.

Consider the DAG in figure 1 (a) and assume that $\tau_i = 1$ for all tasks and the communication cost shown in that figure. The width of this graph is 3. The granularity of the DAG is 1/2 and therefore the DAG is not coarse grain. Figure 1 (b) shows a nonlinear clustering of the DAG in which two parallel nodes n_2 and n_3 are mapped in a cluster and two other parallel nodes n_6 and n_7 in another. This mapping reduces the width from 3 to 2. Scheduling the nonlinear clusters on two processors results in an optimum parallel time of 7, see figure 2 (a). Notice that linear clustering in figure 1 (c) does not affect the width. Scheduling the linear clusters on three processors results in a schedule of parallel time 8 which is higher than the nonlinear clustering. If on the other hand the task weights were taken to be $\tau_i = 10$ then the DAG becomes coarse grain since the granularity $g(G) = 5$. In this case, linear clustering will have shorter parallel time than the nonlinear one. This example gives some intuitive feeling on the impact of the granularity on the decision of sequentialization (corresponding to nonlinear clustering) vs. parallelization (corresponding to linear clustering). It does not, however, explicitly show how the DAG granularity impacts clustering strategies. We will show that with another example.



(a) Join DAG (b) Clustered DAG

Figure 4: An optimum clustering algorithm for the join set with $\tau_j+\beta_j \geq \tau_{j+1}+\beta_{j+1}, \quad j = 1 : m-1$.

We consider the join set J_x with $c_{j,x} = \beta_j$, $j = 1 : m$, see figure 4 (a), and assume that the nodes and edges are sorted such that $\tau_j + \beta_j \geq \tau_{j+1} + \beta_{j+1}, \quad j = 1 : m - 1$.

An algorithm that determines the optimum clustering for this DAG is described below

amount of computation is done between communications will be more efficient than fine-grain algorithms."

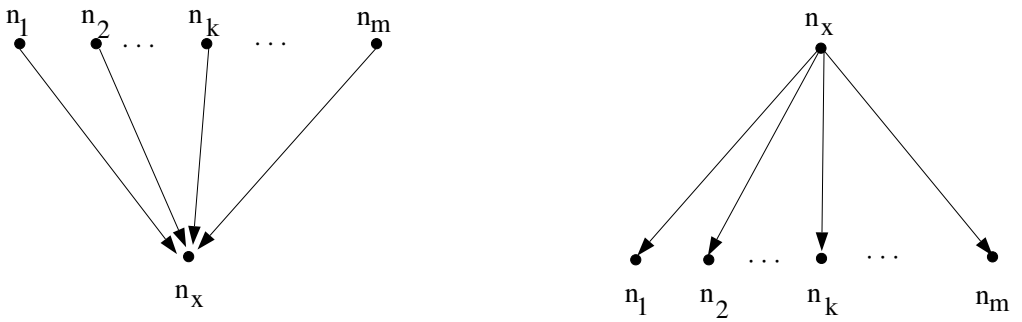
Stone [31] considers a task graph in which every task computes R units of time and communicates with all other tasks at an overhead cost of C. He defines the ratio R/C as the *task granularity* and shows that it is this ratio that determines the optimum trade off point between parallelism and sequentialization.

Gerasoulis and Venugopal [13] have extended Stone's grain definition for arbitrary DAG's and have introduced a new quantity called the *granularity of a DAG*, defined as

$$g = \min_{x=1:v}\{\tau_x / \max_j c_{x,j}\}$$

where $c_{x,j}$ are communication costs of the edges going out from node n_x and τ_x is the computation cost of that node. A detailed discussion of granularity may be found in [13].

Here we introduce a different granularity for a DAG. We will demonstrate that our granularity plays an important role in determining the trade off point between parallelization and sequentialization of tasks in arbitrary DAGs. The reason for this new granularity definition will become apparent with the results of the next few sections. For example, in the next section we prove that linear clustering is better than nonlinear clustering for arbitrary coarse grain DAGs. This is not true for other granularity definitions, and this is the main justification for choosing our granularity definition.



(a) Join set J_x (b) Fork set F_x

Figure 3: Fork and join sets.

