Comparing the Optimal Performance of Different MIMD Multiprocessor Architectures

Lars Lundberg
Department of Computer Science
University of Karlskrona/Ronneby
S-372 25 Ronneby, Sweden
email: Lars.Lundberg@ide.hk-r.se

Håkan Lennerstad
Department of Mathematics
University of Karlskrona/Ronneby
S-371 79 Karlskrona, Sweden

Abstract

We compare the performance of systems consisting of one large cluster containing q processors with systems where processors are grouped into k clusters containing u processors each. A parallel program, consisting of n processes, is executed on this system. Processes may be relocated between the processors in a cluster. They may, however, not be relocated from one cluster to another. The performance criterion is the completion time of the parallel program.

We present two functions: g(n,k,u,q) and G(k,u,q). Provided that we can find optimal or near optimal schedules, these functions put optimal upper bounds on the gain of using one cluster containing q processors compared to using k clusters containing u processors each. The function g(n,k,u,q) is valid for programs with n processes, whereas G(k,u,q) only depends on the two multiprocessor architectures.

By evaluating g(n,k,u,q) and G(k,u,q) we show that the gain of increasing the cluster size from 1 to 2 and from 2 to 4 is relatively large. However, the gain of using clusters larger than 4 is very limited.

1. Introduction

It is relatively easy and cheap to build large MIMD systems, i.e. systems with a large number of processors, by connecting a number of small clusters with a communication network (see figure 1). The MPI (Message Passing Interface) [8] and PVM (Parallel Virtual Machine) [1] environments make it possible to write one distributed program (or distributed application) which runs on a number of clusters. Systems which consist of a number of clusters can easily scale up, i.e. one can simply connect more clusters to the communication network. Another advantage of using multiple clusters is that communication within one cluster does not interfere with communication within other clusters. In bus-based one-cluster systems the shared bus becomes a bottleneck when the number of processors increases. Consequently, systems with multiple small clusters have a number of advantages compared to one-cluster systems. A major disadvantage with multiple clusters is that, since processes may not be relocated between clusters, some clusters may be idle while others are very busy. This problem can be reduced, although not completely eliminated, be careful scheduling of processes to clusters.

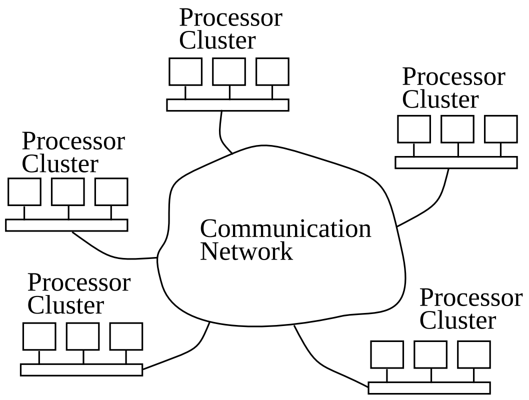


Figure 1: A multiprocessor with clusters.

The problem of finding schedules which result in minimum completion time is NP-hard [3]. However, there are heuristic scheduling techniques which result in near optimal schedules [4][9]. There are two previous results [5][6] which make it possible to put tight bounds on the maximum performance gain of using one large cluster compared to multiple small clusters, provided that we are able to find optimal or near optimal schedules.

In this paper we use these results in order to quantitatively compare the maximum performance gain of using a large one-cluster system with the advantages (e.g. low cost per processor and good scalability) of using multiple small clusters.

2. Previous results

Consider a *parallel program* consists of a set of sequential processes. A process can be either Blocked, Ready or Running. Processes in the Ready and Running states are referred to as *active* processes. The execution of a process is controlled by two synchronization primitives: *Wait(Event)* and *Activate(Event),* where *Event* couples a certain Activate to a certain Wait. When a process executes an Activate on an event, we say that the event has occurred. If a process executes a Wait on an event which has not yet occurred, that process becomes blocked until another process executes an Activate on the same event. However, a process executing a Wait on an event which already has occurred does not become blocked. Each process can be represented as a list of sequential segments, which are separated by a Wait or an Activate (see Figure 2). We assume that, for each process, the length and order of the sequential segments are independent of the way processes are scheduled. All processes are created at the start of the execution. Some processes may, however, be initially blocked by a Wait.

