Parallelizing and Optimizing a Simulator Kernel on a TMS320C40 Multi-DSP Architecture

Authors: A. Riel, E. Brenner

EFRIE, France

December 1995
SPRA309

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain application using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

TRADEMARKS

TI is a trademark of Texas Instruments Incorporated.

Other brands and names are the property of their respective owners.

Contents

Figures

Parallelizing and Optimizing a Simulator Kernel on a TMS320C40 Multi-DSP Architecture

Abstract

This application report describes the parallelization and optimization of a hydraulic simulator kernel on a Texas Instruments (TI™) TMS320C40 multi-DSP (digital signal processor) platform. Research is based on A. Riel's diploma thesis. This report includes references to works directly connected with this thesis.[1]

Three different parallel versions of the initially sequential simulator are implemented and optimized for various hardware and software-related issues. The paradigms applied for parallelization are discussed along with the effects certain optimizations have on simulator performance.

The various parallelization and optimization approaches for real-time implementation are judged by measuring the comparable speedup and simulation times for all parallel versions of the same simulator. This application report includes selected interpreted data of this research.

This document was an entry in TI's DSP Solutions Challenge, an annual contest organized by TI to encourage students from around the world to find innovative ways to use DSPs. For more information on the TI DSP Solutions Challenge, see TI's World Wide Web site at www.ti.com.

TI's web site contains the most up-to-date product information, revisions, and additions. New users must register with TI&ME before they can access the data sheet archive. TI&ME allows users to build custom information pages and receive new product updates automatically via email.

Introduction

The hydraulic simulator kernel Hydsim has been developed over several years at the Graz University of Technology in cooperation with FEST 0-Didactic in Esslingen.[2] Hydsim simulates the dynamic behavior of complex hydraulic circuits in response to user-specified control events. The need to parallelize the simulator arose when the transient analysis facility was integrated into Hydsim to simulate exactly the propagation of hydraulic pressure surges through circuit pipes.

This application report deals with the *dynamic* version of Hydsim, in which a far more sophisticated mathematical model is employed to represent pipes, rather than the original *static* version, in which pipes are modeled as simple hydraulic resistors.[3]

As a pioneer work, Hydsim was ported to the Ariel Hydra V-C40 by G. Wirrer.[4] The Hydra is a widely used high performance multi-DSP system equipped in our configuration with four floating-point TMS320C40 DSPs.[5] The four DSPs are independent processing nodes with private memory banks that communicate with each other as well as with external devices via shared memory and high-speed communication links.

Overview of Hydsim introduces the simulator by focusing on elements that need parallelization. *Optimization* deals with the results of optimizing the simulator and considers both algorithm and source code issues. In this respect, our case study reveals the importance of considering the destination hardware's characteristics when designing time-critical software.

We parallelized the optimized kernel three times for the following two paradigms, which are discussed in *Parallelization*:

- Static partitioning (see *Static Partitioning*)

  Enables processing nodes to carry out simulations of statically assigned parts of hydraulic circuits in their fast local memory banks only.

- Load balancing (see *Load Balancing*)

  Employed in two versions of Hydsim:

  - The first Hydsim version uses private memory and DMA exclusively to quickly transfer publicly required data items in parallel with normal CPU operation.
  - The second Hydsim version keeps the minimum set of public data in shared memory, thereby reducing the overhead required for communication and initialization of DMA channels.

Comparison and Discussion critically compares all three parallel versions to one another.

Finally, *Further Work* briefly discusses other work done for the thesis, including embedding Hydsim in an existing simulation package and contributing to the design and test of a memory-update hardware module.

Overview of Hydsim

Hydsim is the kernel of EDSIM, a comfortable environment for the graphical construction and analytical simulation of complex hydraulic circuits.[6] EDSIM's facilities are updated and extended in WEDSIM, a state-of-the-art generic graphical simulation environment currently being developed at our institute.[7]

The high-level graphical specification of a hydraulic circuit is initially translated to a generic textual description, which Hydsim uses internally to represent the circuit as a directed graph. All equation systems are automatically derived from this representation to be iteratively solved by a modified version of Newton's algorithm.

During transient analysis, the computational expense of the main simulation loop is enormously high, since the simulation time step has to be extremely small (about 250 μs of simulated time). This requirement is imposed by the demand for strict adherence to the relationship between the time step and the length increment prescribed by the model and integration algorithm employed to simulate the transient behavior of pipes.[8]

Optimization

Algorithms

Simulation and careful code analysis reveal that the algorithms used in the dynamic version of Hydsim provide a reasonable tradeoff between execution speed and physical accuracy. In most practical applications, it is not necessary to solve the network equation systems at each time step. Repeatedly, there are phases of quasi-steady flow during which the simulation time step can be enlarged to a multiple of the basic step prescribed by the integration algorithm employed in the pipe model. The enlargement does not seriously affect simulation results.

