A Flexible Clustering and Scheduling Scheme for Efficient Parallel Computation

S. Chingchit , M. Kumar
School of Computing Science,
Curtin University,
Perth, GPO Box U1987, Australia.
chingchi,kumar@cs.curtin.edu.au

L.N.Bhuyan
Department of Computer Science
Texas A&M University
College Station, TX 77843 USA
bhuyan@cs.tamu.edu

Abstract

Clustering and scheduling of tasks for parallel implementation is a well researched problem. Several techniques have been presented in the literature to improve performance and reduce problem execution times. In this paper we present a novel approach where clustering and scheduling of tasks can be tuned to achieve maximal speedup or efficiency. The proposed scheme is based on the relation between the costs of computation and communication of task clusters. In this paper, we show how clustering can be adapted to suit different architectures and number of available processors. The proposed efficient clustering and scheduling algorithm is flexible : the clustering and scheduling can be tuned to suit bounded or unbounded number of processors and/or parallel computing environment. Comparative studies indicate superior efficiency compared to most other schemes proposed in recent years.

1. Introduction

For purposes of practical implementations, a clustering and scheduling algorithm needs to address several issues: 1) availability of processors, 2)granularity of the tasks in the given problem, 3)processor speed, 4) network bandwidth, 5)execution time, 6) efficiency, 7) load balance, 8) complexity of the clustering and scheduling algorithm, and so on.

The main objective of clustering is to combine interdependent tasks for execution on the same processor in order to reduce communication costs. The objective of scheduling is to execute tasks in different processors such that the parallel execution time is reduced. The proposed clustering and scheduling scheme is motivated by the work done by Indurkhya et al. [6] on efficient partitioning of programs and the work of several researchers on clustering and scheduling techniques [4], [8],[9]. In particular, Gerasoulis and Yang [3] have done pioneering work in task scheduling and allocation. Kwok and Ahmad [7] carried out exhaustive studies to compare and contrast a variety of scheduling algorithms. They defined *normalised schedule length* to compare the execution times of various scheduling algorithms on unbounded number of processors. Their studies indicate that no single scheduling technique can be applied to all types of problems.

Eventhough theoretically, a scheduling algorithm may show high performance, however, implementation results may be quite different. Actual implementation results would depend on processor speed, network bandwidth and the communication network[5]. To achieve efficient performance, a scheduling strategy should take into account the characteristics of the underlying architecture.

In [6] Indurkhya et al. proposed two policies for efficient partitioning of tasks: i) even distribution of tasks among processors and ii) efficient number of processors among which to distribute tasks evenly. Further, they demonstrated the significance of the relation between the costs of computation and communication in achieving efficient performance. Huang [5] et al. proposed accurate models to depict execution times on specific multiprocessor systems. Their studies reveal that clustering and scheduling should also consider the underlying architecture for practical implementations. Our approach investigates clustering and scheduling of tasks in the light of the work done in [6] and [5]. However, we consider the ratio of costs of computation and communication of each *task* and then cluster these tasks so as to keep the overall ratio of cost of computation and communication of each *cluster* within predetermined limits. These limits are based on such factors as processor architecture, number of processors, and desired execution time. Our approach is novel because it is flexible and tunable to yield desirable results for all types of application problems and/or architectures. The proposed clustering and scheduling technique compares cluster granularity with processor characteristics. Clusters are formed by gathering individual nodes of a given problem graph into clusters based on costs of computation, communication, precedence relations

among tasks/clusters, and the capacity of the underlying architecture.

The proposed flexible clustering and scheduling scheme can be effectively employed to achieve one or more of the following requirements, 1) efficient clustering and scheduling on bounded number of processors 2) maximal speed up and efficiency 3) minimal execution time on a particular architecture (for different processor speedups and communication channel bandwidth). 4) efficient execution on heterogeneous systems : MIMD, SIMD, and heterogeneous network of workstation and 5) equitable distribution of load among the processors

In our experimental work, we have studied performance characteristics of problems such as Gaussian elimination(GE), LU decomposition, mean value theorem, Floyd-Warshall's shortest path algorithm. Simulation results indicate that mapping based on our technique can be tuned to yield higher efficiency when compared with all other methods in the literature for most application problems.

