Performance of Fork-Join Constructs in Data-Dependent Distributed Environments

Julius Dichter
Computer Science and Engineering
University of Bridgeport
Bridgeport CT 06601
dichter@cse.bridgeport.edu

Howard Sholl
Computer Science and Engineering
University of Connecticut
Storrs CT 06269
hasc@brc.uconn.edu

Abstract

Fork-Join constructs, FJs, are a common way to spawn processes in parallel computations. When branches in a FJ have contention for resources, such as data, placement of data may significantly affect communication time in the execution of the FJ. We examine the data assignment problem in which branches are assigned to distinct processors. We develop a heuristic to assign data which tends to produce the best FJ response time. We also develop an analytical model to find the expected time for the assignment for FJs with independent communication patterns.

Keywords: Parallel and Distributed Systems, Data Assignment, Performance Optimization, Algorithms, Fork-Join Analysis

1. Introduction

The fork-join construct, FJ, is a common way to achieve process spawning and synchronization points in a computation. A FJ occurs when a single process *forks* by spawning multiple branches which execute concurrently. As each branch completes its task, it waits for all others to finish. This synchronization point is called a *join*. The performance of the FJ may depend on the combination of *data assignment*, relative speed of the processors, and communication latency and transmission speed. Data Assignment is the physical placement of data elements (or partitions) on a specific processor(s). When data is not available on the local processor, costly remote access is needed. Previous work in FJ performance has typically been limited to shared memory systems [2,9]. Typically, a data communication focus has not been applied specifically to the FJ construct [2,4,8,9,11]. The goal in optimizing FJ performance is to assign data in a way that *the maximum branch*, the last to arrive at the *join* node, finishes as early as possible. Figure 1 shows a general h-branch FJ structure. We assume that each branch is executing on a distinct processor. In shared memory systems FJ performance is affected by access to a *critical section* which can often be shared data. FJ branches *lock* the *critical section* to guarantee serial access the shared data. When the branch access needs are different, a scheduler may order the branch access to the critical section to optimize performance. In [2,9] algorithms and heuristics were developed to schedule branches of FJs in shared memory systems. Distributed systems are loosely coupled and have inherently more overhead in accessing shared resources. In such systems, we need to limit the network access for data communication. Latency is on the order of 10 ms for IP-based (Internet Protocol) protocols [1]. Parallelism in these systems is usually coarse grained, as the message payload has to be large enough to compensate the message-passing overhead [10]. Network bandwidth and the type of network also affects communication cost. The communication patterns in the branches may be independent or dependent. For *data parallel* problems in a homogeneous environment, the branch communication patterns will be essentially the same, *dependent*. Dependent communication implies that branches will attempt to communicate at approximately the same time. In a shared communication medium like Ethernet, many collisions may occur.

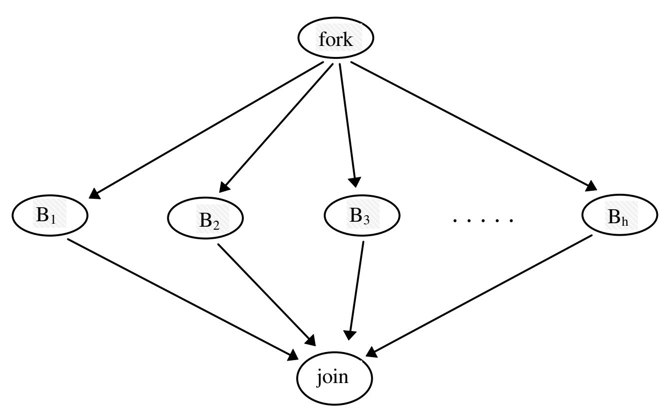

Figure 1. A typical h-branch Fork-Join structure

In *functional partitioning*, where each branch is executing different logic on the shared data, the communication pattern of the branches will be different, *independent*. The independent branches will tend to find the communication medium more available then the dependent ones. The particular network type under consideration also affects how much communication can occur *concurrently*. While Ethernet or Token Ring networks permit no true concurrent communication [3], it may well be that *independent* FJ branches do not interfere to any great degree with each other. A particular branch may be able to communicate when no other branches are using the medium. In ATM networks, true concurrency exists because the ATM switch allows simultaneous communication [3]. Our work is limited to the commonly used standard Ethernet or Token Ring protocol which use a shared communication medium.