Consequently, we implemented a mechanism we call *Dynamic Time Step Control* (DTSC). During quasi-steady phases of simulation, DTSC allows network equation systems to be solved at larger time steps while the pipe flow equations continue to be integrated using the elementary time increment. Hence, DTSC comprises the following tasks:

- Automatically detecting quasi-steady flow during transient analysis
- Gradually increasing the time increments used to solve network equation systems and simulating the behavior of hydraulic elements other than pipes
- Automatically detecting transients during quasi-steady simulation
- Resetting the time increment to the basic time step on resuming dynamic simulation

Numerous simulations reveal that execution times can be cut by a significant amount without seriously compromising the physical accuracy of simulation results.

Code

The fundamental step in optimizing Hydsim for the Hydra was to divide the simulator into two parts:

- Initialization
- Calculation

We designed the initialization part to be executed on the host machine that controls the Hydra as a preprocessor for the calculation part. We constructed the calculation part in a way we considered optimal for the DSP platform.

This approach enabled us to maximize the use of the Hydra's most valuable resources for the benefit of the simulation's most expensive part.

Although requiring a lot of memory, the tasks with which the preprocessor deals are computationally inexpensive and have to be performed only *once* at the beginning of each simulation. Even large hydraulic circuits are handled within a matter of moments on slow PCs or workstations.

We used the following methods to obtain a highly efficient hydraulic simulation engine:

- Made the preprocessor rebuild linked lists as contiguous, linearly addressable memory blocks
- Determined a sophisticated concept to avoid any dynamic allocation of memory in the calculation part
- Hand-optimized the most time-consuming (and maintenance-free) procedures in DSP assembly language
- Modified Hydsim's object-oriented approach of representing hydraulic elements toward increased efficiency
- Optimized the mapping of code, data, and stack sections to the board's and DSPs' memory resources

Interestingly, we observed that the most effective optimizations occur when avoiding pipeline conflicts.[9] [10]

Host Program

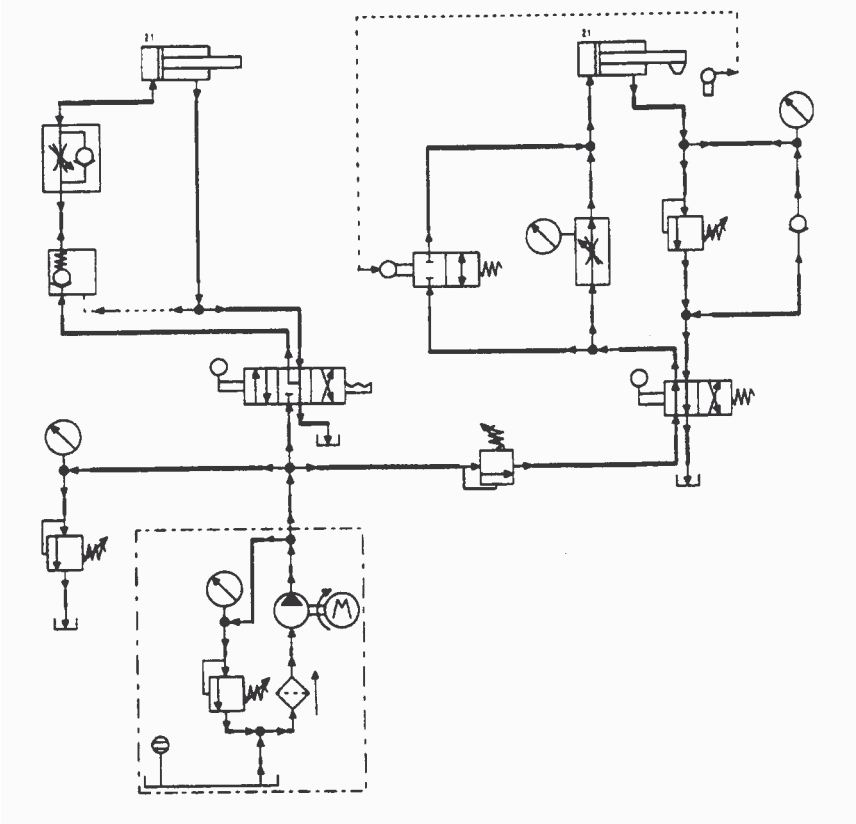
Communication between the host program and the kernel executing on the Hydra is designed never to interfere with the operation of the DSPs. To start simulation, the host program downloads the memory images generated by the preprocessor to the fast private memory banks of the DSPs. During simulation, results produced by the DSPs are buffered in two alternate areas of the fast static local memory of each processor:

- The DSP writes into the active one.
- The host program accesses the passive one via the VME-bus.