2 Preliminaries

A *task graph* can be defined as a tuple $G = (V, E)$ where $V = \{n_j, j = 1 : v\}$ is the set of nodes and $v = |V|$ is the number of nodes, E is the set of communication edges and $|E|$ is the number of edges. An edge between nodes u and v is represented by $e(u, v)$. A task receives all inputs prior to execution, completes its execution without interruption and sends the output to all successor tasks in parallel. A $critical\ path(CP)$ is defined as a set of nodes and edges, forming the longest path that includes a root node and a leaf node in that graph. The final schedule length is the length of the CP of the scheduled graph [7]. Gerasoulis and Yang [3] call this as the dominant sequence (DS) of the clustered DAG. The $as\text{-}soon\text{-}as\text{-}possible(ASAP)$ and the $as\text{-}late\text{-}as\text{-}possible(ALAP)$ bindings are introduced in mobility directed(MD) scheduling [8]. $ASAP$ binding is the earliest possible execution which is created by moving forward through the task graph. Nodes whose $ASAP$ and $ALAP$ are equal are defined as nodes in the $critical\ path(CP)$. Tao et al. [9] define $tlevel$ of a node n as the length of the longest path from an entry node to n excluding the weight of n, and $blevel$ of n as the length of the longest path from node n to an exit node.

Clustering is a mapping of the tasks of a DAG onto clusters [3]. The clustering schemes are presented in [4], [9], [6]. Scheduling is a task to processor assignment and a task to *starting time* mapping. A cluster is a set of tasks which will be executed on the same processor[4]. Tasks of small grain size are gathered into coarse tasks or clusters, in order to reduce the cost of communication during execution on a multiprocessor system. Clustering decreases effective parallelism because tasks in a cluster are executed serially when the cluster is assigned to a single processor.

A task is an indivisible unit of computation which may be an assignment statement, a subroutine or even an entire program [3]. In our model, once started, tasks run to completion without interruption. The size of a task depends on the number of statements or instructions making up the task. The computation load of a task is represented by r time units. A task begins execution upon arrival of all data from its predecessors in the DAG. The communication overhead associated with the arrival of input data is defined as the communication load of the task and is represented by c time units. The value of c depends on the indegree of the task and the weight of each incoming edge. The computation to communication ratio of the task is represented by $\alpha = r/c$. We represent the cost of computation of node x by r_x and the cost of communication associated with that node by c_x. A high value of α implies coarse granularity and low value of α implies fine granularity.

The computation and communication loads of a cluster are represented by R and C respectively. The computation to communication ratio of a cluster is represented by $\beta = R/C$. The values of R and C are dependent on several factors about the DAG representing the given application problem. If β is high then it implies that communication load of the cluster is very low compared to the computation load. We define a system parameter, τ given by ρ/γ, where ρ is a function of the processor speed and γ is a function of the channel bandwidth. In practice, the α values vary depending on the nature of the given problem. As a result the efficient value of β depends on the α s of tasks and the nature of the DAG. A high value of β implies coarse granularity and more number of tasks per cluster. In Figure 1 we summarize several instances for determining R and C of a given graph.

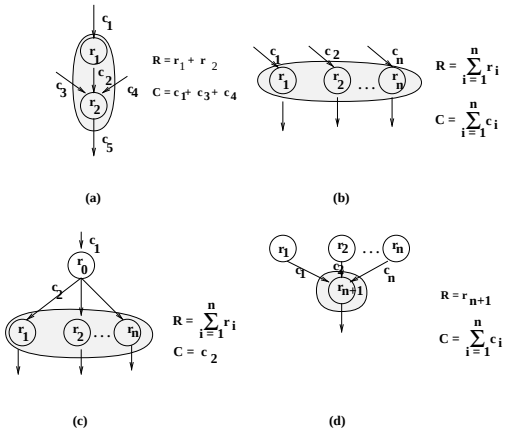


Figure 1. Computation of R and C for different clusters.

