3

AHP: Advanced Hardware for PEIRCE

James D. Roberts[1]
Ron Demara[2]
Gerard Ellis[3]
Richard Hughey[4]
Robert Levinson[5]
Claude Noshpitz[6]

3.1 Introduction

PEIRCE is already being ported to a variety of platforms, both workstations and personal computers, by other researchers. The AHP group is exploring *advanced*, high performance hardware platforms. Our work focuses almost exclusively on parallel processing, including a system under development tailored for CG processing. The goal is to support PEIRCE with hardware platforms that can dramatically improve performance and in turn increasethe size of CG research domains and applications.

To date, we have been unsuccessful in obtaining information from applications that is necessary for evaluating processing needs and characteristics. We urge those who have gathered statistics (averages or distributions on number of concept-types, number of CGs, number of nodes and edges per graph, height and breadth of their hierarchies, etc.) for applications, or who have profiled runtimes versus problem size and characteristics to contact us. We would also like to obtain PEIRCE DBs in order to extract such information ourselves. Finally, we welcome others who would like to join this hardware group.

3.2 Objectives

Our objectives are to:

- Collect application characteristics and performance requirements, including: DB and individual graph characteristics, how application performance scales with problem size, and some target application sizes.
- Identify suitable hardware platforms to achieve the necessary speedups, emphasizing commercially available parallel systems.
- Develop and evaluate systems specifically tailored to CG processing.

[1]Dept. of Computer Engineering, Univ. of California Santa Cruz, Santa Cruz CA 95064, USA. donrob@cse.ucsc.edu
[2]Dept. of Electrical and Computer Engineering, University of Central Florida, Orlando FL 32816, USA. rfd@engr.ucf.edu
[3]Key Center for Software Technology, Dept. of Computer Science, Univ. of Queensland, Brisbane, QLD 4072 Australia. ged@cs.uq.oz.au
[4]Dept. of Computer Engineering, Univ. of California Santa Cruz, Santa Cruz CA 95064, USA. rph@cis.ucsc.edu
[5]Dept. of Computer and Information Sciences, Univ. California Santa Cruz, Santa Cruz CA 95064, USA. levinson@cis.ucsc.edu
[6]Dept. of Computer Engineering, Univ. of California Santa Cruz, Santa Cruz CA 95064, USA. claude@cse.ucsc.edu

- Provide dramatically increased performance with little or no increase in programming difficulty to the PEIRCE user and minimal complication to the PEIRCE programmer.

It is anticipated that uniprocessors, including mainframes, will be too slow for extremely large applications, and that single-purpose application-specific processor designs will lack the flexibility required by an evolving PEIRCE and the CG community. Parallel Prolog and LISP engines do not support the high degree of parallelism required, so other alternatives must be sought.

3.3 Results to Date

Research over the last year has shown that the low-level routines required by PEIRCE can be efficiently parallelized on a SIMD architecture (single-instruction multiple-data stream massive parallelism) with near linear speedup (doubling the number of processors approximately halves computation time). It has also been demonstrated that PEIRCE can be ported to a MIMD (multiple-instruction multiple-data stream multiprocessing) architecture. We have analyzed how processing requirement and speed scales in problem size for CG DB queries and evaluated communication requirements and processor allocation. The most crucial work remaining includes extending these results to other operations (such as maximal join and close matches) and issues, and then to remaining PEIRCE modules.

3.3.1 Large Systems

Roberts, J. D.

At last years workshop, the DB group indicated goals of systems as large as 10^9 graphs averaging 100 nodes each and CGs as large as 10^6 nodes. As we scale from current projects to these enormous sizes, Sowa noted that issues other than just processing power become critical, memory being key among these [25]. Our MSIMD parallelization work indicates that the optimal number of processors grows $O(n^2 \log N)$, where n is the average number of nodes in a graph and N is the number of CGs in the database. Memory scales $\Omega(nN)$ and dominates the cost of a system with a large number of CGs. As Sowa discussed, this indicates the need for *virtual memory* in which CGs are stored on disk rather than in memory (at about 1/10 the cost of memory), possibly organized by contexts. He also notes that as the required hardware becomes large and expensive, access by many users over internet will be appropriate and discusses this in detail. We also propose that large machines could be accomplished through *distributed processing*, connecting several smaller machines over a LAN or the proposed US high-speed optical network. These issues should be hidden from the PEIRCE user, and introduce minimal complications for the PEIRCE programmer.

3.3.2 Processor Allocation

DeMara, Ron

Research on the SNAP semantic network (SN) processor [19] has produced results which should be qualitatively, if not quantitatively, useful in PEIRCE work. Assigning nodes in a SN to a particular processor in a parallel system is analogous to assigning CGs in the partial order (generalization hierarchy) to particular processors. DeMara's results show that below a threshold of available parallelism (which unlike numerical algorithms can be quite high for AI) allocation has only a small impact on performance. Above this threshold, optimal allocation can be formulated in terms of marker propagation parallelism and the critical path through the applicable subgraph of the DB (which is analogous to predecessor and successor fanout in the generalization hierarchy). The best allocation strategy provides roughly a factor of 2 improvement in performance.