In *multiprocessors with clusters*, processors of equal speed are connected in a two-level hierarchy. At the lowest level processors are connected in equally sized clusters. These clusters are connected via a communication network. Processes may not be relocated from one cluster to another. They may, however, be relocated between the processors in a cluster. We assume that only one program may execute on the system at the same time. Moreover, we disregard overhead for process synchronization, context switching and relocation (these assumptions are discussed in section 4). Under these conditions, the minimal completion time for a program P, executed on a system with k clusters, each containing u processors, is denoted $T(P,k,u)$. The performance criterion used in this paper is the completion time of the parallel program.

The left part of Figure 3 shows a parallel program with three processes (p_1, p_2 and p_3). *Work(t)* denotes processing for t time units. Process p_3 cannot start its execution before p_2 has executed one time unit. This dependency is represented with a Wait on Event_1 in p_3 and an Activate on the same event in p_2.

The right part of the figure shows a graphical representation of P and two schedules resulting in minimum completion time for a system with one cluster containing two processors and for a system with two clusters containing one processor each. There are two processors in both systems but in the rightmost case processes may not be relocated from one processor to another. As indicated in Figure 3, the minimum completion time never decreases when process relocation is prohibited. The completion time of a program P using schedule A and a multiprocessor with k clusters containing u processors each is denoted $T(P,k,u,A)$. Consequently, $T(P,k,u) = min_A\ T(P,k,u,A)$.

When there is only one cluster it is relatively easy to find a schedule for which the completion time is close to the optimal case. In fact, the completion time using a schedule in which no processor is idle when there are processes in the Ready state, e.g. self-scheduling using one common ready queue for all processors, is always within a factor of 2 from the optimal case [2]. In most cases the difference between the completion time using self-scheduling and the minimum completion time is marginal. When there is more than one cluster, the problem of finding a schedule with near minimum completion time becomes more difficult. However, there are heuristic techniques which result in near optimal schedules for many configurations [4][9].

In order to compare the performance of two different architectures, we define the function $g(n,k,u,q) = max_{\text{all programs } P \text{ with } n \text{ processes}}\ T(P,k,u)/T(P,1,q)$, i.e. for programs with n processes $g(n,k,u,q)$ is an optimal upper bound on the maximum gain of using a multiprocessor with one cluster containing q processors instead of a multiprocessor with k clusters containing u processors each, provided that we can find optimal or near optimal schedules. The function is defined for all combinations of positive integer values on n, k, u and q.

We also define a function $G(k,u,q) = sup_n\ g(n,k,u,q) = sup_{\text{all programs } P}\ T(P,k,u)/T(P,1,q)$, i.e. $G(k,u,q)$ is an optimal upper bound on the maximum gain of using a multiprocessor with one cluster containing q processors instead of a multiprocessor with k clusters containing u processors each, provided that we can find optimal or near optimal schedules. The function is defined for all combinations of positive integer values on k, u and q. The proofs and the formulas for $g(n,k,u,q)$ and $G(k,u,q)$ have been presented previously [5][6]. However, the formula for $G(k,u,q)$ is shown in Equation 1.

```
                      program P

process p1   process p2             process p3
begin        begin                  begin
  Work(2);     Work(1);               Wait(Event_1);
end p1;        Activate(Event_1);     Work(2);
               Work(1);             end p3;
             end p2;
```

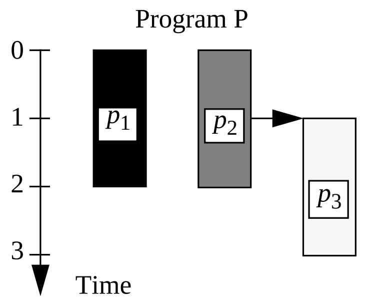


Figure 2: Textual and graphical representation of a parallel program *P*.

