Parallel Processing Research at OGC

David Maier

Oregon Graduate Center
Department of Computer Science
and Engineering
19600 N.W. von Neumann Drive
Beaverton, OR 97006-1999 USA

Technical Report No. CS/E 88-012

Parallel Processing Research at OGC

1. INTRODUCTION

We describe the research programs in the Department of Computer Science & Engineering at Oregon Graduate Center related to parallel processing. We also describe a department-wide research program on design management platforms. This work is included because it directly supports the creation of design environments for parallel programs and systems, and may require database and proof systems with parallel processing power.

This paper is divided into major programs: Large-Grain Data Flow, Graph-Reduction Architectures, Cognitive Architecture Project, and Platforms for Engineering Design, and smaller projects: Parallel Simulation of VLSI, DataCube, Parallel Debugging, and Parallel Prolog. We include a brief description of each project, along with a list of related papers, reports and theses. For further information on the major programs, there are four companion documents giving more detailed descriptions:

"Practical Support for Parallel Programming"
[Large-Grain Data Flow]

"A RISC Architecture for Symbolic Computation"
[Graph-Reduction Architectures]

"The OGC Cognitive Architecture Project: Silicon Implementation of Connectionist/Neural Networks"

"Platforms for Engineering Design"

2. MAJOR PROGRAMS

The research programs in this section have all been underway for a year or more, and each involves multiple doctoral students. The programs are described in chronological order of their inception.

2.1. Large-Grain Data Flow: High-Level Parallel Programming with LGDF2

Investigator: Robert G. Babb II

Parallel processing introduces a qualitative difference in architecture, which has proved difficult to manage by straightforward extensions of sequential programming technology. Large-Grain Data Flow (LGDF) is a model of computation that combines sequential programming with dataflow-like program activation. LGDF applications are constructed using networks of program modules connected by datapaths. Parallel execution is controlled indirectly via the production and consumption of data. With the aid of the LGDF Toolset, LGDF programs are automatically transformed for efficient implementation on a particular (parallel) machine.

2.1.1. Approaches to Parallel Programming

There is a sharp discontinuity in complexity between programming for single and multiple processor systems. Three currently popular approaches to parallel programming all have drawbacks: explicit coding of locks or message-passing lacks the high-level control structures that encourage abstraction, understanding, and architecture independence. Switching to new languages, such as SISAL, ML, or VAL, can allow easier compiler detection of potential concurrency, but at the expense of most of the software engineering experience we have gained for traditional imperative languages. Compiling standard sequential code with parallelizing compilers exploits only a small percentage of the parallelism available.

2.1.2. Large-Grain Data Flow 2

The Large-Grain Data Flow 2 (LGDF2) model provides a graphic language to combine individual sequential processes into a parallel program. Each of the processes is implemented in a standard high-level language (Fortran, C, etc.), compiled with a standard compiler, and will execute in a guaranteed-safe environment—one where all effects of other concurrently-executing processes are hidden from it. Thus, processes can be written without regard to parallel hazards (e.g., unsafe updates) and without explicit reference to hardware-level synchronization mechanisms (locks, events, monitors, barriers, etc.). These processes are composed into a network that specifies how and if they interact through I/O and control. Only restricted, "safe" interactions are allowed. The behavior (semantics) of these interactions is defined formally in the LGDF2 model, and in such a way that they can be implemented efficiently on either shared- or distributed-memory architectures. Because the individual sequential processes are well behaved and interact only through the network, the behavior of the overall parallel program can be understood by representing the processes by partial functions and analyzing the network. An LGDF2 network explicitly represents a parallel program's high-level design in a permanent, visible way.

The LGDF2 research group at OGC has investigated both theoretical foundations and practical aspects of this technology, and has built a series of prototype software tools to support our research. The main avenues of research are:

LGDF2 Language and Theory
: Networks are expressed with the LGDF2 language, a base language from which higher-level network constructs can be formed. We have already investigated abstractions corresponding to loops and subroutines with recursion. LGDF2's formal model provides a foundation for analyzing and verifying the behavior of LGDF2 networks and for studying the concepts of non-determinism, fairness, computational power, decision power, and time complexity in parallel programs.