3.3.3 Communication Requirements

Noshpitz, Claude

A simulator has been built for evaluating communication patterns in a parallel CG system. CG reference profiles for the Morph chess program have been collected, and evaluation has begun. Additional reporting may be added to PEIRCE to collect similar profiles for other applications.

3.3.4 MSIMD Parallelization, Tailored Architecture

Roberts, J. D., Levinson, R., Hughey, R, and Ellis G.

The hardware group members at the University of California, Santa Cruz, are designing a parallel computer, the MISC Machine, tailored specifically to CGs as well as to a variety of other AI paradigms. Its overall architecture features multiple SIMD arrays, each with a supervising microprocessor integrated into a MIMD communication network, MSIMD (multiple SIMD). Its development also includes an associative processing model, in which collections of data are accessed and manipulated by content, to hide low-level parallelism and make programming the multiple levels of parallelism manageable. MISC is described in somewhat more detail in last years workshop proceedings [48].

Our key results are presented in a paper in the main session of this conference [49], which we summarize here. First, we have shown efficient (linear speedup) MSIMD algorithms for subgraph isomorphism (projection), DB query, and reflexive-transitive closure coding of concept-type and generalization hierarchies. Subsumption and other operations on the coded hierarchies can then be performed in constant or logarithmic time. Our current parallel DB query combines Levinson's Method III [43] for DB search pruning and Ullmann's refinement for pairwise graph comparison [59]. Method III empirically reduces the number of necessary graph comparisons to $O(\lg^2 N)$ where N is the number of graphs in the DB. Ullmann's subgraph isomorphism refinement algorithm is empirically $O(n^4)$ expected case, where n is the number of nodes in the larger graph. Willett et al. present a parallelization of refinement to reduce time to $O(n^2)$ using $O(n^2)$ processors [61], and we propose that $O(\lg N)$ projections can be performed in parallel without significant loss of efficiency. We are thereby able to reduce time for a DB query from $O(n^4 \lg^2 N)$ to $O(n^2 \lg N)$, using $O(n^2 \lg N)$ processors. For a DB of 64K graphs averaging 16 nodes each, this represents a parallelization speedup of three orders of magnitude, in addition to the two orders of magnitude search reduction from Levinson's algorithm[7]. Our key analytical result is that the above parallelization is faster than exhaustive search even with a processor for every graph in the KB. After investigating parallelizing the core low-level routines of Method III and hierarchy coding (generalization and concept-type), we are beginning to develop methods specific to the associative MSIMD architecture. We have also begun coding low-level DB routines for the MasPar-2, a commercial SIMD parallel machine which offers peak performance ranging from 4,250 to 68,000 MIPS.

The above asymptotics reveal excellent parallel scaling in N; a linear increase in the number of processors (or individual processor power) provides an exponential increase in the maximum number of graphs in the DB. Small increases in the average size of the CGs works even more strongly against us, however. *Type-definition expansion (join) versus nested comparison in projection becomes a crucial implementation issue.* Although nested comparison would clearly miss possible projections, there is an *exponential* advantage in DB size and a polynomial time advantage.

The MISC Machine is intended to be highly scalable, with implementations ranging from workstation accelerators to "personal superminis" to supercomputers with a million or more SIMD processors. Rough estimates indicate that a workstation accelerator with 1,024 processors could be sold in the neighborhood of $20 thousand, and would offer at least 2 orders of magnitude speedup over current workstations, dramatically increasing feasible DB and graph sizes.

[7] For larger DBs, search reduction would provide greater speedup than the parallelization.

3.4 Interaction With Other Subgroups

With our current focus on retrieval and hierarchy coding, our group's work relates closely with that of the DB subgroup. We encourage further communication with others dealing with hardware-related issues.

As our subgroup's work progresses, it will be appropriate to coordinate with languages and other aspects of the STDS subgroup. The Santa Cruz group is considering the development of an associative programming language for programming the MISC Machine. It would be developed in order to hide low-level parallelism through operations on collections (such as sets and tuples) with notations at the level of AI algorithm pseudocode, and to make the multiple levels of parallelism manageable to the programmer. Ideally it would include a cross-compiler for C++, allowing programs to run on other platforms, and the MISC Machine should also be programmable in C++ with associative and MSIMD objects and methods. We are also considering the language Sather.

> Sather is an object-oriented language that is particularly well suited for the needs of scientific research groups. It is designed to be very efficient and simple while supporting strong typing, garbage collection, object-oriented dispatch, multiple inheritance, parameterized types, and a clean syntax. It compiles into portable C code and easily links with existing C code. [47]

It also features a large class library. Most importantly, a public domain version is currently available, and both MIMD and SIMD parallel versions are under development.