CHAPTER 8: FOUR PROCESSES

8-1. Introduction

When a distributed program is composed of four processes, there are many more arrangements and configurations possible than with two or three processes, making it more challenging to select the best configuration for a program to run in. A program with four processes consumes more resources than one with two or three processes (assuming similar processes), so selecting a good configuration is more critical with four processes than with smaller numbers of processes.

In this chapter, we first describe, in Sections 8-2 and 8-3, the arrangements and configurations possible with four processes and the four-process experiment sets we ran. In Sections 8-4 through 8-6 we discuss analyses of the 4x4, 4x3, and 4x2 situations. In Section 8-7 we describe and analyze the CR-even placement technique. In Section 8-8 we examine the marginal performance improvements due to adding one processor, and in Section 8-9 we examine the relationship of response time to the collocation number, the maximum number of processes collocated on a processing node. In Section 8-10 we discuss the zoomed-in grid and the transition zone with four processes. Finally, in Section 8-11 we summarize and present some conclusions concerning the four-process situation.

8-2. Arrangements and Configurations

A distributed program composed of four processes can be run on up to four processing nodes. When there are four processes to be placed and one, two, three, or four nodes available to run processes, these situations are called *4x1, 4x2, 4x3,* and *4x4* (read ‘‘4-by-1,’’ ‘‘4-by-2,’’ ‘‘4-by-3,’’ and ‘‘4-by-4’’), respectively.

There is only one possible arrangement of four processes on four processors (*4@4*, read ‘‘4-on-4’’): 1:1:1:1, or one process on each processor. There is only one possible configuration in this arrangement: 0/1/2/3 (process 0 on one processor, process 1 on a second processor, process 2 on a third, and process 3 on a fourth processor; with identical processors it does not matter which processor runs which process). This is called the four-process APART configuration.

There is also only one possible 4@3 arrangement: 2:1:1:0, or two processes on one of the processors, one process on a second, and one on a third. There are six configurations in this arrangement; they are listed in Figure 8-1. We call these configurations *triples,* because they use three processors. They are labelled according to which two processes are collocated, for example, configuration 01/2/3 is called *TRIPLE01*.

In the 4@2 situation there are two different arrangements, something which has not happened in the two-process or three-process situations. The 3:1:0:0 arrangement has four configurations, listed in Figure 8-1. These are called *loner* configurations because each places one process alone on a processor, and they are labelled according to which process is the loner (similar to the 3@2 2:1:0 loner configurations). The other 4@2 arrangement is 2:2:0:0, or *pairwise,* with two processes on each of two processors. This arrangement has three configurations, called *pair-type* configurations, labelled as shown in Figure 8-1.

In the 4x1 case, of course, there is only one arrangement, 4:0:0:0, and one configuration, 0123/-/-/-. There are no choices in a 4x1 situation, all processes must be run on the one available processor in the

four-process TOGETHER configuration. Since there are no placement decisions to be made in such a situation no further discussion of the 4x1 situation is necessary.

Situations				Usage	Arrangement	Configuration	Label
4x4	—	—	—	4@4	1:1:1:1	0 / 1 / 2 / 3	APART
"	4x3	—	—	4@3	2:1:1:0	01 / 2 / 3 / -	TRIPLE01
"	"	—	—	"	"	02 / 1 / 3 / -	TRIPLE02
"	"	—	—	"	"	03 / 1 / 2 / -	TRIPLE03
"	"	—	—	"	"	12 / 0 / 3 / -	TRIPLE12
"	"	—	—	"	"	13 / 0 / 2 / -	TRIPLE13
"	"	—	—	"	"	23 / 0 / 1 / -	TRIPLE23
"	"	4x2	—	4@2	3:1:0:0	0 / 123 / - / -	LONER0
"	"	"	—	"	"	1 / 023 / - / -	LONER1
"	"	"	—	"	"	2 / 013 / - / -	LONER2
"	"	"	—	"	"	3 / 012 / - / -	LONER3
"	"	"	—	"	2:2:0:0	01 / 23 / - / -	PAIR01
"	"	"	—	"	"	02 / 13 / - / -	PAIR02
"	"	"	—	"	"	03 / 12 / - / -	PAIR03
"	"	"	4x1	4@1	4:0:0:0	0123 / - / - / -	TOGETHER

Figure 8-1:
Arrangements and Configurations of 4 Processes

Figure 8-2 shows the number of arrangements and configurations with each number of processors.

	M Processors Available (4xM Situation)		M Processors Used (4@M Usage)	
M	Number of Arrangements	Number of Configurations	Number of Arrangements	Number of Configurations
4	5	15	1	1
3	4	14	1	6
2	3	8	2	7
1	1	1	1	1

Figure 8-2:
Numbers of Arrangements and Configurations Possible
with Four Processes in 4xM Situations and 4@M Usages

The numbers of arrangements and configurations given here, which were found by enumerating all possibilities, match the numbers of arrangements and configurations computed according to the formulas given in Chapter 3.

8-3. Experiment Sets

8-3.1. Introduction

As in the two-process and three-process work, the experiment sets used for four-process experiments are based on the full grid and the zoomed-in grid. Experiment set A is based on the full grid, experiment set B is based on the zoomed-in grid, and both have four processes per program trace. Communication is uniform among processes; that is, each sending process is equally likely to choose as destination any of the other processes; there are no communication clusters.

8-3.2. Experiment Set A

Experiment set A is based on the full grid. In set A, the 26 cells of the full grid are labelled **aa**, **ab**, and so on, through **az**; see Figure 8-7.

In set A, to save time and disk storage space, we simulated every second case, in a checkerboard pattern. In this pattern, at least two cases in each row and two in each column were simulated; overall, half of the possible cases were simulated. Actually, we simulated 14 of the 26 cases; case **az** did not fit into the checkerboard pattern but we simulated it because it appears to be in an important position.

We had previously tried simulating a subset of the cases in set U (three processes, full grid). In that instance we simulated 10 of 26 cases, completely skipping every other row and every other column. Specifically, we simulated cases **ua**, **uc**, **ue**, **uk**, **um**, **uo**, **uu**, **uw**, **uy**, and **uz** (see Figure 6-1; set U uses the same full grid as set N). This gave us results so odd that we had to simulate the remaining cases, completing the grid. However, had we simulated in set U the equivalent of the 14 cases simulated here, we would have obtained very nearly the same results as were obtained from the entire set U grid. So we are confident that the 14 case subset is a good one.

With 14 cases each repeated twice, there are 28 runs in set A. Each trace is simulated in each of the 15 configurations, for a total of 420 simulations.

8-3.3. Experiment Set B

Experiment set B is based on the zoomed-in grid, again with four processes per trace. The zoomed-in grid has 36 cells; in set B they are labelled **baa**, **bab**, and so on, through **bbj** (see Figure 8-39, or see Figure 6-4 since set B is based on the same zoomed-in grid as set R). There are two repetitions per cell, and each trace is simulated once in each of the 15 configurations.

Most of the material in this chapter is based on analyses of set A. Only Section 8-10 is based on experiment set B. We defer further discussion of set B to that section.

8-4. Four Processes with Four Processors (4x4)

8-4.1. Introduction to the 4x4 Situation

Our objectives are, first, to find a placement procedure which will select the best configuration in which to run any given multi-process program, and, second, to understand the factors affecting the response times of the different configurations. Our approach is to analyze the results of simulating each trace in set A in every configuration. In Section 8-4.2 we present the winning configurations in 4x4 and the response times of the simulations. Sections 8-4.3 and 8-4.4 explore the 4x4 relative percentage differences, specifically, step differences and best/order differences. In Section 8-4.5 we briefly discuss dyadic placement for four-process programs. Finally, in Section 8-4.6 we summarize the 4x4 results.

8-4.2. 4x4 Winning Configurations and Response Times

When four processors are available, APART is the best configuration in all cases except the upper left corner cell, **aa** (see Figure 8-3). The always-APART policy is thus as successful in 4x4 situations as in 2x2 and 3x3 situations.

After APART, the triples and pairs are the next best configurations. In **aa-1** and **aa-2** the triples and pairs are the best configurations. (The relative merits of triples and pairs are examined more closely in Sections 8-5.2 and 8-5.3).

The loner configurations are next best after the triples and pairs. The four loner configurations are always 11th, 12th, 13th, and 14th best out of the 15 configurations in every run, including the **aa** runs.

TOGETHER is always the worst configuration, in every run, including the upper left corner.

As examples of the response time orders of the configurations, and as examples of the response times of the four-process runs, Figure 8-3 shows the response times of all 15 configurations, ordered by response time, for four runs in set A.

Order	Configuration	Response Time (ms)	Order	Configuration	Response Time (ms)
aa-1			**ai-2**		
1	PAIR02	1,484	1	APART	26,659
2	PAIR01	1,513	2	TRIPLE01	28,206
3	TRIPLE02	1,529	3	TRIPLE02	28,674
4	TRIPLE01	1,537	4	TRIPLE03	29,869
5	PAIR03	1,541	5	TRIPLE12	30,104
6	TRIPLE23	1,544	6	TRIPLE13	31,827
7	TRIPLE13	1,556	7	PAIR03	32,537
8	TRIPLE12	1,560	8	TRIPLE23	32,845
9	TRIPLE03	1,561	9	PAIR02	32,959
10	APART	1,643	10	PAIR01	33,644
11	LONER2	1,673	11	LONER3	35,295
12	LONER1	1,695	12	LONER2	36,205
13	LONER0	1,706	13	LONER1	37,324
14	LONER3	1,708	14	LONER0	40,268
15	TOGETHER	2,003	15	TOGETHER	45,862
aq-1			**az-2**		
1	APART	40,667	1	APART	40,671
2	TRIPLE02	46,628	2	TRIPLE02	79,214
3	TRIPLE01	49,318	3	TRIPLE01	79,760
4	TRIPLE03	49,639	4	TRIPLE03	79,876
5	TRIPLE23	53,151	5	TRIPLE12	80,565
6	TRIPLE12	53,270	6	PAIR03	80,565
7	TRIPLE13	56,341	7	TRIPLE23	80,681
8	PAIR01	56,999	8	PAIR01	80,681
9	PAIR03	58,195	9	TRIPLE13	81,226
10	PAIR02	58,657	10	PAIR02	81,226
11	LONER1	67,829	11	LONER3	119,769
12	LONER3	68,111	12	LONER1	119,885
13	LONER2	71,035	13	LONER2	120,431
14	LONER0	75,243	14	LONER0	121,236
15	TOGETHER	91,372	15	TOGETHER	160,440

Figure 8-3:
Response Times, in Order,
For Four Runs in Set A

The mean response time over all 420 configurations in all runs in set A is 48,051 ms, with a standard deviation of 30,702 ms; the fastest response time in set A is 1,484 ms (the PAIR02 configuration in **aa-1**), the slowest response time is 163,936 ms (the TOGETHER configuration in **az-1**).

Summary statistics on the response times are shown in Figure 8-4.

Type of Configuration	# of Simulations	Response Time (ms)			
		Mean	Std. Dev.	Minimum	Maximum
APART	28	28,601	13,945	1,643	52,099
triples	168	41,276	23,163	1,529	84,589
pairs	84	44,520	24,447	1,484	84,589
loners	112	58,545	34,739	1,673	125,525
TOGETHER	28	76,761	46,362	2,003	163,937

Figure 8-4:
Summary Statistics on
Response Times in Set A

8-4.3. Step Differences

For each run, we compute the relative percentage difference between the best response time and the second best, between the second best and the third best, and so on. As in the 3x3 investigation, we call these *step differences* since they are the differences in response time as we step from one configuration to the next.

The mean relative percentage step difference, averaged over the 392 step differences in set A, is 10.85%, with standard deviation 18.61%; the smallest step difference is zero, the largest is 96.39%.

The means of the individual step differences are shown in Figure 8-5.

Step Differences				
Step	Mean	Std. Dev.	Minimum	Maximum
Best – 2nd-Best	30.00	25.14	0.74	94.77
2nd-Best – 3rd-Best	4.39	3.77	0.14	15.33
3rd-Best – 4th-Best	2.24	2.21	0.20	10.45
4th-Best – 5th-Best	4.39	4.46	0.25	17.36
5th-Best – 6th-Best	2.07	2.77	0.00	11.09
6th-Best – 7th-Best	2.77	2.32	0.07	9.46
7th-Best – 8th-Best	1.64	2.22	0.00	9.21
8th-Best – 9th-Best	3.71	2.88	0.04	10.11
9th-Best – 10th-Best	2.29	2.39	0.00	7.42
10th-Best – 11th-Best	32.02	26.54	0.06	94.77
11th-Best – 12th-Best	4.54	4.43	0.17	17.13
12th-Best – 13th-Best	5.48	3.35	0.50	14.29
13th-Best – 14th-Best	7.08	6.26	0.19	25.80
14th-Best – 15th-Best	49.28	22.08	15.78	96.39
All Step Differences	10.85	18.61	0.00	96.39

Figure 8-5:
Mean Step Differences in Set A
as Relative Percentage Differences

The mean step difference of almost 11% is smaller than the mean 3x3 step difference of 24.51%, but it is still a significant fraction of the best response time, indicating that the selection of a good configuration is important.