Dusty-Deck to LGDF2 Conversion Tools
: For LGDF2 to be useful for the existing base of older scientific programs, tools must exist to help convert them. Conversion is easier than a total rewrite, since the base sequential language can be preserved. We are involved in a joint research project with David Klappholz, Stevens Institute of Technology, to investigate application of Refined Language tools to aid in this task. We have used LGDF techniques to restructure medium-sized parallel physics codes.

Schedulers for Various Architectures
: We have developed a scheduler for LGDF2 on a shared memory multi-processor, the Sequent Balance, which should serve as a basis for other shared-memory processors. We expect a distributed-memory scheduler to be more difficult to implement, though the LGDF2 model leaves many options for optimizing available bandwidth.

Cooperative Software Engineering
: LGDF2 networks specify and illustrate all possible interfaces between processes. This feature of the model can serve as the basis for a large-scale multi-user software engineering tool to automatically prevent concurrent editing of related process code by multiple programmers.

LGDF2-Specific Architecture
: The LGDF2 computation model has attributes which enhance performance on certain parallel architectures. The model provides maximum flexibility in where/when/what process to schedule next. Processes never block, so pre-emption is unnecessary. No state needs to be saved when a process finishes. Hardware locking for processes is not required, since they are event driven.

2.1.3. LGDF References

Books

R. G. Babb II (ed. and author), *Programming Parallel Processors,* Addison-Wesley, 1987.

Articles

R. G. Babb II, "Parallel processing with large-grain data flow techniques," *Computer 17,* 7, July 1984.

R. G. Babb II, "Programming the HEP with large-grain data flow techniques," in *MIMD Computation: HEP Supercomputer and its Applications* (J. S. Kowalik, ed.), MIT Press, 1985.

R. G. Babb II, "A data flow approach to unifying software specification, design, and implementation", in *Proc. Third Int. Workshop on Software Specification and Design,* 1985.

R. G. Babb II, "Parallel processing on the CRAY X-MP with large-grain data flow techniques," in *The Book of Supercomputers* (S. Fernbach, ed.), North-Holland, 1986.

R. G. Babb II, L. Storc, and W. Ragsdale, "A large-grain data flow scheduler for parallel processing on CYBERPLUS," *Proc. 1986 Int. Conf. on Parallel Processing.*

R. G. Babb II and R. Hamlet, "Proposal for a software Design Control System (DCS)", in *Proc. Fourth Annual Pacific Northwest Software Quality Conference,* Nov. 1986, pp. 265-271.

D. C. DiNucci and R. G. Babb II, "Practical Support for Parallel Programming", in *Proc. 21st Hawaii Int. Conf. on System Sciences,* Jan. 1988. *Software Track,* Jan. 1988, pp. 109-118.

R. G. Babb II and D. C. DiNucci, "Design and implementation of parallel programs with Large-Grain Data Flow", in *The Characteristics of Parallel Algorithms* (ed. by L. H. Jamieson, D. B. Gannon, and R. J. Douglass). Cambridge, MA: The MIT Press, 1987, pp. 335-349.

Reports

R. G. Babb II and L. Storc, "Parallel Processing on the Denelcor HEP with Large-Grain Data Flow Techniques", OGC Tech. Report No. CS/E 85-010, May 1985.

R. G. Babb II and R. Hamlet, "Software Design Revision Control, or, How to Keep Too Many Cooks from Spoiling the Broth", OGC Tech. Report CS/E-88-004, Jan. 1988.

D. C. DiNucci, "A Formal Model of Parallel Computation: LGDF2", OGC Tech. Report No. CS/E-88-006, Jan. 1988.

R. G. Babb II and D. C. DiNucci, "A Parallel Architecture Based on LGDF2", OGC Tech. Report no. CS/E-88-008, Jan. 1988.

D. C. DiNucci, "Building LGDF2 Nets from High-Level Constructs", OGC Tech. Report No. CS/E-88-009, Jan. 1988.

Theses

L. Storc, "Parallelization Schemes for 2D Hydrodynamics Codes Using the Independent Time Step Method", M. S. Thesis in preparation. A paper on this work (with R. G. Babb II and P. G. Eltgroth) was presented at Vector and Parallel Processors in Computational Science III, Liverpool, UK, Aug 1987, to appear *Parallel Computing.*