A DAG consists of *fork* or/and *join* sets such as the ones shown in figure 3. The join set J_x consists of all immediate predecessors of node n_x. The fork set F_x consists of all immediate successors of node n_x. Let $J_x = \{n_1, n_2, \ldots, n_m\}$ and define

$$g(J_x) = \min_{k=1:m}\{\tau_k\} / \max_{k=1:m}\{c_{k,x}\}.$$

Similarly let $F_x = \{n_1, n_2, \ldots, n_m\}$ and define

$$g(F_x) = \min_{k=1:m}\{\tau_k\} / \max_{k=1:m}\{c_{x,k}\}.$$

newly-zeroed edge is in the critical path then the critical path length may be reduced. If the zeroed edge is not in the critical path then the critical path length does not change. ■

Property 2 *(***Simplification of schedule computation***). The length of the optimal schedule for a linear clustering on a DAG can be derived in $O(e+v)$ time complexity.*

Proof: When m linear clusters are scheduled on exactly m separate processors, task ordering for each processor is uniquely determined by the single dependence chain of each cluster. An optimum scheduled graph is derived by zeroing the edges in each cluster. The edge costs between clusters remain unchanged. The parallel time is the length of the critical path of the scheduled graph. The length of the critical path for a DAG can be determined in $O(v+e)$ time complexity by topologically traversing the DAG, see Manber [23] p. 202. ■

Property 3 *(***Preservation of Parallelism***). The number of linear clusters for any linear clustering on a DAG is greater or equal to the the width of this DAG.*

Proof: Assume that the number of linear clusters m is less than w, the width of this DAG. We consider the maximal independent set of this DAG, $MIS = \{n_1, n_2, ..., n_w\}$. Because $m < w$, there exist at least two nodes in MIS which are in the same linear cluster. Then there is a path between these two nodes, namely, they are not independent, which is a contradiction. ■

The preservation of parallelism property provides flexibility in the second step of two step approach to scheduling DAGs. Better efficiency can be achieved if there is sufficient parallelism in the clustered DAG. In conclusion of this section, linear clustering has many desirable properties and it is of interest to investigate when linear clustering is a better choice than nonlinear clustering. We will do that in the next few sections.

3 The granularity of a task graph

In this section, we define the grain of a task and the granularity of a DAG. The grain definition is based on a local ratio between computation and communication while the granularity is a global quantity characterizing a DAG. It is well known that a trade off between parallelization and sequentialization is needed to avoid unnecessary communication and obtain the optimum schedule, e.g. Golub and VanLoan [9], pp. 270-271, Stone [31] p. 309. For example, Heath and Romine [10] p. 559 write

> " Another important characteristic determining the overall efficiency of parallel algorithms is the relative cost of communication and computation. Thus, for example, if communication is relative slow, then coarse grain algorithms in which relative large

same processor. Initially, each task is in a separate processor and no edge has been zeroed as of yet, the parallel time PT_0 is the length of the critical path including communication delay.

As an example of a clustering algorithm we consider Sarkar's [30], pp. 123-131, heuristic "if the parallel time at step i, PT_i, does not increase by zeroing the highest edge cost then zero this edge". The edges are first sorted according the highest edge cost first principle. Then at each clustering step the edges are visited in the sorted list order and zeroed or not depending if the parallel time increases or not. Assume that $\tau_i = 1$ and apply Sarkar's algorithm on the DAG in figure 1. The sorted edges list is

$$\{< n_1, n_2 >, < n_1, n_3 >, < n_1, n_4 >, < n_5, n_8 >, < n_6, n_8 >, \quad < n_7, n_8 >, < n_2, n_5 >, \\ < n_3, n_6 >, < n_4, n_7 >\}.$$

In the first step of Sarkar's algorithm $PT_0 = 9$. Zeroing the edge $< n_1, n_2 >$ then $PT_1 = 9 \leq PT_0$ and this edge is zeroed. In the next step the edge $< n_1, n_3 >$ is zeroed and again $PT_2 = 9$ and so on. Finally all edges will be zeroed which implies that all tasks are mapped in one cluster whose parallel time is $PT_9 = 8$. The complexity for Sarkar's algorithm is $O(e(v + e))$. This cost is due to the re-computation of the parallel time PT_i at each clustering step since the scheduled DAG must be topologically traversed and there are e such traversals with worst time complexity of $O(e + v)$ for each traversal.

Let us now impose the linearity constraint on Sarkar's algorithm. This implies that at each step the edges that will create nonlinear clusters, if they are zeroed, do not have to be visited. For example, once the edge $< n_1, n_2 >$ is zeroed, the edges $< n_1, n_3 >$ and $< n_1, n_4 >$ do not have to be visited. The next edge to be visited and zeroed will be $< n_5, n_8 >$ and the following up will be $< n_2, n_5 >$. The resulting linear clusters are

$$M_0 = \{n_1, n_2, n_5, n_8\}, \quad M_1 = \{n_3, n_6\}, \quad M_2 = \{n_4, n_7\}$$

with parallel time equal to 8.