The individual step differences vary greatly and define clusters of configurations. There is a large difference between the best configuration and the second best; this is almost always the difference between APART and the best triple-type configuration, indicating that APART is almost always by far the best configuration, 30.00% better than the next best configuration. The next eight step differences are small, no

more than 4.39%; these are almost always the differences between the triple-type and pair-type configurations, indicating that there is not a large difference between adjacent triples and pairs. This is not to say that any triple or pair is as good as any other, only that the adjacent step differences are small; the eight small step differences add up to a 23.50% difference between the best triple or pair and the worst. The 10th-best/11th-best step difference, almost always the difference between the worst of the triples and pairs and the best loner, is again substantial. The next three step differences, always the differences between the adjacent loner-type configurations, are small. The difference between the worst loner and TOGETHER, the 14th-best/15th-best step difference, is very large, almost 50%. Thus the clusters of configurations are APART, the triples and pairs, the loners, and TOGETHER. The differences in response time between one cluster and another are large, the differences in response time between configurations within the same cluster are much smaller. We observed the same clustering phenomenon in the 3x3 situation (see Section 7-2.2.4.2).

8-4.4. Best/Order Differences

The importance of selecting the best configuration, or at least a good configuration, can be gauged by examining the differences between the response times of the configurations. If the differences between the fastest response time and the slowest response time in each run are small, then all configurations have similar response times, and it is not important to select a configuration carefully. The placement procedure could choose randomly or arbitrarily. Even if the fastest response time is much faster than the slowest, it may be that a number of configurations have similarly fast response times. In that case the placement procedure should select one of the fast configurations, but it is not important which.

To test whether this is the case, in each run we compute the difference between the response time of the fastest configuration and that of the second-fastest configuration, between the response time of the fastest configuration and that of the third-fastest configuration, and so on, through the difference between the fastest response time and the slowest response time in each run. These differences are called *best/order differences*. If they are large, then it is important to select the very best configuration.

The mean relative percentage best/order differences are shown in Figure 8-6. Each row in the figure summarizes 28 best/order differences, one from each run in set A.

Best/Order Differences				
	Mean	Std. Dev.	Minimum	Maximum
Best – 2nd-Best	30.00	25.14	0.74	94.77
Best – 3rd-Best	34.39	26.08	1.51	96.11
Best – 4th-Best	36.63	25.92	1.81	96.39
Best – 5th-Best	41.02	25.55	2.15	98.09
Best – 6th-Best	43.09	24.58	2.65	98.09
Best – 7th-Best	45.86	24.25	3.02	98.37
Best – 8th-Best	47.50	23.99	3.16	98.37
Best – 9th-Best	51.21	25.40	3.75	99.71
Best – 10th-Best	53.50	24.16	8.06	99.71
Best – 11th-Best	85.52	49.94	8.12	194.48
Best – 12th-Best	90.07	48.91	9.05	194.77
Best – 13th-Best	95.55	49.56	9.56	196.11
Best – 14th-Best	102.63	48.85	10.95	198.09
Best – Worst	151.91	70.33	26.73	294.48

Figure 8-6:
Mean Best/Order Differences in Set A
as Relative Percentage Differences

These results show again the clusters we saw in the step difference results. The best/second-best through the best/10th-best differences differ from one to the next by only about 2-4 percentage points. Since APART is almost always best, and the triple- and pair-type configurations are almost always second best through 10th best, these small differences show the clustering of triples and pairs. Again, although the differences from one best/order difference to the next are small, they do add up, so that the difference between the best triple or pair best/order difference and the worst is 23.50 percentage points.

There is a large jump of 32.02 percentage points between the best/10th-best order difference and the best/11th-best, then there are small differences of 5-7 percentage points between the next four order differences (best/11th-best through best/14th-best). This is the difference between the triple-and-pair cluster and the loner cluster, followed by the cluster of loner-type configurations. The difference between the best/14th-best and the best/worst difference is 49.28 percentage points; the gap between the loner cluster and the TOGETHER cluster of one.

It is clearly worthwhile avoiding the worst configuration, TOGETHER, which on average is 152% slower than the fastest configuration; at worst, it is almost 300% worse. This is especially so when one considers that large programs often run for many hours; response times of 10 hours instead of 30 hours are worth working for. The loner configurations are about 100% slower than the fastest configuration; so one should expect response times of, say, 20 hours instead of 10. It is certainly worth expending some resources to avoid the worst configurations. Many of the triple- and pair-type configurations are 50% slower than the best configuration. Even the mean best/second-best difference is 30%, so it is very likely to be worth some cost (in the form of computation and delay) to find the very best configuration.

In 2x2 full grid set N, the differences between the response times of APART and TOGETHER range in magnitude from 5.83% to 49.60%, with mean magnitude 30.16% and standard deviation 9.92% (see Section 6-5 and Figure 6-13). The three-process mean best/second-best difference is 35.72% (see Section 7-2.2.4.3 and Figure 7-6), which is larger than the four-process mean best/second-best difference of 30.00%, but the four-process best/order differences grow much larger than the three-process differences; the three-process mean best/worst difference is 98.05%, while the equivalent four-process best/order difference is 151.91%. The largest three-process best/order difference is about 190%; the largest four-process difference is about 290%. The best/order differences seem to grow with the number of processes. If so, as the number of processes grows, it is increasingly important to choose the configuration carefully.

Figure 8-7 shows the best/second-best and the best/worst relative percentage differences for each run in set A.

	IO-COMM									
	IO	COMM	IO	COMM	IO	COMM	IO	COMM	IO	COMM
CPU	0	100	25	75	50	50	75	25	100	0
0	**aa**				**ac**				**ae**	
	2.0	35.0			11.8	105.7			61.0	244.6
	0.7	26.7			14.3	112.1			64.3	246.2
20			**ag**				**ai**			
			10.1	80.6			5.5	65.2		
			5.4	78.8			5.8	72.0		
40	**ak**				**am**				**ao**	
	16.3	138.8			12.7	105.9			40.6	177.2
	24.1	134.9			17.2	134.5			30.0	169.9
60			**aq**				**as**			
			14.7	124.7			16.1	146.1		
			24.4	145.9			24.4	138.2		
80	**au**				**aw**				**ay**	
	39.4	183.3			30.2	170.3			64.9	249.9
	46.3	191.7			17.7	151.8			60.4	246.8
100	**az**									
	84.9	282.1								
	94.8	294.5								

Figure 8-7:
Best/Second-Best and Best/Worst
Relative Percentage Differences
in Set A

In each cell, the upper row is run 1, the lower row run 2. The left column in each cell is the best/second-best order difference and the right column is the best/worst order difference, both given as relative percentage differences.

Even in the upper left corner the worst configuration is about 30% slower than the best. In most of the grid the difference is 100-150%, although it sometimes exceeds 200%. At the bottom of the grid, the difference is close to 300%. In the lower left region, into which we suspect most scientific distributed programs will fall, the difference between the best and the worst response time is about 180-280%.

The differences between the response time of the best configuration and the response time of the second-best configuration are less dramatic, but still substantial in most cases. In the upper left, the difference is only 1-2%. Through most of the middle of the grid the difference is about 5-15%, but on the right and bottom edges the difference is 30-60%. At the very bottom, it is 85-95%. In the lower left region, the difference between the best and second-best response time is 40-95%.

8-4.5. Dyadic Placement

For cases in the top, zero percent CPU row of the grid, dyadic placement always recommends the TOGETHER configuration, which is in fact the worst configuration. Below the top row, dyadic placement always recommends the APART configuration, which is in fact the best configuration. This is the same behaviour as in 3x3 set U (see Section 7-2.2.7).

Triples, pairs, and loners are never recommended, nor are there any inconclusive, conflicting results. If we claim that real programs will never fall in the first row — a reasonable claim — we can say that dyadic placement works well. Of course, always choosing APART (the always-APART policy) works even better.

8-4.6. Summary

We have seen that when there are four processes to be placed and there are four processors available, the APART configuration is nearly always the fastest configuration. Because of this, the always-APART placement policy would be very successful.

The four-process configurations can be clustered by response times. APART is fastest, the next fastest cluster includes the triples and the pairs, followed by the loners, and the slowest cluster consists of TOGETHER. The differences in response time between one cluster and another are large; the differences in response time between configurations within the same cluster are much smaller. There is a similar clustering phenomenon with three-processes.

The configuration in which a distributed program is run has a very strong effect on its response time. With four processes, the mean difference between the fastest configuration's response time and that of the slowest is 152%, and the largest difference is almost 300%. Even the difference between the fastest configuration's response time and that of the second-fastest configuration averages 30%.

The sizes of the differences between configuration response times grow with the number of processes. With two processes, the mean fastest/slowest difference is about 30%, with three processes it is about 98%, and with four processes it is 152%.

Dyadic placement always picks the worst configuration, TOGETHER, in the top row of the grid. In the rest of the grid, it always picks APART, which is the fastest configuration. Since real programs will probably never fall in the first row, dyadic placement is usable, but the always-APART policy is more successful, easier to implement, and faster to use.

8-5. Four Processes with Three Processors (4x3)

8-5.1. Introduction to the 4x3 Situation

When there are fewer processors than processes, some processes must be collocated. Which configuration is best in such a case, and how can the best configuration be selected? In this and the following section, we discuss such situations: in this section we discuss the 4x3 situation; in the following section the 4x2 situation. This section is devoted to the examination of a number of aspects of the 4x3 situation and the configurations available in it, and to reporting on the success of various placement procedures.

APART is the best configuration for four processes when there are four processors available. In a 4x3 situation the APART configuration cannot be used, so the placement procedure must select one of the other 14 four-process configurations listed in Figure 8-1. The investigations of the 4x3 situation discussed in this section use the same set A traces and simulations as were used in the preceding section, except that the APART simulation is ignored.

8-5.2. 4x3 Winning Configurations

8-5.2.1. Hypothesis

Based on the 3x2 results, our hypothesis is that the best configuration is the one which spreads out the CPU demand as much as possible and as evenly as possible. Using three processors spreads out the CPU demand more than any way of using two processors, so we expect that the triples configurations will be the best 4x3 configurations. In fact, we expect to find that all of the triple-type configurations are better than any of the 4x2 configurations (pairs, loners, and TOGETHER).

8-5.2.2. Results and Triple/Pair Interspersion

The six triple-type and three pair-type configurations are the best configurations available in 4x3 situations. In almost all cases, the 4x3 winner (the fastest 4x3 configuration) is a triple. In fact, in almost all cases the fastest four 4x3 configurations are all triples. In almost all cases, the ninth fastest configuration (the last of the triples and pairs) is always a pair. (''In almost all cases'' means in all cases except case **aa** in the upper left corner.)

In between the four fastest triples and the slowest pair, the remaining two triple and two pair configurations are ordered in one of the patterns shown in columns (a)-(e) of Figure 8-8. We call these patterns in which triples and pairs are interspersed *interspersion patterns.* Although the pattern in column (f) is possible, it does not occur. Figure 8-8 also lists as an example one run in which each interspersion pattern occurs, and a label for each pattern.

The labels shown in Figure 8-8 are two-part characterizations of the interspersion patterns. The first part is the number of triple-type configurations that are faster than any non-triple configuration. The second part is the number of non-triples that occur in the order before the last triple. The patterns shown in columns (d) and (e) of Figure 8-8 are both characterized as 4/2; we label the column (e) pattern ''4/2b'' to differentiate.

	(a)	(b)	(c)	(d)	(e)	(f)
1	triple	triple	triple	triple	triple	triple
2	triple	triple	triple	triple	triple	triple
3	triple	triple	triple	triple	triple	triple
4	triple	triple	triple	triple	triple	triple
5	triple	triple	triple	pair	pair	pair
6	triple	pair	pair	triple	pair	triple
7	pair	triple	pair	pair	triple	triple
8	pair	pair	triple	triple	triple	pair
9	pair	pair	pair	pair	pair	pair
Example	**aq-1**	**aq-2**	**aw-1**	**az-1**	**as-1**	(none)
Label	6/0	5/1	5/2	4/2	4/2b	4/1

Figure 8-8:
Interspersion Patterns of Triple and Pair Configurations

The interspersion characterization of each run in set A is shown in Figure 8-9.

	IO-COMM									
CPU	IO 0	COMM 100	IO 25	COMM 75	IO 50	COMM 50	IO 75	COMM 25	IO 100	COMM 0
0	**aa** 0/3 0/3				**ac** 5/2 6/0				**ae** 4/2 4/2	
20			**ag** 6/0 6/0				**ai** 6/0 5/1			
40	**ak** 6/0 6/0				**am** 6/0 6/0				**ao** 4/2 4/2	
60			**aq** 6/0 5/1				**as** 4/2b 5/1			
80	**au** 5/1 6/0				**aw** 5/2 5/2				**ay** 4/2 4/2	
100	**az** 4/2 4/2									

Figure 8-9:
Interspersion Patterns in Set A

The 0/3 characterization of the **aa** runs means that the best pair is better than the best triple, and all three pairs are better than the worst triple (see Figure 8-3).

There does not appear to be any pattern in the distribution of the interspersion patterns.

8-5.2.3. Conclusions

It is not true that every triple-type configuration is faster than any other type of 4x3 configuration, although the fastest four 4x3 configurations are always triples. The placement procedure cannot arbitrarily select any triple-type configuration and achieve one of the fastest response times. For example, in run **aq-1** shown in Figure 8-3, the difference between the fastest triple and the slowest is close to 10,000 ms (9,713 ms), which is close to a 25% relative percentage difference (23.88%). On the other hand, we know that the placement procedure only needs to identify the fastest of the triples; it need not pay any attention to pairs or other types of configurations.

In the preceding chapter we saw the same phenomenon in the 3x2 situation, in which a loner configuration in the 2:1 arrangement is the fastest 3x2 configuration. We see here that it is true for 4x3, and we will see in Section 8-6, below, that it is true for 4x2, so it seems to be true for all NxM situations.

