Competitive Dynamic Multiprocessor Allocation for Parallel Applications

Timothy Brecht, Xiaotie Deng, Nian Gu
Department of Computer Science
York University, Toronto, Ontario, Canada M3J 1P3
email: brecht@cs.yorku.ca / deng@cs.yorku.ca / gu@cs.yorku.ca

Abstract
In this paper we use competitive analysis to study preemptive multiprocessor allocation policies for parallel jobs whose execution time is not known to the scheduler at the time of scheduling. The objective is to minimize the makespan (i.e., the completion time of the last job to finish executing). We characterize a parallel job, $J_i$, by two parameters: its execution time, $l_i$, and its parallelism, $P_i$, which may vary over time. The preemption and reallocation of processors can take place at any time.

We devise a preemptive policy which achieves the best possible competitive ratio and then derive upper and lower bounds for scheduling $N$ parallel jobs on $P$ processors.

1. Introduction
A number of studies have applied competitive analysis to sequential job scheduling on multiprocessors when the scheduler has no a priori information about job arrivals or execution times [5][4][6][12]. In this paper we address the processor allocation problem for parallel jobs executing on multiprocessors.

We characterize a parallel job using a parallelism profile, which is defined as the number of processors an application is capable of using at any point in time during its execution [7][11]. That is, a job, $J_i$, is characterized by its execution time, $l_i$, and its parallelism, $P_i$, which may vary with time. If the parallelism of an application varies with time during its execution, its parallelism profile is said to have multiple phases as shown in Figure 1. If the parallelism does not change during job execution, the parallelism profile is said to have a single phase. Our research differs from previous studies by Turek, et al. [15][16][17] in that (a) the exact length of the profile (which is equivalent to the execution time of the job) is not known to the scheduler a priori and (b) the parallelism may vary with time, and the variations in parallelism are not known to the scheduler before the job reaches the point at which it changes parallelism. In other words, the instantaneous parallelism of jobs is the only information available to the system when making scheduling decisions. We allow processors to be preempted and reallocated during the execution of a job and assume that the cost of preemption is negligible. We assume that the execution time of a job does not decrease if it is allocated more processors than its parallelism. Therefore, no job should be allocated more processors than its parallelism. However, when less than $P_i$ processors are allocated to job $J_i$, we assume that the job’s execution will be prolonged proportionally.

We first construct an optimal competitive policy using competitive analysis for scheduling two parallel jobs without a priori information about job execution.
times. Then we compare our optimal competitive policy with the Dynamic Equipartition policy (abbreviated DEQ in this paper), which is reported to perform very well in simulation and experimental studies [9][18][8][10][1] and is considered to possess the desired properties of a good scheduler on multiprocessors [9][8]. DEQ is first introduced to parallel scheduling by Tucker and Gupta as a process control policy [14], and modified by Zahorjan and McCann [18][10]. The main idea behind this approach is to distribute processors evenly provided they have sufficient parallelism. Our result shows that the competitive ratio of DEQ is very close to that of the optimal policy we devise for two jobs.

The competitive analysis of algorithms is an approach to studying algorithms operating with incomplete information, first introduced by Sleator and Tarjan [13] in the study of a system memory management problem. Let $M_A(JS)$ be the makespan for a scheduling algorithm $A$ on job set, $JS$, and $Opt(JS)$ be the makespan for the optimal algorithm which has complete information about the jobs. The competitive ratio of algorithm $A$ is defined as $\max_{all JS} \left( \frac{M_A(JS)}{Opt(JS)} \right)$. The goal is to find an algorithm which leads to the minimum competitive ratio.