2.2. Graph-Reduction Architectures

Investigator: Richard B. Kieburtz

A reduction architecture evaluates an expression by transforming it through a series of intermediate forms until it cannot be further transformed, under a set of rewrite rules. The expression is then said to be in normal form, which is the value attributed to the original

expression. In a *pure reduction* system (such as beta-reduction, combinator reduction or string reduction), the control—the selection of the next reduction step—is derived dynamically from the form of the current expression. An alternative is *programmed reduction*, in which the steps of a computation are still applications of reduction rules, but where control is derived from a static analysis of the original expression.

By using a representation for an expression that is a graph, rather than a simple string, multiple occurrences of a common subexpression are captured as multiple references to a common subgraph. *Graph reduction* refers to a reduction process in which expressions are represented as such graphs, which avoids redundant re-evaluations of a common expression. In a graph-reduction architecture, a reducer routines walks though the graph of an applicative expression to locate a redex (a subexpression for rewriting), chosen according to the computation rule followed. A graph-rewrite rule is then applied to replace the redex with a subgraph representation of its value. Other parts of the graph that referenced the redex now have access to this value. In a programmed graph-reduction architecture, the reducer executes a program derived from the definition of the function used at the redex to perform the replacement.

The G-machine is a particular architecture for programmed graph reduction. It was defined by Thomas Johnsson and Lennart Augustsson (Gothenburg) as the evaluation model for a compiler for lazy ML (LML). In it, the reduction step is quite efficient, as the program computes in terms of a stack of pointers (the P-stack) into the expression graph.

2.2.1. The G-Machine

The original G-machine architecture was realized as a virtual machine that interpreted instructions (G-code) generated by the LML compiler. The G-machine project at OGC is producing a hardware design for this architecture, realized on a single-board subsystem composed of custom VLSI chips. The principal features of the design are hardware support for graph traversal, a vertically microcoded, pipelined internal architecture, an instruction fetch and translation unit with very low latency, and a new memory architecture specifically suited to graph reduction and very large memories.

It is the memory architecture in which the G-machine exhibits the most parallelism. The strategy is to provide a storage module that behaves as a dynamically allocatable, list-structured memory. This goal is achieved by having a G-memory with its own processor that garbage collects and allocates free list nodes in parallel with the reduction processor. The garbage collection is based on a modified reference-counting scheme that works in the presence of graph cycles, and requires time proportional to the size of structure collected, not the total amount of memory in use.

The G-machine design has been extensively simulated at the macroarchitecture level, and most components have been simulated at lower levels of detail. The simulations have been carried out for several different computation rules: lazy evaluation, strict evaluation and call-by-value. The general conclusion is that the G-machine has the relative performance of a 3.6 Mips VAX, using conservative assumptions about the VLSI technology used. Another observation is that the G-memory as currently designed is fast enough to collect and allocate list nodes without any waiting by the reduction processor.

2.2.2. G-machine References

Articles

R. B. Kieburtz, "The G-machine: a fast, graph-reduction evaluator," *Proc. of IFIP Conf. on Functional Prog. Lang. and Computer Arch.*, Sept. 1985.

R. B. Kieburtz, "Performance evaluation of a G-machine implementation," In *Graph Reduction*, (R. B. Keller and J. H. Fasel, eds.), Springer-Verlag, LNCS series, 1987

R. B. Kieburtz, "The G-machine: an architecture for symbolic processing," *Proc. MCC University Research Symposium,* Austin, July 1987.

R. B. Kieburtz, "A RISC architecture for symbolic computation," *Proc. ASPLOS II,* Palo Alto, Oct. 1987.

Reports

R. B. Kieburtz, "Incremental collection of dynamic, list-structure memories," OGC Tech. Report CS/E-85-008, Feb. 1985.

Theses

A. G. Sarangi, "Simulation and performance evaluation of a graph reduction machine architecture" M. S. Thesis, July 1984.