Stated more generally, in an NxM situation where M<N, the best configuration will be an N@M configuration using all of the M available processors. The placement procedure need only select the fastest (or one of the fastest) of the N@M configurations; it need not pay any attention to N@m configurations where where m<M. In short, all available processors should be used.

In an NxN situation, APART will always be the best configuration (except for extreme cases in the very upper left corner of the full grid), so using all the available processors is a good policy not only for M<N, but for all M≤N.

It is quite surprising that 4@2 configurations such as the pair-type configurations would ever be faster than 4@3 configurations such as the triple-type configurations. This demonstrates that, outside of the

transition zone, there are configurations using fewer processors that are faster than configurations which use more processors.

8-5.3. Triple- and Pair-Type Configurations

8-5.3.1. Differences Between Best Pair and Next Better Triple

In this section we examine the differences between the response times of the triple-type and pair-type configurations. In particular, we wish to determine whether the response time of the best pair in each run is much larger or only slightly larger than that of the next faster configuration, which is (nearly) always a triple. If there is a significant gap, it could indicate that the cluster of triples and pairs is composed of two smaller clusters, the fastest triples and the remaining triples and pairs.

Figure 8-10 lists, for each run in set A, the fastest pair-type configuration and the position of that configuration in the 4x3 response time order, and how much slower that pair-type configuration is than the next faster triple configuration. The difference is expressed both as an absolute number of milliseconds and as a relative percentage. (This percentage is computed relative to the response time of the fastest configuration for the run, which is usually APART.)

Run	Fastest Pair-Type Config.			Diff. from Next Faster Config.	
	Place	Config.	Resp. Time	Absolute Diff. (ms)	Relative Diff. (%)
aa-1	1	PAIR02	1,484	(fastest configuration)	
aa-2	1	PAIR02	1,585	(fastest configuration)	
ac-1	6	PAIR02	13,025	951	9.46%
ac-2	7	PAIR03	12,907	160	1.64%
ae-1	5	PAIR02	20,845	0	0.00%
ae-2	5	PAIR01	20,639	0	0.00%
ag-1	7	PAIR02	21,191	38	0.22%
ag-2	7	PAIR02	22,729	518	2.88%
ai-1	7	PAIR02	27,808	295	1.27%
ai-2	6	PAIR03	32,537	710	2.66%
ak-1	7	PAIR01	37,296	597	2.32%
ak-2	7	PAIR03	39,340	2,115	7.62%
am-1	7	PAIR03	41,902	541	1.69%
am-2	7	PAIR03	39,143	241	0.88%
ao-1	5	PAIR01	38,012	0	0.00%
ao-2	5	PAIR02	37,210	0	0.00%
aq-1	7	PAIR01	56,999	658	1.62%
aq-2	6	PAIR02	50,877	768	2.14%
as-1	5	PAIR01	56,381	4,341	11.09%
as-2	6	PAIR02	58,463	2,148	5.26%
au-1	6	PAIR03	71,706	2,288	5.03%
au-2	7	PAIR03	71,156	4,074	9.21%
aw-1	6	PAIR01	72,965	1,058	2.23%
aw-2	6	PAIR01	76,443	3,199	6.14%
ay-1	5	PAIR03	67,144	0	0.00%
ay-2	5	PAIR02	66,675	0	0.00%
az-1	5	PAIR03	82,618	0	0.00%
az-2	5	PAIR03	80,565	0	0.00%

Figure 8-10:
Response Time Differences
Between the Fastest Pair and the Next Faster Triple

The mean of the differences is 882 ms, or 2.62%, with standard deviation 1237 ms, or 3.24%, in both cases counting the **aa-1** and **aa-2** differences as zero. In some runs the difference is sizable (9-11%), in others it is small (0.2-1.6%). These figures are very similar to the relative percentage differences between the response times of other adjacent triple and pair configurations, whether the configurations are both triples, both pairs, triple-pair or pair-triple. Apparently there is only one cluster of triples and pairs; there does not seem to be a cluster of faster triples and a cluster of slower triples and pairs.

8-5.3.2. Pairs and Triples with Equal Execution Times

In certain runs, pair-type configurations have exactly the same response times as triple-type configurations. Some instances of this appear in Figure 8-10, above, as rows in which there is no difference between the fastest pair and the next faster triple. Figure 8-11 lists all instances of this phenomenon.

Run	Places in Order		Configurations		Response Time
ae-1	4	5	TRIPLE13	PAIR02	20,845
ae-1	6	7	TRIPLE12	PAIR03	21,067
ae-1	8	9	TRIPLE23	PAIR01	21,875
ae-2	4	5	TRIPLE23	PAIR01	20,639
ae-2	6	7	TRIPLE12	PAIR03	21,089
ae-2	8	9	TRIPLE13	PAIR02	21,661
ao-1	4	5	TRIPLE01	PAIR01	38,012
ao-1	6	7	TRIPLE02	PAIR02	38,635
ao-1	8	9	TRIPLE03	PAIR03	40,393
ao-2	4	5	TRIPLE13	PAIR02	37,210
ao-2	6	7	TRIPLE03	PAIR03	37,262
ao-2	8	9	TRIPLE23	PAIR01	39,781
ay-1	4	5	TRIPLE12	PAIR03	67,144
ay-1	6	7	TRIPLE02	PAIR02	67,291
ay-1	8	9	TRIPLE23	PAIR01	70,204
ay-2	4	5	TRIPLE02	PAIR02	66,675
ay-2	6	7	TRIPLE23	PAIR01	67,321
ay-2	8	9	TRIPLE03	PAIR03	71,161
az-1	4	5	TRIPLE03	PAIR03	82,618
az-1	6	7	TRIPLE02	PAIR02	83,842
az-1	8	9	TRIPLE23	PAIR01	84,589
az-2	4	5	TRIPLE12	PAIR03	80,565
az-2	6	7	TRIPLE23	PAIR01	80,681
az-2	8	9	TRIPLE13	PAIR02	81,226

Figure 8-11:
Pairs and Triples with the Same Response Times

These equalities occur in exactly those runs whose interspersion pattern is 4/2. Thus, the 4/2 interspersion pattern shown in column (d) of Figure 8-8 is actually a ‘‘3, 3-equal’’ pattern as shown in Figure 8-12. Each succeeding line in Figure 8-12 has a slower response time than the line above, so pair (a) has a faster response time than triples (b) or (c), and so on.

1	triple	
2	triple	
3	triple	
4, 5	triple=pair	(a)
6, 7	triple=pair	(b)
8, 9	triple=pair	(c)

Figure 8-12:
Interspersion Pattern
with Equal Triples and Pairs

Two important observations we had made earlier are still true despite these triple-pair equalities: (1) the fastest few 4x3 configurations are still triples, although now only the fastest three configurations are triples rather than the fastest four, and (2) there are still some pairs that are faster than some triples.

In the cases in which they occur at all, cases **ae**, **ao**, **ay**, and **az**, equal response times occur in both repetitions of the case. Cases **ae**, **ao**, and **ay** are all the cases in the right-most column of the grid, the column of 100% I/O and 0% communication (see Figure 8-9). Case **az** is the whole of the bottom row, the row of 100% CPU. In all four cases (all eight runs), there is no communication. In case **ae** there is only I/O (and its indirect CPU demand), there are no communication or execute actions; in cases **ao** and **ay** there is a mix of CPU and I/O, but still no communication; in case **az** there is only CPU use, no communication or I/O actions.

In each of these eight runs, there are three instances of a pair and a triple with equal response times. Since there are only three pair-type configurations in each run, every pair-type configuration has the same response time as some triple-type configuration, in those runs in which equal pair-triple response times occur.

Which triples are the pairs equivalent to, equivalent in the sense of having the same response time? Figure 8-11 shows that each pair-type configuration is equivalent to one of two particular triple-type configurations. These are listed in Figure 8-13, both by configuration label and by the numerical representation of the configuration.

Labels		
Pair	Triples	
PAIR01	TRIPLE01	TRIPLE23
PAIR02	TRIPLE02	TRIPLE13
PAIR03	TRIPLE03	TRIPLE12

Configurations		
Pair	Triples	
01/23	01/2/3	23/0/1
02/13	02/1/3	13/0/2
03/12	03/1/2	12/0/3

Figure 8-13:
Pair-type and Triple-type Configurations
with Equal Response Times

Figure 8-14 depicts PAIR01 and its associated triples.

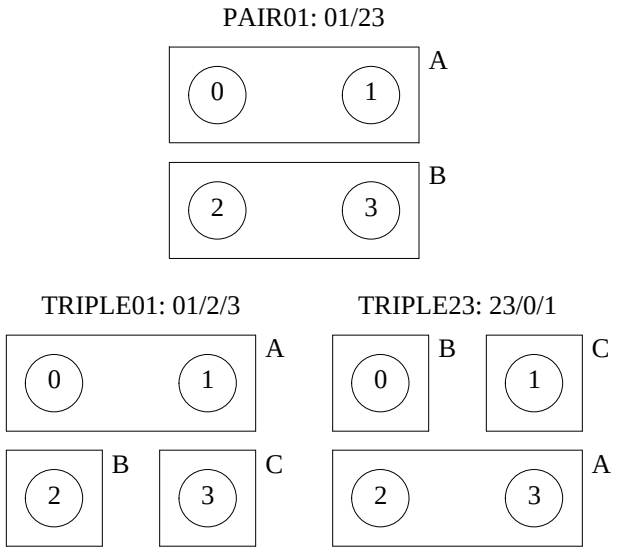

Figure 8-14:
Some Pair- and Triple-Type Configurations Depicted.
The numbered circles are processes and the lettered boxes are processors.

(In the following, when we speak of ‘‘the response time of processor A’’ or say ‘‘when processor A finishes,’’ we mean ‘‘when the processes assigned to processor A finish’’ or ‘‘the response time of the longest-running process on processor A.’’)

In the TRIPLE01 configuration, the processes on processor A will take much longer to finish than either the process on B or the process on C; this is clearly true, since A has two processes running on it while B and C each have one, and all four processes are similar. Similarly, in TRIPLE23, processor A with two processes will take longer to finish than either B or C, each with one process.

We are considering only cases in which there is no communication, so the response time of processes running on a processor depends only on the processes and on the processor itself; it does not depend on anything that happens outside the processor. Recall that we are considering identical processors, and that each processor has associated with it an I/O facility (say, a disk), which is identical to all other I/O facilities.

In the PAIR01 configuration, either processor A finishes last, and therefore determines the overall response time, or processor B finishes last. Let us say for the moment that processor A finishes last, determining the response time of the configuration. The response time of processor A, running processes 0 and 1, does not depend on anything outside A, so it must be identical with the response time of processor A in the TRIPLE01 configuration, which also runs processes 0 and 1 and does not depend on anything outside of the processor. As discussed above, processor A determines the overall response time of the TRIPLE01 configuration, so the response time of the TRIPLE01 configuration must be exactly the same as the response time of the PAIR01 configuration, assuming that A finishes last in PAIR01.

If, in the PAIR01 configuration, processor B finishes last, then by similar reasoning the response time of processor B in PAIR01 will be the same as the response time of processor A in TRIPLE23, and those two processors determine the response times of the configurations, so in this case PAIR01 and TRIPLE23 will have identical response times.

Thus, the response time of the PAIR01 configuration not only can be, but must be identical to the response time of either TRIPLE01 or TRIPLE23 when there is no communication. Similarly, PAIR02 must be equivalent to either TRIPLE02 or TRIPLE13, and PAIR03 must be equivalent to either TRIPLE03 or TRIPLE12, when there is no communication.

When there is communication between processes, the response time of a processor depends not only on the processor and the processes running on it, but also on other processors and processes with which communication takes place. In particular, the amount of time processes spend waiting for remote (off-machine) messages to arrive will depend on when those messages are sent. Thus, when there is communication, a pair-type configuration need not be equivalent to a triple-type configuration.

8-5.3.3. Response Time Ranges of Pairs and Triples

We had hypothesized that all triples were faster than all pairs, as illustrated in Figure 8-15(a). In fact, we find that some pairs are better than some triples, as shown in Figure 8-15(b).

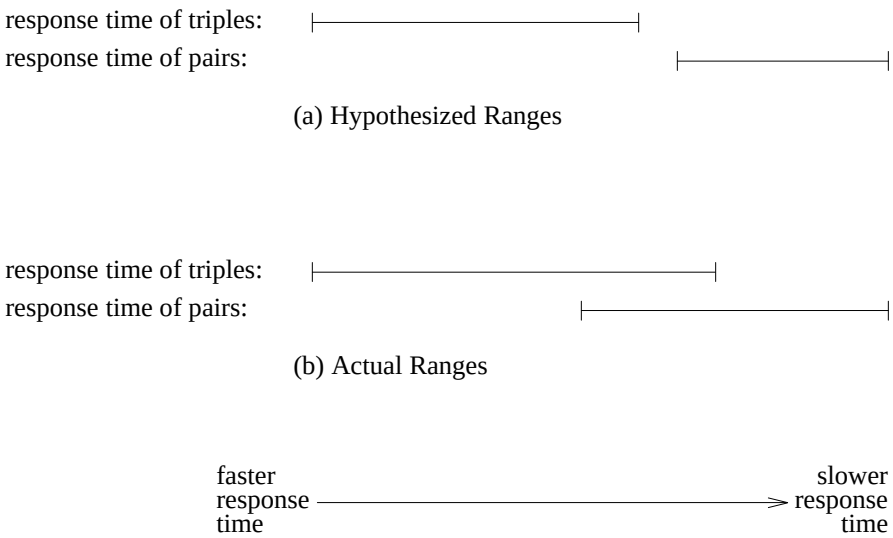

Figure 8-15:
Hypothesized and Actual Response Time Ranges of Triples and Pairs

This happens all over the grid, and it happens to various extents, from not at all — as in the 6/0 interspersion pattern, to quite a bit — as in the 4/2b interspersion pattern (see Figure 8-9). Often it happens on one repetition of a case and not on the other (for example, **au-1** has interspersion pattern 5/1, while **au-2** has 6/0).

