Benchmarking Performance of Massively Parallel AI Architectures †

Ronald F. DeMara
Department of EE - Systems
University of Southern California
Los Angeles, CA 90089-2562
demara@gringo.usc.edu

Hiroaki Kitano ‡
Center for Machine Translation
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
hiroaki@cs.cmu.edu

Abstract

We address architectural evaluation of massively parallel machines suitable for Artificial Intelligence. Our approach was to identify the impact of specific algorithm features by measuring execution time on SNAP-1 and Connection Machine-2 using different knowledge base and machine configurations. Since a wide variety of parallel AI languages and processing architectures are currently in use, we developed a portable benchmark set for Parallel AI Computational Efficiency. PACE provides a representative set of processing workloads, knowledge base topologies, and performance indices. We also analyze speedup and scalability of fundamental AI operations in terms of the massively parallel paradigm.

1 Introduction

A primary motivation for parallel AI architectures is to obtain a significant increase in speed and tractable problem size over sequential implementations. Massively parallel systems have achieved some success towards this goal [3] [4] [1] [2]. To date, their performance evaluation has emphasized the reduction in overall execution time for an entire application. In this paper, we evaluate the interaction between characteristics of marker-passing algorithms and the processing features provided by massively parallel machines.

Our approach has been to develop a benchmark set for *Parallel AI Computational Efficiency* called *PACE*. Kernels are specified at the algorithm level to compensate for the lack of a common programming language and provide flexibility with respect to hardware and implementation. Performance is analyzed for a set W of workloads consisting of core AI operations. Since results depend heavily on the knowledge base used, we employ a separate set K of knowledge bases. The PACE benchmark consists of the cartesian product $W \times K$. For each (W, K) pair, speedup and execution time are measured while an independent variable, such as knowledge base depth or fanout, is varied.

Massively parallel AI centers around *memory-based reasoning* and *marker-passing* [6]. Knowledge is represented as a *semantic network*. Each node of the network represents a generic or specific concept in the domain. Edges of the graph, called *links* or *relations*, enforce relationships and constraints between concepts. *Markers* are propagated to perform inferencing operations. Simple *bit markers* indicate that a node represents an active candidate for some hypothesis. *Complex markers* contain a numeric weight, source address of the origin node, and an ALU operation to be performed at each propagation step. Intra-propagation parallelism, or *α-parallelism*, is derived from the searching of relation links and transmission of markers within a single propagate operation. The corresponding nodes are active in a data parallel fashion. Inter-propagation parallelism, called *β-parallelism*, also exists because multiple propagation statements can be executed simultaneously when there are no data dependencies between markers.

Array processor architectures (CM-2, SNAP-1) attain parallelism by the number of physical processors available. The associative processors (IXM-2) achieve degrees parallelism much greater than the number of physical processors. However, only bit-marker operations can be performed using the associative memories. The array processors can parallelize the activation of more than one node, but cannot multicast markers from a source node along multiple links simultaneously. While associative memories provide the multicast capability, they cannot process the activation of multiple source nodes at the same time.

We have analyzed workloads based on massively parallel AI applications such as DmSNAP Natural Language Parsing, Knowledge Classification, PASS

†This research has been supported by National Science Foundation grants MIP-90/09109 and MIP-90/09111 and performed in conjunction with the International Consortium on Massively Parallel Advanced Computing Technology (IMPACT).

‡This author also with NEC Corporation.

0-8186-2772-7/92 $3.00 © 1992 IEEE

Speech Understanding System, and DmDialog [5]. The knowledge base topologies used consist of 4 types: tangled networks, trees, meshes, and chains of concepts. F_{in} and F_{out} denote fan-in and fan-out, respectively. The parameter l denotes the length of a relation sequence. Ideally, markers are replicated and transmitted upon arrival for propagation time $O(l_{max}) = O(\log_{F_{out}} M)$ due to the hierarchical organization of the KB, where M is KB size in nodes. Results are presented below for various KB configurations using 4 basic workload classes.