M. H. Foster, "Design of a list-structure memory using parallel garbage collection," M. S. thesis, Sept. 1985.

Richard P. Vireday, "Bit slice design of a graph reduction processor," M. S. Thesis, May 1986.

L. J. Rankin, "A dual-ported real memory architecture for the G-machine," M. S. thesis, Aug. 1986.

R. Tenneti, "A retargetable code generator using a machine description table and attributes" M. S. thesis, Dec. 1986.

W. Hostmann, "An examination of designs for the instruction pipeline of the G-Machine," M. S. Thesis, Jan. 1988.

S. L. Kuo, "A high performance instruction fetch and translation unit for the G-processor of the G-machine," M. S. Thesis, Jan. 1988.

2.2.3. The Parallel Graph Reduction (PGR) Project

The PGR project is a relatively new program to map graph-reduction computation to a distributed memory multiprocessor. The PGR system will run on an Intel iPSC Hypercube, roughly as a parallel emulation of the G-machine. It will be programmed in a new language, F+L, designed at OGC. This language combines functional- and logic-programming semantics, and includes parallel constructs. The PGR compiler for F+L generates G-code (G-machine instructions), which is then translated into 80286-code and linked to a run-time system for the cube nodes being written in-house.

This work focuses on fine-grain concurrency, in contrast to the LGDF project. A virtual processor is allocated to every reducible expression in the program graph. Virtual processors are multiplexed onto physical processors. Reduction of a single expression is very simple, so it is possible to use "flyweight" tasks in which the cost of a context switch is extremely low. At each node, task scheduling is viewed as an instruction pipeline, with tasks waiting for communications shunted to a side pool. When there are more tasks than are needed to fill the pipeline, the excess tasks are offloaded to other processors, using diffusion scheduling. Diffusion scheduling is a heuristic method for dynamic distribution of workload in a multiprocessor system, with control of load balancing distributed among all the participating nodes.

To prevent expressions whose value is never needed from "stealing" too much processor time from useful expressions, expressions for reduction are tagged with a priority. At each node, reductions are scheduled according to this priority. There are a number of aspects of the concurrent reduction scheme explored. One significant factor is the volume of useless work that will result from speculative evaluation. We hope to characterize the proportion of useless work from various problem classes to decide when speculative evaluation is profitable. The effects of communication limitations are also of interest. The design is intended to minimize the performance penalty from latency in data exchange among application tasks. However, delays can

indirectly affect the system, forcing the diffusion scheduler to work with old information.

2.2.4. PGR References

Reports

R. B. Kieburtz, "Functions+Logic in theory and practice," OGC Tech. Report CSE-87-002, Feb. 1987.

Theses

B. Schaefer, *Massive Asynchronous Concurrency Through Parallel Combinator Reduction,* Ph.D. Thesis in preparation.

2.3. The Cognitive Architecture Project (CAP)

Investigator: Daniel W. Hammerstrom

Important practical problems are often incompletely specificed and characterized by many weak constraints requiring large solution spaces. Visual processing and speech recognition are problems of this sort. For example, speech processing is difficult because speech is a high dimensional data space with a large number of subtle, loosely connected, spatial and temporal stochastic relationships (correlations) among the data elements. Due to differences in individual voices and speech styles, the information in the speech stream is primarily contained in these high-order correlations. Processing this large data space overwhelms traditional sequential or low-level parallel computation. Inexpensive hardware is required for most envisioned applications, thus limiting even further the computational power available for typical speech processing applications. (At $1,000,000 per unit, you cannot put a Connection Machine on a secretary's desk for speech processing.)

2.3.1. Massively Parallel Neural Network Emulation Using ULSI

Recently, a new style of massively parallel computation has been receiving attention as a candidate for processing these large dimensional spaces. Massively parallel networks, often called *connectionist* or *neural network* models, are characterized by a large number of highly connected, simple processors, and are loosely based on biological information processing systems. A variety of neural network computational structures have been proposed for modeling such cognitive phenomena as visual image analysis and speech recognition. Though still experimental, the results are encouraging: even simple models have demonstrated great power at analyzing complex, stochastic data spaces.