The execution times of clusters on parallel computers depend on such factors as processor speed, channel bandwidth and the inter-processor communication network. In this paper we consider the processor speed and channel bandwidth for clustering and scheduling. We define a system parameter, τ given by ρ/γ, where ρ is a function of the processor speed and γ is a function of the channel bandwidth.

3 Flexible Clustering and Scheduling

In this section we introduce a novel clustering and scheduling algorithm that takes into account the cost of computation and communication of the underlying tasks, gathers tasks into clusters such that the ratio of the cost of computation and communication is suited for execution on the underlying architecture. One of the strong points of this method is the use of optimal number of processors for execution of the given problem.

The proposed Flexible Clustering and Scheduling(FCS) algorithm enables to 1) find the best clustering and scheduling given the DAG, p, α, and τ. 2) find the best clustering and scheduling for execution in minimum time given the DAG, α, τ, and unbounded number of processors. 3) find a mapping on efficient number of processors for achieving highest efficiency given the DAG, α, τ, and bounded number of processors.

3.1 Clustering

The clustering and scheduling algorithm comprises two phases, i)*clustering* during which tasks of the graph are gathered into clusters, and ii)*scheduling* during which clusters are allocated to available processors. Clustering is based on the computation and communication costs of individual tasks and precedence relations. In the following section we consider sample precedence relations and illustrate computation of R and C for sample graph segments.

3.2 Computation of R and C

Nodes of a DAG can be gathered into clusters in many ways depending on computation and communication costs. The two nodes of Figure 1a , r_1 and r_2, are sequentially executed. Total cost of computation of this cluster is given by $R = r_1 + r_2$ and total communication cost C, is given by $(c1+c3+c4)$. Tasks r_1 and r_2 are executed in the same cluster which means that these tasks are allocated to the same PE. We assume that there is no communication cost between the tasks. Communication cost of c_2 is zero. We consider only input costs because when a cluster sends data to another cluster the communication cost is considered at the receiving end. The effective β value for a DAG of Figure 1a is $\frac{r_1+r_2}{c_1+c_3+c_4}$, when nodes in a cluster have precedent nodes as shown in Figure 1b , the order of execution is dependent on the arrival of inputs from precedent nodes. In other words, order of execution depends on input communication time, $c_1, c_2, ..., c_n$. The effective β is $\frac{\sum_{i=1}^{n} r_i}{\sum_{i=1}^{n} c_i}$. In the third case r_0 broadcasts data to nodes $n_1, n_2, ...n_n$ which are in the same cluster, as shown in Figure 1c , node n_0 sends data only once. The effective β of this structure is $\frac{r_1+r_2+...r_n}{c_2}$. In contrast, in the cluster of Figure 1d , a node in the cluster receives data from many sources, $n_1, n_2, ...n_n$, the effective β is $\frac{r_{n+1}}{\sum_{i=1}^{n} c_i}$. As can be seen, β depends on r and c, and the clustering mechanism.

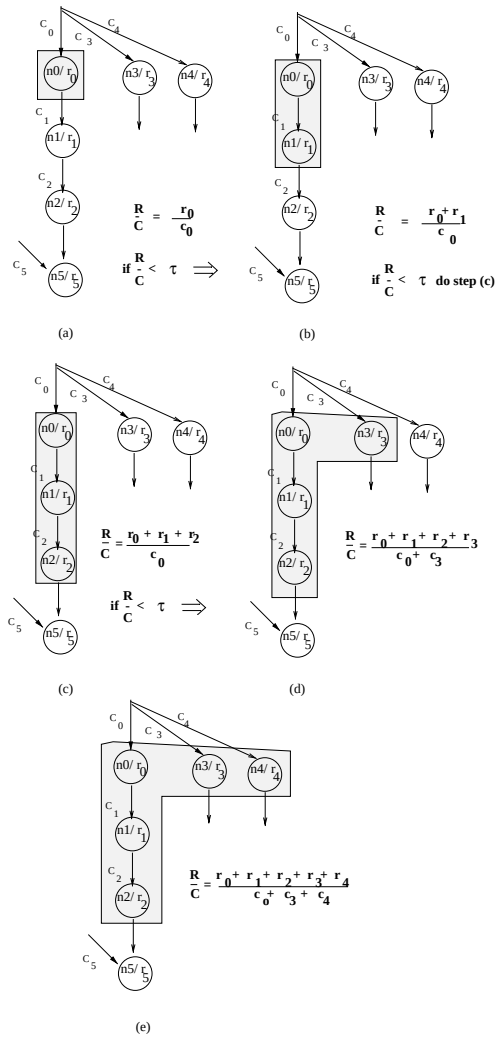


Figure 2. Clustering mechanism.

