The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

OPTIMIZING SW/HW ARCHITECTURE FOR PARALLEL EMBEDDED SYSTEMS - A CASE STUDY

Vaclav DVORAK

Dept. of Computer Science and Engineering, University of Technology Brno,
Bozetechova 2, 612 66 Brno, CZECH REPUBLIC, *dvorak@dcse.fee.vutbr.cz*

Abstract. *The paper addresses the issue of prototyping hw/sw architecture of application-specific multi-processor systems (recently on a chip). Performance prediction of these systems, either bus-based SMPs or message-passing networks of DSPs, is undertaken using a CSP-based tool Transim. Variations in processor count, clock rate, link speed, bus bandwidth, cache line, as well as in partitioning and mapping the resulting sw components to processors can be easily accounted for. The technique is demonstrated on parallel FFT on 2 to 8 processors.*

Key Words. *Parallel Embedded Systems, Multiprocessor Simulation, Hardware Description Language*

1. INTRODUCTION

The design of mixed hw/sw systems for embedded applications has been an active research area in recent years. Hw/sw co-synthesis and co-simulation have been mainly restricted to a single processor and programmable arrays attached to it, that were placed incidentally on a single chip (SoC). A new kind of systems, application-specific multi-processor SoC, is emerging with frequent applications in small-scale parallel systems for high-performance control, data acquisition, and analysis, image processing, wireless, networking processors, and game computers. Typically several DSPs and/or microcontrollers are interconnected with an on-chip communication network and may or may not use an operating system. In this paper, we want to concentrate on performance prediction of various configurations of hw and sw, since performance guarantees must be complied with before anything else can be decided. Other difficult problems such as system validation at the functional level and at the cycle-accurate level, software and RTOS synthesis, task scheduling and allocation, overall system testing, etc., are not considered.

In this paper we will study only application-specific multiprocessors and modeling their performance. As a suitable application for performance comparison, we have selected

a parallel FFT (1024) benchmark (1024 points, one dimension), in real-time environment, with the goal of maximizing the number of such FFTs per second.

2. ARCHITECTURES OF PARALLEL EMBEDDED SYSTEMS AND CMP

The performance race between a single large processor on a chip and a single-chip multiprocessor (CMP) is not decided yet. Applications such as multimedia point to CMP with multithreaded processors [1] for the best possible performance. The choice between application-specific (systolic) architectures or processors on one hand and CMP on the other is yet more difficult. CMP architectures may also take several forms such as:

- a bus-based SMP with coherent caches similar to Pentium Pro quad pack, with an atomic bus or a split-transaction bus;
- a SMP with a crossbar located between processors and a shared first-level cache which in turn connects to a shared main memory;
- a distributed memory architecture with a direct interconnection network (e.g. a hypercube) or an indirect one (the multistage crossbar).

As the number of processors on the chip will be typically lower than 10, at least in a near future, we do not have to worry about scalability of these architectures. Therefore the bus interconnection will not be seen as too restrictive in this context.

Some more scalable architectures such the SMP with processors and memory modules interconnected via a multistage interconnection network (the so called „dancehall“ organization) or a hw-supported distributed shared memory will not be considered as candidates for small-scale parallel embedded systems or SoCs.

Let us note, that the choice of architecture can often be also dictated by a particular application to be implemented in parallel. E.g. broadcasting data to processors, if not hidden by computation, may require a bus for speed, but on the contrary, all-to-all scatter communication of intermediate results will be serialized on the bus and potentially slower than on a direct communication network. Some decisions can be supported by back-of-the-envelope calculations, others are more difficult due to varying message lengths or irregular nature of communications. This is where simulation fits into.

For the sake of the presented case study, we will investigate the following (on-chip) communication networks:

1. fully connected network
2. SF hypercube
3. WH hypercube
4. Crossbar switch
5. Atomic bus.

The number of processors p = 2, 4, and 8. The problem size of a benchmark (parallel 1D-FFT) will be n = 1024 points.