2 Path Traversal

Path Continuity

Markers are propagated to check for transitive closure between two concepts in the semantic network, e.g. perform a yes/no inheritance query. The traversal must be constrained along specific relations, e.g. using only **is-a** and **has-part** links. Given a knowledge base K, two individual or generic concepts C_1 and C_2, and one or more relation types $R_1, R_2, ..., R_n$, determine if there is a path in K which connects C_1 and C_2 using any combination of R_i, $1 \leq i \leq n$.

```
/* transitive closure with link restriction */
 parbegin
   set m1 on node C1;
   set m2 on node C2;
 parend
 parfor i := 1 to n do
   propagate m1 from: m1 path: Ri dir: forward;
 set m3 := m1 AND m2;
 retrieve nodes with m3;
```

Results for a $L_x = L_y$ mesh in Figure 3 show that the parallel machines exhibit much better scalability as the KB grows with execution time for CM-2 less than 10 seconds [1] and SNAP-1 less than 1 second. The low execution time on SNAP-1 was due to the MIMD capability to perform selective propagation where as CM-2 had to iterate between the controller and array after each propagation step on the critical path. Chains of concepts are used to evaluate the multiple source activations by varying the number of distinct chains C is varied with $L_{reltype}$ fixed. Alternatively, as shown in Figure 4 for SNAP-1, C is fixed and $L_{reltype}$ is increased to evaluate speed of parallel propagations versus individual source propagation. To achieve a set degree of α-parallelism with M constant, the number of source nodes marked with $m1$ at the head of each chain is varied.

Enumeration

Markers are propagated to perform an exhaustive search along a specific relation. For example, following a path forward along **is-a** links determines all subsumers of a concept. This workload is defined as: given a knowledge base K, a concept C, and relations R_i, identify all concepts in K related to C by any number of applications of each R_i $1 \leq i \leq n$. This creates a degree of β-parallelism $= n$ with α dependent on the KB used.

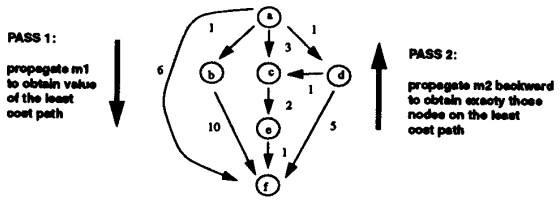


Figure 1: Least Cost Path Construction

```
/* find nodes from concept C by relations Ri */
 set m1 on node C;
 parfor i := 1 to n do
   propagate m1 from: m1 path: Ri dir: forward;
 retrieve nodes with m1;
```

Figure 5 shows that a $\beta = 2$ propagate from the root of 10-ary tree caused a rapid increase in SNAP-1 execution time due to network congestion. For the mesh, while time increased superlinearly in the size of the KB, it grew slower because the lower F_{out} did not saturate the network routers.

Path Construction

Rather than check if a path exists, markers are propagated to construct and retain the least cost path path between concepts. Given a knowledge base K, two individual or generic concepts C_1 and C_2, and a relation types R, find and retain the least cost path in K between C_1 and C_2 via R. Propagation is performed in two passes as shown in Figure 1.

```
/* single-source least cost path with link restriction */
configure node function := MIN on all nodes;
parbegin
 begin
  /* Forward Pass: val(m2) = cumulative minimum cost */
  set m1 on node C1;
  propagate m2 from: m1 link: Ri dir: forward op: ADD
  let minval := val(m2) on node C1;
 end
 begin
  /* Reverse Pass: val(m4) = cumulative minimum cost */
  set m3 on node C2;
  propagate m4 from: m3 link: Ri dir: reverse op: ADD
 end
parend
let val(m5) := val(m2) + val(m4);
set m6 on nodes with val(m5) = minval;
retrieve nodes with m6;
```

3 Structure Matching

Markers are propagated to match entire subgraphs of relations and concepts. Given an input concept-relation graph G_1 and a knowledge base K, test the compatibility of vertices and edges in G_1 and K to identify the best fit subgraph $G_2 \epsilon K$. G_2 is specified

by that graph $G_2 \epsilon E_k \times V_k$ such that $| E_1 \bigcap E_2 | + | V_1 \bigcap V_2 | \geq | E_1 \bigcap E_n | + | V_1 \bigcap V_n | \ \forall n \neq 2$. We have measured scale-up performance for a 4 concept pattern with 2 variables. Results in Figure 6 show that not only is the match performed quickly over an 8K node KB, but also the increase in execution time as the size of the KB is increased is negligible with respect to the overall execution time.

```
/* Exact match for 4-node subgraph against KB */
parbegin
 begin
  set m1 on node A;
  propagate m2 from: m1 path: R1 hops: 1 dir: forward;
  propagate m3 from: m2 path: R2 hops: 1 dir: forward;
 end
 begin
  set m4 on node B;
  propagate m5 from: m4 path: R3 hops: 1
    dir: forward;
 end
parend
set m6 := m3 AND m4;
propagate m7 from: m6 path: R2 hops: 1 dir: reverse;
propagate m8 from: m7 path: R1 hops: 1 dir: reverse;
retrieve nodes with m8;
```