Researchers need to emulate ever larger networks. However, they are approaching the computational limits of existing computer hardware. To some degree this computational wall is due to the extreme connectivity of these networks (every node in even a moderate sized network has hundreds of input and output connections). Since the number of connections grows geometrically, and since the number of processors grows linearly, the emulation of large networks with traditional architectures becomes impractical for even moderately sized networks.

The Cognitive Architecture Project (CAP) at OGC was founded in 1985 to examine radically new silicon-based computing structures that are optimized for the emulation of connectionist and neural network architectures. The project was based on several basic assumptions:

(1) Real applications of neural networks will require the emulation of large networks in close to real time (equivalent to their biological counterparts) with inexpensive hardware.

(2) Only existing, state-of-the-art technology is to be considered, in particular CMOS, since neural networks will only be commercially viable when implemented in cheap, mass producible technology. Because of its ease of use and functional density, existing CMOS provides a significant cost/performance advantage for the next decade. This constraint

precludes GaAs and a number of optical technologies.

(3) The emulation of large networks with CMOS technology will require ultra-large-scale and possibly even wafer-scale integration. This level of integration will be possible because of the inherent fault-tolerance of neural network models.

Over the last two years we have developed a set of hybrid analog/digital technologies for solving efficiently the connectivity and computational problems. Several prototype chips have been designed and fabricated by MOSIS, and four patent applications submitted (additional patents are in preparation).

In addition to our VLSI architecture work, we have developed a neural network simulation environment for the Intel iPSC Hypercube at OGC. This software system includes a comprehensive network compiler that compiles a network description language, NDL, to an intermediate form, BIF (Beaverton Intermediate Form), a network partitioning facility that maps parts of the BIF network to each iPSC processor, and a fault simulation program that injects faults probabilistically into the BIF description of the network. The fault simulation capability is used to understand the effect of typical silicon fabrication faults on network performance. The most important part of the system is the ANNE (Another Neural Network Emulator) system that reads in the partitioned (and possibly faulted) BIF file and generates an iPSC simulation system. We also have a silicon architecture emulator that reads in the BIF file and emulates our physical architecture models on the iPSC.

In addition to silicon architecture, we have also studied network model scaling issues (in both learning time and connectivity), and are developing a number of applications: magnetic and optical character recognition, speech recognition, and simple dynamic control of a moving object. By studying applications in parallel to hardware design, we can more realistically simulate our silicon architectures and provide feedback on applications requirements for architecture functionality. Real applications also provide a context for the network scaling studies.

We are now exploring with several industrial partners the sponsorship of the design and fabrication of a small single board computing system using chips based on our architecture. This board would emulate moderate sized networks at several billion connection updates per second for a range of algorithms.

2.3.2. CAP References

Articles

D. Hammerstrom, S. Thakkar and D. Maier, "The OGC Cognitive Architecture Project," SIGARCH, *Computer Architecture News 14*, 1, Jan. 1986.

D. Hammerstrom, "Connecting with the human mind," *OGC Visions Magazine*, Winter 1987.

R. D. Geller and D. W. Hammerstrom, "A VLSI architecture for a neurocomputer using higher-order predicates," *Proc. Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence*, Oct. 1987.

Reports

J. Bailey and D. Hammerstrom, "How to make a billion connections," Jim Bailey and Dan Hammerstrom, Tech. Report CS/E-86-007, July 1986.

D. Hammerstrom and R. Kravitz, "A design methodology for high-performance, general-purpose VLSI," Tech. Report CS/E-86-008, July 1986.

D. Hammerstrom, "The connectivity analysis of a class of auto-associative connection networks," Tech. Report CS/E-86-009, Aug. 1986.

D. Hammerstrom, C. Bahr, J. Bailey, T. Baker, G. Beaver, K. Jagla, J. Mates, N. May, H. McCartor, M. Rudnick, "The OGC Cognitive Architecture Project: Silicon implementation of

connectionist/neural networks," Cognitive Architecture Project White Paper, 1987.

D. Hammerstrom, "The connectivity requirements of simple association, or How many connections do you need?", 1987 IEEE Conference on Neural Network Information Processing, Poster Session.

M. Rudnick, "VLSI implementation considerations of the broadcast hierarchy interconnect architecture," Tech. Report, January, 1988.