2. Fixed vs. Variable Costs

In our model, we assume that the number of required reads and writes for each branch is known. For example, $r_{i,dk}$ is the number of read operations needed by branch i for data element d_k. Similarly, $w_{i,dk}$ is the write requirement. A *data assignment set* **A** is the physical association for each data element d_k to a subset of the FJ branches. Each FJ branch has *fixed* and *variable* costs. Fixed costs do not depend on the data assignment. CPU time and local data access are fixed costs. Local data resides with and is used exclusively by one branch. The assignment of *shared data* affects FJ branch cost. In previous work, we developed an Optimal Data Assignment algorithm, ODA, which minimized the total communication in a parallel computation [5]. Such an assignment is achieved when for each data element d_k, a single process satisfying a *First Assign Condition, FA,* and any other processes satisfying the *ADD* Condition are assigned a data element. The conditions are given in section 2.1. Below, SLD|**A** and SRD|**A** correspond to the time cost of accessing *shared local data* and *shared remote data* given data assignment **A**. The time cost to execute branch i given an assignment **A** has a lower bound given by

$$T_{Bi|\mathbf{A}} \geq CPU_{Bi} + LD_{Bi} + SLD_{Bi}\,|\mathbf{A} + SRD_{Bi}\,|\mathbf{A} \qquad (1)$$

It is a lower bound to account for communication medium contention.

2.1 The Data Communication Factor

Searching the set of all possible assignments may not be feasible [7]. It would be useful to apply our previous ODA results to find an assignment to minimize (1) for the *maximum branch*. In distributed systems, *FA and ADD* conditions are given by $MAX_{i\varepsilon FJ}\{r_{,dki} + \beta w_{i,dk}\}$ and $\frac{r_{i,dk}}{\beta} + w_{i,dk} > \omega$. Above, β is the ratio of write to read cost and $\omega = \Sigma\ w_{i,dk}$. The ODA assignment minimizes the *total* FJ communication, but it may not optimize FJ performance because a particular *maximum* branch may have a very high cost. We might settle for a higher total branch cost, such that the *maximum* branch cost is minimized. If branches had low communication medium conflict, then an assignment with higher total cost, but lowest *maximum* branch cost may be the best-choice. In [6], we give a detailed example of such a tradeoff.

3. Finding a Data Assignment

In this section we find an assignment which is likely to produce the best maximum branch cost, by focusing on SRD costs. Given a FJ construct with h branches and n data elements, we have a possible number of $(2^h\text{-}1)^n$ assignments [7]. For each data element, assignments can be classified as k-process, meaning the best assignment will consist of some number $k \leq h$ processes assigned the data element. The best 1-assignment has the data element is assigned to a single branch. For data element d_k, if we assign process i, the communication costs for the h branches are given by

$$\text{COMM}|\mathbf{A} = \{B_i\} \;\; \mathbf{=} \{(r_1+\beta w_1)\ ,\ (r_2+\beta w_2),\ \ldots\ ,(r_{i-1}+\beta w_{i-1}),\ (r_{i+1}+\beta w_{i+1}),\ \ldots,\ (r_h+\beta w_h)\}$$

This shows that the communication costs for branch i with any 1-assignment is just fa_i, its FA value. Then if we order the branches in descending *FA* costs such that $fa_1 > fa_2 > fa_3 > \ldots > fa_h$, we conclude that the best 1-assignment is $\mathbf{A} = \{p_1\}$. This is because for any branch $i \neq 1$, our ordering results in the following trivial tautology

$$\text{MAX}\ \{\text{COMM}|\mathbf{A} = \{B_1\}\} < \text{MAX}\ \{\text{COMM}|\mathbf{A} = \{B_i\}\ (r_2+\beta w_2), < (r_1+\beta w_1)$$

To find the best 2-assignment, we define an expression $fa2_i$ which is similar to fa_i, but accounts for the additional update required for data elements assigned to two processes. Thus $fa2_i = (r_i+2\beta w_i)$. The $fa2_i$'s can be ordered as before in descending order such that

$$fa2_1 > fa2_2 > fa2_3 > \ldots > fa2_h$$

Given the best 1 and 2-assignments, we can choose the better of the two. This can be done for each data element independently. While we can determine the best k-assignment for $k > 2$, it is unlikely that a 3^+-assignment will show any improvement due to a likely sharp increase in communication [5]. Simply put, when many branches are assigned d_k, any write translates to updates to multiple branches, increasing communication. The exception is read-only data which is a special case of input data, and can be identified by the program designer [5].