The linearity constraint ensures that the parallel time PT_i does not increase at each zeroing step. Therefore, Sarkar's algorithm does not have to compute the parallel time at each clustering step. The complexity of the algorithm reduces from $O(e(v + e))$ to $O(e \log e)$ which is the cost for sorting the edges if the linearity is imposed as a constraint in the algorithm. In the next subsection, we present some fundamental properties of linear clustering.

2.3 Fundamental properties of linear clustering

Property 1 (**Monotonicity in parallel time reduction**). *The parallel time at each clustering step does not increase by zeroing one edge of a given DAG linearly, namely,* $PT_i \leq PT_{i-1}$.

Proof: Because of linearity constraint the task ordering in the scheduled graph remains unchanged by the clustering. The parallel time is the length of the critical path of the scheduled graph. If the

and $ST(n_j)$ by topologically traversing the DAG in $O(v+e)$ time, see Manber [23]. For example, a dominant sequence in figure 2 (b) is $(n_1, n_4, n_7, n_6, n_8)$ with a length of 7. A clustering algorithm based on the reduction of the dominant sequence, has been shown to have the best performance in terms of computational complexity and accuracy compared to many other clustering algorithms, see Gerasoulis, Venugopal and Yang [12].

To determine clusterings using the min-max model, we need to specify an execution model and the architecture. We will make the following assumptions:

- *Duplication* of the execution of tasks in separate clusters is not allowed. This is because duplication results in an increase in the space complexity since data must also be duplicated.
- The task communicates only before starting and after completing execution. The tasks communicate via asynchronous message passing and they can receive and send *data in parallel*. The task starts execution as soon as all of its input data have arrived in the local message buffer. It sends its output data to successor tasks immediately after completion of its execution. This task model is the same as the compile time macro-dataflow model in which communication may overlap with the computation of each task, e.g. Wu and Gajski [32].
- The architecture is a completely connected *clique* network of processors with local memory and message buffers. The clique has an unbounded number of processors and the communication protocol is via asynchronous message passing.

Another problem related to clustering is to determine the optimum schedule and its parallel time once a clustering is given. The constraints for this minimization problem are that all tasks in a cluster must execute in the same processor and that every cluster must execute in a separate processor. We do not know if this problem is NP-complete or not for a general clustering. There is, however, a class of clusterings, the linear clusterings, for which determining the parallel time can be done in polynomial time.

2.2 Clustering algorithms

In this subsection, we briefly describe a clustering algorithm using a framework for clustering DAGs presented in [12] and demonstrate the usage of linear clustering strategy. Many clustering algorithms have been proposed in the literature, e.g. Sarkar [30], Kruatrachue and Lewis [18], McGreary and Gill [22], Kim and Browne [17], Kim [16], Wu and Gajski [32], Gerasoulis, Venugopal and Yang [12] and Girkar and Polychronopoulos [14]. Most of the proposed clustering algorithms perform a sequence of graph traversal steps where at each step a set of edges is examined and based on a heuristic the edge costs are zeroed. A zeroed edge implies that the tail and head nodes in that edge will be mapped in the same cluster. PT_i is the parallel time after the completion of step i, estimated by a scheduling algorithm that assumes that the tasks of each cluster are executed in the

of the nodes of G onto m clusters $\{M_0,\ M_1, \ldots,\ M_{m-1}\}$, so that a certain goal is achieved. One of the goals commonly used in the literature is the minimization of the parallel time on an *unbounded number* of processors, Sarkar [30]. This is also known as the parallel time minimization or min-max criterion for clustering,

$$\min_{S}\{\ \max_{j=1:v} CT(n_j)\ \}$$

where S is the set of all possible schedules on an unbounded number of processors and $CT(n_j)$ is the completion time of node n_j. The clustering that attains the minimum parallel time is called the *optimum clustering*. This problem has been shown to be NP-complete by Sarkar [30], Chretienne [2] and Papadimitriou and Yannakakis [26].

A clustering is called *linear* if every cluster consists of tasks that are *one* simple directed path in G without considering transitive edges; otherwise it is called *nonlinear*, see figure 1. A simple path is a path in which all nodes except possibly the first and the last are distinct. If an algorithm produces a linear clustering for any DAG then we say that it satisfies the *linearity constraint*, see Kim [16] for one such algorithm.