3. THE SIMULATION TOOL AND THE DESCRIPTION LANGUAGE

A performance modeling has to take characteristics of the machine (including an operating systems, if any) and application and predict the execution time. Generally it is much more difficult to simulate performance of an application in shared address space than in message passing, since the events of interest are not explicit in the shared variable program. In the shared address space, performance modeling is complicated by the very same properties that

make developing a program easier: naming, replication and coherence are all implicit, i.e. transparent to the programmer, so it is difficult to determine how much communication occurs and when, e.g when cache mapping conflicts are involved [5].

Sound performance evaluation methodology is essential for credible computer architecture research to evaluate hw/sw architectural ideas or trade-offs. Commonly used shared-memory simulators rsim, Proteus, Tango, limes or MulSim [2], beside their sophistication, are not suitable for message passing systems. This made us to reconsider the simulation methodology for shared-memory multiprocessors. Here we suggest using a single CSP-based simulator both for message passing as well as for shared address space. It is based on simple approximations and leaves the speed vs. accuracy tradeoff on the user, who can control the level of detail and accuracy of simulation.

The CSP-based Transim tool can run simulations written in Transim language [3]. It is a subset of Occam 2 with various extensions. Transim is naturally intended for message-passing distributed memory systems. Nevertheless, it can be used also for simulation shared memory bus-based (SMP) systems - bus transactions in SMP are modeled as communications between node processes and a central process running on an extra processor. Transim also supports shared variables, which are used in modeling locks and barriers. Until now, only an atomic bus model has been tested; the split-transaction bus requires more house-keeping and its model is going to be developed in a near future.

The input file for Transim simulator tool contains descriptions of software, hardware and mapping to one another. In software description, control statements are used usual way, computations (integer only) do not consume simulated time. That is why all pieces of sequential code are completed or replaced (floating point) by special timing constructs SERV ( ). Argument of SERV( ) specifies the number of CPU cycles taken by the task. Granularity of simulation is therefore selectable from individual instructions to large pieces of code. Explicit overhead can be represented directly by WAIT() construct. Data-dependent computations can be simulated by SERV construct with a random number of CPU cycles. Some features of a RT distributed operating system kernel, originally supported by hw in transputers, are also built into the simulator, such as process management, process priorities (2 levels only), context switching, timers, etc.

The NODE construct in hardware description is used to specify the CPU speed, communication model and other parameters; otherwise the default values are used. The mapping between software and hardware, between processes and processors, is made through the MAP construct. Parallel processes on different processors, one process per processor, are created by PLACED PAR construct for MPMD or by replicated PLACED PAR for SPMD model of computation.

4. THE PARALLEL FFT BENCHMARK PROGRAM

We will illustrate the technique of optimization of hw/sw multiprocessor architecture on the problem of computing the 1D-, n-point-, discrete Fourier transform on p processors in O((n log n)/p) time. Let p divides n, $n = 2^q$ is a power of two and $n \geq p^2$. Let the n-dimensional vector x [n×1] be represented by matrix X[n/p×p] in row-major order (one column per processor). The DFT of the vector x is given by

$$Y = W_n X = W_p \left[S * W_{n/p} X \right]^T \tag{1}$$

where S [n/p× p] is the scaling matrix, * is elementwise multiplication and the resulting vector y [n×1] = W_n x is represented by matrix Y [n/p×p] in column major order form (n/p^2 rows per processor). Operation denoted by T is a generalized matrix transpose that corresponds to the usual notion of matrix transpose in case of square matrices [4].

The algorithm can be performed in the following three stages. The first stage involves a local computation of a DFT of size n/p in each processor, followed by the twiddle-factor scaling (elementwise multiplication by S). The second stage is a communication step that involves a matrix transposition. Finally, n/p^2 local FFTs, each of size p, are sufficient to complete the overall FFT computations on n points. The amount of computation work for the sequential FFT of an n-element real vector is n log n / 2 "butterfly" operations, where one butterfly represents 4 multiplications and 6 additions/subtractions (20 CPU clocks in simulation). In parallel implementation the computation work done by p processors is divided into stage 1 and 3, but the total amount of work is the same,

$$p \ [\frac{n}{2p}\log\frac{n}{p} + \frac{n}{p^2}\frac{p}{2}\log p \ ] = \frac{n}{2}\log n . \qquad (2)$$