$$G(k,u,q) = \frac{k!q!}{k^q u}\sum_{l=1}^{q} max(l,u)\sum_{I}\frac{1}{\prod_{j_1=1}^{k} i_{j_1}!\prod_{j_2=1}^{b(I)} a(I,j_2)!}$$

,where the second sum is taken over all sequences of nonnegative integers $I = \{i_1,...,i_k\}$ which are decreasing, i.e. $i_j \geq i_{j+1}$ for all $j = 1,...,k\text{-}1$, and for which $i_1 = l$ and $\sum_{j=1}^{k} i_j = q$. The functions *a(I,j)* and *b(I)* are defined in the following way:

a(I,j) = the number of occurrences of the *j:th* distinct integer in *I*. *b(I)* = the number of distinct integers in *I.*

Equation 1: The formula for *G(k,u,q)*.

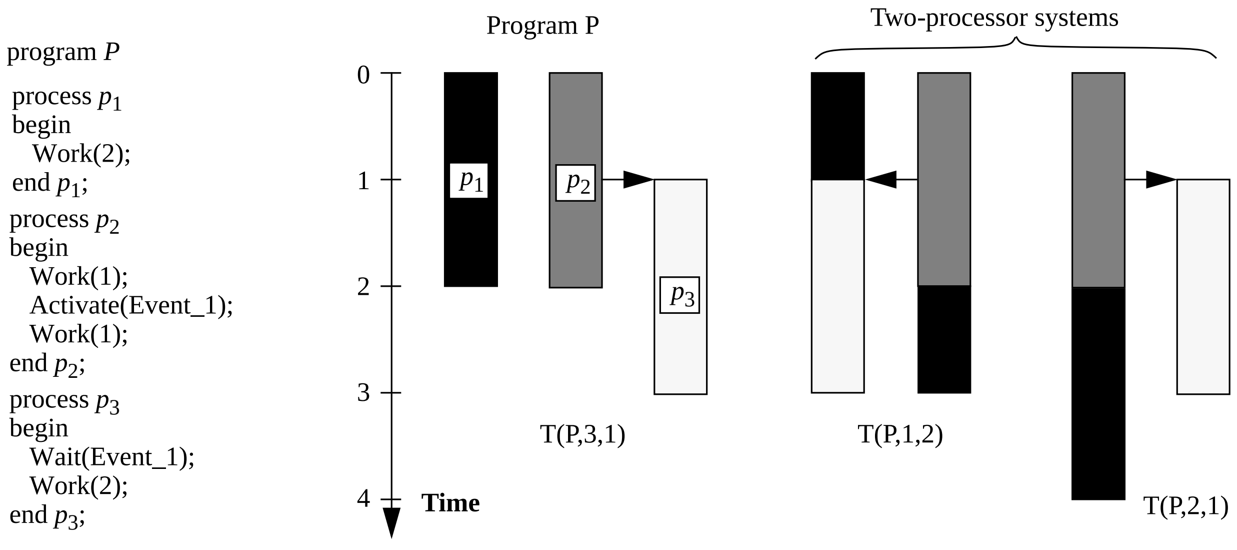
3. Comparing different multiprocessor architectures

In this section we demonstrate the applicability of *g(n,k,u,q)* and *G(k,u,q)* by comparing different multiprocessor architectures. Figure 4 shows the function *G(k,u,ku)* in the interval $k, u \leq 8$, i.e. in this case $q = ku$. *G(k,u,ku)* is the maximum performance loss of using a system with *k* clusters containing *u* processors each compared to using one cluster with *ku* processors provided that we can find optimal or near optimal schedules for both architectures, i.e. in this case we compare systems with equal numbers of processors but with different architectures.