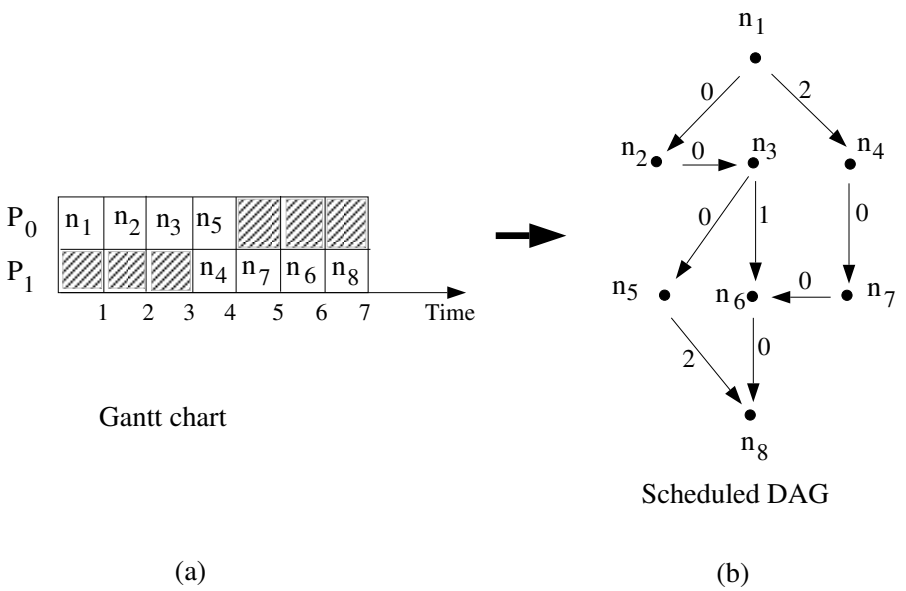


Figure 2: (a) is a Gantt chart. (b) is the scheduled graph derived from the Gantt chart.

A schedule is defined by a processor assignment mapping, $PA(n_j)$, of the nodes of G onto the p processors and by the starting times, $ST(n_j)$, of all nodes. In figure 2, we show a *Gantt chart* of a schedule for the DAG in figure 1 (b) on $p = 2$ processors. The Gantt chart completely describes the schedule since it defines both $PA(n_j)$ and $ST(n_j)$, see Stone [31]. Another way of describing a schedule is through its *scheduled DAG*. The scheduled DAG is a weighted DAG which is derived from the Gantt chart using the execution ordering of the nodes imposed by the schedule within each processor and the communication edges between processors. Communication edges between two nodes mapped in the same processor are assigned zero edge cost. For example, the scheduled graph corresponding to the Gantt chart in figure 2 (a) is given in figure 2 (b). The critical path of the scheduled DAG is called a *dominant sequence* of the initial DAG. The length of the dominant sequence represents the parallel time. From the scheduled DAG we can derive $PA(n_j)$

Finally, in section 7 we discuss some open questions and present an example which shows that the results of the paper could be useful even in cases that the DAG is not coarse grain. For example, there are cases in which the granularity is less than one for a small subgraph of the DAG and greater than one for the major part of the DAG. One such case is the Gaussian-Elimination and linear clustering can be shown to be asymptotically optimal in this case.

2 The clustering problem

2.1 Definitions and assumptions

A weighted DAG is a tuple $G = (V, E, \mathcal{C}, \mathcal{T})$, where $V = \{n_j, j = 1 : v\}$ is the set of nodes and $v = |V|$, $E = \{e_{i,j} =< n_i, n_j >\}$ is the set of communication edges and $e = |E|$. The set $\mathcal{C}$ is the set of edge communication costs and $\mathcal{T}$ is the set of node computation costs. The value $c_{i,j} \in \mathcal{C}$ is the communication cost incurred along the edge $e_{i,j} \in E$, which is zero if both nodes are mapped in the same processor. The value $\tau_i \in \mathcal{T}$ is the computation cost for node $n_i \in V$. The *width* of a DAG is the size of the maximal set of independent tasks. The *length* of a path is the summation of all node computation and edge communication costs in that path. The *critical path* is the path with the longest length in the DAG. For example, if we assume that the computation weights are $\tau_i = 1$ for all nodes of the DAG in figure 1 (a), then $\{n_1,\ n_3,\ n_6,\ n_8\}$ is a critical path with length 9. The width of the DAG is 3.