We start our study by making the following two assumptions: (a) the parallelism profiles of all parallel jobs have only one phase, and (b) all jobs arrive at the system simultaneously. In Section 2, we obtain the optimal solution $Opt(JS)$ for scheduling jobs on $P$ processors, assuming that complete information about job execution time is known to the scheduler. In Section 3, we use competitive analysis to devise a policy with an optimal competitive ratio for scheduling two parallel jobs, whose parallelism does not change during execution. In Section 4, we study the problem of scheduling $N$ parallel jobs by generalizing our analysis in Section 3 and the results for scheduling sequential jobs on multiprocessors by Graham [5], Hall and Shmoys [6] and Shmoys, et al. [12]. We prove that the matched upper and lower bound is $2 - 1/P$, for scheduling $N$ single-phased parallel jobs on $P$ processors. Then we consider the cases where the job parallelism profiles have multiple phases and there are new job arrivals. We prove that the competitive ratio for scheduling $N$ parallel jobs with multi-phased parallelism profiles is $2 - 1/P$. As well, we prove that the competitive ratio for scheduling $N$ parallel jobs (with single or multiple phased parallelism profiles) with new arrivals is also $2 - 1/P$. In Section 5, we conclude our work.

2. Optimal scheduling with complete information

A scheduling policy that minimizes makespan will not leave processors idle if there exists a job in the system capable of using it. (Such a policy is also called a work-conserving policy). We start by considering the case when the job parallelism does not change during execution. Assume that $P_1$ is the parallelism of job $J_1$, and $P_2$ is the parallelism of job $J_2$. Let $l_1$ represent the time $J_1$ needs to complete its execution if it is allocated $P_1$ processors. Similarly, $l_2$ is the time required for $J_2$ to execute if allocated $P_2$ processors. There are $P$ processors in the system. The total amount of work to be executed by $J_1$ and $J_2$ can be denoted by $W_1 + W_2$ which is equal to $P_1l_1 + P_2l_2$. Let $p_i$ be the number of processors that are actually allocated to $J_i$.

**Theorem 1:** The optimal makespan for scheduling two parallel jobs on $P$ processors is

$$Opt(JS) = \max \left( l_1, l_2, \frac{P_1l_1 + P_2l_2}{P} \right)$$

**Proof:** Processor allocation is trivial when $P_1 + P_2 \leq P$: allocate $P_1$ processors to $J_1$. The theorem holds in this case. Therefore, we only need to prove the theorem when $P_1 + P_2 > P$. Without loss of generality, we assume that $P_1 \geq P_2$. We make use of the following three policies:

Least Parallelism First (LPF)

Allocate $\min(P, P_2)$ processors to $J_2$ and allocate the remaining to $J_1$. When one job finishes execution, allocate $\min(P, P_1)$ processors to the other job, where $P_i$ is the parallelism of the job which is still executing.

Most Parallelism First (MPF)

Allocate $\min(P, P_1)$ processors to $J_1$ and allocate the remaining to $J_2$. When one job finishes execution, allocate $\min(P, P_1)$ processors to the other job, where $P_i$ is the parallelism of the job which is still executing.

Dynamic Proportional Partition (DPP)

The number of processors allocated to each job is proportional to the job parallelism. That is, allocate $(P_1P)/(P_1 + P_2)$ processors to $J_1$, and allocate $(P_2P)/(P_1 + P_2)$ processors to $J_2$. After one job has finished execution, allocate $\min(P, P_1)$ processors to the remaining job.

LPF and MPF deal with two extreme situations, and therefore the allocation of processors using any other policy will lie between these two extremes. To obtain the optimal makespan we consider three possible relations among $P_1$, $P_2$, and $P$. 
The above analysis shows that max \( P_2 \leq P_1 < P \)
Obviously, neither \( J_1 \) nor \( J_2 \) can use all \( P \)
processors. The processor allocation varies with
the relation between \( l_1 \) and \( l_2 \). Therefore, we
consider the three possibilities in terms of \( l_1 \) and \( l_2 \).

(1a) \( l_1 = l_2 \)
Since \( P_1 + P_2 > P \), the processors allocated to
at least one of the jobs will be less than
its parallelism. DPP yields the optimal
makespan, since it keeps all processors busy,
and both jobs finish execution at the same
time. The makespan for DPP is \((P_1 l_1 + P_2 l_2)/P\).