By looking at the point $k = 8$ and $u = 1$ in Figure 4, we see that the performance loss of using a system with 8 clusters each containing one processor each compared to one cluster containing 8 processors cannot be more than a factor of 2.5. If we look at the point $k = 8$ and $u = 8$ we see that the performance loss of using a system with 8 clusters each containing 8 processors compared to one cluster containing 64 processors cannot be more than 1.5.

Figure 5 shows three plots corresponding to the functions *g(n,k,*1*,k)*, *g(n,k,*2*,*2*k)* and *g(n,k,*4*,*4*k)*. Consequently, Figure 5 compares systems with equal amounts of processors but with different architectures, i.e. systems with cluster sizes 1, 2 and 4 respectively. One important difference compared to Figure 4 is that in Figure 5 the number of processes in the parallel program (*n)* is considered. If $n \leq ku$, there are at least as many processors as processes in the system consisting of *k* clusters containing *u* processors each. In that case, each process can be scheduled to its own processor, i.e. it is not possible to obtain a completion time shorter than *T(P,k,u)*. Consequently, *T(P,k,u)/T(P,*1*,q)* ≤ 1, when $n \leq ku$. This is the reason why $g(n,k,1,k) = 1$ when $n \leq k$, $g(n,k,2,2k) = 1$ when $n \leq 2k$ and $g(n,k,4,4k) = 1$ when $n \leq 4k$.

Figures 4 and 5 both show that the gain of increasing the cluster size from 1 to 2 and from 2 to 4 is relatively


```
program P

process p1
begin
  Work(2);
end p1;
process p2
begin
  Work(1);
  Activate(Event_1);
  Work(1);
end p2;
process p3
begin
  Wait(Event_1);
  Work(2);
end p3;
```

Figure 3: The minimal completion times for two different systems for a parallel program *P*.

large. However, the gain of using clusters larger than 4 is very limited. This is a strong argument for building large multiprocessors by connecting a number of small clusters, containing 2 or 4 processors each.

4. Discussion

The results are based on the assumption of zero synchronization overhead, i.e. Waits and Activates take zero time to execute. The bounds are, however, also valid if there is a fixed overhead for process synchronizations, i.e. if the execution of a Wait or an Activate takes a certain amount of time. In that case there are programs executing on systems with zero synchronization overhead which are equivalent to the programs executing on the system with fixed overhead (see figure 6). In such programs the synchronization events are separated by some minimum amount of processing. All programs executing on systems with fixed synchronization overhead can be mapped on a subset of all programs executing on systems with zero synchronization overhead, viz. the subset of programs where all synchronization events are separated by a some minimum amount of processing. We know that *g(n,k,u,q)* and *G(k,u,q)* are valid for all programs executing on systems with zero synchronization overhead. Consequently, they are valid (however, not necessarily optimal) for any subset of these programs.

Validations using multithreaded Solaris programs and a multiprocessor with eight processors indicate that the bounds are valid for coarse grained as well as fine grained programs [7]. The measurements show that the results are optimal for coarse grained programs, but not for fine grained programs [7]. The reason for this is that in coarse grained programs the overhead for process synchronization can be neglected, thus making our results optimal.

References

[1] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM: Parallel Virtual Machine*, MIT press, 1994.

[2] R. L. Graham, *Bounds on Multiprocessor Timing Anomalies*, SIAM Journal of Applied Mathematics, 17, 2(1969), pp. 416-429.

[3] M. Garey and D. Johnson, *Computers and Intractability*, W.H. Freeman and Company, 1979.

[4] E.S.H. Hou, R. Hong and N. Ansari, *Efficient multiprocessor scheduling based on genetic algorithms*, 16th Annual Conference of the IEEE Industrial Electronics Society, Vol II, 1990, pp. 1239-1243.

[5] H. Lennerstad and L. Lundberg, *An Optimal Execution Time Estimate of Static versus Dynamic Allocation in Multiprocessor Systems*, SIAM Journal of Computing, Vol. 24, No. 4, pp. 751-764, Aug. -95.