Optimization Results

Figure 1 shows circuit A19, which is used for the sample measurements throughout this application report. During 20 seconds of simulated time, the outline function of A19 allows both cylinders to move asynchronously in response to a series of valve switchings. The cylinder on the right side controls a valve by itself. A series of tests showed that measurement results of this expensive simulation may be accepted as representative for the quality of the simulator version involved.

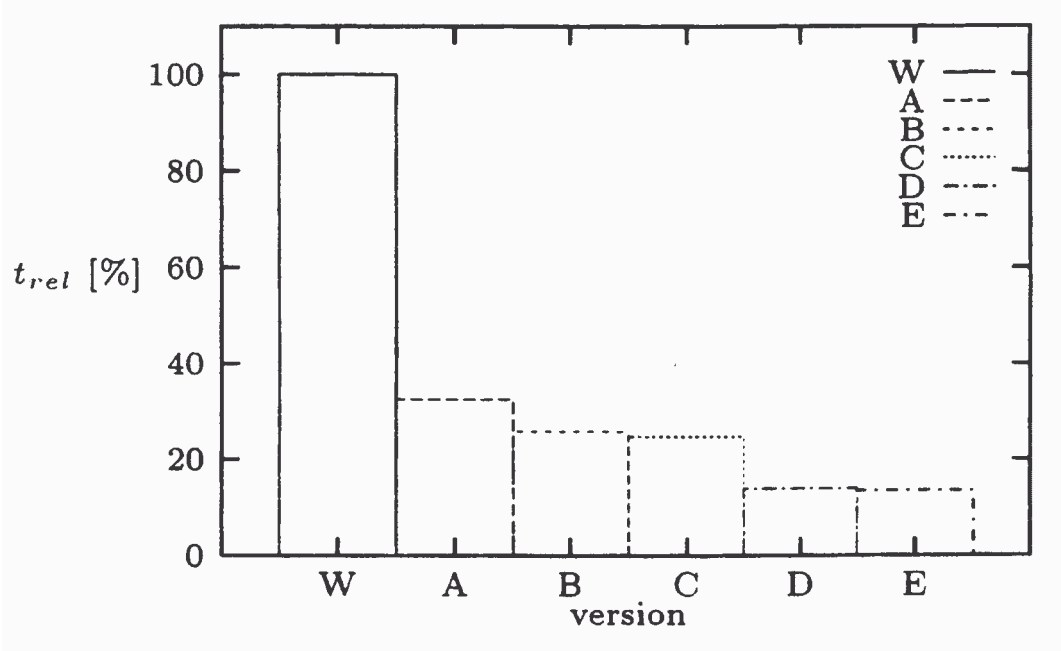
Figure 1. Circuit A19



The optimized version without DTSC executes about four times faster than the original single DSP version (see Figure 2). Furthermore, the results reveal that hand optimizing the module implementing Newton's algorithm was not substantially fruitful but optimizing the graph-traversal procedures was worth the effort. This can be explained by the fact that the Newton module primarily consists of algebraic operations that are optimized by the compiler's code optimizer reasonably well.

On the contrary, graph traversal is highly recursive, requiring a lot of subroutine calls, jumps, and argument passing, and thus induces a large amount of pipeline conflicts to code execution. We experienced that the code optimizer is able to avoid such conflicts only in some more or less trivial cases.[11]

Figure 2. Circuit A19 Simulation Times for Original Port



Legend:
- W Wirrer's original Hydsim-port
- A Optimized version without assembler-optimizations
- B Optimized version with graph-traversal modules hand-optimized
- C Optimized version with graph-traversal and Newton-modules hand-optimized
- D Version B with DT5C enabled
- E Version C with DT5C enabled

Parallelization

Static Partitioning

The initial port of Hydsim to the Hydra maintains all data required for simulation in the board's large bank of shared memory.[12] This approach allows both network equation solving and behavior simulation of hydraulic elements to be easily parallelized when applying *dynamic task attraction* with the DSPs working in a master-slave configuration. In the case of Hydsim, the validity of this principle relies on all DSPs taking part in the simulation having access to *all of the most recent* computation results.