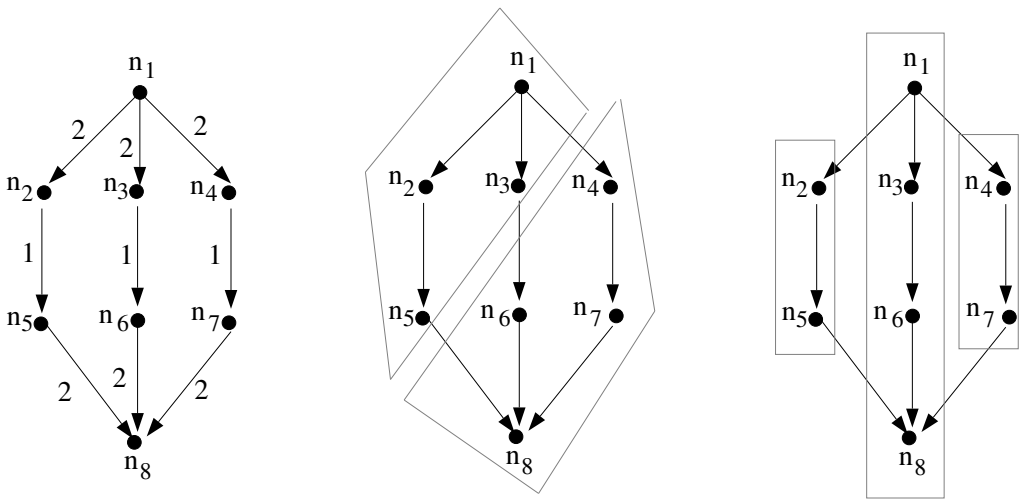


Figure 1: (a) is a weighted DAG. (b) is a nonlinear clustering and (c) is a linear clustering.

As we mentioned in the introduction, clustering is a mapping of the tasks of a DAG onto m clusters. More specifically, the problem is to determine a mapping

$$map(n_j) = i, \quad j = 1 : v, \quad i = 0 : m - 1$$

Graham [15] has shown that list scheduling, a polynomial time algorithm, is within 50% of the optimum, see also Sarkar [30] p. 60. For the asynchronous message passing communication model, in which a task receives its input in parallel from its predecessors before starting its execution and sends its output in parallel to all of its successors immediately after completing its execution, this question still remains open with the exception when task duplication is allowed, Papadimitriou and Yannakakis [26].

Assuming task duplication and an unbounded number of processors, Papadimitriou and Yannakakis [26] have shown that a polynomial time algorithm exists which is within 50% of optimum. Task duplication, however, could result in considerable increase in the space complexity of the clustering algorithms and data duplication in the actual program execution. For example, Anger, Hwang and Chow [1] propose an optimum algorithm for scheduling coarse grain DAGs using duplication. Their space complexity, however, increases from $O(v)$ to $O(v^2)$ where v is the number of nodes in the DAG. In this paper, we will analyze the first step, in the two step approach to scheduling above, and derive upper bounds for performance for the case of coarse grain DAGs.

Gerasoulis and Venugopal [13] have proposed a granularity definition for DAGs. Using this granularity definition and assuming no task duplication, it is shown that the ratio of the parallel time for any linear clustering to that of the optimum parallel time for a DAG is bounded above by one plus the inverse of the granularity. This implies that if the granularity is at least one, the ratio is at most two. This result is similar to Graham's [15] bound for list scheduling without communication delays, but it is true only for coarse grain DAGs.

In section 2, we review some basic definitions and assumptions on the task model, clustering and scheduling. We also introduce a linear and nonlinear clustering. We show that linear clustering possesses several desirable properties. For example, the complexity of linear clustering is linear in the size of edges and nodes of the DAG for certain algorithms.

In section 3, we discuss various granularity definitions. We introduce a new granularity definition and show how this definition better captures the trade off point between parallelization (corresponding to linear clustering) and sequentialization (corresponding to nonlinear clustering) of DAGs. In section 4, we show that linear clustering is better than nonlinear clustering for arbitrary coarse grain DAGs. This is not true for other granularity definitions, and this is the main justification for choosing our granularity definition. In section 5, we show that the performance bound of linear clustering by Gerasoulis and Venugopal [13] can be extended to our granularity definition. These two results provide a theoretical justification for using linear clustering in numerical computing and VLSI processor design.