Figure 2 illustrates the process of cluster creation using the proposed algorithm. Nodes that execute in a serial fashion are collected first because these nodes should be in the same cluster. Nodes n1, n2 are collected into one cluster

with $\frac{R}{C} = \frac{r_0+r_1+r_2}{c_0}$. After collecting serial nodes, if the value of β is less than the value of τ, sibling nodes of n_0 are collected as shown in Figure 2d. The value of $\frac{R}{C}$ at this stage is $\frac{r_0+r_1+r_2+r_3}{c_0+c_3}$ and if $\frac{R}{C}$ or β is less than τ node n_4 will be collected as shown in Figure 2e. If the value of β of the cluster is greater than or equal to the value of τ, the cluster is assigned to an available PE, otherwise, parent nodes of those nodes in the cluster are collected.

3.3 The Algorithm

The flexible clustering and scheduling algorithm is shown in the Figure 3. The algorithm comprises three components. Firstly, we determine the critical path (CP) of the given task graph G(V,E) by computing the ASAP and ALAP times for all nodes in G. The nodes of the CP are grouped into one cluster and assigned to the first PE (PE0 in this case). The nodes of the graph are ordered lexicographically. The second part of the algorithm is the most important contribution of the proposed FCS scheme. During this part zero or more clusters are created out of the remaining nodes of G. Everytime a cluster is formed, the nodes of the cluster are removed from the lexicographic list, L. To form a new cluster, the first node x in L is picked. The β of x is given by $\frac{r_x}{c_x}$ and is compared with τ. If $\beta \geq \tau$ then the single node x forms a cluster. However, if β is less than τ then other nodes in L are clustered with x until $\beta \geq \tau$ or L = 0. The order in which nodes are picked to form clusters is as follows. After gathering each node in the cluster we recompute β and go to the next node if and only if β is less than τ. The objective of the scheme is to reduce the cost of communication among nodes. The nodes are gathered based on precedence relations.

Algorithm CLUSTERING() compute ASAP and ALAP for all nodes in $G(V, E)$

S = Set of nodes in CP
L = List of nodes not in $CP = (V - S)$ in increasing order lexicographically
map $S \rightarrow PE0$
While $L <> 0$
x = first unmarked node in L
  FORM_A_CLUSTER()
  if β of $CL < \tau$
    map $CL \rightarrow PE0$
  else map $CL \rightarrow SELECTED\_PE()$

FORM_A_CLUSTER(x)
  remove x from L;
  place x in set CL; CL is the set of nodes in the cluster
  Compute β of CL
  While $\beta < \tau$ and $L <> 0$
    if $h(x)$ is in L
      INCLUDE(h(x))
    While there exists a $h^k(x)$ in L and $d(h^k(x)) = 1$
      INCLUDE($h^k(x)$)
    if there exists an $s(x)$ in L
      INCLUDE(s(x))
    if there exists an $p(y)$ in L
      INCLUDE(p(y))
    if there exists an $n(x)$ in L
      INCLUDE(n(x))

number of PEs = P or ∞ if unlimited
i = 0
SELECTED_PE()
 $i = i + 1$
 if $i \leq P$
   SELECTED_PE = PE_i
   $\beta(PE_i) = \beta\ of\ CL$
 else
   SELECTED_PE = PE_j such that $\frac{R}{C}(PE_j) = Min\{\beta(PE_0), \beta(PE_1) \ldots \beta(PE_n)\}$