Since blocks of data simultaneously accessed by the DSPs do not overlap, there are no race conditions to take into account. However, accesses to the board's shared DRAM are much slower intrinsically than accesses to the private SRAM-banks of the DSPs.

These accesses constitute a large sequential portion within concurrent execution. This is because memory is shared over a common bus (the internal shared bus, or ISB) that is automatically arbitrated by an on-board bus controller.

A series of measurements and an extensive theoretical treatment both confirmed our expectations that shared memory becomes a serious bottleneck in parallel simulation. Our aim was to parallelize the optimized simulator's code in a way that shared memory is not used at all.

Overview

Dynamic task attraction relies on all DSPs having access to the entire data set of simulation, as tasks are distributed *dynamically* during each time step. By contrast, partitioning the network graph statically among the processing nodes requires the equation systems and hydraulic elements assigned to processing nodes to be computed prior to the start of the simulation on the multi-DSP platform. Because the graphs representing hydraulic circuits are always connected, partitioning the graphs requires communication among the DSPs involved in the simulation.

Hence, we were faced with the problem of extending the preprocessor by a partitioning algorithm with the following two main general characteristics:

- The communication among the processing nodes should be minimized.
- The utilization of processing power should be maximized during each time step.

Fulfilling both these requirements is difficult, if not impossible. First, the hydraulic circuit graph unpredictably can change its structure anytime during simulation in response to certain events within the circuit. These events can be either user-specified, induced internally, or both.

This implies that any valid (that is, invariant during runtime to ensure minimal communication activity) partition is generally a superset of the components that can be treated in parallel at the moment the graph is being partitioned. The graph's components are uniquely determined by the scheme used to classify its edges. Each change in the graph's structure induces a new classification. With our static partitioning approach, classifications can be determined in parallel.

Secondly, network equations are solved *iteratively* by a variant of Newton's algorithm, in which the speed of convergence depends on the numerical values involved and, hence, is unknown in advance.[13]

As a consequence, our partitioning algorithm has to find those pieces of the network graph that will *not* change their border structures when their internal components reshape dynamically. Further, the algorithm has to somehow estimate the computational complexity of each piece to determine a mapping of pieces to processors. The mapping minimizes DSP idle times that are induced by synchronization at the end of each time step.

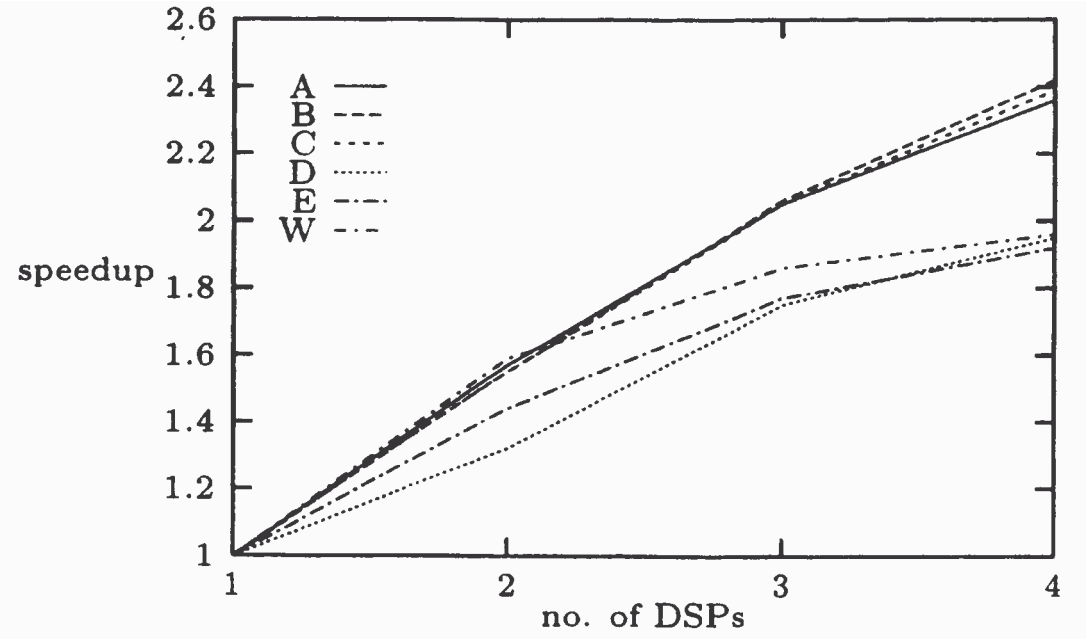
All four DSPs on the Hydra are interconnected by high-speed communication links. Our mapping algorithm makes use of this feature. However, it can easily be extended to consider restrictions imposed by other topologies. Moreover, this is supported by our expectation that circuit graphs can be rather sparse, since practically used and physically feasible hydraulic circuits will never be heavily meshed.