Lemma

The best 2-assignment for a FJ construct such that that the maximum branch has the minimum communication costs, we assign the data element to the two process with the two highest $fa2_i$ values. The proof is given in [6].

3.1 The Add-Push-Pull Heuristic (APP)

Given a FJ construct with data interdependencies among the branches, we would like to find an assignment for all data elements which would minimize the time cost of the maximum branch. Applying the best assignments for each data element, in turn, does not mean that the sum total of the communication for each branch will be minimized. The **Add-Push-Pull** heuristic can be applied to balance the branches and bring to a minimum the cost of the maximum branch. Unequal fixed costs pose no additional complexity.

APP Heuristic

1. For each data element d_k, determine the assignment the minimizes the cost of the maximum branch. Compute one assignment based on fa_i, the second on $fa2_i$. Then choose the better.
2. Apply all *best* assignments as defined in step 1.
3. Determine the maximum branch, $branch_{MAX}$, and its cost.
4. For all data elements d_k accessed by $branch_{MAX}$ which are *not* assigned to it do the following:

   i. If d_k is assigned to a single branch i , consider adding $branch_{MAX}$ to the assignment, or pulling it from branch i and pushing it into $branch_{MAX}$

   ii If d_k is assigned to two branches, i and j, consider pulling it from either branch independently with or without pushing it into $branch_{MAX}$.

   For each configuration, determine the cost of $branch_{MAX}$. It may now be a *different* branch than before. Apply the move which gets the maximum reduction in the $branch_{MAX}$ cost.
5. Repeat steps 3 and 4 until no more improvement is possible

A detailed example of the AP Heuristic is given in [6].

3.2 Refining the Communication Cost

Given an assignment, we can refine our time estimate for the maximum branch by considering the components of interprocessor communication, namely message-passing latency, the transmission speed, the granularity of the data, and the particular network characteristics. Separating the fixed costs, the latency and branch transmission costs, gives the abstract graphic of the FJ branches, shown in Figure 2. In the figure, Cp_i is the time for branch i to perform all data transmission; λ is the latency time for sending a message; FC_i is the branch fixed cost and Tp_i is the sum for fixed, latency and

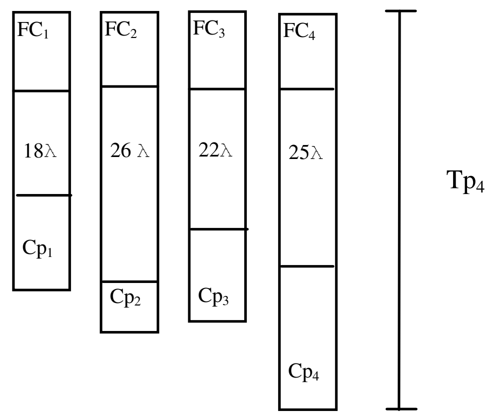

Figure 2. Shared data costs shown broken down onto latency and transmission times

transmission costs by branch i. The latency costs are from step 3 above; transmission cost can be computed from data size and transmission speed; and here fixed costs are assumed to be equal. Using the time costs given in Figure 3, we can define a lower and upper bound for the FJ construct given a particular network type. We will focus for the time on an Ethernet network. Communication times may not be additive w.r.t. the FJ construct. It is possible, for example, for a process to be transmitting data while all other processes are doing local computation. In this case a process never waits to transmit data. The FJ can be executed in at least $MAX\{Tp_i\}$ time. The upper limit of the FJ execution allows all latency to occur concurrently but also accounts for all communication sequentially, and adds the time penalty for collisions that occur on a CSMA/CD protocol [3]. The time for the FJ is then bounded by the following interval

$$t_{FORK} + t_{JOIN} + MAX\{T_{pi}\} \le T_{FJ} \le t_{FORK} + t_{JOIN} + T_{p_{\max}} + \sum_{\substack{B_i \\ i \ne \max}} \sum_{d_k} C_{B_i,d_k} + b|C|\Pi$$

where

b - expected number of collisions per transmission
|C| - total number of communications
Π - time penalty per collision
$C_{Bi,dk}$ - total cost of transmitting data d_k by branch *i*