Theses

R. Geller, "A VLSI architecture for a neurocomputer that uses higher-order predicates," M. S. Thesis, Apr. 1987.

N. May, "Fault simulation of a wafer-scale integrated neurocomputer," M. S. Thesis in preparation.

J. Bailey, "*A VLSI Interconnect Structure for Neural Networks,*" Ph.D. Thesis in preparation.

C. Bahr, "ANNE, a general-purpose neural network emulator for the Intel iPSC Hypercube," M. S. Thesis in preparation.

K. Jagla, "A broadcast hierarchy simulator for the Intel iPSC," M. S. Thesis in preparation.

2.4. Platforms for Engineering Design

Investigators: Robert G. Babb II, Goetz Graefe, Richard G. Hamlet, Daniel W. Hammerstrom, Richard B. Kieburtz, David Maier

Engineering design is one of the applications in which computer support is indispensable: were CAD tools stripped away, many modern design efforts would be literally impossible. But as machines have extended the range of the possible, new demands are made for yet more support. The hallmarks of state-of-the-art design are complexity and change. A large real-time embedded computer system, for example, requires the dedicated attention of dozens of people over years of time, and the resulting software may amount to millions of lines of code. During development, thousands of decisions are made that significantly alter the course of the project, and many of these decisions require modifications to past efforts on the design. Existing computer design systems are composed of isolated software tools, with human beings performing the actions necessary to interface each to a particular design, and to each other. Proposals for "integrated environments" today are typically no more than the definition of standard interfaces so that tools can communicate more easily and with fewer errors. Existing and proposed systems do not address the question of controlling and facilitating change, nor the problem of *managing* complexity; at best they seek to *record* it.

The principles of design are not so very different in different applications such as VLSI and software engineering; nor do they differ much among projects within one application. Yet methods are usually tailored to each project, with a high cost in human organization and a high potential for error. By investigating *generators* for design environments, rather than the environments themselves, we believe that common principles can be identified, yet the final environments adapted to the details of each project. We call the collection of methods and generic tools necessary to generate an environment, a Design and Abstraction Management Platform (DAMP). Its components are:

(1) Design methods of refinement and representation derived from data abstraction in the programming-language discipline.

(2) The formalism of constructive second-order logic and its proof theory, to express and verify properties of designs.

(3) An object-oriented database capable of the performance needed to support complex objects and their dependencies.

(4) Generic tools and tool generators built upon these foundations, which can be easily tailored to particular applications, and even to particular designs.

3. SMALL PROJECTS

The projects in the section are of smaller scope or are just in their early stages.

3.1. Parallel Simulation

Investigators: Daniel W. Hammerstrom, William Bain (Intel Scientific Computers and Block Island Technologies)

Dr. Bill Bain has created an efficient, object-oriented, multi-tasking computing environment called *Interwork* (copyright Block Island Technologies). Interwork provides large numbers of small, lightweight processes. To create a computational framework, there are a number of powerful intertask communication and synchronization primitives. Interwork is used in a number of research projects and courses at OGC. For example, in conjunction with Block Island Technologies, we have developed a Register Transfer Level simulation environment (Microsimulator), called BitSim, that is used in the Advanced VLSI Design class and several research projects at OGC.

Interwork has been available on a large number of sequential machines for some time. Recently, Block Island Technologies has ported Interwork to the Sequent Balance and Symmetry systems and to the Intel iPSC Hypercube. The hypercube port is particularly attractive, since it provides transparent multiprocessing of applications on the cube. For example, our existing BitSim programs will run directly on the cube without modification.

3.2. DataCube: A Cube-Connected Database Machine

Investigators: Goetz Graefe, Leonard Shapiro (Portland State), Joseph Brandenburg (Intel Scientific Computers)

The performance of highly parallel database machines depends on the bandwidth between permanent storage devices and the processing elements. Through the addition of I/O facilities and file system software to individual nodes in hypercubes, this architecture becomes a suitable vehicle for database applications. We are currently working on the design and implementation of a new database machine called DataCube. The goal of the design is to provide fast response for small transactions and high throughput for large queries by applying dataflow techniques. Each file is distributed over some or all disks, and can be partitioned using round-robin-, key-range-, or hash-partitioning. When optimizing database queries, data distributions and correlations are taken into account to reduce the amount of data transferred between processors, giving preference to transfers between neighboring nodes. Range queries and exact-match queries on partitioning attributes are evaluated only on the required nodes, an important feature for processing small update and retrieval transactions.