Let us note, that the work done in stage 1 proportional to (log n - log p) is much larger than the work done in stage 3, proportional to log p. The only overhead in parallel implementation is due to a matrix transposition. The matrix transposition problem is equivalent to all-to-all scatter (AAS) group communication. Clearly, it requires 1 step in a fully connected topology, p/2 steps (a lower bound) in the SF-hypercube, p-1 steps in the WH-hypercube or a crossbar, and finally p(p-1) bus transactions on a bus.

FFT processing will be done continuously in real time. Therefore loading of the next input vector from outside and writing the previous results from processors to environment will be carried out in the background, in parallel with three stages of processing of the current input vector (with the first stage of processing only in the shared memory case).

5. PARAMETERS OF SIMULATED ARCHITECTURES AND RESULTS OF SIMULATION

Six architectures simulated in the case study are listed in Tab.1 together with the execution time. The CPU clock rate is 200 MHz in all 6 cases, the external channel speed of 100 Mbit/s (12 MB/s) is used for serial links in all message-passing architectures, whereas bus transfer rate for SMP is 100 MB/s. Downloading and uploading of input data and results were supposed to continue in the background in all processors simultaneously at 8-times higher rate than the link speed, which is almost equivalent to the bus speed in SMP case. In message-passing architectures the AAS communication was overlapped with submatrix transposition as much as possible. Optimum routing algorithm for SF hypercube and AAS communication requires p/2 steps and uses a schedule tables at Fig.1. In case of WH hypercube, dimension-ordered routing is used in every step i, i = 1, 2,…, p-1, in which src-node and dst-node with the relative addresses RA = src ⊕ dst = i exchange messages.

The small cluster of (digital signal) processors, referred to as COSP in Tab.1, uses a centralized router switch (Omega type) with sw/hw overhead of 5 μs, the same as a start-up cost of serial links, and WH routing. The algorithm for AAS was designed to avoid contention using cyclic permutations [e.g (01234567), (0246)(1357), …, (07654321)] for p=8.

Finally a bus-based shared memory system with coherent caches (SMP) has had 100MB/s bus bandwidth, 50 MHz bus clock, and the miss penalty of 20 CPU clocks. We will assume an

step	RA in dimension	
	0	1
1	3	2
2	1	3

step	relative addr. used in dimension		
	0	1	2
1	3	6	4
2	1	7	6
3	7	2	5
4	5	3	7

Fig. 1. . Optimum schedule for AAS in all-port full-duplex 2D- and 3D- SF hypercubes

atomic bus for simplicity and fair bus arbitration policy. Other types of bus arbitration (priority-based, random, etc.) are also feasible. The cache block size is 16 bytes and the size of the cache is assumed to be sufficient to hold input data (a real vector), intermediate data after the first stage of FFT (a complex vector) as well as the results (a complex vector). In the worst case (p=2) the size of all these vectors will be around 10 kB, if we use REAL32 format. We assume I/O connected via a bus adapter directly to the cache. To avoid arbitration between CPU and I/O, the next input and previous results are transferred in/out during the first stage of the FFT algorithm.

The results summarized in Tab.1 and plotted in Fig.2 deserve some comments. A fully connected network of processors is the fastest architecture for 8 processors, but the slowest for 2 processors. The reason is that communication is mostly seen as an overhead, but gets better overlapped with communication when p increases. The cluster of DSPs (COSP row in Tab.1) starts with p=4 and increasing the number of processors from 4 to 8 does not make much sense because it has small influence on speed.