4 Set Intersection

Associative processors can perform set intersection in time independent of the set cardinalities. Scaleup results for SNAP-1, as shown in Figure 7, indicate a slight increase in intersection time as the KB grows from 250 to 8000 nodes for Post-Propagation Intersection.

Post-Propagation Intersection

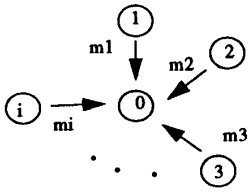
Upon completion of propagation, the set intersection operation is performed on all concepts simultaneously: given a set of concepts A consisting of C_{A1}, C_{A2}, ... C_{Am} and a set of concepts B consisting of C_{B1}, C_{B2}, ... C_{Bn}, form the set of concepts $C_I = C_A \bigcap C_B$.

```
/* Full KB intersection with stationary markers */
   parbegin
      parfor i := 1 to m do
         set m1 on node i
      parfor i := 1 to n do
         set m2 on node i
   parend
   set m3 := m1 AND m2;
```

Stepwise Intersection

Stepwise Intersection executes the intersection operation at each hop along the propagation. Since this required involvement from the central controller, processing time increased proportionally to the number of propagation steps as shown in Figure 7.

```
/* Intersection at each propagation step */
parbegin
 parfor i := 1 to m do
    set m1 on node i
```



Nodes 1 to i propagate marker mi to star node 0. ALU function must be performed for each incoming marker.

Figure 2: Marker Collisions at Node 0.

```
 parfor i := 1 to m do
    set m2 on node i
propagate m1 from: m1 path: is-a dir: forward
 op: m3=m1 AND m2
```

5 Network Modification

Static Update

A typical update involves the addition of a new concepts to the knowledge base. The static update workload is defined as the addition of $2 \times n$ new concepts to the KB each related by m links.

```
/* Add to KB 2*n nodes, each having relation R */
parfor i := 1 to n step 2 do
  create node i
  create node i + 1
  parfor j := 1 to m do
    create relation R from: node j to: node j + 1
```

Impingement

When markers arrive at the destination node, they modify the local value of the markers and perform an ALU operation. Several markers arrive simultaneously causing a collision at the destination node as shown in Figure 2. Each marker is requesting an ALU operation: Add, Multiply, Divide, Maximum Minimum, Sigmoid, or Boolean functions. Performance is measured as the number of simultaneous arrivals is increased.

```
/* total of n Marker Collisions Impinging on
Destination Node #0 requesting ALU operation */
parfor i := 1 to n do
   set m1 on node i
propagate m1 from: m1 path: R1 dir: forward op: MUL;
```

We are currently extending the PACE benchmark to include composite tasks such as Sequential Pattern Matching and Case Based Reasoning for implementation on the CM, SNAP, and IXM parallel processors.

References

[1] S. Chung, D. Moldovan, and Y. Tung "Modeling Semantic Networks on the Connection Machine" Technical Report PKP-90-2, University of Southern California, Dept. EE-Systems.

[2] R. F. DeMara and D. I. Moldovan, "The SNAP-1 Parallel AI Prototype," in *Proceedings of Eighteenth Annual International Symposium on Computer Architecture*, Toronto Canada, 1991.