If all the triple configurations have low response time and all the pair configurations have high response time, then there will be no overlap, and a 6/0 interspersion pattern, as shown in Figure 8-16(a).

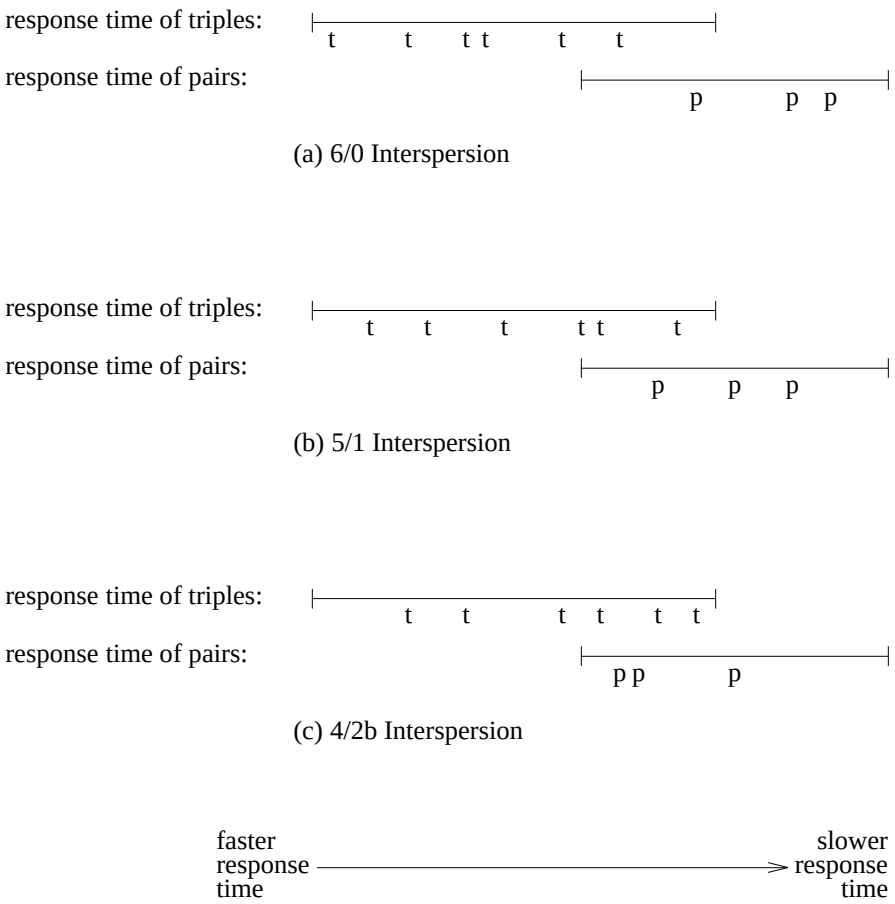

Figure 8-16:
Overlap of Triple and Pair Configurations' Response Times

If, on the other hand, some of the pair configurations have better response times, and/or some of the triple configurations have worse response times, then there can be overlap. The overlap can be of various extents, as illustrated in Figure 8-16(b) and (c). The overlap of the ranges is partial, not total, so we can have 4/2b, but not 3/2 or 0/3 (except for the upper left corner cell, which is anomalous and we ignore it). The non-overlapping portions of the ranges allow the 6/0 interspersion pattern to exist.

Where in its range a particular configuration falls probably has to do with how well the characteristics of the particular processes suit the configuration. For example, the PAIR01 configuration (01/23) is very well suited to a distributed computation in which there happens to be much communication between processes 0 and 1, and between 2 and 3, but little communication between other pairs of processes. If the total CPU demand of processes 0 and 1 is approximately equal to the sum of the demands of processes 2 and 3, so that one will not take much longer than the other, then the processes are even better suited to that configuration. Similarly, the TRIPLE01 configuration (01/2/3) would be especially unsuitable if processes 0 and 1 had heavy CPU demands but little communication between them, while there was a great deal of communication between the other pairs of processes.

Recall that all four processes in each run are quite similar, so it appears that even small differences in the characteristics of the processes can have a significant effect on which configurations are better or worse. We could probably devise a successful placement procedure that examines which processes have slightly higher or lower CPU demand and which pairs of processes have slightly more or less communication between them, and determines which configurations would suit such characteristics; we have not looked into this possibility. On the other hand, we are mostly interested in finding only the best

configuration for any given situation, and since we believe we can do that by noting that the best 4x3 configuration is a triple and then finding the best triple, there is no need to look into picking configurations by suitability.

Concerning how it can be that in many cases the two repetitions have different interspersion patterns, recall that the trace generating program, GENACTIONS, is accurate only to within 2-3%. So the two repetitions, although generated using the same parameters, will be somewhat different. Also, as noted above, it appears that even small differences in the processes make a difference as to which configuration is better.

8-5.4. Summary

We hypothesized that the more the CPU demand is spread out, and the more evenly it is spread out, the better the configuration performs. This is not true; the response times of the triple-type configurations and the pair-type configurations are interspersed, even though the triples spread the CPU demand over three processors while the pairs use only two. It is surprising that there are configurations which use fewer processors which are faster than configurations which use more processors. On the other hand, the fastest 4x3 configurations are always triples, so the hypothesis may be partially correct.

In every situation studied so far, the fastest configuration uses all available processors. This is true in NxN situations, including 2x2, 3x3, and 4x4, and in Nx(N-1) situations, including 3x2 and 4x3.

The cluster of triple and pair configurations, all of which have similar response times, does not consist of two sub-clusters, one with the fastest triples and one with the slower triples and the pairs.

When there is no communication between processes, so that response time depends only on the processes and the processors, not on a communication medium, and there is no waiting for messages, then certain triples will have exactly the same response time as certain pair configurations.

It might be possible to devise a placement procedure which examines which processes have slightly higher or lower CPU demand and which pairs of processes have slightly more or less communication between them, and determines which configurations would suit such characteristics.

In these experiments, all processes in a trace are similar to one another. Nonetheless, the differences between one process and another do have an effect on the response times of the configurations, and on which configuration is best. In situations where the processes of a program are less similar to each other, it would be even more important to choose the best configuration.

We have noted that the best 4x3 configurations are triples, and we have examined the extent to which that is true. We have yet to discuss whether we can select the best of the six triple configurations, and how we might do so. We will discuss this further in Section 8-7. For now, though, let us turn to the 4x2 situation.

8-6. Four Processes with Two Processors (4x2)

8-6.1. Introduction to the 4x2 Situation

When there are four processes to be placed and only two processors available, a 4x2 situation, then the fastest configurations, APART and the triple-type configurations, are not available. Only the pair-type configurations, the loner-type configurations, and the TOGETHER configuration are possible (see Figure 8-1). In this section, we explore which of the 4x2 configurations is fastest, and why.

The 4x2 investigations use the same set A traces and simulations as were used in the 4x4 and 4x3 work discussed above. The only difference is that APART and the triple-type configurations cannot be used; only the pairs, loners, and TOGETHER are used.

8-6.2. 4x2 Winners

The best configuration for a given situation almost always uses as many processors as are available (the exceptions being in the upper left corner cell in the full grid and in the transition zone of the zoomed-in grid). We have seen that this is true in 2x2, 3x3, and 4x4 situations, in which APART is fastest; in 3x2 situations, in which a 2:1 loner configuration is fastest; and in 4x3 situations, in which a 2:1:1 triple configuration is fastest. Our hypothesis, previously stated, is that these types of configurations are best because they spread the CPU demand out as much and as evenly as possible.

Based on this hypothesis, we expect that in 4x2 situations all pair-type configurations will be faster than all loner-type configurations. If not, then we expect that at least the fastest 4x2 configuration will be a pair. Recall that in these experiments, all processes in each trace have similar characteristics, so that a 2:2 pair-type configuration will distribute the CPU demand more evenly than a 3:1 loner-type configuration possibly could. Either of the 4@2 configuration types, pairs and loners, will spread the CPU demand across more processors than the 4@1 TOGETHER configuration, so TOGETHER should be worse than either pairs or loners. If the processes in a trace were not all similar, if instead there were one process with very heavy CPU demand and three with light demands, a loner configuration might produce the most even distribution of CPU demand.

In every run in set A, the response time order of the 4x2 configurations is as shown in Figure 8-17.

Order	Config. Type	
1	pair	2:2
2	pair	2:2
3	pair	2:2
4	loner	3:1
5	loner	3:1
6	loner	3:1
7	loner	3:1
8	TOGETHER	4:0

Figure 8-17:
Order of 4x2 Configurations
from Fastest to Slowest

In every run in set A, the fastest 4x2 configuration is a pair, as expected. In fact, the three pair-type configurations are the fastest 4x2 configurations in every run, also as expected. The four loner-type configurations are next fastest after the pairs, and TOGETHER is always slowest. This supports the hypothesis about evenly distributing the CPU demand.

8-6.3. Step Differences

Figure 8-18 shows the mean step differences between the 4x2 configurations, that is, the means of the relative percentage differences between the fastest 4x2 configuration and the second fastest, between the second and third fastest, and so on.

4x2 Step Differences				
Step	Mean	Std. Dev.	Minimum	Maximum
Best Pair – Second-Best Pair	2.57%	2.02%	0.04%	9.03%
Second-Best Pair – Worst Pair	4.35%	3.25%	0.08%	10.11%
Worst Pair – Best Loner	32.49%	26.05%	0.19%	94.77%
Best Loner – Second-Best Loner	4.54%	4.43%	0.17%	17.13%
Second-Best Loner – Third-Best Loner	5.48%	3.35%	0.50%	14.29%
Third-Best Loner – Worst Loner	7.08%	6.26%	0.19%	25.80%
Worst Loner – TOGETHER	49.28%	22.08%	15.78%	96.39%
All 4x2 Step Differences	15.12%	21.59%	0.04%	96.39%

Figure 8-18:
Mean Step Differences between 4x2 Configurations
as Relative Percentage Differences

The step differences between configurations of the same type are small, averaging 3.46% between pair-type configurations and 5.70% between loner-type configurations. The step differences between the best of one configuration type and the worst of another are much larger: 32.49% between pairs and loners, 49.28% between loners and TOGETHER. These relationships are illustrated in Figure 8-19.

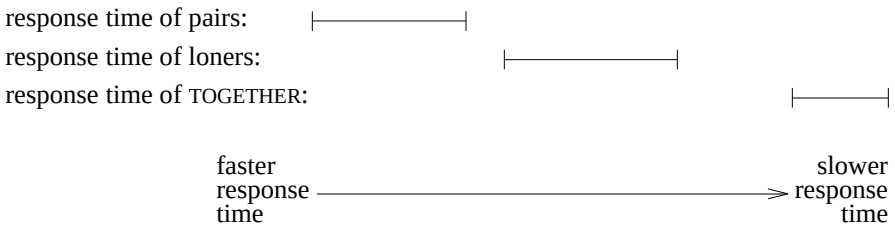

Figure 8-19:
Response Time Ranges of Pairs, Loners, and TOGETHER

The large gaps between the response times of the pairs, loners, and TOGETHER lends strong support to the hypothesis about evenly distributing CPU demand.

8-6.4. Summary

We have seen that in a 4x2 situation, the fastest configuration will be a pair-type configuration; in fact, the fastest three 4x2 configurations will be the three pair-type configurations, with about 3.46% difference between them. The four loner-type configurations will be next fastest, the best of them being 32.49% slower than the slowest pair. This is in accord with the hypothesis about spreading out the CPU demand as much and as evenly as possible.

In the next section we explore this hypothesis more directly.

8-7. The CR-Even Approach

8-7.1. Introduction, Background, and Review

We now turn to the question of how to select the best configuration out of a group of the same type of configurations, that is, how to find the fastest of the triple-type configurations, the fastest of the pair-type configurations, and the fastest of the loner-type configurations. The best 4x3 configuration will be one of the triple-type configurations, the best 4x2 configuration will be a pair, and the best 3x2 configuration will be a loner; if we can identify the fastest triple, pair, and loner, we will be able to select

the best available configuration in those situations.

In this section we explore a technique for selecting the best configuration and predicting the response time order of configurations based on evenly distributing the CPU service requirements, the *CR-even* technique. After explaining the hypothesis on which the technique is founded and the method of computing results, we present the all-hit and the best-hit results, that is, how successful CR-even is at predicting the complete response time order of configurations and how successful it is at selecting the best configuration, respectively. In addition to using the CR-even technique on the four-process triples, pairs, and loners, we also try it on all 4x4 configurations taken together, all 4x3 configurations, and all 4x2 configurations. We then discuss two systematic patterns of errors in the predictions and their implications, and we investigate how the size of the CPU demand differences effects the CR-even predictions. In the final section we draw some conclusions about the CR-even technique.