In section 6, we apply the results of the paper to the Gauss-Jordan linear algebra algorithm and prove why the ring architecture is the best architecture for the Gauss-Jordan DAG when the number of processors is equal to the number of the columns of the matrix. This is done by showing that the optimum parallel time on a ring is equal to the optimum parallel time on the clique for this special case.

for each task do not have to be given. Every task in a cluster must execute in the same processor. A clustering is called linear if every cluster is one simple directed path in the task graph; otherwise is called nonlinear. Clustering has been used as a pre-processing step in scheduling task graphs on parallel architectures. For example, Sarkar [30], p. 124, has proposed a two step approach to compile-time scheduling:

1. Find clusters by scheduling the task graph on an unbounded number of processors on a completely connected *virtual* architecture. Sarkar calls this step the "internalization pre-pass".

2. Schedule the clusters on p *physical* processors.

Sarkar [30] shows that determining the clustering that minimizes the parallel time on an unbounded number of processors is NP-complete. A similar result has been shown by Chretienne [2] and Papadimitriou and Yannakakis [26]. Polynomial time heuristic algorithms for clustering have been proposed by Sarkar [30], Kruatrachue and Lewis [18], Kim and Browne [17], Kim [16], McGreary and Gill [22], Wu and Gajski [32] and others. A comparison of these algorithms is given in Gerasoulis, Venugopal and Yang [13].

The two step approach above has also been used to schedule DAGs on MIMD architectures and in VLSI processor design, S. Y. Kung [20]. Examples are the Gaussian-Elimination and Gauss-Jordan algorithms in which the DAG is first clustered by using the data locality assumption, i.e. processors only modify data that they own. Next, clusters are mapped onto the p processors using wrap mapping and scheduled using either the naive or compute-ahead task orderings, e.g. Ortega [25], pp. 88 and 241, Gerasoulis and Nelken [11], Robert, Tourancheau and Villard [27], Moler [24], Geist and Heath [8] and Davis [5]. A reference for VLSI processor design is S. Y. Kung [20] pp. 133-136, p. 165, where the two step approach is known as (1) Processor Assignment or processor projection and (2) Scheduling. Linear clustering is widely used in the first step of VLSI processor design along with linear schedulings in the second step in which nodes in a hyperplane execute in parallel. As far as we know, there has not been a theoretical justification for linear clustering based on the granularity of the DAG.

A question that arises is how good is the two step scheduling approach above. Girkar and Polychronopoulos [14] have given an example that demonstrates that the two step approach may fail to determine the optimum schedule. However, since the scheduling problem is NP-complete, a polynomial time algorithm that is "close" to the optimum is desirable. By "close" we mean that the parallel time is at least within a factor of two of the optimum. Kim and Browne [17] have presented experiments showing empirically that the two step approach is superior to direct scheduling when communication is present.

For the shared memory communication model in which a task reads its input before starting execution from the shared memory and writes the output after completion to the shared memory,

ON THE GRANULARITY AND CLUSTERING OF DIRECTED ACYCLIC TASK GRAPHS

Apostolos Gerasoulis* and Tao Yang
Department of Computer Science
Rutgers University
New Brunswick, NJ 08903.

September 12, 1990

Abstract

Clustering has been used as a compile time pre-processing step in the scheduling of task graphs on parallel architectures. A special case of the clustering problem arises in scheduling an unbounded number of completely connected processors. Using a generalization of Stone's granularity definition, the impact of the granularity on clustering strategies is analyzed. A clustering is called linear if every cluster is one simple directed path in the task graph; otherwise is called nonlinear. For coarse grain directed acyclic task graphs (DAGs), a completely connected architecture with unbounded number of processors and under the assumption that task duplication is not allowed, the following property is shown: For every nonlinear clustering there exists a linear clustering with less or equal parallel time. This property, along with a performance bound for linear clustering algorithms, shows that linear clustering is the best choice for coarse grain DAGs. It provides a theoretical justification for linear clustering which has been widely used in numerical computing and VLSI processor design. Finally, an example is given to demonstrate the applicability of the property. It is shown that the optimal parallel time for the coarse grain Gauss-Jordan algorithm on a ring architecture is equal to the optimum parallel time on the clique, when the number of processors is equal to the number of columns n of the matrix.

Index Terms– Clustering, DAGs, Gauss-Jordan algorithm, granularity, partitioning, task scheduling.

1 Introduction

In this paper, we study the problem of clustering directed acyclic task graphs (DAGs). Scheduling is defined by a mapping of the nodes of a DAG onto p processors plus the starting times for each task. Clustering is also a mapping of the nodes of a DAG onto m clusters, but the starting times

*Supported by a Grant No. DMS-8706122 from NSF.