Parallelization Results

We already provided a measure for the efficiency of the single DSP version of the optimized simulator kernel in *Optimization Results.* The quality of parallelization schemes is usually judged by the speedups attained by actual implementations when executed on a specific hardware platform.

Figure 3 shows the speedup diagram obtained by simulating our test circuit with the simulator versions. Although the optimized versions generally attain somewhat higher speedups than the original port, the curves clearly deviate from linearity. As discussed in *Load Balancing*, this is a logical consequence of partitioning the computational load to the processors statically rather than dynamically.

Figure 3. Circuit A19 Speedup Curves for Static Partitioning Versions



Legend:
- W Original Hydsim-port
- A Optimized version with static partitioning and no assembler-optimization
- B Optimized version with static partitioning and graph-traversal modules hand-optimized
- C Optimized version with static partitioning and both graph-traversal and Newton-modules hand-optimized
- D Version B with DTSC enabled
- E Version C with DTSC enabled

Load Balancing

The key advantage of partitioning the network graph statically is that it enables the exclusive use of the fast local memory banks of the DSPs. However, the main drawback of this approach was known in advance: the unpredictable computational complexity of solving independent network equations meant that the processing nodes may have to idly wait for the others to complete their computation tasks.

Because of the numerous advantages the private memory model offers compared to the shared memory implementation, we were therefore striving for a private memory version of Hydsim that would replace the strictly static distribution of equation systems by a more dynamic approach.

Unifying Static and Dynamic Partitioning

To successfully adapt load balancing to the private memory version, one must ensure that the local memory blocks of all DSPs are updated with recent computation results. The most important restriction on the design of the new version is that the amount of data to be transferred among the DSPs during the memory update phases must be as small as possible. Since network equations are assigned dynamically to the processors instead of statically, the following information must be made public to all processing nodes:

- Public structural information
  - All data items concerning the current classification of the network graph
  - Updated only after the graph has been newly classified
- Public behavioral information
  - All object data that are possibly modified when network equations are solved
  - It is important to find the *minimal* set of items that belong to it, since this group comprises all data that have to be exchanged among DSPs whenever equation systems have been solved.

However, one phase remains that must be executed during each time step: the dynamic update of hydraulic elements; that is, the simulation of the time dependent behavior of the elements. Although they are entirely independent of one another, it is not wise to distribute the tasks of dynamic updates dynamically, since this requires updating *all* object data that may be modified throughout the simulation. Even for small circuits containing only a few hydraulic elements, these object data sets comprise considerable amounts of words.

The key issue is that the dynamic update methods of objects representing hydraulic elements take predictable computation times. Furthermore, even the most complicated dynamic update process, which is that of a cylinder, is computationally far less expensive than solving the network equations of an average sized graph component.

Consequently, we can still distribute the pieces of the network graph statically to the processing nodes using estimates of the total computational complexity of the dynamic update procedures of each piece's hydraulic objects as a measure of the piece's weight. In other words, the preprocessor assigns a well-defined set of the pieces of the network graph to each processor to statically distribute the dynamic update tasks.

Unifying load balancing and static partitioning using this approach seems efficient because

- Private behavioral information remains local to each DSP.
- The possibility is preserved of classifying the network graph in parallel.

The principle of dynamic task attraction utilized to balance the load of solving equation systems introduces an asymmetric relationship of master and slaves among the DSPs involved in the calculation.

To minimize the cost of the increased communication activity, we extended the preprocessor to organize data in a way that permits communication to be driven by the internal DMA coprocessors of the DSPs, which may pursue their tasks independently of CPU operation.

The CPUs use the communication links only to transmit initialization data to remote coprocessors. Their activities are synchronized by the operation of the DMA channels. To obtain fast access to both public and private information, we extended the Hydsim's object-oriented principle of representing hydraulic elements to directly support public and private data segments. In addition, this improvement greatly alleviates the integration of new hydraulic object models into the existing kernel.

Utilizing Shared Memory

The private memory version discussed in *Unifying Static and Dynamic Partitioning* combines the advantages of keeping the whole data set needed for simulation local to the DSPs while solving equation systems by dynamic task attraction. Although the amount of transferred data during each time step is minimized, this scheme runs the risk that calculation costs may be exceeded by communication overhead. This is especially true in systems with more than four processing nodes.