Recall that for three-process full-grid set U we achieved up to 83% success rates at selecting the best 2:1 loner configuration by using the matching technique and the variable/direction ‘‘most CPU demand’’ (see Section 7-3.6.1 and Figure 7-34). Matching on most CPU demand involves identifying the process with the largest CPU demand; that process is chosen as the loner process, and the other two processes are collocated. For example, if process 1 has the largest CPU demand, then the configuration selected is 1/02. Matching on other variable/directions had lower success rates.

Recall also that in these experiments, all processes in a particular trace are quite similar, so the differences in CPU demand between one process and another are small. If the processes in real-world distributed programs are less similar to each other than the processes in these experiments, then the success rates of matching and related techniques should be even higher than the 83% success rate we found.

8-7.2. Hypothesis

We hypothesize that the best configuration will spread the CPU demand across as many processors as possible, and that the configuration in which the CPU demand on each CPU is most nearly equal will be the best configuration.

This hypothesis (first mentioned in Section 8-5.2.1) is based on previous results in 3x2, 4x3, and 4x2. We believe that the reason for most CPU matching’s success in 3x2 situations is that it spreads the CPU demand across the two available processors as evenly as possible: putting the process with the largest CPU demand alone on a CPU leaves the other two processes, with smaller CPU demands, on the other CPU, which is the most even way to distribute the CPU demand. Triple-type configurations are the best 4x3 configurations because using three processors spreads out the CPU demand more than any way of using two processors, and 2:2 pair-type configurations distribute the CPU demand more evenly than 3:1 loners can in 4x2 situations.

If this hypothesis is correct, the best four-process 3:1 loner-type configuration can be selected by choosing the process with the largest CPU demand to be the loner process. As in 3x2, that would spread the CPU demand across the two processors as evenly as possible (although with one process on one processor and three on the other the CPU demand will tend to be less evenly spread out than with one on one and two on the other).

The hypothesis could also explain the overlap in performance of 2:2 pair-type configurations and 2:1:1 triple-type configurations, although we have not verified it. Some triple configurations cluster high-CPU demand processes together, while some pair configurations split them up. For example, if processes 0 and 1 have high CPU demands while processes 2 and 3 have low demands, then the triple 01/2/3 clusters the high CPU demand processes, while the pair 02/13 spreads out the high CPU demand processes.

8-7.3. Computing CPU Demand Differences

In each run, all 15 4x4 configurations are sorted into order from fastest simulated response time to slowest. The 14 4x3 configurations are similarly ordered from fastest to slowest (the same simulations, but excluding APART), as are the 4x2 configurations (again the same simulations, but excluding APART and the triples). Separate orderings of the triple-type, pair-type, and loner-type configurations are also made. Thus, for each run, there are six lists of configurations ordered by response times.

The *difference in CPU demand* in our experiments was computed for each configuration in the following manner. The total CPU service demand for each processor was found from the simulation output, and the difference in CPU demand between each pair of processors was computed. The largest of these differences was taken as the difference in CPU demand of the configuration. (No check was made for ties, equally large differences.) This is equivalent to finding the CPU with the largest total CPU demand and the CPU with the smallest CPU demand for each configuration, and computing the difference.

In each run, the differences in CPU demand for each configuration are sorted into order from smallest to largest. The hypothesis says that the configuration with the smallest CPU demand difference should be the fastest configuration, and that the configuration with the largest CPU demand difference should be the slowest configuration. As with the response time orderings, six ordered lists are produced, each in order from smallest CPU demand difference to largest.

The response-time-ordered list and the CPU-demand-difference-ordered list for each run are then compared, entry by entry. If the entries match, it is a ‘‘hit,’’ if they do not match it is a ‘‘miss.’’ For example, the response time and CPU difference lists, and the hit or miss outcomes, for 4x4, 4x2, and triples in run **aw-2** are shown in Figure 8-20.

List		Response Time Order	CPU Demand Diff. Order	Hit or Miss
4x4	1	APART	PAIR01	—
	2	TRIPLE03	PAIR02	—
	3	TRIPLE23	APART	—
	4	TRIPLE02	PAIR03	—
	5	TRIPLE01	TRIPLE03	—
	6	TRIPLE13	TRIPLE02	—
	7	PAIR01	TRIPLE01	—
	8	PAIR02	TRIPLE23	—
	9	TRIPLE12	TRIPLE13	—
	10	PAIR03	TRIPLE12	—
	11	LONER1	LONER1	hit
	12	LONER2	LONER2	hit
	13	LONER3	LONER3	hit
	14	LONER0	LONER0	hit
	15	TOGETHER	TOGETHER	hit
4x2	1	PAIR01	PAIR01	hit
	2	PAIR02	PAIR02	hit
	3	PAIR03	PAIR03	hit
	4	LONER1	LONER1	hit
	5	LONER2	LONER2	hit
	6	LONER3	LONER3	hit
	7	LONER0	LONER0	hit
	8	TOGETHER	TOGETHER	hit
triples	1	TRIPLE03	TRIPLE03	hit
	2	TRIPLE23	TRIPLE02	—
	3	TRIPLE02	TRIPLE01	—
	4	TRIPLE01	TRIPLE23	—
	5	TRIPLE13	TRIPLE13	hit
	6	TRIPLE12	TRIPLE12	hit

Figure 8-20:
CR-Even Hits and Misses for **aw-2**
Hits are labelled ‘‘hit,’’ misses are labelled ‘‘—.’’

The hit-or-miss results are analyzed in a number of ways in the following sections.

8-7.4. All-Hit Counts

The first approach to evaluating the success of the CR-even technique is to count the number of hits in each pair of lists. We call this the *all-hit* result; it shows how close CR-even comes to correctly predicting the entire response time order of configurations. We first discuss the all-hits results summarized over all runs, then the results in individual runs. Lastly, we compare the all-hit success of CR-even with the success of random selection.

The number of hits in each pair of lists, summed over all runs in set A, is shown in Figure 8-21. There are 28 runs in set A, and 15 4x4 configurations per run, so for 4x4 there are a total of 420 configurations; for 122 of them the two lists match, a hit. The number of configurations and the number of hits is computed similarly for each of the other lists.

CR-Even All-Hits			
List	Total # of Configs.	# of Hits	Percent Hits
4x4	420	122	29.05%
4x3	392	118	30.10%
4x2	224	164	73.21%
triples	168	100	59.52%
pairs	84	46	54.76%
loners	112	90	80.36%

Figure 8-21:
All-Hit Count Sums in Set A

The success rates for 4x4 and 4x3 are very low, for triples and pairs they are moderately low, and for 4x2 and loners they are moderately high. We had not expected great success for the mixed-type lists (4x4, 4x3, and 4x2), but we expected better success for the single-type lists (triples, pairs, and loners).

We next examine the number of all-hits in each individual run. The number of hits can be very different in the two repetitions of a case, for example, **ae-1** has six triple hits while **ae-2** has two triple hits, or they can be the same, as in **az** where both repetitions have six triple hits.

We displayed the number of hits in each run on the grid in order to search for patterns in the distribution of higher and lower numbers of hits. Figures 8-22 and 8-23 show the grids for the 4x4 and triples lists.

	IO-COMM									
CPU	IO 0	COMM 100	IO 25	COMM 75	IO 50	COMM 50	IO 75	COMM 25	IO 100	COMM 0
0	**aa** 4	5			**ac** 5	3			**ae** 5	4
20			**ag** 2	5			**ai** 4	4		
40	**ak** 5	3			**am** 3	3			**ao** 5	3
60			**aq** 5	5			**as** 4	5		
80	**au** 5	5			**aw** 5	5			**ay** 5	5
100	**az** 5	5								

Figure 8-22:
CR-Even: Number of 4x4 All-Hits in Set A
Number of Hits Per Run Out of 15

	IO-COMM									
CPU	IO 0	COMM 100	IO 25	COMM 75	IO 50	COMM 50	IO 75	COMM 25	IO 100	COMM 0
0	**aa** 0	1			**ac** 4	2			**ae** 6	2
20			**ag** 4	4			**ai** 6	2		
40	**ak** 4	1			**am** 1	4			**ao** 4	6
60			**aq** 4	6			**as** 0	2		
80	**au** 3	3			**aw** 4	3			**ay** 6	6
100	**az** 6	6								

Figure 8-23:
CR-Even: Number of Triples All-Hits in Set A
Number of Hits Per Run Out of 6

There do not appear to be any strong patterns in the grid. For the most part, there are fairly good results along the right edge and the bottom of the grid, but not always.

In each list, only certain numbers of hits occur, as shown in Figure 8-24.

Numbers of Hits Which Occur		
List	Numbers of Hits Which Occur	Number of Hits Possible
4x4	2, 3, 4, 5	15
4x3	2, 3, 4, 5	14
4x2	3, 4, 6, 8	8
triples	0, 1, 2, 3, 4, 6	6
pairs	0, 1, 3	3
loners	2, 4	4

Figure 8-24:
Numbers of Hits which Occur in Each CR-Even List

The chance of randomly selecting a particular configuration for a particular position in the order, out of a list of C configurations, is $1/C$.† If H is the percentage of hits by the CR-even technique in a certain list, the ratio $H/(1/C) = HC$ compares the success of CR-even with the success of random selection, showing how many times more successful than random the CR-even technique is.

Figure 8-21, above, shows H for each of the six lists. In Figure 8-25, below, we have added columns showing C, $1/C$, and the ratio HC for each list.

† One may say there are $C!$ orders only one of which is the correct one, or that there is a $1/C$ chance of a hit in each of C list positions, so that the chance of a hit is $1/(C^C)$, but consider the following. In a list with three configurations (for example, the pairs list) there is one actual response time order and there is one CR-even order. There are six possible orders of three configurations, so by random selection there is a 1/6 chance of selecting exactly the actual order. But two of the six orders have PAIR01 in the first position, two have PAIR02, and two have PAIR03, so there is a 1/3 chance of a hit in the first position. Similarly, two of the six have PAIR01 in the 2nd position, and so on, so there is a 1/3 chance of a hit in the 2nd position, and similarly a 1/3 chance of a hit in the 3rd position. The mean of 1/3, 1/3, and 1/3 is 1/3.

CR-Even All-Hits						
List	# of Configs. in List	# of Hits in List	Percent Hits by CR-Even (H)	Configs. Per Run (C)	Random Success ($1/C$)	Times CR-Even Better Than Random ($H/(1/C) = HC$)
4x4	420	122	29.05%	15	6.67%	4.36
4x3	392	118	30.10%	14	7.14%	4.21
4x2	224	164	73.21%	8	12.50%	5.86
triples	168	100	59.52%	6	16.67%	3.57
pairs	84	46	54.76%	3	33.33%	1.64
loners	112	90	80.36%	4	25.00%	3.21

Figure 8-25:
Comparing the Success of CR-Even All-Hits with Random

The performance of CR-even is not as good as we had hoped, but it is always better than random, usually several times better. The lists with the lowest hit rates, 4x4 and 4x3, have some of the largest advantages over random. On the other hand, lists with high hit rates have both high and low *HC* ratios.

8-7.5. Best-Hit Counts

While we would like to be able to completely predict the response time order of each list with the CR-even technique, our main objective is to find the single best triple-type configuration, the best pair, and the best loner, and, if possible, the single best 4x4, 4x3, and 4x2 configurations. Although there are substantial numbers of misses using CR-even, it is possible that the misses are often lower down in the lists, and the success rates of selecting the best configuration in each list are higher.

For each run and each list, we checked whether the fastest configuration is also the configuration with the smallest difference in CPU demand between processes, in short, whether CR-even has a hit for the best configuration. We call this a *best-hit*. In this section we examine and discuss the best-hit results and compare them to the all-hit results.

Figure 8-26 shows the best-hits results for all six lists. There are 28 runs in set A, with one best configuration each, so the maximum possible number of hits in each list is 28.

CR-Even Best-Hits					
List	# of Hits	Percent Hits (H)	Configs. Per Run (C)	Random Success ($1/C$)	Times CR-Even Better Than Random ($H/(1/C) = HC$)
4x4	0	0.00%	15	6.67%	0
4x3	0	0.00%	14	7.14%	0
4x2	16	57.14%	8	12.50%	4.57
triples	18	64.29%	6	16.67%	3.86
pairs	16	57.14%	3	33.33%	1.71
loners	25	89.29%	4	25.00%	3.57

Figure 8-26:
Best-Hit Success Rates

For 4x4 and 4x3, the results are terrible. The configurations with the smallest CPU demand differences are always pairs, whereas the best 4x4 configuration is APART, and the best 4x3 configuration is always a triple (except in the upper left corner, where pair-type configurations are best for both 4x4 and 4x3; as it happens, though, not the same pair-type configurations that have the smallest CPU demand differences). The 4x2 and pair results are identical because the best 4x2 configuration is always a pair. Other than in the 4x4 and 4x3 lists, the CR-even results are several times better than random.

Figure 8-27 compares the best-hit success rates to the all-hit success rates.

List	Percent Hits			Times Better Than Random		
	Best Hits	All Hits	Difference	Best Hits	All Hits	Difference
4x4	0.00%	29.05%	-29.05%	0.00	4.36	-4.36
4x3	0.00%	30.10%	-30.10%	0.00	4.21	-4.21
4x2	57.14%	73.21%	-16.07%	4.57	5.86	-1.29
triples	64.29%	59.52%	4.77%	3.86	3.57	0.29
pairs	57.14%	54.76%	2.38%	1.71	1.64	0.07
loners	89.29%	80.36%	8.93%	3.57	3.21	0.36

Figure 8-27:
Comparing Best-Hits to All-Hits

For groups of configurations of the same type (triples, pairs, and loners) the CR-even success rates are moderately high to fairly high, about 57% to about 89%; these success rates are about 1.7 to about 3.9 times better than random. For some groups of configurations of mixed types (4x4 and 4x3) the success rates are very low, from zero to about 30%; the zero success rates are worse than random, but 30% successes is 4.2-4.4 times better than random. The 4x2 group results are quite similar to the pairs group results.