INCLUDE(y)
  move y from L to CL
  recompute $\beta\ of\ CL$

where
$h(x)$ = child of x
$h^k(x)$ = child kth of x where $k > 1$; for example $h^3(x) = h(h(h(x)))$
$p(y)$ = parent of y :y is any node in CL
$s(x)$ = sibling of x
$d(x)$ = in degree of x
$n(x)$ = next node, lexicographic lists

Figure 3. Clustering and Scheduling Algorithm

If there is a child of x, $h(x)$ then there exists an edge from x to $h(x)$. By clustering x and $h(x)$ together, we reduce the cost of $e(x, h(x))$ to zero. By doing so, β of the cluster changes to $\frac{r_x+r_{h(x)}}{c_x+c_{h(x)}-e(x,h(x))}$. If the cost of $e(x, h(x))$ is considerable then the effective cost of communication is reduced.

We represent a child of $h(x)$ by $h(h(x))$ or $h^2(x)$, and a child $h^2(x)$ by $h^3(x)$ and so on. If there is a $h^k(x)$ then there exists an edge from $h^{k-1}(x)$ to $h^k(x)$. By clustering $h(x)$ with $h^2(x)$ the cost of $e(h(x), h^2(x))$ is reduced to zero. In general, by clustering $h^{k-1}(x)$ and $h^k(x)$ together, the cost of $e(h^{k-1}(x), h^k(x))$ is reduced to zero. By doing so β changes to $\frac{r_x+r_{h(x)}+r_{h^2(x)}+\ldots r_{h^{k-1}(x)}}{c_x+c_{h(x)}-e(x,h(x))}$. Child nodes are clustered until a node with additional input edges is found. If we include a child node with additional input edges then the effective cost of communication increases in addition precedence constraints.

Sibling of x is denoted by $s(x)$ and their common parent node $p(x)$ has an edge to $s(x)$. By clustering $s(x)$ with x, we reduce the cost of the edge from $p(x)$ to $s(x)$ if $p(x)$ is broadcasting. Clustering of $s(x)$ results in additional reduction in communication costs associated with the clustering of $s(x)$ with x. The cost of new β is $\frac{r_x+r_{h(x)}+r_{h^2(x)}+\ldots r_{h^{k-1}(x)}+r_{(s(x))}}{c_x+c_{h(x)}-e(x,h(x))+e(p(x),s(x))}$.

There is an edge from parent node of y, $p(y)$ to y where node y is already in the cluster CL. The communication cost of $e(y, p(y))$, will be reduced to zero when they are in the same cluster. By doing this, the β will change to $\frac{r_x+r_{h(x)}+r_{h^2(x)}+\ldots r_{h^{k-1}(x)}+r_{(s(x))}+r_{p(y)}}{c_x+c_{h(x)}-e(x,h(x))+e(p(x)+c_{s(x))}}$.

In case β of the cluster is still less than τ, we consider the subsequent nodes in the list L.

Finally, the algorithm allocates processors to clusters in the third part. On completion of a cluster, if the cluster's β is $\geq \tau$, the cluster is allocated to the next available PE. However, if there is no PE available and/or if the β of the cluster is $< \tau$ then we merge this cluster with one of the

already allocated clusters such that the partner cluster has the least β.

3.3.1 Complexity of the algorithm

The time complexity for computation of ASAP and ALAP is given by $O(v^2)$ [8], where v is the number of nodes in the task graph. The procedure FORM_A_CLUSTER takes $O(v - S)$ time. Therefore, the complexity of the clustering algorithm is $O(v^2)$. The complexity of the proposed algorithm is lower than those of other scheduling algorithm.