Consequently, we have extended the private memory load balancing version of Hydsim to optionally maintain behavioral information in shared memory. This makes all data transfers unnecessary during the most time-critical phases of equation solving and dynamic update. Transfers of the locally stored public structural information still have to be carried out after the graph has been newly classified.

Load Balancing Results

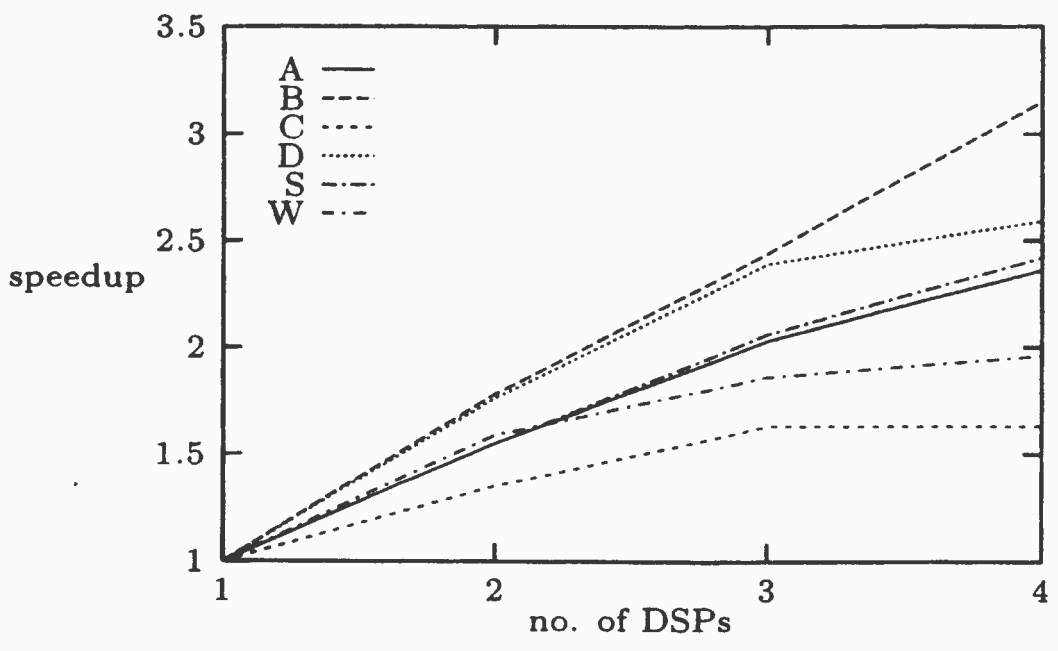
This section compares the speedups of the two load balancing versions of Hydsim introduced previously. Figure 4 clearly shows that version A's speedup is dramatically limited by the communication activity required to update public behavioral information after each time step. The more processors involved, the more it drops below even the static partitioning version's speedup curve.

At the other extreme, version B's speedup is absolutely superior. Recall that version B differs from version A only in the location of public behavioral information. This proves that shared memory, when cleverly used, does not necessarily introduce a bottleneck in parallel applications.

Comparison and Discussion

The most striking aspect revealed by the examples presented and the numerous other measurements we undertook is that the load balancing versions generally outdo the otherwise equal static partitioning version with regard to speedup.