(1b) \( l_1 < l_2 \)
In order to reduce the number of idle
processors during the execution of both
jobs, the initial allocation of \( J_2 \) should be
\( P_2 \), and therefore \( J_1 \) should be \( P - P_2 \). In
other words, the processors should be
initially allocated according to LPF until the
jobs reach a point at which the remaining
execution time of both the jobs are the same.
Once they reach that point, the remainder of
the execution is identical to case (1a).
Therefore, at that point the processors
should be allocated according to DPP so
that both jobs finish simultaneously.
The completion time is \((P_1 l_1 + P_2 l_2)/P\) if there is
work remained after using LPF.
Otherwise, \( J_1 \) will finish execution before
\( J_2 \) and the makespan will be \( l_2 \).

(1c) \( l_1 > l_2 \)
This case is symmetric to (1b). The initial
allocation of processors is done according to
MPF. Once the remaining execution time of
both jobs is equal, the processors should be
reallocated according to DPP.

(2) \( P_2 \leq P \leq P_1 \)
In this case LPF yields the optimal makespan regardless of the values of \( l_1 \) and \( l_2 \), because LPF
keeps all processors busy until both jobs have
finished execution. Therefore, the makespan is \((P_1 l_1 + P_2 l_2)/P\).

(3) \( P \leq P_2 \leq P_1 \)
In this case, any work-conserving policy yields the
optimal makespan, since either \( J_1 \) or \( J_2 \) by itself
can utilize all \( P \) processors. In other words, there
are no idle processors until both jobs have finished
execution. The makespan is \((P_1 l_1 + P_2 l_2)/P\).

The above analysis shows that max \( l_1, l_2, \frac{P_1 l_1 + P_2 l_2}{P} \)
is an upper bound on the makespan. When \( P_1 + P_2 \leq P \)
the makespan is \( \max(l_1, l_2) \geq \frac{P_1 l_1 + P_2 l_2}{P} \). For other
cases, we have:

Case (1a):
The makespan is \( \frac{P_1 l_1 + P_2 l_2}{P} \geq \max(l_1, l_2) \).

Case (1b):
Either case (1a) doesn’t occur and the makespan is:
\( l_2 \geq l_1, \ l_2 \geq \frac{P_1 l_1 + P_2 l_2}{P} \) or case (1a) does occur
and the makespan is \( \frac{P_1 l_1 + P_2 l_2}{P} \geq \max(l_1, l_2) \).

Case (1c):
This case is symmetric to case (1b).

Case (2):
Either \( J_1 \) finishes first and the makespan is:
\( l_2 \geq \max \left( \frac{P_1 l_1 + P_2 l_2}{P}, l_1 \right) \) or \( J_1 \) finishes first and
the makespan is \( \frac{P_1 l_1 + P_2 l_2}{P} \geq \max(l_1, l_2) \).

Case (3):
The makespan \( \frac{P_1 l_1 + P_2 l_2}{P} \geq \max(l_1, l_2) \).

It is not difficult to see that
\( \max(l_1, l_2, \frac{P_1 l_1 + P_2 l_2}{P}) \) is also a lower bound on the
makespan. Therefore, the optimal makespan for
scheduling two jobs is \( \max(l_1, l_2, \frac{P_1 l_1 + P_2 l_2}{P}) \).

This result can be extended to schedule \( N \) parallel
jobs on \( P \) processors.

**Theorem 2:** The optimal makespan for scheduling \( N \)
parallel jobs on \( P \) processors is

\[
\min \left( \left\{ l_1, l_2, \ldots, l_i, \frac{\sum_{i=1}^{N} P_i l_i}{P} \right\} \right)
\]

**Proof:** Combining the previous analysis in this section
and the optimal algorithm for scheduling sequential jobs
on multiprocessors by Shmoys, et al. [12], we construct
the following optimal algorithm for scheduling \( N \) parallel
jobs on \( P \) processors. Suppose that there are \( N \) parallel
jobs: \( (P_1, l_1), (P_2, l_2), \ldots, (P_N, l_N) \). Without loss of
generality, we assume that \( l_1 \leq l_2 \leq \cdots \leq l_N \). First, we
allocate processors to jobs according to the following
recursive rules:
(1) (Base case): $l_1 = l_2 = \cdots = l_N$. Allocate processors to jobs in proportion to their parallelism.