The bounds for the FJ are helpful, but they may define a very wide interval. It would be helpful to be able to find the expected time through the FJ, $E[T_{FJ}]$.

4. Expected Time for the FJ Construct

For every branch of the FJ, we have C_{pi} and T_{pi}. The ratio of C_{pi} to T_{pi} gives us the percent of time during the execution of branch i that the processor requires the communication medium. If we assume that the communication patterns are independent and uniformly distributed, we can compute the likelihood that the medium is available. If we let the percentage of time required by processor *i* to access the medium be defined by

$$\rho_i = C_{pi} / T_{pi}$$

then the likelihood of processor *i* needing to access the medium and finding it available is

$$a_i = \prod_{j \ne i} (1 - \rho_j)$$

Suppose that process *i* needs medium access to transmit d_k. Then with probability a_i, it will take $B_{dk}*\tau$ time for the transmission. But with probability $(1- a_i)$ it will have to wait for the completion of the current transmission. Then branch *i* repeats the attempt. The time to transmit d_k by branch *i* is given by the recurrence equation

$$T_{dk,pi} = a_i*B_{dk}\tau + (1-a_i)[½m\tau + T_{dk,pi}]$$

where *m* is the average message length in bytes and B_{dk} is the number of bytes in data element d_k, including the overhead, and τ is the transmission speed in seconds per byte. Expanding the recursive relation three times results in

$$T_{dk,pi} = a_i*B_{dk}\tau + (1-a_i)[½m\tau+ a_i*B_{dk}\tau + (1-a_i)$$

$$[½m\tau+a_i*B_{dk}\tau + (1-a_i)[½m\tau+T_{dk,pi}]]]$$

and gives away the general form for the solution. The solution is simply the sum

$$T_{dk,pi} = \frac{1}{2}\sum_{j=1}^{\infty}(1-a_i)^j m\tau + \sum_{j=0}^{\infty}(1-a_i)^j a_i B_{dk}\tau$$

which can be simplified to

$$T_{dk,pi} = \frac{m\tau(1-a_i) + 2a_i B_{dk}\tau}{2a_i}$$

The *initial* expected time through the FJ for any branch can then be stated as follows

$$E[B_i] = \sum_{d_k}\{T_{dk,pi} * f_{dk,pi}\} + \lambda_i + FC_i$$

where

FC_i - the fixed time costs for branch *i*
$f_{dk,pi}$ - the number of times branch *i* transfers element d_k

Taking the maximum branch, i.e. MAX { $E[B_i]$ } will not get us the expected time through the FJ, however. That is because $T_{dk,pi}$ accounts that a_i for all branches to be constant. That is not true, however. For example, suppose we order the branches 1 through *h*, so that $E[B_1] < E[B_2] < E[B_3] < ... < E[B_h]$. When branch 1 completes, at time $E[B_1]$, branches 2 through *h*, no longer experience any branch 1 contention for the transmission medium. We assume that the branches have fixed costs, and that their communication is uniformly spread across a branch process. Thus ρ_i is constant for any branch *i*. In order to get the exact expected time for branches 2 through *h*, we need to consider the rate of completing the work as a function of the number of branches which are *still* executing. Therefore, we modify our definition of expected time for branches as follows. Let $E[B_i]_j$ the expected time to execute branch *i* assuming that branches *j* through *h* are active. $E[B_i]_j$ is then defined as

$$E[B_i]_j = \sum_{d_k}\{T_{dk,pi,j} * f_{dk,pi}\} + \lambda_i + FC_i \qquad 1 \le j \le i$$