4 Experimental Results

The main objective in all cases is to determine the efficient number of processors to be employed for a given problem size and value of τ, in order to minimize the normalized schedule length. We implemented the GE problem for sizes 16, 32, and 64. All problem instances used 1 to 10 processors based on the value of τ as indicated in Table 4. For example when $\tau = 0.50$, the system uses 10 processors for all the three problem sizes and yields maximum speedup and minimum normalized schedule length for the problem size of 8. The efficiency increases with problem size when τ is between 0.25 and 2.0. On the other hand when $\tau = 11.00$, the system uses two processors for n = 16, six processors for n = 32, and ten processors for n = 64. The efficiency decreases with problem size. It is clear from the Figure, that for a given problem size, schedule length and maximal speedup can be achieved at one value of τ; in this case it is τ = 1.75 for n = 32, and τ = 3.0 for n = 64. After this point, if τ increases, the speedup reduces, but efficiency increases until $\tau = 12.0$, and then the system uses only one processor. This shows that when $\tau = 12.0$, (for n = 16) it is better to execute the whole problem on one processor rather than multiple processors. The results of Laplace equation implementation for problem sizes 8, 16, and 32 are shown in Figure 4.The efficiency is greater at large problem sizes. In contrast, when $\tau = 2.50$, the system uses five processors for n = 8, fourteen processors for n = 16, and thirty processors for n = 32. The efficiency reduces with problem size . Mean-value analysis problem for problem sizes 16, 32, and 64 was implemented and the results are shown in Figure 4.The efficiency increases with problem size and the speedup is maximum when τ is low. We implemented the Floyd-Warshall's algorithm for sizes 8, 16, and 32. The maximal speedup and efficient schedule length are achieved at τ = 2.0 for n = 8, τ = 5 for n = 16, and τ = 11 for n = 32.

5 Conclusion

In this paper, we have proposed a flexible clustering and scheduling algorithm to determine efficient cluster size and schedule tasks onto bounded or unbounded number of processors in order to achieve maximal efficiency or minimal execution time. The proposed technique can be used to tune cluster size based on such system characteristics as speed of processor and communication bandwidth. We present simulation results to show the versatility of the clustering and scheduling technique under various conditions and applications. In each case, the implementation can be tuned to achieve high efficiency or minimal execution time.

The FCS algorithm can be effectively used for implementing application problems on a variety of parallel computers and/or heterogeneous network of workstations. The FCS scheme provides a mapping that is most suitable for a given system. We are in the process of evaluating the FCS scheme in implementing irregular problem. A realtime version of the FCS scheme is under development to enhance the application to realtime problem. We are also carrying and experiment studies on various parallel computing platforms in order to validate simulation results.

References

[1] M. Al-Mouhamed. Lower bound on the number of processors and time for scheduling precedence graphs with communication costs. *IEEE Transactions on Software Engineering*, 16:1390–1401, December 1990.

[2] J. H. et al. Scheduling precedence graphs in systems with interprocessor communication times. *Siam Journal of Computer*, 18:244–257, April 1989.

[3] A. Gerasoulis and T. Yang. A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors. *Journal of Parallel and Distributed Computing*, 16:276–291, 1992.

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

[5] W. Huang, X. Chen, L. Bhuyan, and F. Lombardi. Accurate communication models for task scheduling in multicomputers. 1995.

[6] B. Indurkhya, H. Stone, and L. Cheng. Optimal partitioning of randomly generated distributed programs. *IEEE Transactions on Parallel and Distributed Systems*, SE-12:483–495, March 1986.

[7] Y. Kwok and I. Ahmad. Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 7:506–521, May 1996.

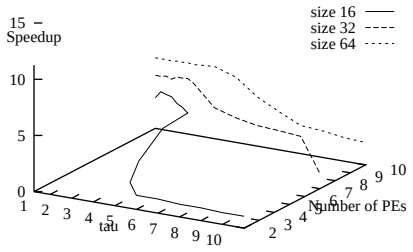
[8] M. Wu and D. Gajski. Hypertool: A programming aid for message-passing systems. *IEEE Transactions on Parallel and Distributed Systems*, 1:330–343, July 1990.