The purpose of the DataCube project is threefold: to explore designs for a scalable, high-performance database machine on a hypercube, to develop query optimization techniques for architectures that have a concept of "near" and "far" between processors, and to investigate building a powerful and flexible Integrated Data and Compute Server for a workstation environment. The design builds on the experience of the GAMMA project, in which Goetz Graefe participated at the University of Wisconsin.

Baru and Frieder introduced data dynamic re-distribution in their hypercube database machine design. Rather than using dynamic re-distribution, we concentrate on reliable optimization and estimation techniques to avoid the need for dynamic re-distribution. Bratbergsengen is currently porting single operator algorithms from the Cross-8 database machine to hypercube connected hardware, with justified expectations of excellent performance. While the individual algorithms are basically the same, the DataCube effort goes far beyond this work as it includes complex queries, concurrent queries, and query optimization.

3.2.1. DataCube References

Articles

D. J. DeWitt, R. H. Gerber, G. Graefe, M. L. Heytens, K. B. Kumar, and M. Muralikrishna, "GAMMA - A high performance dataflow database machine," *VLDB,* Kyoto, Japan, Aug. 1986.

G. Graefe, "Software modularization with the EXODUS optimizer generator," *IEEE Database Engineering,* Nov. 1987.

Reports

G. Graefe, "Selectivity estimation using moments and density functions," Nov. 1987.

G. Graefe and L. D. Shapiro, "DataCube: A cube-connected dataflow database machine," submitted to the International Conference on Parallel Processing 1988.

G. Graefe, "Query processing in DataCube, a cube-connected dataflow database machine," Feb. 1988.

Theses

G. Graefe, "*Rule-Based Query Optimization in Extensible Database Systems,*" Ph.D. Thesis, University of Wisconsin, Madison, Aug. 1987.

3.3. Parallel Debugging

Investigators: Richard G. Hamlet, Robert G. Babb II, Shreekant S. Thakkar (Sequent)

Computer architects have found in parallel processing the medium to exploit today's VLSI technology, which is approaching theoretical and technological speed limits. Gate times cannot easily be reduced, but it is easy to add processors. The software to control parallel hardware is notoriously difficult to write and debug. The underlying model of concurrent computation, communicating processes, is so low-level that programmers find it extremely difficult to retain intellectual control of their designs. Parallel programming has been supported largely by modified conventional languages, with added hardware-level primitives for communication. These languages are used because they accept existing sequential programs. They are error-prone partly because the parallel versions are poorly defined and poorly implemented; a more lasting difficulty is that they follow the difficult low-level model too closely.

Conventional debugging techniques have been uneasily adapted to parallel languages, but they do not address the significant new problems of concurrent computation. These arise from the information explosion of multiple instruction streams overlapping in time. In particular:

(1) Conventional high-level debugging ideas such as variable monitoring and breakpointing do not provide enough information to be helpful. Message traffic is new important information.

(2) Brute-force methods such as tracing and snapshot dumps founder when the number of events and values multiply.

(3) Because the detailed sequencing of events is not fixed in parallel execution, bugs may be difficult to reproduce and study. Worse, the sequences that display bugs may not arise when code is instrumented for debugging.

(4) The information explosion makes data reduction and statistical processing crucial parts of presentation.

The most urgent need in parallel debugging is for better models of concurrent computation. One of our efforts, using the conventional process model, isolates each process, tests it independently of all others, and then induces representative patterns of cooperative execution. Dataflow scheduling of processes is a candidate for reducing the number of possible execution

sequences. The Large-grain Dataflow model (LGDF) shows promise both in understanding parallelism, and as a practical tool for design and programming.