For single-type groups, the best-hit success rates and *HC* ratios are slightly better than the all-hit rates; for mixed-type groups, best-hit is worse than all-hit.

As with all-hits, we wondered whether there was a pattern in the way best-hit successes and failures were distributed on the grid.

In the loner best-hit grid, there seems to be a weak area along the top and pushing down into the center. Other than that, no patterns are apparent, either within grids or between them.

8-7.6. Error Patterns in CR-Even

Is there any pattern to the way CR-even misses? When the *i*-th configuration on the response-time-ordered list is not the same as the *i*-th configuration on the list ordered by CPU demand differences, where in the CPU-demand-difference list is the matching configuration? One would expect that the non-matching entries would be scattered about with no order to them. We examined the CR-even misses for triple-type configurations, and it turns out there are two patterns.

Figure 8-28 shows the triples lists and the hit-or-miss outcomes for run **am-1**, which illustrates both patterns of misses.

Run	List		Order According to Response Time	Order According to Diffs. in CPU Demand	Hit or Miss
am-1	triples	1	TRIPLE03	TRIPLE01	—
		2	TRIPLE01	TRIPLE03	—
		3	TRIPLE13	TRIPLE13	hit
		4	TRIPLE12	TRIPLE02	—
		5	TRIPLE23	TRIPLE12	—
		6	TRIPLE02	TRIPLE23	—

Figure 8-28:
Triple-type Configurations in Run **am-1**

In the first pattern, two adjacent entries swap places, causing two misses. Entries 1 and 2 in the figure are an example of this pattern. In the other pattern, an entry moves up or down by two places, causing three misses. Entries 4, 5, and 6 in the figure are an example of this pattern: TRIPLE02 moves from position 6 to

position 4, pushing TRIPLE12 and TRIPLE23 down one position each.

In the 28 runs in set A there are 22 instances of the first pattern (adjacent swap) and four instances of the second pattern (move by two positions). Several runs exhibit a number of occurrences of one or both patterns. There are only three runs in which combinations of the two patterns do not completely account for all the misses: **aa-1**, **aa-2**, and **au-2**. Except for those three runs, all CR-even misses in the triples lists are off by at most two positions from their correct positions, and in most cases are off by only one position. If this is true of the other lists as well, it implies that, although the hit counts and hit percentages are fairly low, the error in using the configurations selected by CR-even might be small.

8-7.7. Sizes of CR Differences

The CR-even hit rates are rather lower for pair-type configurations than they are for triples and loners. We hypothesize that the CPU demand differences of the pair-type configurations are smaller than those for triple and loner configurations, and that these small differences are the cause of CR-even's poor performance for pairs. In these experiments, all processes in a trace are similar but somewhat different. In pair-type configurations, with two processes on each of two processors, the variations in processes on the same processor could tend to cancel each other out, leading to small differences in CPU demand on the processors.

To verify this hypothesis, we computed summary statistics for the CPU demand differences for each type of configuration separately; these summary statistics are shown in Figure 8-29. Since TOGETHER uses only one processor, there are no CPU demand differences for TOGETHER.

In each run, the configurations are ordered from smallest maximum CPU demand difference to largest, and each configuration is given an order number between 1 (for the smallest difference) and 15 (for the largest difference). We computed summary statistics for these order numbers, for each type of configuration separately. If the mean order number for one type of configuration were 1, it would mean that that type of configuration had, on average, the smallest CPU demand difference; if the mean order number were 6.5, it would mean that that type of configuration had, on average, a CPU demand difference that was between 6th and 7th smallest. The summary statistics for the order numbers are also shown in Figure 8-29.

CPU Demand Differences (ms)						
List	Arrangement	Number	Mean	Std. Dev.	Minimum	Maximum
APART	1:1:1:1	28	5,397	4,532	58	19,918
triples	2:1:1	168	19,455	14,469	353	55,165
pairs	2:2	84	4,242	4,548	2	25,940
loners	3:1	112	35,948	26,235	910	87,113

CPU Demand Difference Order						
List	Arrangement	Number	Mean	Std. Dev.	Minimum	Maximum
APART	1:1:1:1	28	3.18	0.38	3	4
triples	2:1:1	168	7.49	1.94	2	11
pairs	2:2	84	2.43	1.39	1	6
loners	3:1	112	12.39	1.35	8	14

Figure 8-29:
CPU Demand Differences and Difference Orders
by Configuration Type

The APART and pairs lists have the smallest CPU demand differences, triples are next smallest, and loners are largest. The differences between the lists are substantial compared to the mean response time in

set A of 48,051 ms. In APART and pairs, all processors run the same number of processes; in triples, one processor runs one more process than the others; and in loners, one processor runs two more processes than the others. This explains the sizes of the CPU demand differences.

The CPU demand differences of pair-type configurations are much smaller than those for triples or loners, which supports the hypothesis that CR-even has less predictive power when the differences in CPU demand between processors are small. The hypothesis is further supported by the fact that CR-even is least successful for pairs, next most successful for triples, and most successful for loners. The order matches the order of sizes of CPU demand difference from smallest to largest. The size of the CPU demand difference does determine the predictive power, and hence the usefulness, of CR-even.

The CPU demand differences of the pair-type configurations are even smaller than that of APART. This confirms the conjecture that variations in resource usages of the processes sharing a processor tend to cancel each other out.

8-7.8. Summary

We have seen that using the differences in CPU demand between processors to predict the full response time order, the CR-even technique, is generally not greatly successful. For the loner configurations only, CR-even is 80% successful; for all 4x2 configurations, 73% successful; the other all-hit success rates, triples, pairs, all 4x4 configurations, and all 4x3 configurations, are all less than 60%. All the success rates are better than randomly selecting configurations, usually 3-4 times better.

Some of the best-hit rates are better: the success rate for loner is 89%, for triples, 64%, and for pairs and 4x2, 57%. For 4x4 and 4x3, however, the best-hit success rates are zero. Except for 4x4 and 4x3, the success rates are again 3-4 times better than random, except pairs, for which CR-even is 1.7 times better than random.

For triple-type configurations, when CR-even misses, it is usually off by only one position in the order, sometimes two positions. It is very rarely off by more than two positions. This implies that, although the CR-even hit counts and hit percentages are not as high as we would like, the error in using the configurations selected by CR-even might be small.

The predictive power of CR-even depends on the size of the CPU demand differences of the configurations between which it is choosing. CR-even's predictive power is low for choosing the fastest 2:2 pair-type configuration, higher for the 2:1:1 triples, and highest for the 3:1 loners. This is the same order as the sizes of the CPU demand differences for different types of configurations.

The CPU demand difference is even lower for pairs than it is for APART, confirming our hypothesis that the two processes sharing processors in pair-type configurations tend to even out each others' differences.

8-8. Marginal Advantage of Adding One Processor

8-8.1. Introduction

We have shown that the best configurations use all available processors, and that performance, measured by response time, improves as the number of available processors increases (up to the number of processes). In this section we explore how much the response time improves as each additional processor is made available. In other words, what is the *marginal advantage* of adding one more processor? To answer this question we compare the performance of the 4@4, 4@3, 4@2, and 4@1 configurations, and compute the differences between their response times.

There is only one 4@1 configuration, TOGETHER, but there are several 4@2 configurations (see Figure 8-1). We consider three ways of computing the marginal advantage of adding a second processor:

compare the performance of TOGETHER to the performance of the best 4@2 configuration, to the worst 4@2 configuration, or to the mean performance of the 4@2 configurations.

There are several 4@3 configurations; so, in computing the marginal advantage of adding a third processor, there are nine ways of comparing best, mean, and worst; these are listed in Figure 8-30.

compare	best 4@2	to	best 4@3
compare	best 4@2	to	mean 4@3
compare	best 4@2	to	worst 4@3
compare	mean 4@2	to	best 4@3
compare	mean 4@2	to	mean 4@3
compare	mean 4@2	to	worst 4@3
compare	worst 4@2	to	best 4@3
compare	worst 4@2	to	mean 4@3
compare	worst 4@2	to	worst 4@3

Figure 8-30:
Margins Between 4@2 and 4@3

For 4@4 there is again only one configuration, APART, so there are only three ways to compute the marginal advantage of adding a fourth processor, as shown in Figure 8-31.

compare	best 4@3	to	only 4@4
compare	mean 4@3	to	only 4@4
compare	worst 4@3	to	only 4@4

Figure 8-31:
Margins Between 4@3 and 4@4

There are a total of 3+9+3=15 different margins.

While computing margins, it is also convenient to examine the margins between the two types of 4@2 configurations, pairs and loners. There are a number of pair-type configurations and a number of loner-type configurations, so there are nine ways of comparing best, mean, and worst performance; these nine combinations are listed in Figure 8-32.

compare	best loner	to	best pair
compare	best loner	to	mean pair
compare	best loner	to	worst pair
compare	mean loner	to	best pair
compare	mean loner	to	mean pair
compare	mean loner	to	worst pair
compare	worst loner	to	best pair
compare	worst loner	to	mean pair
compare	worst loner	to	worst pair

Figure 8-32:
Margins Between Pair-Type and Loner-Type Configurations

The loners and pairs are compared only among themselves.

8-8.2. Computing Marginal Advantages

For each run, all 15 margins are computed, as well as the nine pair/loner margins. Each margin is expressed in several ways: as an absolute number of milliseconds, as a percentage relative to the response time with the smaller number of processors, as a percentage relative to the response time with the larger number of processors, and as a percentage relative to the APART response time.

The first of these percentages is the improvement in response time as a processor is added, expressed as a percentage of the response time with the smaller number of processors. We call this the *percentage up.* A positive value means that the response time improves (gets smaller) as a processor is added; a negative value means the response time worsens (gets larger) as a processor is added.

The second percentage is the increase in response time when one processor is removed, expressed as a percentage of the response time with the larger number of processors. We call this the *percentage down.* A positive value means that the response time worsens (gets larger) as a processor is removed; a negative value means that the response time improves (gets smaller) as a processor is removed.

The percentages relative to APART are computed as a standard, so that one margin can be compared to another. We used the response time of the APART configuration as the standard because in almost all cases it is the desired, fastest, response time. The response time of APART is smaller than some of the margins, so some of the percentages relative to APART are greater than 100%.

8-8.3. Mean Margins Over Set A

We computed the mean, over all 28 runs in set A, of each margin. The margins in runs **aa-1** and **aa-2** are very different from the means. An example is shown in Figure 8-33, comparing a few of the **aa-1** and **aa-2** margins with the mean margins excluding those runs.

aa-1			
Margin	Percent Up	Percent Down	Percent Rel. APART
only 4@1 – best 4@2	25.94%	35.017%	31.62%
only 4@1 – mean 4@2	19.27%	23.863%	23.49%
only 4@1 – worst 4@2	14.70%	17.238%	17.92%

aa-2			
Margin	Percent Up	Percent Down	Percent Rel. APART
only 4@1 – best 4@2	423.74	21.09%	26.730%
only 4@1 – mean 4@2	332.60	16.56%	19.840%
only 4@1 – worst 4@2	250.08	12.45%	14.218%

Mean Excluding **aa-1** and **aa-2**			
Margin	Percent Up	Percent Down	Percent Rel. APART
only 4@1 – best 4@2	41.20%	72.10%	111.57%
only 4@1 – mean 4@2	30.33%	43.96%	81.55%
only 4@1 – worst 4@2	19.05%	23.76%	51.70%

Figure 8-33:
Comparing Example Margins in **aa-1** and **aa-2**
with Mean Margins

Because of this difference, we computed the means of the margins over all of set A and also the means over all of set A except **aa-1** and **aa-2**. Both sets of means are shown in Figure 8-34.

Margin	Percent Up		Percent Down		Percent Rel. APART	
	All	No-**aa**	All	No-**aa**	All	No-**aa**
only 4@1 – best 4@2	39.94	41.20	69.15	72.10	105.61	111.57
only 4@1 – mean 4@2	29.45	30.33	42.38	43.96	77.26	81.55
only 4@1 – worst 4@2	18.66	19.05	23.19	23.76	49.17	51.70
best 4@2 – best 4@3	11.25	12.11	13.10	14.11	16.21	17.45
best 4@2 – mean 4@3	4.91	5.56	5.35	6.02	7.01	7.80
best 4@2 – worst 4@3	-1.71	-1.50	-1.57	-1.36	-2.63	-2.52
mean 4@2 – best 4@3	24.80	26.18	33.73	35.76	44.57	47.48
mean 4@2 – mean 4@3	19.43	20.65	24.56	26.17	35.37	37.82
mean 4@2 – worst 4@3	13.82	14.68	16.40	17.45	25.73	27.50
worst 4@2 – best 4@3	34.65	36.43	54.82	58.03	72.65	77.32
worst 4@2 – mean 4@3	30.02	31.68	44.13	46.81	63.46	67.67
worst 4@2 – worst 4@3	25.21	26.57	34.60	36.63	53.81	57.35
best 4@3 – only 4@4	19.76	22.00	29.29	32.20	29.29	32.20
mean 4@3 – only 4@4	25.22	27.60	38.48	41.86	38.48	41.86
worst 4@3 – only 4@4	30.05	32.73	48.13	52.18	48.13	52.18
best loner – best pair	19.03	19.77	24.94	26.06	39.35	41.64
best loner – mean pair	17.29	17.99	22.29	23.32	36.19	38.34
best loner – worst pair	15.24	15.88	19.32	20.24	32.44	34.40
mean loner – best pair	22.68	23.62	30.49	31.93	47.26	50.06
mean loner – mean pair	21.02	21.92	27.72	29.06	44.10	46.76
mean loner – worst pair	19.07	19.92	24.60	25.82	40.35	42.82
worst loner – best pair	26.39	27.53	36.90	38.74	56.44	59.87
worst loner – mean pair	24.81	25.92	33.99	35.72	53.29	56.57
worst loner – worst pair	22.95	24.02	30.71	32.31	49.54	52.63

Figure 8-34:
Margins in Set A, With and Without Runs **aa-1** and **aa-2**

For the margins between 4@3 and 4@4, the percent down and percent relative to APART values are the same. This is because the only 4@4 configuration is APART, so the percent down results are computed relative to APART.