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

Gaussian elimination: n = 16, 32, and 64

τ	Norm.Sched.Length			Efficiency			No. of PEs		
	16	32	64	16	32	64	16	32	64
0.25	1.64	1.69	2.41	0.27	0.47	0.63	10	10	10
0.50	1.33	1.70	2.41	0.34	0.47	0.63	10	10	10
0.75	1.39	1.66	2.40	0.32	0.48	0.63	10	10	10
1.00	1.45	1.73	2.42	0.31	0.46	0.63	10	10	10
1.25	1.74	1.63	2.39	0.26	0.49	0.63	10	10	10
1.50	1.93	1.63	2.39	0.23	0.49	0.63	10	10	10
1.75	2.38	1.62	2.38	0.19	0.49	0.63	10	10	10
2.00	2.24	1.72	2.41	0.25	0.46	0.63	8	10	10
3.00	2.92	2.91	2.35	0.30	0.28	0.64	5	10	10
4.00	3.25	3.68	2.60	0.46	0.22	0.58	3	10	10
5.00	3.53	4.40	3.47	0.63	0.18	0.44	2	10	10
6.00	3.53	4.71	4.46	0.63	0.17	0.34	2	10	10
7.00	3.95	5.18	6.04	0.56	0.15	0.25	2	10	10
8.00	3.91	5.53	6.39	0.57	0.16	0.24	2	9	10
9.00	4.25	5.60	7.19	0.52	0.18	0.21	2	8	10
10.00	4.21	5.92	7.59	0.53	0.19	0.20	2	7	10
11.00	3.79	6.02	8.11	0.59	0.22	0.19	2	6	10
12.00	4.44	6.42	8.57	1.00	0.25	0.18	1	5	10

Gaussian elimination

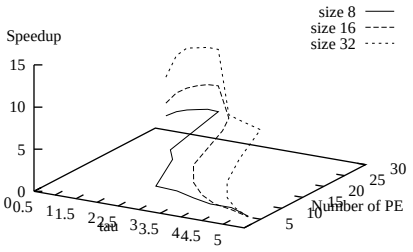


Algorithms	Norm.Sched.Length			Efficiency			No. of PEs		
	16	32	64	16	32	64	16	32	64
MCP	1.03	1.41	1.43	0.25	0.17	0.16	16	32	64
MD	3.80	4.87	11.03	0.29	0.41	0.14	3	3	9

Laplace equation: n = 8, 16, and 32

τ	Norm.Sched.Length			Efficiency			No. of PEs		
	8	16	32	8	16	32	8	16	32
0.25	2.54	2.57	2.59	0.06	0.11	0.21	30	30	30
0.50	1.76	1.78	1.79	0.08	0.15	0.30	30	30	30
0.75	1.50	1.52	1.60	0.09	0.18	0.34	30	30	30
1.00	1.38	1.39	1.56	0.10	0.20	0.35	30	30	30
1.25	1.27	1.31	1.52	0.11	0.21	0.36	30	30	30
1.50	1.30	1.30	1.52	0.11	0.21	0.35	30	30	30
1.75	1.65	3.04	5.22	0.16	0.09	0.10	16	30	30
2.00	1.97	3.37	6.04	0.15	0.09	0.09	14	26	30
2.50	2.41	4.24	8.21	0.35	0.14	0.07	5	14	30
3.00	2.61	5.22	10.49	0.32	0.17	0.08	5	9	20
3.50	2.86	6.26	12.30	0.37	0.22	0.11	4	6	12
4.00	2.92	6.80	14.00	0.48	0.30	0.17	3	4	7
4.50	2.91	6.83	14.44	0.48	0.40	0.28	3	3	4
5.00	3.74	7.75	15.75	0.56	0.53	0.51	2	2	2
5.50	3.74	7.74	15.75	0.56	0.53	0.51	2	2	2
6.00	3.74	7.74	15.75	0.56	0.53	0.51	2	2	2
6.50	3.74	7.74	15.75	0.56	0.53	0.51	2	2	2

Laplace Equation

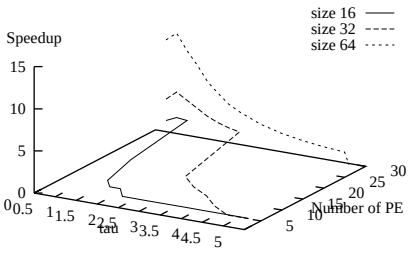


Algorithms	Norm.Sched.Length			Efficiency			No. of PEs		
	8	16	32	8	16	32	8	16	32
MCP	1.30	1.50	-	0.41	0.34	-	7	15	-
MD	2.31	4.54	-	0.31	0.61	-	5	2	-