(2) (Recursive case): $l_m < l_{m+1} = l_{m+2} = \cdots = l_N$. If $\sum_{i=m+1}^{N} P_i < P$, allocate $P_i$ processors to $J_i$, where $m + 1 \leq i \leq N$. Then recursively allocate the remaining $P - \sum_{i=m+1}^{N} P_i$ processors to the remaining jobs. Otherwise, allocate $\frac{P_iP}{\sum_{i=m+1}^{N} P_i}$ processors to $J_i$, $m + 1 \leq i \leq N$.

This process continues until one of the following cases occurs:

(a) All applications have finished execution.

(b) One job finishes execution, after which the processors are reallocated according to the above rules.

(c) Some job $J_j$ reaches a point at which its remaining execution time becomes equal to that of one or more other jobs (which were not previously the same as $J_j$). Then we reallocate the processors using the above recursive rules.

The correctness follows easily by an inductive proof. □

3. Scheduling two jobs on $P$ processors

Without loss of generality, we assume that $P_1 \geq P_2$. The worst case competitive ratio for MPF occurs when $J_1$ finishes execution before $J_2$ does. After this point, $P - P_2$ processors will be idle. Similarly, the worst case for LPF occurs when $J_2$ finishes execution first, after which $P - P_1$ processors will be idle. Since there are more idle processors in the worst case using MPF than using LPF, the competitive ratio for MPF is larger than that of LPF. If we consider the number of processors allocated to $J_1$ (or $J_2$) to be a continuum, with the allocations of MPF and LPF being the two extremes, the policy that yields the smallest competitive ratio will be closer to LPF than to MPF. DPP can be viewed as a combination of LPF and MPF. Its worst case competitive ratio occurs when $J_1$ finishes execution first, since there will be more idle processors than the case when $J_2$ finishes first. The policy with the optimal competitive ratio will be a combination of LPF and DPP in the form of $\alpha$LPF + $(1 - \alpha)$DPP. Let $p_1$ be the number of processors allocated to $J_1$, and $p_2$ be the number of processors allocated to $J_2$. Then $p_1 = \alpha(P - P_2) + (1 - \alpha) \frac{P_1P}{P_1 + P_2}$, and $p_2 = \alpha P_2 + (1 - \alpha) \frac{P_2P}{P_1 + P_2}$. The policy that yields the minimum competitive ratio will yield the same competitive ratio no matter which job finishes execution first. We can thus determine $\alpha$ and the optimal policy accordingly.

Define $M_\alpha(JS)$ as the makespan of the combined policy, $Policy(\alpha)$, with parameter $\alpha$ on a job set $JS$. Its competitive ratio is $\max_{\alpha} \frac{M_\alpha(JS)}{Opt(JS)}$. If $J_1$ finishes first using $Policy(\alpha)$, the optimal allocation policy for the worst competitive ratio will be LPF because the number of idle processors is the smallest. That is, $Opt(JS) = l_2$. In this case, $l_1$ and $l_2$ satisfy the following relation: $l_2 = (P_1l_1)/(P - P_2)$. Denote the competitive ratio as $R_1(P_1, P_2, \alpha), 0 \leq \alpha \leq 1$. Similarly, $Opt(JS) = l_1$ if $J_2$ finishes first, where $l_1 = (P_2l_2)/(P - P_1)$. Denote the competitive ratio in this case as $R_2(P_1, P_2, \alpha), 0 \leq \alpha \leq 1$. When $R_1(P_1, P_2, \alpha) = R_2(P_1, P_2, \alpha)$, we can obtain a function of $\alpha(P_1, P_2)$ which yields the smallest competitive ratio.