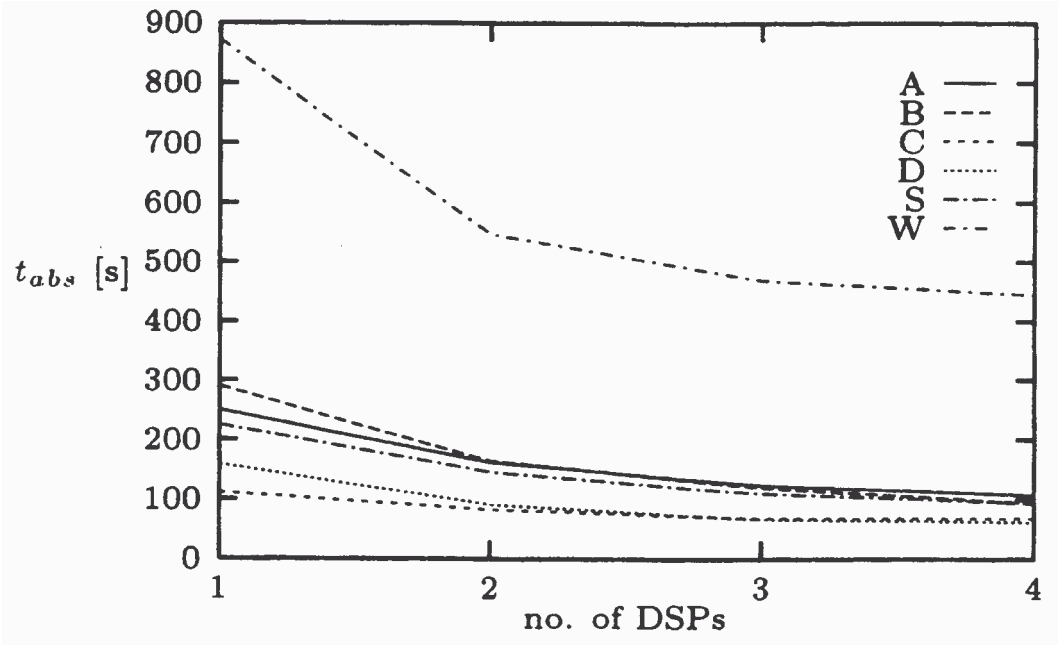
Figure 4. Circuit A19 Speedup Curves for Load Balanced Versions



Legend	W	Original Hydsim-port
	A	Optimized version with load balancing, both public behavioral and public structural information in local memory, and graph-traversal module hand-optimized
	B	Optimized version with load balancing, public behavioral information in shared memory, public structural information in local memory, and graph-traversal module hand-optimized
	C	Version A with DTSC enabled
	D	Version B with DTSO enabled
	S	Optimized version with static partitioning and graph-traversal modules hand-optimized

With the number of DSPs involved in the simulation increasing, the speedup factors of the load balancing versions exceed those of version S, which uses static partitioning.

Figure 5. Absolute Simulation Times for Circuit A19



Legend
- W Original Hydsim-port
- A Optimized version with load balancing, both public behavioral and public structural information in local memory, and graph-traversal module hand-optimized
- B Optimized version with load balancing, public behavioral information in shared memory, public structural information in local memory, and graph-traversal module hand-optimized
- C Version A with DTSC enabled
- D Version B with DTSO enabled
- S Optimized version with static partitioning and graph-traversal modules hand-optimized

With only one or two DSPs employed, static partitioning provides the best results. The difference in behavior is especially drastic between versions S and B. As shown in Figure 5, for circuit A19 and one DSP version, the execution time of version B exceeds that of version S by more than one minute.

Nevertheless, version B will end up with the same simulation time as version S if the same circuit is simulated on all four DSPs with the same control events. Clearly, this situation is impressively reflected in the respective speedup factors shown in Figure 4 (2.42 for version S and 3.15 for version B).

We may draw the conclusion that combining static partitioning and load balancing works best for parallelization. Although load balancing with the exclusive use of private memory provides comparable results, the speedup is obviously limited by both the higher communication load and the larger overhead required to set up communication.

Figure 4 indicates that the speedup grows linearly for a circuit of acceptable size, as the theoretically ideal case suggests. Most importantly, we may expect this curve to continue its growth if even more than four DSPs are used for simulation, since the speedup of this version is primarily limited by the physical bandwidth of the ISB. Because we have cut ISB accesses to a minimum, this limit is out of reach for circuits intended for simulation by Hydsim.

Furthermore, the results confirm H. Braun's theoretical result that real-time hydraulic simulation cannot be achieved with the given hardware platform, although we believe we have come as close as possible to achieving this ultimate aim.[14]

Further Work

WEDSIM

To ease the testing and analysis processes and to make software compliant with modern user-interface standards, the optimized Hydsim package is integrated into a comfortable, object-oriented graphical construction and simulation environment called WEDSIM. WEDSIM is currently being developed at the Institute for Technical Informatics. The main part of this work involves:

- Implementing a powerful interface that links the output of the simulation environment with the Hydsim preprocessor and kernel.
- Extending the environment by the hydraulic domain; that is, constructing hydraulic elements and sample hydraulic networks.

The concept of preprocessor and kernel greatly supports the extension of Hydsim by such interfaces, since only the preprocessor is affected.

Hardware Extension

E. Brenner conceived and C. Mandl implemented a hardware extension that should allow the simulation of shared memory with private memory for applications where parallelism is coined by independent data regions that are modified at any moment of concurrent execution.[15] [16] [17] Basic operation periodically unifies the contents of private memories via the DSP's external communication ports.