[6] H. Lennerstad and L. Lundberg, *Optimal Combinatorial Functions Comparing Process Allocation Strategies in Multiprocessor Systems*, Technical report 1996, Department of Computer Science, University of Karlskrona/Ronneby.

[7] L. Lundberg, *Evaluating the Performance Implications of Binding Threads to Processors*, in Proceedings of the IEEE Conference on High Performance Computing, Bangalore, India, December, 1997, pp. 393-400.

[8] MPI Forum, *MPI: A message-passing interface standard*, International Journal of Supercomputer Application, 8 (3/4), pp. 165-416, 1994.

[9] A.K. Nanda, D. DeGroot, D.L. Stenger, *Scheduling Directed Task Graphs on Multiprocessors using Simulated Annealing*, in Proceedings of the IEEE 12th International Conference on Distributed Systems, 1992, pp. 20-27.

.

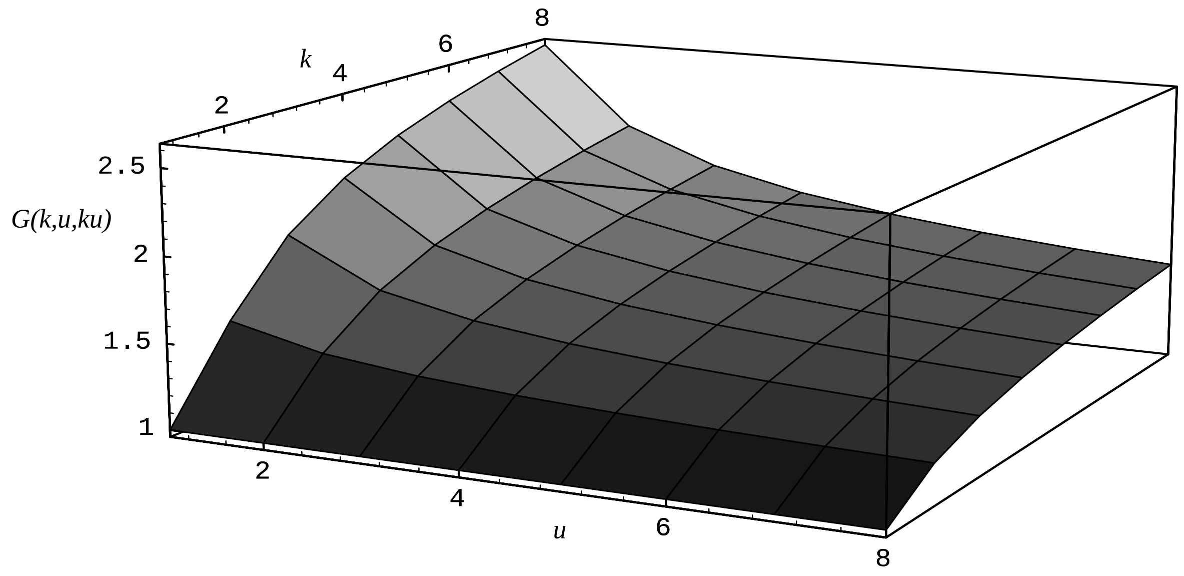


Figure 4: The function *G(k,u,ku)* in the interval *k, u* ≤ 8.

