Abstract—This paper explores the problem of efficiently ordering interprocessor communication operations in both statically and dynamically-scheduled multiprocessors for iterative dataflow graphs with probabilistic execution times. In most digital signal processing applications, the throughput of the system is significantly affected by communication costs. We explicitly model these costs within an effective graph-theoretic analysis framework. We show that ordered transaction schedules can significantly outperform both self-timed schedules and dynamic schedules for moderate task execution time variability. As the task execution time variability increases, we show that first self-timed and then dynamic scheduling policies are preferred. We perform an extensive experimental comparison on both real and simulated benchmarks to gauge the effect of synchronization and communication overhead costs on these crossover points.

I. INTRODUCTION

Interprocessor communication (IPC) operations are responsible for significant execution time and power consumption penalties in multiprocessor embedded systems. This paper compares trade-offs for different IPC ordering strategies in both statically-scheduled and dynamically-scheduled multiprocessors for iterative dataflow specifications. We target lower-cost, shared memory embedded architectures in which IPC is assumed to take place through shared memory. Such simple communication mechanisms are common in embedded systems due to their simplicity and low cost.

II. PREVIOUS WORK

High-level exploration of interprocessor communication in multiprocessor architectures has received more attention as the complexity and size of these architectures has increased. Kogel [1] and Pasricha [2], for example, have recently presented tools for early exploration of bus-based on-chip communication architectures. They show that such exploration early in the design is essential for developing efficient implementations.

Lee and Ha [3] discuss four general scheduling strategies—fully-static (FS), self-timed (ST), static-assignment (SA), and fully-dynamic (FD)—for multiprocessors. Multiprocessor scheduling can be divided into three steps—assigning actors to processors (processor assignment), ordering the actors assigned to each processor (actor ordering), and determining when each actor should commence execution. All of these tasks can either be performed at run-time or at compile time to give us different scheduling strategies.

In the FS strategy, all three scheduling steps are carried out at compile time, including the determination of an exact firing time for each actor. The FS strategy only works when tight worst-case execution times are available, and forces system performance to conform to the available worst-case bounds. In the ST strategy, on the other hand, processor assignment and actor ordering are performed at compile time, but run-time synchronization is used to determine actor firing times—an ST schedule executes by firing each actor invocation $A$ as soon as it can be determined via synchronization that the actor invocations on which $A$ is dependent have all completed execution. In the SA strategy, the processor assignment is performed at compile time, but the ordering of actors on each processor is determined at run time. In the FD strategy, all three steps (processor assignment, actor ordering, and firing times) are determined at run time.

The ordered transaction (OT) method [9] falls between the FS and ST strategies. It is similar to the ST method but also adds the constraint that a linear ordering of the communication actors be determined at compile time, and enforced at run-time. The linear ordering imposed is called the transaction order of the associated multiprocessor implementation.

Sriram [5] shows that optimal transaction orders can be derived in polynomial time if IPC costs are negligible; however, the performance of the self-timed schedule is an upper bound on the performance of corresponding ordered transaction schedules under negligible IPC costs. Conversely, Khandelia and Bhattacharyya [7] show that when IPC costs are not negligible, the problem of determining an optimal transaction order is NP-hard, but at the same time the performance of a self-timed schedule can be exceeded significantly by a carefully-constructed transaction order.

In this paper, we examine the performance of OT and ST as a function of the variability of task execution times, and compare them with the FD strategy.

III. EXPERIMENTS

We developed a software simulator of the execution of self-timed and dynamic iterative schedules. The simulated system is a shared-memory architecture, where synchronizations are
**Communication Strategies for Shared-Bus Embedded Multiprocessors**

**Army Research Laboratory, 2800 Powder Mill Road, Adelphi, MD, 20783-1197**

Approved for public release; distribution unlimited

Security classification:
- a. Report: unclassified
- b. Abstract: unclassified
- c. This Page: unclassified

Limitation of abstract: unclassified

Number of pages: 4

Approved for public release; distribution unlimited
performed by accessing the shared memory bus. Data tokens associated with the IPC constraints are transferred via the shared memory. When a processor tries to gain access to the bus for a synchronization or interprocessor data communication, the system permits access if the bus is not in use, otherwise it denies access and the processor waits for a specified back-off time, then tries again.