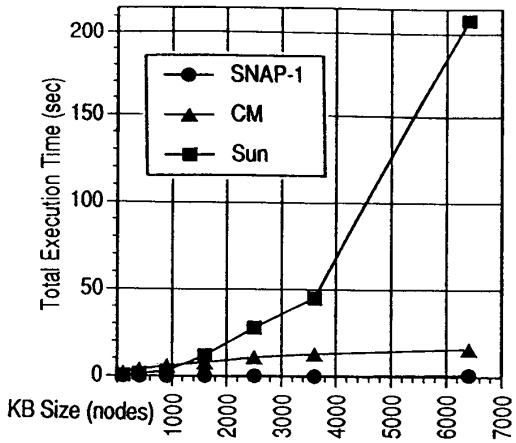


Figure 3: Machine Scaleup T=2D Mesh ($l_x = l_y$)

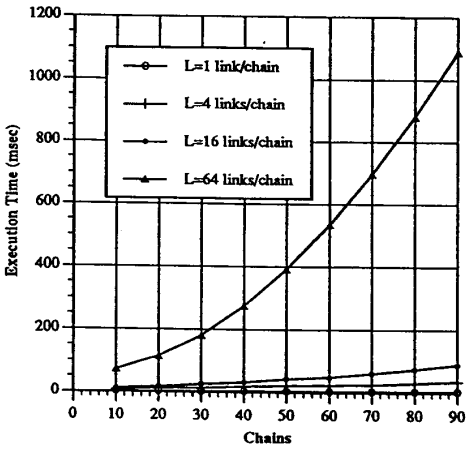


Figure 4: Effect of Propagation Length on Processing Time (SNAP-1)

[3] M. Evett, J. Hendler and L. Spector, "PARKA: Parallel Knowledge Representation on the Connection Machine," Technical Report UMIACS-TR-90-22, University of Maryland, 1990.

[4] T. Higuchi, *et al*, "The Prototype of a Semantic Network Machine IXM," *Proceedings of 1989 International Conference on Parallel Processing.*

[5] H. Kitano, "DM-Dialog: An Experimental Speech-to-Speech Dialog Translation System," *Computer* **24**(6), pps. 36-50, June 1991.

[6] Waltz, D.,"Massively Parallel AI," *Proceedings of AAAI-90*, 1990.

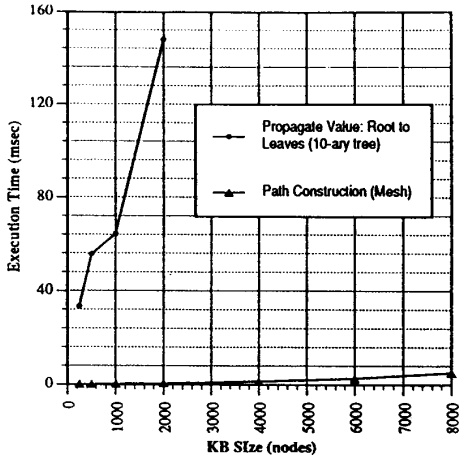


Figure 5: Path Traversal on Tree ($F_{out}^{ave} = 10$) and 2D Mesh ($l_x = l_y$)

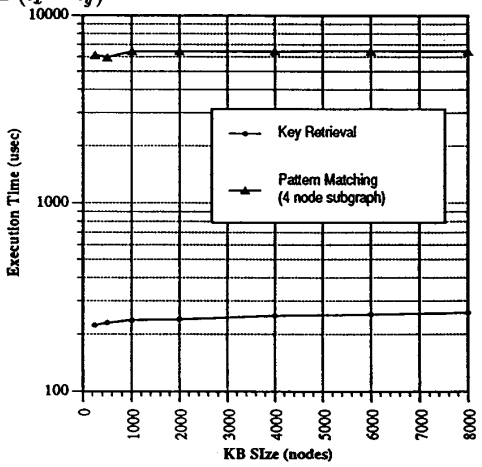


Figure 6: Scaleup under Structure Matching

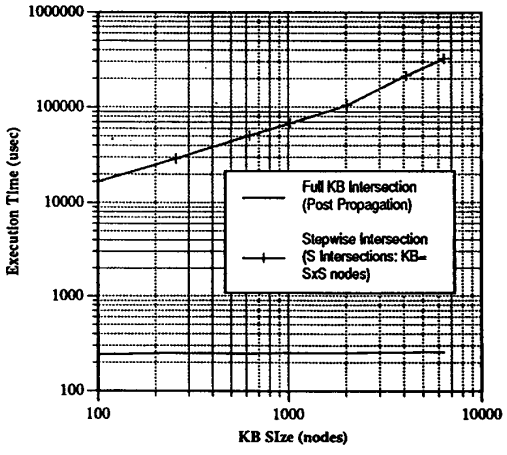


Figure 7: Scaleup for Full KB Set Intersection (T = *mesh*)