Mean-value: n = 16, 32, and 64

τ	Norm.Sched.Length			Efficiency			No. of PEs		
	16	32	64	16	32	64	16	32	64
0.25	4.52	2.92	2.02	0.04	0.13	0.36	30	30	30
0.50	3.16	2.31	1.85	0.06	0.16	0.40	30	30	30
0.75	3.40	2.73	2.19	0.06	0.14	0.33	30	30	30
1.00	4.13	3.33	2.59	0.10	0.11	0.28	14	30	30
1.25	4.61	4.27	3.27	0.22	0.09	0.22	6	30	30
1.50	4.70	5.33	3.93	0.32	0.07	0.19	4	30	30
1.75	4.69	6.49	4.83	0.32	0.06	0.15	4	30	30
2.00	5.51	7.64	5.67	0.54	0.05	0.13	2	30	30
2.50	5.51	8.82	7.81	0.54	0.11	0.09	2	12	30
3.00	5.51	9.13	10.07	0.54	0.14	0.07	2	9	30
3.50	5.51	9.49	12.58	0.54	0.17	0.06	2	7	30
4.00	5.51	9.84	15.19	0.54	0.29	0.05	2	4	30
4.50	5.51	10.84	17.44	0.54	0.52	0.04	2	2	30
5.00	5.51	10.84	17.83	0.54	0.52	0.05	2	2	26

Mean-value equation

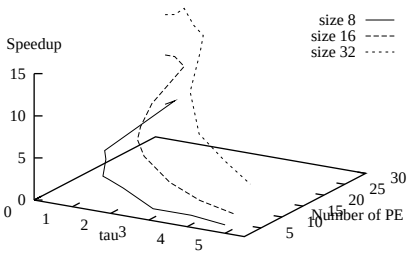


Algorithms	Norm.Sched.Length			Efficiency			No. of PEs		
	16	32	64	16	32	64	16	32	64
MCP	1.80	1.46	-	0.41	0.37	-	7	20	-
MD	2.51	-	-	0.20	-	-	11	-	-

Floyd-Warshall's Algorithm: n = 8, 16, and 32

τ	Norm.Sched.Length			Efficiency			No. of PEs		
	8	16	32	8	16	32	8	16	32
0.25	1.60	1.37	1.89	0.14	0.33	0.49	30	30	30
0.50	1.39	1.37	1.86	0.16	0.33	0.50	30	30	30
0.75	1.67	1.52	1.76	0.35	0.30	0.53	11	30	30
1.00	1.83	1.86	1.98	0.40	0.37	0.47	9	20	30
1.25	2.57	2.24	2.14	0.42	0.40	0.43	6	15	30
1.50	2.46	2.47	2.31	0.53	0.46	0.44	5	12	27
2.00	2.92	2.89	2.92	0.56	0.52	0.48	4	9	20
3.00	6.00	4.28	3.91	0.54	0.53	0.55	2	6	13
4.00	5.90	5.79	5.06	0.55	0.59	0.55	2	4	10
5.00	6.50	7.47	6.51	1.00	0.61	0.61	1	3	7
6.00	6.50	9.07	8.09	1.00	0.50	0.57	1	3	6
7.00	6.50	13.00	9.90	1.00	0.52	0.56	1	2	5
8.00	6.50	12.95	11.89	1.00	0.52	0.58	1	2	4
9.00	6.50	12.84	14.52	1.00	0.53	0.48	1	2	4
10.00	6.50	11.65	16.59	1.00	0.58	0.42	1	2	4
11.00	6.50	13.58	14.68	1.00	1.00	0.63	1	1	3
12.00	6.50	13.58	16.09	1.00	1.00	0.58	1	1	3

Floyd's algorithm



Algorithms	Norm.Sched.Length			Efficiency			No. of PEs		
	8	16	32	8	16	32	8	16	32
MCP	1.96	2.01	-	0.47	0.45	-	6	14	-
MD	3.71	6.32	-	0.35	0.13	-	4	16	-

Figure 4. Experimental results.