Tab.1. Parallel FFT execution times in μs for six analyzed architectures

p =	2	4	8
full	436,8	180,8	138
COSP		230,4	173,1
SMP	363,5	304,7	321,6
SF cube		272	182,8
WH cube		230	174,4

In the SMP with shared bus, processors write the results of the n/p-point FFT computed in stage 1 into the local caches and do the transposition at the same time. This means that consecutive values of FFT will be stored with a stride required by the rule of matrix transposition. The following read requests by other processors at the start of stage 3 will generate read misses: at cache block size 16 bytes, one miss always after 3 hits in a sequence. Fresh cache blocks will be loaded into requestor's cache and simultaneously into the shared memory. A prefetch of cache blocks has been simulated without an observable improvement in speed, most probably due to bus saturation. This is even worse for 8 processors than for 4, see Fig.2.

As for hypercubes, the WF hypercube is superior and gives the same results as a cluster of DSPs. Slightly worse performance than that of a fully connected processors is balanced by much simpler interconnection and by a lower number of communication ports.

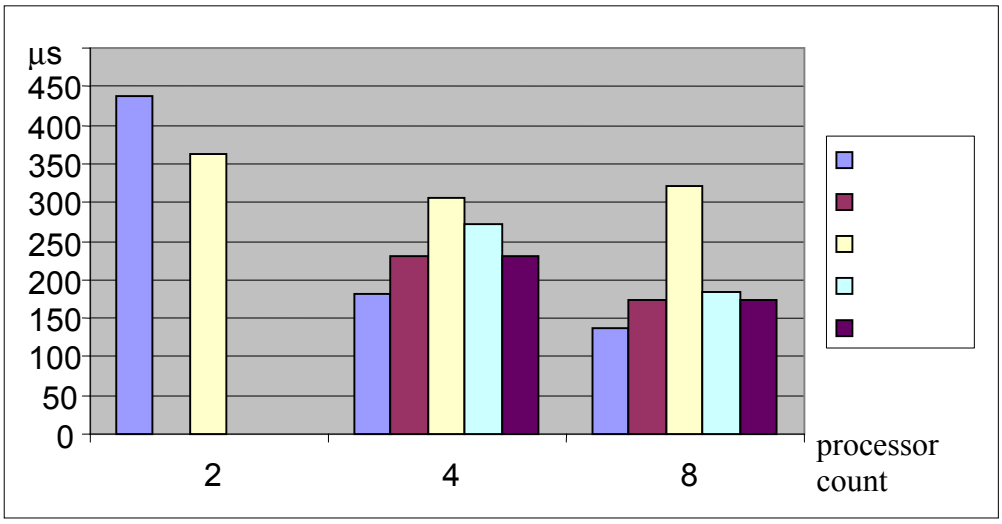

Fig.2. Comparison of execution times [µs] for six architectures.

6. CONCLUSIONS

The performance study of the parallel FFT benchmark on a number of architectures using Transim tool proved to be a useful exercise. Even though the results of simulations have not been confronted with real computations, they can certainly serve to indicate serious candidate architectures that satisfy certain performance requirements. The approximations hidden in simulation are limiting accuracy of real-time performance prediction, but the level of detail in simulation is given by the user, by how much time he or she is willing to spend on building the model of hw and sw. For example, modeling the split-transaction bus or the contention in interconnection network for WH routing could be quite difficult. The latter was not attempted in this case study since the FFT benchmark requires only regular contention-free communication. This, of course, will not be generally the case. Nevertheless, simulation enables fast varying of sw/hw configuration parameters and studying the impact of such changes on performance, free from the second-order effects. In this context, the CSP-based Transim simulator and language proved to be very flexible, robust and easy to use. The future work will continue to include other benchmarks and analyze the accuracy of performance prediction.

REFERENCES

[1] Silc, J.- Robic, B.-Ungerer, T.: Processor Architecture: From Dataflow to Superscalar and Beyond. Springer-Verlag, 1999, ISBN 3-540-64798-8.

[2] http://heather.cs.ucdavis.edu/~matloff/mulsim.html

[3] Hart, E.: TRANSIM - Prototyping Parallel Algorithms. London, University of Westminster Press (2nd edition), 1994.

[4] Zomaya, A.: Parallel and Distributed Computing Handbook. McGraw Hill, 1996, p.344.

ACKNOWLEDGEMENT

This research has been carried out under the financial support of the Research intention no. CEZ: J22/98: 262200012 - Research in information and control systems.