The synchronization cost for OT is much lower than the synchronization costs for ST. In the OT strategy a shared bus access takes no more than a single read or write cycle on the processor, and the overall cost of communicating one data sample is two or three instruction cycles [9].

Our simulator for ST operation implements both the Unbounded Buffer Synchronization (UBS) and the Bounded Buffer Synchronization (BBS) protocols [9]. In the BBS protocol, the protocol requires one local memory increment operation (the local write pointer) and one write to shared memory (store write pointer) occur after the source node of the synchronization edge has executed.

We assume an architecture where all synchronization and memory accesses occur in a single shared memory. We define a parameter \( \beta \) to be the ratio of the synchronization time to the IPC time. Since we are considering HSDF graphs with one data token produced per IPC operation, we have \( \beta \geq 2 \) for BBS (at least 2 memory accesses for synchronization for every data memory access) and \( \beta \geq 4 \) for UBS.

The simulator for FD assumes a centralized scheduler with separate control signals to each processor. The scheduler keeps track of ready tasks and ready processors. A task is ready when all its predecessors in the application graph have completed. A processor is ready if it is not executing a task or IPC operation. Tasks are prioritized according to ready time (the earliest time at which all predecessors have completed) with ties broken by static level [9]. The scheduler attempts to place the highest priority task on the lowest number ready processor whenever a new ready task is detected.

We implemented the heuristic transaction partial order (TPO) algorithm [7] to determine the OT task ordering. This heuristic simultaneously takes IPC costs and the serialization effects of transaction ordering into account when determining the transaction order.

A. Task Execution Times

For many DSP applications, it is possible to obtain accurate statistics on task execution times. Probabilities for events such as cache misses, pipeline stalls, and conditional branches can be obtained by using sampling techniques or simulation of the target architecture [10]. We utilize the task execution model in [11], where each task \( v_i \) in the task graph \( G = (V, E) \) is associated with three possible execution times \( e_0, e_1, \) or \( e_2 \) with probabilities \( p_0, p_1, \) and \( p_2 \) respectively. Here, \( e_0 \) is the task execution time given in the benchmark specification, \( e_1 = 2e_0 \) and \( e_2 = 4e_0 \). We define a single parameter \( p \) for the degree of randomness of the task execution times, where \( p_0 = (1 - p), p_1 = p(1 - p), \) and \( p_2 = p^2 \). Note that for this probability distribution, \( p = 0.5 \) corresponds to the highest degree of randomness. Under these assumptions, we note that the FS strategy is not practical for any \( p > 0 \), no matter how small, since an FS architecture must operate with \( e_2 \) for all tasks in order to assure correctness.

B. Benchmarks

The benchmark application graphs used were fairly complicated, ranging from between 9–764 nodes, and the numbers of processors involved ranged from 3 to 8. We examined a combination of real and synthetic benchmarks. For the synthetic benchmarks, we used the TGFF [12] algorithm.

The examples \( fft1, fft2, \) and \( fft3 \) result from three representative schedules for Fast Fourier Transforms based on examples given in [13]: \( karpi0 \) is a music synthesis application based on the Karplus Strong algorithm in 10 voices; the video coder is taken from [14], and \( eddat \) is a CD to digital audio tape converter.

IV. Results

Experiments were carried out to compare the OT, ST, and FD methods, and to measure the performance of the TPO heuristic in finding transaction orders. For the OT and ST methods, the benchmarks were scheduled using the DLS algorithm [18].

We used the task execution model from Section III-A, and calculated the average iteration periods over 10000 iterations.

We define a parameter \( \alpha \) that quantifies the IPC overhead in a given schedule. It is calculated from the ratio of the total IPC time (synchronization plus data communication) to the total execution time spent on computational tasks over all processors. Thus, \( \alpha \) is a function of \( p \), the schedule, and the relative speed of processor to memory. We note that VLSI is trending toward higher relative processor-to-memory speeds (higher \( \alpha \)) as gate lengths decrease.