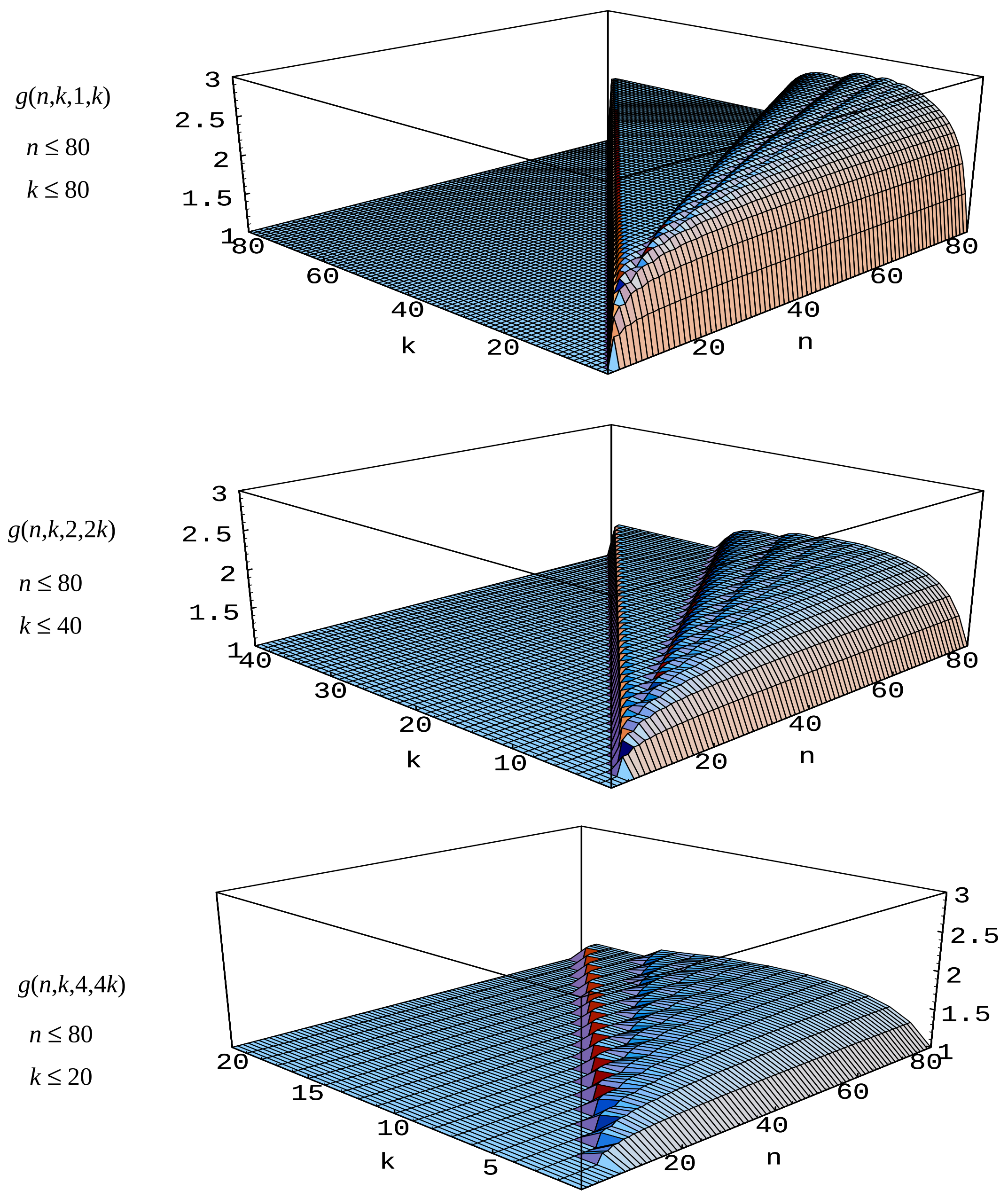


Figure 5: Three plots corresponding to *g(n,k,*1*,k)*, *g(n,k,*2*,*2*k)* and *g(n,k,*4*,*4*k)*.

System with fixed overhead for process synchronization

```
process $p_x$
  Work($x$)
  Wait(Ev_1)
  Activate(Ev_2)
  Work($y$)
end $p_x$
```

In this systems the execution of a Wait takes *ow* time units and the execution of an Activate takes *oa* time units

$\longleftarrow$ equivalent processes $\longrightarrow$

System with zero overhead for process synchronization

```
process $p_x$
  Work($x+ow$)
  Wait(Ev_1)
  Work($oa$)
  Activate(Ev_2)
  Work($y$)
process $p_x$
```

Figure 6: Two equivalent processes, which have the same execution time (*x+y+ow+oa*).