![Figure 2: Finding $\alpha$ when $P_1 = 0.75$ and $P_2 = 0.6$.](image)

Let $P_1 = mP$, and $P_2 = nP$, $0 \leq m \leq 1$ and $0 \leq n \leq 1$. Then $R_1$ and $R_2$ can be represented as functions of $m, n$ and $\alpha$. The minimum competitive ratio is achieved by setting $R_1 = R_2$ which eliminates $\alpha$. The solution is presented in the appendix. Figure 2 illustrates how a proper $\alpha$ value is selected when $m = 0.75$ and $n = 0.6$. The two curves in Figure 2 are $R_1$ and $R_2$ respectively. The proper value of $\alpha$ corresponds to the point at which $R_1$ intersects with $R_2$. In this case, $\alpha$ is approximately 0.2.

Figure 3 illustrates the competitive ratio of the optimal policy, $R_1$ (note that $R_1 = R_2$), as a function of $m$ and $n$. It shows that, across all possible $m$ and $n$ where $0 \leq m, n \leq 1$, $R_1$ reaches the maximum of $4 - 2\sqrt{2}$ (approximately 1.175729) when $m = n = \sqrt{2}/2$. This
means that, in the worst case, the makespan for Policy(α) is within a constant of 1.175729 times the optimal (i.e., the competitive ratio of Policy(α) is 1.175729).

**Figure 3: Competitive ratio of OptComp**

As well, in the process of deriving this policy we have shown that 1.175729 is the best possible competitive ratio among all possible policies. That is, Policy(α) has the optimal competitive ratio. Therefore, we call it OptComp. (Note that OptComp is different from Opt(JS).) Opt(JS) refers to the policy which has the optimal makespan when it has information about the execution time of the jobs being executed, while OptComp refers to the policy which has the optimal competitive ratio when it does not know job execution times at the time of scheduling.

Now we compare OptComp with the dynamic equipartition (DEQ) policy. The purpose of this comparison is to find out the difference between DEQ and OptComp for scheduling two parallel jobs when the job execution time is not known to the scheduler a priori. We find that the competitive ratio for DEQ is the same as for OptComp when m = n = \( \sqrt{2}/2 \). For all other cases, the competitive ratio of DEQ is slightly larger than that of OptComp but never more than 4 – 2√2 = 1.175729. Taking the complexity of OptComp into consideration (see the appendix) one would likely prefer DEQ to OptComp.

### 4. Scheduling N jobs on P processors

In Section 3, we have devised an optimal competitive policy for scheduling two parallel jobs on P processors when the job execution time is not known to the scheduler a priori. In this section, we study the problem of scheduling N jobs on P processors in the same environment.

#### 4.1. N jobs with single-phased profiles

Let \( P_i \) denote the parallelism of \( J_i \), and let \( l_i \) denote the time it takes to execute \( J_i \) if it is allocated \( P_i \) processors, \( 1 \leq i \leq N \). We first assume that all N jobs arrive simultaneously, and there are no new arrivals. (We will relax these assumptions later.) We also assume that \( \sum_{i=1}^{N} P_i > P \). (It is trivial to schedule N jobs if \( \sum_{i=1}^{N} P_i \leq P \).)

Similar to studies for sequential job scheduling problems by Graham [5], Hall and Shmoys [6] and Shmoys, et al. [12], we have the following theorem.

**Theorem 3:** The competitive ratio for scheduling N jobs with single-phased parallelism profiles is \( 2 - 1/P \).

**Proof:** Suppose that all jobs arrive at the system at time \( t_0 \). Assume that \( J_i \) finishes at time \( t' \), which makes, for the first time, the sum of the parallelism of the remaining jobs in the system less than \( P \). Assume that the last job finishes execution at time \( t_N \). The execution of these N jobs can be divided into two parts. The first part is from time \( t_0 \) to \( t' \). The second part is from time \( t' \) to \( t_N \). Let \( \tau = t' - t_0 \) and \( \tau' = t_N - t' \). Therefore, the makespan is \( \tau + \tau' \). Let \( W_i \) be the amount of work done on job \( J_i \) up to time \( t' \), \( 1 \leq i \leq N \). Obviously this is no more than \( P l_i \), \( 1 \leq i \leq N \). Therefore, we have \( \tau \leq \frac{\sum_{i=1}^{N} W_i}{P} \leq \frac{\sum_{i=1}^{N} P l_i}{P} \).