The time-sequence aspect of parallel computation can be attacked using screen display techniques to hide much of the information explosion. If necessary, the animated playback of events can be driven by a trace so that it may be slowed down, replayed, or edited. Two preliminary studies of pictorial debugging systems have been presented as Master's research. However, without altering the computation model, far too much information must still be displayed and grasped by the debugger. Designs using LGDF have significantly smaller execution descriptions; indeed, the information required to "play back" a trace from the source is limited to the outcome of competition for input. A debugger is being developed to exploit this feature of LGDF, and to display results in terms of the high-level source description.

Finally, a probabilistic theory of testing shows promise of being able to compare parallel programming methods in terms of the number of test points needed to guarantee their reliability.

3.3.1. Parallel Debugging References

Articles

R. G. Hamlet, "Step-wise debugging," *ACM Symposium on High-level Debugging,* Pacific Grove, CA, Mar. 1983.

R. G. Hamlet, "Probable correctness theory," *Info. Proc. Letters 25,* Apr. 1987.

R. G. Babb II and D. C. DiNucci, "Design and implementation of parallel programs with Large-Grain Data Flow," in *Characteristics of Parallel Algorithms,* Cambridge, MA: MIT Press, 1987.

D. C. DiNucci and R. G. Babb II, "Practical support for parallel programming," *Proc. 21st Hawaii Int'l Conf. for System Sciences,* Jan. 1988.

Reports

R. G. Babb II, D. C. DiNucci, and L. Storc, "Multi-level monitoring of parallel programs," Tech. Report CS/E 87-013, Oct. 1987.

Theses

R. C. Brandis, "IPPM: Interactive Parallel Program Monitor," M. S. Thesis, May 1987.

3.4. Parallel Prolog

Investigator: Peter Borgwardt (Tektronix)

Several student projects on the parallel evaluation of logic programs have been conducted at OGC under the direction of Peter Borgwardt, an adjunct faculty member from the Imaging Research Laboratory of Tektronix. This work is an outgrowth of research started by Borgwardt at University of Minnesota, and an NSF grant has supported the continuation of it at OGC.

One project builds on Borgwardt's work on a parallel Prolog implementation for shared-memory multiprocessors. His method exploits stack-based execution, with pieces of the stack distributed among multiple processors. AND-parallelism is supported via processors maintaining lists of unsolved goals, which idle processors can examine for work. A compiler from Prolog to intermediate code was produced at Minnesota. The work at OGC produced a parallel interpreter for instructions in the intermediate code, which runs on a Sequent Balance multiprocessor. Several benchmarks were run through the interpreter, with the result that parallel speed up is achievable, but at sub-linear rates. The speed-up was very program and data dependent. On occasion, adding a processor resulted in longer elapsed time, because of added synchronization constraints for goal transfer.

Two other projects are looking at logic programming on distributed-memory multiprocessors. The first is developing an execution model for PARLOG, based on the work of Crammond, but extending the AND/OR-tree model to a processor network. The current plan is for an implementation of PARLOG on a network of INMOS transputers, with each transputer running two processes, one for message direction and the other an interpreter.

The other distributed-memory project concentrates on load-balancing in the OR-parallel execution of Prolog. Diffusion scheduling is dynamically distribution scheme for message-passing multiprocessor systems when static distribution is too expensive or ineffective. The effectiveness of diffusion scheduling is dependent upon the accuracy of each node's perception of its neighbor's load. The research seeks an analytical model of the effects of perceptive error on diffusion scheduling (a la Keller and Lin), to identify the sources and costs of perceptive error, and to determine how to reduce sensitivity to perceptive error. The model will be simulated on an Intel Hypercube to validate and tune it.

3.4.1. Parallel Prolog References

Articles P. Borgwardt, "Parallel Prolog using stack segments on shared-memory multiprocessors," *Proc. Int. Symp. on Logic Programming,* Feb. 1984.

Reports

D. Pase, "Load balancing heuristics and network topologies for distributed evaluation of Prolog," Tech. Report CS/E 87-005, April 1987.

B. Schaefer and P. Borgwardt, "A model for execution of PARLOG on a distributed processor network," Tech. Report CS/E 87-007, June 1987.

Theses

C. Hakansson, "The design and implementation of a parallel Prolog opcode-interpreter on a multiprocessor architecture," M. S. Thesis, Dec. 1987.