The two sets of means do not differ by much, usually by one or two percent, occasionally by two or three percent. The largest difference is six percent. The no-**aa** values are larger, that is, more positive. The response time orders of configurations for the **aa** runs are very different from the orders for the other runs. Rather than the response times of configurations using the same number of processors being clustered, with large gaps between clusters, as is usual, in **aa** they are mixed together; so, the margins for **aa-1** and **aa-2** are much smaller than those for other runs, and the means including them are smaller.

8-8.4. Analysis

In this section we examine the margins shown in Figure 8-34 in three ways: mean-to-mean, best-to-best, and best-to-worst.

The mean-to-mean margins, which compare the mean performance of all configurations with a certain number of processors to the mean performance of all configurations with one more processor, show the importance of each additional processor.

Our main objective is to find and use the best configuration of each type. Presuming we can do this, then the marginal advantage of adding a processor is to be found by comparing the best configuration with N processors to the best configuration with N+1 processors.

Also of interest is the marginal improvement from the best configuration with N processors to the worst configuration with N+1 processors. This margin is the smallest possible advantage, the minimum ‘‘guaranteed’’ benefit, of adding a processor.

8-8.4.1. Analysis of Mean-to-Mean Margins

Figure 8-35 shows the average mean-to-mean margins; this is a subset of the data presented in Figure 8-34, above.

All Runs			
Margin	Percent Up	Percent Down	Percent Relative to APART
only 4@1 – mean 4@2	29.45	42.38	77.26
mean 4@2 – mean 4@3	19.43	24.56	35.37
mean 4@3 – only 4@4	25.22	38.48	38.48
mean loner – mean pair	21.02	27.72	44.10

No-**aa**			
Margin	Percent Up	Percent Down	Percent Relative to APART
only 4@1 – mean 4@2	30.33	43.96	81.55
mean 4@2 – mean 4@3	20.65	26.17	37.82
mean 4@3 – only 4@4	27.60	41.86	41.86
mean loner – mean pair	21.92	29.06	46.76

Figure 8-35:
Average Mean-to-Mean Margins

The percent up varies from about 20% to about 30%; so, on average, adding a processor improves response time by 20-30%. This assumes that configurations are chosen randomly and uniformly, which of course will not be true. Similarly, the percent down, the response time penalty for giving up the use of one processor, averages about 25-45%.

Comparing margins using the percentage relative to APART, the biggest margin is between one processor and two; it is about twice as large as the other margins.

The margin between loner-type configurations and pair-type configurations is about 20-30%, depending on direction. Both loners and pairs are 4@2 configurations, so selecting the configuration to be used wisely can improve response time as much as adding a processor.

8-8.4.2. Analysis of Best-to-Best Margins

As mentioned above, we hope to be able to find the best configuration for each number of available processors; thus of great interest are the best-to-best margins, that is, the margin between the best configuration with a certain number of processors and the best configuration with one more processor. In this section we examine the best-to-best margins, first discussing the means of the margins, then displaying the margins for individual runs on the grid.

8-8.4.2.1. Average Best-to-Best Margins

Figure 8-36 shows the mean best-to-best margins; this is a subset of the data presented in Figure 8-34, above.

All Runs			
Margin	Percent Up	Percent Down	Percent Relative to APART
only 4@1 – best 4@2	39.94	69.15	105.61
best 4@2 – best 4@3	11.25	13.10	16.21
best 4@3 – only 4@4	19.76	29.29	29.29
best loner – best pair	19.03	24.94	39.35

No-**aa**			
Margin	Percent Up	Percent Down	Percent Relative to APART
only 4@1 – best 4@2	41.20	72.10	111.57
best 4@2 – best 4@3	12.11	14.11	17.45
best 4@3 – only 4@4	22.00	32.20	32.20
best loner – best pair	19.77	26.06	41.64

Figure 8-36:
Mean Best-to-Best Margins

The improvement in the response time from adding a second processor is quite substantial, about 40%. The improvement from adding a third processor is a much smaller but still significant 11%. The benefit of a fourth processor is intermediate at 21%.

The mean of the three no-**aa** best-to-best margins is 25.10%, so one could say that the mean benefit of adding a processor is a 25% improvement (reduction) in response time. Because of the large variance in the values, however, this is a statement to be made with a great degree of caution.

The pattern of the three averages is surprising, and displeasing. We had expected that either the best-to-best margins would all have similar values, or, more likely, that the margins would start large and decrease steadily, showing a diminishing return on adding processors. The reason for the ‘‘large, small, intermediate’’ pattern is not clear. Furthermore, the pattern does not allow us to predict by extrapolation what the margins might be for larger numbers of processes and processors. We might guess that there is a large oscillation in the margins as the first few processors are added, and that eventually the margins settle down to values close to the average (note that the best-4@2/best-4@3 margin of 22% is close to the average of 25%), but this is merely a supposition needing further confirmation.

8-8.4.2.2. Individual Best-to-Best Margins on the Grid

Having considered the mean best-to-best margins, we now examine the best-to-best margins for each run. Any patterns in the distribution of the values on the grid may lead us to a better understanding. The individual best-to-best margins for each run are displayed in Figure 8-37.

	IO-COMM									
	IO	COMM	IO	COMM	IO	COMM	IO	COMM	IO	COMM
CPU	0	100	25	75	50	50	75	25	100	0
0	**aa**				**ac**				**ae**	
	25.9	21.1			37.0	37.7			49.1	49.8
	0.0	0.0			13.7	13.5			8.3	5.5
	-10.8	-8.1			10.6	12.5			37.9	39.1
20			**ag**				**ai**			
			32.4	29.4			27.5	29.1		
			9.8	16.5			11.9	13.3		
			9.1	5.1			5.2	5.5		
40	**ak**				**am**				**ao**	
	39.4	39.7			36.3	39.4			45.6	45.1
	19.5	12.4			14.1	17.6			6.8	12.2
	14.0	19.4			11.2	14.7			28.9	23.1
60			**aq**				**as**			
			37.6	42.4			41.5	39.9		
			18.2	12.1			19.4	13.1		
			12.8	19.6			13.9	19.6		
80	**au**				**aw**				**ay**	
	44.3	44.9			43.2	41.7			49.3	49.4
	11.7	9.0			15.2	19.8			7.1	8.6
	28.3	31.7			23.2	15.0			39.4	37.7
100	**az**									
	49.6	49.8								
	4.0	1.7								
	45.9	48.7								

Figure 8-37:
Best-to-Best Percent-Up Margins in Set A

The left column in each cell is run 1, the right column is run 2. In each column, the top number is the only-4@1/best-4@2 margin, the middle number is the best-4@2/best-4@3 margin, and the bottom number is the best-4@3/only-4@4 margin.

At first glance, there do not appear to be any patterns in the grid of Figure 8-37. On closer examination, however, two patterns appear. One pattern is that the 4@1-4@2 and 4@3-4@4 margins are somewhat smaller in the upper, left, and center parts of the grid, and larger on the right and bottom. The second pattern appears in the 4@2-4@3 margins, and is the inverse of the first pattern, larger in the upper, left, and center parts of the grid and smaller on the right and bottom.

The first pattern makes sense. The right and bottom are regions of lower or no communication. Since remote communication takes longer than local communication, adding a processor would be less advantageous if that meant a lot of remote communication. Thus we expect the margins to be larger where there is little or no communication.

It is not clear why the 4@2-4@3 margin should have a different pattern than the other two margins, or why adding a third processor should be less advantageous where there is little communication.

The only negative best-to-best margins are in case **aa**. This means that (except possibly for some pathological programs falling into cell **aa**) it is always advantageous to add a processor, if you can find the best way to use it, that is, if you can select the best configuration.

In the lower left, the region into which we expect large, scientific programs to fall, the 4@1-4@2 and 4@3-4@4 margins are large: 30-50%. Since large, scientific programs are most likely to be parallelized, and are likely to have long run times, we are pleased to see that the marginal advantages of adding a processor are large for such programs, at least for two of the margins.

8-8.4.3. Analysis of Best-to-Worst Margins

The best-to-worst margin is the minimum improvement (reduction) in response time one could possibly obtain by adding a processor. Figure 8-38 shows the mean best-to-worst margins over all runs in set A; this is a subset of the data presented in Figure 8-34, above. Since there is only one 4@4 configuration, the best-to-best 4@3-4@4 margin and the best-to-worst 4@3-4@4 margin are the same.

All Runs			
Margin	Percent Up	Percent Down	Percent Relative to APART
only 4@1 – worst 4@2	18.66	23.19	49.17
best 4@2 – worst 4@3	-1.71	-1.57	-2.63
best 4@3 – only 4@4	19.76	29.29	29.29
best loner – worst pair	15.24	19.32	32.44

No-**aa**			
Margin	Percent Up	Percent Down	Percent Relative to APART
only 4@1 – worst 4@2	19.05	23.76	51.70
best 4@2 – worst 4@3	-1.50	-1.36	-2.52
best 4@3 – only 4@4	22.00	32.20	32.20
best loner – worst pair	15.88	20.24	34.40

Figure 8-38:
Mean Best-to-Worst Margins

The 4@1-4@2 and 4@3-4@4 best-to-worst margins are about 20%, which is the lower end of the range of mean-to-mean margins, 20-30%. The 4@2-4@3 best-to-worst margins are negative, reflecting the response time overlap of the 4@2 pair-type configurations and the 4@3 triples. This shows that it is possible to get worse response time when adding a processor, if the configurations are chosen poorly.

8-8.5. Summary

We have computed the mean-to-mean marginal advantages of adding one processor by comparing the mean response time of all configurations using some number of processors to the mean response time of all configurations using one more processor. The mean-to-mean margins show an improvement of about 20% to 30% in response time by adding one processor. The largest margin is going from one processor to two: it is about twice as large as the other margins.

The best-to-best margins, comparing the best response time using using some number of processors to the best response time using using one more processor, range from about 11% to about 40%. They average 25%, but they vary too much to make use of this figure. Disappointingly, the margins do not have a clear trend, they go from larger to smaller to intermediate.

The best-to-worst margins compare the best response time using some number of processors to the worst response time using one more processor. This is the minimum ‘‘guaranteed’’ benefit of adding a processor. When going from one to two processors, or three to four, the best-to-worst margins are about 20%. Between two and three processors, however, the mean margin is about -1.60%, an increase in the response time rather than a decrease. This reflects the overlapping response time ranges of triple- and pair-type configurations.

8-9. Maximum Number of Processes Collocated on a Processing Node

8-9.1. Introduction

In Section 7-2.2.5 we introduced the collocation number, the maximum number of processes collocated on a processing node, and discussed the relationship between response time and collocation number. In this section we examine response times and collocation numbers for all two-, three-, and four-process arrangements over the full grid.

8-9.2. Analysis

Figure 8-39 shows the mean response time of each arrangement over the full grid, grouped by collocation number.

Number of Processes	Experiment Set	Arrangement Label	Arrangement	Mean Response Time (ms)
1-Collocated Arrangements:				
2-processes	set N	APART	1:1	24,758.49
3-processes	set U	APART	1:1:1	26,165.56
4-processes	set A	APART	1:1:1:1	28,600.97
2-Collocated Arrangements:				
2-processes	set N	TOGETHER	2:0	37,643.79
3-processes	set U	loners	2:1:0	38,302.28
4-processes	set A	triples	2:1:1:0	41,275.79
4-processes	set A	pairs	2:2:0:0	44,520.36
3-Collocated Arrangements:				
3-processes	set U	TOGETHER	3:0:0	54,713.13
4-processes	set A	loners	3:1:0:0	58,545.34
4-Collocated Arrangements:				
4-processes	set A	TOGETHER	4:0:0:0	76,761.15

Figure 8-39:
Mean Response Times of Arrangements
Over the Full Grid
By Collocation Number

The response times of the 1-collocated arrangements are similar, as are those of the 2-collocated arrangements, the 3-collocated arrangements, and (trivially) of the 4-collocated arrangement. Furthermore, the groups of response times do not overlap, and are well-separated from each other.

Figure 8-40 shows the summary statistics of the mean response times shown in Figure 8-39.

Collocation Number	Number of Arrangements	Response Time (ms)			
		Mean	Std. Dev.	Minimum	Maximum
1-collocated	3	26,508.34	1,587.30	24,758.49	28,600.97
2-collocated	4	40,435.56	2,726.56	37,643.79	44,520.36
3-collocated	2	56,629.24	1,916.11	54,713.13	58,545.34
4-collocated	1	76,761.15	0.00	76,761.15	76,761.15

Figure 8-40:
Means of Mean Response Times of Arrangements
By Collocation Number
Over the Full Grid

The means of the mean response times are clearly quite different. The mean of the mean response times of the 2-collocated arrangements is 52.54% larger than that of the 1-collocated arrangements; that of the 3-collocated is 40.05% larger than that of the 2-collocated arrangements; that of the 4-collocated is 35.55% larger than that of the 3-collocated arrangements.