Obviously, after time \( t' \), the number of processors allocated to each remaining job will be equal to its parallelism. Denote the remaining execution time of each job after \( t' \) with \( r_{t'} \). Then \( \tau' = \max( r_{t_1}, r_{t_2}, \ldots, r_{t_N} ) \)

Therefore, \( \tau + \tau' \leq \frac{\sum_{i=1}^{N} W_i}{P} + \max( r_{t_1}, r_{t_2}, \ldots, r_{t_N} ) \).

Let \( L_1 = \max( r_{t_1}, r_{t_2}, \ldots, r_{t_N} ) \). Since the minimum parallelism of a job is one, \( \sum_{i=1}^{N} P_i l_i \geq L_1 + \sum_{i=1}^{N} W_i \). It follows that \( \text{Opt(JS)} \geq x L_1 + y \geq ( x + \frac{y}{P} ) L_1 \) Since \( \tau + \tau' = L_1 + \frac{\sum_{i=1}^{N} W_i}{P}, \) we let \( y = \frac{1}{P} \) so that we can simplify the term \( \frac{\tau + \tau'}{\text{Opt(JS)}} \). Therefore, we have an upper bound on the competitive ratio: \( \frac{\tau + \tau'}{\text{Opt(JS)}} \leq 2 - 1/P \).

Shmoys, et al. [12] prove that a lower bound on the competitive ratio for scheduling N sequential jobs on P multiprocessors is \( 2 - 1/P \). Since scheduling sequential jobs can be viewed as a special case of scheduling parallel jobs, a lower bound on the competitive ratio for
scheduling \( N \) jobs on \( P \) processors is also \( 2 - 1/P \). Since both the upper and the lower bounds are \( 2 - 1/P \), the competitive ratio for scheduling \( N \) parallel jobs on \( P \) processors is \( 2 - 1/P \).

\[ \square \]

4.2. \( N \) jobs with multi-phased profiles

We have studied the problem of scheduling \( N \) parallel jobs, whose parallelism does not change during execution. In this section, we study the problem of scheduling parallel jobs with multi-phased parallelism profiles. The analysis is similar to that used to prove the upper bound on the competitive ratio for scheduling parallel jobs in Section 4.1. Without loss of generality, let \( J_i \) be the last completed job. We can divide the total execution time into two phases: Phase 1 is when the number of processors allocated to \( J_i \) is the same as its full parallelism. Denote the total length of time of periods of this type with \( \tau \). Phase 2 occurs when the number of processors allocated to \( J_i \) is smaller than its full parallelism. Denote the length of this time period with \( \tau' \). The fact that DEQ performs well in the case with two jobs is no coincidence. In subsequent work, we show that DEQ achieves the optimal competitive ratio for mean job response time [2].

Let the optimal completion time (using complete information) be \( OPT \). Then, we have

\[ OPT \geq t_0 + t_{\text{full}} + \frac{W_{\text{equi}} + W'}{P}. \]

Without loss of generality, we may assume that there is at least one job in the system between time zero and time \( t_0 \). Therefore, the total amount of work done during time interval \([0, t_0]\) is at least \( t_0 \). Let \( W' \) be the work done on the other jobs executed during the equipartition phase. The total amount of work done is at least \( t_0 + t_{\text{full}} + W_{\text{equi}} + W' \). Then, we have

\[ OPT \geq \frac{t_0 + t_{\text{full}} + W_{\text{equi}} + W'}{P}. \]

On the other hand, all of the jobs are executed by time \( t_0 + t_{\text{full}} + \frac{W_{\text{equi}} + W'}{P} \), according to DEQ. By a linear combination of the above inequalities for \( OPT \), it is easy to verify that the makespan for DEQ is bounded by \((2 - 1/P) OPT \).