Figure 1 plots \( T_{OT}, T_{ST}, \) and \( T_{FD} \) versus the parameter \( p \) that governs the degree of randomness of the task execution times for a synthetic application benchmark generated by TGFF. It can be seen that the dynamic scheduling approach performs significantly worse than OT and ST for low task execution time randomness (low \( p \)), but that FD is much less sensitive to \( p \). The dynamic scheduling algorithm is able to adapt (through different task orderings and assignments) to changes in execution times, while the task orderings and assignments are fixed for ST and OT. This behavior of FD compared to OT and ST was observed with all the benchmarks.

Figure 2 plots \( T_{OT} \) and \( T_{ST} \) versus \( p \) for different values of \( \beta \), the ratio of synchronization-to-IPC overhead described in Section III. The calculations for \( \beta = 0 \) do not correspond to any synchronization protocol in our architectural model, but are given as a point of reference. Values of \( \beta < 2 \) would be possible if a separate (faster) memory were allocated to the synchronization variables.

The iteration period increases with \( p \) since the average execution time increases with \( p \). For many DSP applications \( p < 0.1 \) is a reasonable assumption. For example, if \( e_2 \)
corresponds to a cache miss, $e_1$ to a processor pipeline stall,
and $e_0$ to the base execution time for a task. $p = 0.1$
corresponds to a 1% cache miss probability, a 9% pipeline
stall probability, and a 90% probability for the base execution
time.

From Figure 2 we see that $T_{ST}$ increases more slowly as
a function of $p$ than does $T_{OT}$. This is because the self-
timed schedule has more flexibility than the OT schedule (the
OT schedule imposes a pre-determined, global ordering of
all the IPC while the ST does not) and thus is better able
to adapt to changes in task execution times. This behavior
was observed with all the benchmarks. We also see that it is
possible for OT to outperform ST for $\beta = 0$, but only for small
$p$. Comparing Figure 2(a) and Figure 2(b), we see that the
slopes of the curves decrease as $\alpha$ increases. This is because
the IPC operations are not random, and so as IPC increases,
a smaller fraction of the overall execution time comes from
random tasks.

We also observe that the relative improvement of OT over
ST increases as $\alpha$ increases. Figures 3 and 4 plot the ratio
$T_{ST}/T_{OT}$ versus $\alpha$ for the irr and fft1 benchmarks. For the
fft1 benchmark, $T_{ST}/T_{OT} > 1$ for all $\beta \geq 2$ and $p \leq 0.4$. For the
fft1 benchmark with $\beta = 2$ and $p = 0.4$, $T_{ST}/T_{OT} > 0.96,
and $T_{ST}/T_{OT} > 1$ elsewhere. As discussed above, $p = 0.4$
represents a high degree of uncertainty for task execution times
in DSP applications (with $p = 0.5$ representing the highest
possible degree of randomness in the probability distribution).

Table I compares the performance (iteration period) of the
ST and the OT schedules. In all cases, we observe that the
OT strategy outperforms the ST strategy for $\beta \geq 2$. As noted
before, $\beta = 2$ and $\beta = 4$ represent lower bounds for the BBS
and UBS synchronization protocols, respectively. The results

---

Fig. 1. OT, ST, and FD scheduling for an application graph generated by
TGFF.

Fig. 2. Iteration periods $T_{OT}$ and $T_{ST}$ for the irr benchmark versus amount
of randomness in task execution times (parameter $p$).

Fig. 3. Ratio of $T_{ST}$ to $T_{OT}$ versus IPC overhead for fft1 with $\beta = 4$.

Fig. 4. Ratio of $T_{ST}$ to $T_{OT}$ versus IPC overhead for fft1 with $\beta = 2$. 
for the (synthetic) TGFF benchmark are an average over 50 different graphs generated by the TGFF program [12].