8-9.3. Conclusions

Arrangements with the same collocation number have similar response times, on average over the full grid, even when the number of processes and the number of processing nodes are different. The smaller the maximum number of processes on any node, the faster the mean response time. This does not mean that the collocation number is sufficient information to predict response time; programs in different part of the full grid have very different response times. It does mean that, for a fixed number of processes, using more processing nodes and so reducing the collocation number will lead to a faster response time; in other words, on average it is good to spread the processes out as much as possible.

All processes in a trace are similar, so the collocation number is a crude measure of maximum CPU demand. The similarity of the average response times of arrangements from different experiment sets which have the same collocation number further supports the idea that CPU demand determines response time.

Mean response times do increase with increasing collocation numbers, but the increase is much less than linear. For example, the increase in mean response time between 1-collocated and 2-collocated arrangements is about 53%, not 100%; the increase between 1-collocated and 3-collocated arrangements is about 114%, not 200%; and the increase between 1-collocated and 4-collocated arrangements is about 190%, not 300%. This shows that, when processes share a processing node, they are able to overlap use of the facilities; for example, one process might be using the CPU while another is performing an I/O. The larger the number of processes sharing the node, the more they are able to overlap in this manner.

Although CPU demand is the strongest determinant of response time, communication does have an influence. The mean response time over the full grid of the two-process 2:0 TOGETHER configuration is faster than that of the three-process 2:1:0 loner configurations, though both types are 2-collocated. We believe the slowdown is due to additional synchronization delay, time spent waiting for messages from the third process. The four-process 2:1:1:0 triples and 2:2:0:0 pairs are all 2-collocated, but the mean response time of the triples is faster. We believe this is because the two processes which are spread out in the triple configurations send their messages earlier than when the processes are collocated, so other processes experience less synchronization delay. These results contradict Indurkhya, Stone, and Xi-Cheng [ISX86], and agree with Nicol's mathematical analysis, though not with the simulations Nicol reports [Nic89] (see Section 2-3.2).

8-10. Experiment Set B

8-10.1. Introduction, Objectives, and Background

Experiment set B is based on the zoomed-in grid, with four processes per trace. It is similar to experiment sets R and V, which are also based on the zoomed-in grid, but which have two and three processes per trace, respectively (see Section 8-3.3).

The only result needed from the set B simulations is to locate the transition zone boundary, that is, to identify the region in which APART is not always the best configuration. The transition zone boundary moved upward and to the left between two-process set R and three-process set V. We expect it to move further upward and to the left between set V and set B.

In sets R and V the transition zone boundary runs through the middle of the zoomed-in grid (see Section 6-2.3.1 and Figure 6-5 concerning set R, and Section 7-2.3.3 and Figure 7-12 concerning set V). Since we only need to find the boundary, we do not need to simulate all the cases in set B. We simulate cases **baa-bbd** (see Figure 8-41, below).

8-10.2. Wobble

The GENACTIONS program uses random numbers in order to generate different traces, otherwise it would generate exactly one trace for each set of parameters. This means, however, that the resulting traces do not have exactly the characteristics called for by the parameters. In particular, the actual characteristics can differ from the parameter-specified characteristics by several tenths of a percent, and in some cases by a percent or two, or even three. To correct this, we checked the difference between the called-for and the actual characteristics, the *wobble*. Traces that had a difference between specified and actual value in any characteristic greater than or equal to 1% were discarded, and a new trace generated. This procedure was repeated until no trace had a wobble in any characteristic of more than 1%.‡

8-10.3. Winning Configurations

Figure 8-41 shows the winning configurations for the runs in set B that were simulated.

‡ It might have been well to reduce wobble in this way for the other experiment sets, but we did not develop this procedure until experiment set B was generated.

	IO-COMM											
CPU	IO 0	COMM 100	IO 1	COMM 99	IO 2	COMM 98	IO 3	COMM 97	IO 4	COMM 96	IO 5	COMM 95
0	**baa** P03	T02	**bab** P03	P02	**bac** P03	T02	**bad** P02	T12	**bae** T13	T12	**baf** AP	T01
1	**bag** T13	P02	**bah** T01	T23	**bai** T23	AP	**baj** AP	T02	**bak** T12	AP	**bal** AP	AP
2	**bam** AP	T03	**ban** T02	AP	**bao** AP	T13	**bAP** AP	AP	**baq** AP	AP	**bar** AP	AP
3	**bas** AP	AP	**bat** AP	AP	**bau** AP	AP	**bav** AP	AP	**baw** AP	AP	**bax** AP	AP
4	**bay** AP	AP	**baz** AP	AP	**bba** AP	AP	**bbb** AP	AP	**bbc** AP	AP	**bbd** AP	AP
5												

Figure 8-41:
Winning Configurations in Set B

Key:

AP = APART
T01 = TRIPLE01; T02 = TRIPLE02; T03 = TRIPLE03
T12 = TRIPLE12; T13 = TRIPLE13; T23 = TRIPLE23
P01 = PAIR01; P02 = PAIR02; P03 = PAIR03

As in previous experiment sets, the transition zone is defined as consisting of those cells in which APART is not always the best configuration. The transition zone for set B is shown in Figure 8-42.

	IO-COMM											
CPU	IO 0	COMM 100	IO 1	COMM 99	IO 2	COMM 98	IO 3	COMM 97	IO 4	COMM 96	IO 5	COMM 95
0	**baa** P03	T02	**bab** P03	P02	**bac** P03	T02	**bad** P02	T12	**bae** T13	T12	**baf** AP	T01
1	**bag** T13	P02	**bah** T01	T23	**bai** T23	AP	**baj** AP	T02	**bak** T12	AP		
2	**bam** AP	T03	**ban** T02	AP	**bao** AP	T13						
3												
4												
5												

Figure 8-42:
The Transition Zone in Set B

The transition zone in set B is very nearly the same as in set V. The only difference is that **bak** is in the transition zone, whereas **vak** is not in it (see Figure 7-12). Even so, it is for only one of **bak**'s two repetitions that APART is not best.

Within the set B transition zone, APART is best in one of the two runs per case in those cases just inside the boundary; APART is best in neither of the two runs in those cases not on the boundary row. When APART is not the best configuration, one of the triple-type or pair-type configurations is best; neither a loner-type configuration nor TOGETHER is ever the best configuration.

In every run in set B, TOGETHER is the worst configuration. This is true even in the transition zone, even in **baa**.

8-10.4. Conclusions

We had hypothesized that the transition zone would continue to shrink as the number of processes continued to grow beyond three, and that the transition zone boundary would continue to move upward and to the left. We had hoped to find that the number, and hence the importance, of cases in which APART is not best was small and grew smaller as the number of processes grew larger.

Based on set B, it now appears that the transition zone might be of a fixed size. With only two processes it is a bit larger, but with more than two it is a standard size. The standard size, observed in sets V and B, includes 13-14 cells in the zoomed-in grid. If the full grid were drawn at the same resolution as the zoomed-in grid, it would have 1,000 cells. A transition zone of 13-14 cells represents about 1.4% of the cells. That is a very small fraction of the grid, even if real programs were uniformly distributed on the grid. We do not believe real programs are uniformly distributed on the grid; on the contrary, we believe that real programs fall in regions of the grid far away from the transition zone. So, even though it appears that the transition zone does not continue to shrink with larger numbers of processes, we believe that the transition zone can be disregarded with impunity; that is, we can still recommend the APART configuration for any NxN situation.

Another possibility is that, rather than the transition zone being of a standard, fixed size, the size of the transition zone does continue to shrink with larger numbers of processes, but quite slowly. It may be that between the three-process experiment set and the four-process set the shrinkage is too small to be detected.

8-11. Summary of Four-Process Work

In set A, based on the full grid, when there are four processors available, the APART configuration is always the fastest configuration, except for unrealistically communication-intensive programs in the extreme upper left of the grid. A placement policy of always using APART would be very successful, very simple to implement, and very fast to run.

The configuration in which a distributed program is run has a very strong effect on its response time, and the sizes of the differences between one configuration's response time and another's grow with the number of processes. With two processes, the mean difference between the fastest configuration's response time and that of the slowest is about 30%, with three processes it is about 98%, and with four processes it is about 152%; the largest four-process fastest/slowest difference is almost 300%. Even the difference between the fastest and the second-fastest four-process configurations' response times averages 30%.

The mean response times of the four-process configurations can be clustered, and the clusters correspond to the arrangements, with one exception. The APART arrangement is fastest, the next fastest cluster includes the triples and the pairs, followed by the loners, and the slowest cluster consists of TOGETHER. The differences in response times between one cluster and another are large; the differences in response times between configurations within the same cluster are much smaller. There is a similar clustering phenomenon with three-processes.

The response times of the triple- and pair-type configurations overlap partly, but not completely; the fastest three configurations of the pairs and triples group are always triples. There is no significant gap between the response times of the fastest triples and the rest of the triples and pairs. Even though the fastest configurations are triples, it is surprising that there are configurations using fewer processors (2:2 pairs) that are faster than configurations using more processors (2:1:1 triples).

When there is no communication between processes, so that response time depends only on the processes and the processors, not on a communication medium, and there is no waiting for messages, then certain triples will have exactly the same response time as certain pair configurations.

In all NxM situations studied, the fastest configurations use all available processors. This is true for all M≤N. In NxN situations (2x2, 3x3, 4x4), the fastest configuration is (nearly) always APART. In Nx(N-1) situations (3x2, 4x3), the fastest configuration is in an arrangement of the form 2:1:...:1. In the one Nx(N-2) situation studied, 4x2, the fastest configuration was a 2:2 pair.

We hypothesized that the response time of a configuration is determined principally by how much it spreads out the CPU demand of the processes, and how evenly it spreads out the demand. If the first part of this hypothesis is correct, it will indicate, for any NxM situation, which arrangement will contain the fastest configuration. The placement decision then only needs to select one of the configurations in that arrangement. The second part of the hypothesis indicates that the configuration which spreads the CPU demand across the available processors most evenly will be best.

The fastest configurations in any situation use as many processors as are available, spreading out the CPU demand over as many processors as possible, so the first part of the hypothesis seems to be correct. On the other hand, the triple and pair response times overlap, even though triples use three processors and pairs use only two.

The CR-even placement technique is based on the second part of the hypothesis. A measure of the evenness of CPU demand on processors, the maximum CPU demand difference, is computed for each configuration, and lists of configurations ordered by CPU demand difference are compared to lists ordered by response time. For triples and pairs, the success rates of CR-even at predicting the complete response time order are about 55-60%, for loners they are about 80%. These triple and loner rates are 3.2-3.6 times better than random selection of a configuration. The pair success rate is also better than random, but only 1.6 times better.

The success rates of CR-even at predicting the fastest configuration of each type are 57-89%. Again, for triples and loners, CR-even is 3.6-3.9 times better than randomly selecting a configuration, and for pairs it is only 1.7 times better.

The predictive power of CR-even depends on the size of the CPU demand differences of the configurations between which it is choosing. CR-even's predictive power is lowest for the pair-type configurations, which have the smallest CPU demand differences. The CPU demand differences in the pair-type configurations are even lower than those in APART; evidently, two processes sharing a processor tend to even out each other's variations.

When the CR-even order of triples and the response time order of triples do not match, the configurations are almost always off by one position because adjacent entries switched positions, for example, [1, 2, 3, 4, 5] compared to [1, 2, 4, 3, 5]. In a few instances, configurations are off by two positions because one entry moved by two positions, moving two other entries by one position each, for example, [1, 2, 5, 3, 4]. In only three runs do these two patterns fail to account for all the misses in the triples list. This suggests that, although the best-hit rates are not as high as we would like, the penalties for using the configurations selected by CR-even may be small, perhaps in the range 2-5%, the typical size of response time differences within an arrangement cluster.

We have computed the marginal advantages, the reduction in response time obtained by adding one processor. The mean-to-mean margins compare the mean response time of all configurations using some number of processors to the mean response time of all configurations using one more processor. The mean-to-mean margins are about 20% to 30%. The largest margin corresponds to going from one processor to two; and is about twice as large as the other margins.

The best-to-best margins, comparing the best response time using some number of processors to the best response time using one more processor, are the ones which should be used to compare the performance with one number of processors to that with another number of processors if it were possible to identify the best configurations using those numbers of processors. The best-to-best margins range from

about 11% to about 40%. Disappointingly, the margins do not have a clear trend, they go from larger to smaller to intermediate as the number of processors increases.

The best-to-worst margins compare the best response time using some number of processors to the worst response time using one more processor. This is the minimum ‘‘guaranteed’’ benefit of adding a processor. When going from one to two processors, or three to four, the best-to-worst margins are about 20%. Between two and three processors, however, the margin is about -1.60%, an increase in the response time rather than a decrease; this reflects the overlapping response time ranges of the triple- and pair-type configurations.

In set B, based on the zoomed-in grid, we found that the transition zone does not continue to shrink as the number of processes increases; at least, there is no shrinkage between the three-process and four-process transition zones. Nevertheless, the transition zone includes only 13-14 cells, less than 1.4% of the full grid, so we think that it can be ignored. More importantly, we do not believe that any real programs are likely to fall in the transition zone; we believe real distributed programs are most likely to fall in the lower left part of the full grid.