The hardware's unification process lies in a simple minority vote over the memory words arriving from the input ports and a subsequent broadcast of the winning word to all output ports. (The winning word is distinct from all others and thus has been changed.)

The load balancing versions of our simulator have data organized in a way suitable for optimal support of this hardware extension. This is the result of consequent hardware-software codesign pursued throughout the research process. Unfortunately, hardware compatibility problems have hindered attempts to include the module in our tests.

Conclusions

We believe that the research supporting the thesis of this application report is based on in the consequent practical application of *general* optimization, partitioning, and scheduling paradigms to one *stereotype* piece of simulation software.[18] This software is to be executed in a *typical* multiprocessor environment:

- Optimizations range from careful considerations on the highest (algorithmic) level to improving code on the lowest (assembly language) level. None of the optimizations restrict the extensibility and portability of Hydsim; some even improve the software's flexibility and usability.
- Partitioning principles have been applied to software and network graphs.
- Scheduling paradigms have been implemented and evaluated in their static, dynamic and hybrid forms.

Due to the consequent use of compiler directives, all versions can be crafted from *one single* set of source modules (using the make-utility), such that Hydsim is still open for further extensions and improvements.

Researchers may be interested in the extensive evaluations and concise theoretical treatments provided for each simulator version. Students and practitioners will find a valuable resource of multiprocessor- and DSP-related programming skills.

References

[1] Andreas Riel. Parallelizing and Optimizing a Hydraulic Simulator Kernel in a Multi-DSP Environment. Master's thesis, Graz University of Technology, March 1995.

[2] E. Brenner, G. Neumann. and P. Seifter. *EDSim-H.* Festo Didactic KG, 1990.

[3] Kurt Schlacher. Systemanalytische Methoden für hydrodynamische Netze. Habilitationsschrift, Graz University of Technology, May 1990.

[4] Gerhard Wirrer. A Parallel Hydraulic Simulator Kernel Based on Multi-DSP Hardware. Master's thesis, Graz University of Technology, September 1993.

[5] Ariel Corporation, Highland Park, NJ 08904. *User's Manual for the V- C40 Hydra,* 1993. Document Version 0.4.

[6] E. Brenner and R. Weiß. Ein interaktives Entwurfs- und Simulationssystem für Hydraulikanwendungen. In F. Breitenecker, I. Troch, and P. Kopacek, editors, *Simulationstechnik,* pages 377-384. Vieweg, Braunschweig, September 1990.

[7] Andreas Riel. Parallelizing and Optimizing a Hydraulic Simulator Kernel in a Multi-DSP Environment. Master's thesis, Graz University of Technology, March 1995.

[8] E. Benjamin Wylie and Victor L. Streeter. *Fluid Transients in Systems.* Prentice Hall, Englewood Cliffs, New York, 1993.

[9] Texas Instruments, *TMS32OC4x: User's Guide,* preliminary edition, May 1991.

[10] Hans-Peter Messmer. *Pentium.* Addison-Wesley, Reading, Mass., 1994.

[11] Texas Instruments. *TMS32O Floating-Point DSP Optimizing C Compiler User's Guide,* October 1991.

[12] Gerhard Wirrer. A Parallel Hydraulic Simulator Kernel Based on Multi-DSP Hardware. Master's thesis, Graz University of Technology, September 1993.

[13] Kurt Schlacher. Systemanalytische Methoden für hydrodynamische Netze. Habilitationsschrift, Graz University of Technology, May 1990.

[14] Helmut Braun. Bewertungsmethoden für ein Mehrprozessorsystem mit konstantem Auslastungsprofll. Master's thesis, Graz University of Technology, December 1994.

[15] E. Brenner and R. Weiß. A New Concept for Shared Memory Update in Parallel DSP and Transputer Systems. In F. Breitenecker and I. Husinsky, editors, *EUROSIM* '95, pages 279-284. Elsevier Science, September 1995.

[16] E. Brenner and R. Weiß. A New Shared Memory Concept for Multiprocessor Systems with High-Speed Communication Links. In *Proc. of the ISCA International Conference on Parallel and Distributed Computing Systems,* pages 210-215, September 1995.

[17] Christian Mandl. FPGA-Schaltung zum schnellen Update von shared memory Bereichen in einem Multi-DSP System auf der Basis TM5320C40. Master's thesis, Graz University of Technology. September 1994.

[18] Andreas Riel. Parallelizing and Optimizing a Hydraulic Simulator Kernel in a Multi-DSP Environment. Master's thesis, Graz University of Technology, March 1995.