The time to transmit data element d_k by process *i* with processes *j* through *h* active is

$$T_{dk,pi,j} = \frac{\tau * [m(1-a_{i,j}) + 2a_{i,j}B_{dk}]}{2a_{i,j}}$$

where $a_{i,j}$ is

$$a_{i,j} = \prod_{\substack{k \ne i \\ k \ge j}} (1 - \rho_k)$$

To get the rates of completion for each branch, we simply define the rate of completing branch i with j active branches as

$$r_{Bi,j} = \frac{1}{E[Bi]j}$$

Given t_i is the time to complete branch i, we can define the t_i's as follows

$t_1 = t_0 + x_1$ s.t. $\mathbf{x_1} r_{B1,1} = 1$
$t_2 = t_1 + x_2$ s.t. $t_1 r_{B2,1} + \mathbf{x_2} r_{B2,2} = 1$
....
$t_h = t_{h-1} + x_h$ s.t. $t_1 r_{Bh,1} + t_2 r_{Bh,2} + t_3 r_{Bh,3} + \ldots + t_{h-1} r_{Bh,h-1} + \mathbf{x_h} r_{Bh,h} = 1$

Given that t_0 is the initial time, typically zero, t_h is simply the sum of x_1 through x_h. Then for an h-branch FJ, we can compute $E[T_{FJ}] = t_h$ using $O(h^2)$ multiplications and additions.

4.1 An Example for Expected Performance

Let us consider a 3-branch FJ with the following expected times

$E[B_1] = 10$
$E[B_2] = 15$ $E[B_2]_2 = 13$
$E[B_3] = 20$ $E[B_3]_2 = 19$ $E[B_3]_3 = 17$

The completion rates then follow immediately
$r_{B1,1} = 1/10$
$r_{B2,1} = 1/15$ $r_{B2,2} = 1/13$
$r_{B3,1} = 1/20$ $r_{B3,2} = 1/19$ $r_{B3,3} = 1/17$

Solving for $E[T_{FJ}] = t_3$, we get 18.96. The initial expected value $E[B_3] = 20$, is a 5.5% difference.

5. Conclusions

We have developed a method to assign data to branches in FJs which execute in distributed systems. The method is based, in part, on the Optimal Data Assignment Algorithm. Next, the APP heuristic was used to minimize the execution time of the maximum branch. Finally, we defined a model to find the expected time through a FJ. Our model applies to FJs executing on shared communication medium distributed systems, such as bus or ring-based TCP-oriented communication protocols.

References

[1] G. Almasi, A. Gottlied, *Highly Parallel Computing*, The Bejamin Cummings Publishing Company, Inc., 1994

[2] R. Ammar et al, "Heuristic Scheduling Algorithms to access the critical Section in Shared Memory Environment", ISCA Conference on Parallel and Distributed Systems, Sept. 1996.

[3] D. Comer, *Computer Networks and Internets*, Prentice Hall, 1997.

[4] P. Crandall and M. Quinn, "A Partitioning Advisory System for Networked Data-Parallel Processing", Concurrency: Practice and Experience, Vol 7(5) August 1995.

[5] J. Dichter and H. Sholl, "A New Optimal Algorithm For Data Assignment in Parallel Computation", ISCA Conference on Computers and Their Applications, March 1996.

[6] J. Dichter, "Optimizing Performance of Parallel Computations from a Data Communications Perspective", Ph.D. Dissertation, University of Connecticut, 1997.

[7] S. Garg, "Joint Utilization of Control and Data Partitioning in Determining Performance Optimum Software", Ph.D. Dissertation, University of Connecticut, 1993.

[8] R. Hanxleden, L.R. Scott, "Load Balancing on Message Passing Architectures", Journal of Parallel and Distributed Computing, 13, 312-324, 1991.

[9] M. Neilforoshan et al, "Optimizing the Time Cost of Parallel Structures by Scheduling Access to Critical Sections", Technical Report, University of Connecticut, BRC-TR-94-11, 1994.

[10] Mischa Schwartz, *Computer-Communication Network Design and Analysis*, Prentice Hall, Inc, 1977.

[11] T. Znati, R. Melhem, "A Uniform Framework for Dynamic Load Balancing Strategies in Distributed Processing Systems", Journal of Parallel and Distributed Computing 23, 1994.