V. CONCLUSIONS
We have demonstrated that the ordered transaction method—which is superior to the self-timed method in its predictability, and its total elimination of synchronization overhead—can significantly outperform self-timed and fully dynamic implementations for low task variability, even though the ordered transaction implementation offers less run-time flexibility due to a fixed ordering of communication operations. When synchronization cost is taken into account, the ordered transaction method performs significantly better than the self-timed and fully dynamic methods.

We have studied the relative behavior of OT, ST, and FD implementations under a realistic model for task execution times. The OT strategy performs better relative to the ST and FD strategies for lower $p$ (degree of randomness in task execution times), higher $\beta$ (synchronization costs), and higher $\alpha$ (IPC overhead). The ranges in which OT favors ST encompass the design space that we are targeting—namely, low-cost shared bus embedded multiprocessor DSP systems.

We have also developed a detailed simulator to measure the performance of the self-timed schedule under different constraints.

The benefits of OT can be expected to increase with the general trend in VLSI technology for increasing processor/memory performance disparity. Some of this benefit may be offset, however, by another trend, which is for decreasing predictability in application behavior (and thus execution times) due to the use of more and more sophisticated and adaptive types of algorithms. The evolution of the MPEG standards is an example of this. Further research on OT methods to efficiently handle such lower degrees of predictability is therefore an interesting and important direction for further study.

REFERENCES

<table>
<thead>
<tr>
<th>Application</th>
<th>$\alpha$</th>
<th>$\beta = 0$</th>
<th>$\beta = 2$</th>
<th>$\beta = 4$</th>
<th>$\beta = 0$</th>
<th>$\beta = 2$</th>
<th>$\beta = 4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>tgff avg.</td>
<td>0.21</td>
<td>1.057</td>
<td>1.266</td>
<td>1.342</td>
<td>1.046</td>
<td>1.190</td>
<td>1.321</td>
</tr>
<tr>
<td>fft1</td>
<td>0.04</td>
<td>0.970</td>
<td>1.024</td>
<td>1.097</td>
<td>0.997</td>
<td>1.064</td>
<td>1.088</td>
</tr>
<tr>
<td>fft1</td>
<td>0.26</td>
<td>0.858</td>
<td>1.146</td>
<td>1.483</td>
<td>0.872</td>
<td>1.135</td>
<td>1.452</td>
</tr>
<tr>
<td>fft2</td>
<td>0.02</td>
<td>0.995</td>
<td>1.035</td>
<td>1.083</td>
<td>1.042</td>
<td>1.058</td>
<td>1.115</td>
</tr>
<tr>
<td>karp10</td>
<td>0.04</td>
<td>1.020</td>
<td>1.665</td>
<td>2.147</td>
<td>1.001</td>
<td>1.632</td>
<td>2.104</td>
</tr>
<tr>
<td>video coder</td>
<td>0.19</td>
<td>0.896</td>
<td>1.372</td>
<td>1.895</td>
<td>0.896</td>
<td>1.273</td>
<td>1.687</td>
</tr>
<tr>
<td>cddad</td>
<td>0.735</td>
<td>1.014</td>
<td>1.117</td>
<td>1.441</td>
<td>0.933</td>
<td>1.028</td>
<td>1.326</td>
</tr>
<tr>
<td>irr</td>
<td>0.72</td>
<td>1.013</td>
<td>1.405</td>
<td>2.140</td>
<td>1.001</td>
<td>1.414</td>
<td>2.130</td>
</tr>
<tr>
<td>irr</td>
<td>0.24</td>
<td>0.956</td>
<td>1.246</td>
<td>1.645</td>
<td>0.954</td>
<td>1.082</td>
<td>1.470</td>
</tr>
<tr>
<td>laplace</td>
<td>0.03</td>
<td>0.837</td>
<td>1.692</td>
<td>2.406</td>
<td>0.847</td>
<td>1.603</td>
<td>2.248</td>
</tr>
<tr>
<td>laplace</td>
<td>0.14</td>
<td>1.025</td>
<td>1.316</td>
<td>1.608</td>
<td>0.979</td>
<td>1.186</td>
<td>1.376</td>
</tr>
</tbody>
</table>

TABLE I

$T_{ST}/T_{OT}$ FOR ST AND OT SCHEDULES.