5. Conclusions

In this paper we address the problem of scheduling parallel jobs on multiprocessors in order to minimize the makespan, when the scheduler does not have a priori information about job arrival times, execution times, or the variation in job parallelism. We use competitive analysis to devise an optimal policy for scheduling two parallel jobs on multiprocessors. In this case, DEQ produces a competitive ratio very close to that of the optimum. However, DEQ is much simpler than the complicated formula obtained in the appendix for the optimal competitive ratio in this case. Therefore, DEQ is a good policy to use for scheduling two jobs.

Following the work by Hall and Shmoys [6] and Shmoys, et al. [12] in sequential job scheduling, we generalize our result of scheduling two parallel jobs to scheduling \( N \) parallel jobs and prove that the upper and lower bounds for scheduling \( N \) parallel jobs is \( 2 - 1/P \). As well, we prove that the competitive ratio for scheduling parallel jobs is \( 2 - 1/P \) when the job parallelism changes and there are new arrivals.

The fact that DEQ performs well in the case with two jobs is no coincidence. In subsequent work, we show that DEQ achieves the optimal competitive ratio for mean job response time [2]. Mean response time is an important performance metric because it is of interest in general purpose multiprocessors and because makespan is unable to differentiate a number of scheduling policies
(e.g., any work-conserving policy achieves the minimum possible makespan). Our result for mean response time has also been extended to include interactive jobs, which are defined to be jobs that enter a blocked or sleeping phase while waiting for user input (which occurs at an unspecified time, waking the job).

6. Acknowledgments

This work was partially supported by the Natural Sciences and Engineering Research Council of Canada.

7. References


Appendix

The following are the equations for $\alpha$ and $R_1$, for the optimal competitive policy Policy($\alpha$) discussed in Section 3. Note that $\alpha$ is obtained by setting $R_1 = R_2$. They are presented in terms of $x$ and $y$ to simplify the formulæ. Since $P_1$ and $P_2$ can be expressed as a fraction of $P$, we let $P_1 = m P$ and $P_2 = n P$.

\[
x = 4nm - 8n^2m - 8nm^2 + 5n^3m + 10n^2m^2 - 3n^3m^2 - 3n^2m^3 + 5nm^3 - 4m^4n + (4n^3m + 8n^2m^2 + 4n^m^3 - 48n^2m^2 - 16n^4m - 16m^4n + 104n^4m^2 + 160n^3m^3 - 232n^4m^3 - 232n^3m^4 + 104n^2m^4 - 104n^5m^2 - 104n^2m^5 + 160n^5m^3 \]
\[ +230n^4m^4 + 49n^6m^2 + 160n^3m^5 + 49n^2m^6 - 50n^6m^3 \\
- 100n^5m^3 - 10n^7m^2 - 100n^3m^5 - 50n^3m^6 + 15n^6m^4 \\
+ 20n^5m^5 + 6n^7m^3 + 15n^4m^6 + 6n^3m^7 - 10n^2m^7 \\
+ n^8m^2 + m^8n^2 + 24n^5 - 16n^m^6 + 4n^m^7 + 24n^5m \\
- 16n^6m + 4n^7m \right)^{1/2} \]

\[ y = nm + n^2m^2 - n^3m + n^3m - 2n^2m - n^2 + 2n^3 - n^4 \]

\[ \alpha = \frac{x}{2y} \]

Replacing \( \alpha \) in \( R_1 \) (or \( R_2 \) equivalently), we have:

\[ R_1 = \frac{\frac{x}{2y} - 2m + \frac{x}{2y} + 1 - 2n - \frac{x}{2y} + n^2 + n^m}{- \frac{x}{2y} + \frac{xn}{2y} + \frac{xn}{2y} - m} \]