


CSE@UTA

Research Experiences for Undergraduates (REU) in Dynamic Distributed Real-Time Systems Summer 2000 Program



Dr. Behrooz Shirazi
Jeff Marquis
Geoff Dale
Kshiti Desai
Chris Forrest

August 9, 2000 1

CSE@UTA

Student Introductions

- This summer 3 REU 2000 students were supported in CSE@UTA:
 - Chris Forrest.
 - Graduating senior in second semester of REU support.
 - Will enter the CSE@UTA graduate program, Fall 2000.
 - Kshiti Desai
 - Senior at CSE@UTA.
 - Will receive REU support in Fall 2000.
 - Geoff Dale
 - Senior at Texas Christian University.
 - Will receive REU support in Fall 2000.

August 9, 2000 2

CSE@UTA

Introduction to REU 2000

- REU 2000 was designed to give the students a rich and rewarding experience including:
 - Classroom instruction.
 - Exposure to state-of-the-art programming tools.
 - Applied research on an on-going research project.
- This approach was designed to allow the REU 2000 students to participate in and materially contribute to the overall program.
- We believe the students have gained valuable experience with REU 2000.

August 9, 2000 3

CSE@UTA

Classroom Instruction

- The REU 2000 students were required to enroll in CSE 4392, Parallel Processing.
 - Taught by Dr. Bob Weems, CSE@UTA Professor.
 - Provide the basic knowledge needed to participate in the program.
 - The course combined theory and hands-on programming experience.
- REU 2000 student Geoff Dale will present the course details.

August 9, 2000 4

CSE@UTA

PARSA and ThreadMan

- The REU 2000 students were tasked with learning state-of-the-art parallel programming tools.
 - The PARSA™ Software Development Environment.
 - The ThreadMan™ Thread Manager.
- These commercial tools were developed between CSE@UTA and Prism Parallel Technologies (www.prismpti.com).
 - Access to the tools provided an ideal setting for the students to understand the inner workings of a commercial software product.
- REU 2000 student Kshiti Desai will present an introduction to PARSA and ThreadMan.

August 9, 2000 5

CSE@UTA

Applied Research

- The REU 2000 students were tasked with working with a CSE@UTA graduate student who was extending the capabilities of PARSA.
 - Support for distributed cluster computers.
 - Mix of multi-threading and message passing.
- The REU 2000 students validated distributed cluster computing using PARSA.
- The REU 2000 students did a performance analysis of distributed cluster computer using PARSA.
- REU 2000 student Chris Forrest will present the results of their applied research.

August 9, 2000 6

CSE@UTA

An Introduction to Parallel Programming Presented by Geoff Dale

August 9, 2000 7

CSE@UTA **CSE 4392: Parallel Processing**

- Taught by Dr. Bob Weems
- Met for 2 hours on Tuesdays and Thursdays for the entire summer session
- G.R. Andrews, *Foundations of Multithreaded, Parallel, and Distributed Programming*, Addison-Wesley, 2000.
- P.S. Pacheco, *Parallel Programming with MPI*, Morgan Kaufmann, 1997.

August 9, 2000 8

CSE@UTA

Objective of the Class:

An introduction to the variety of topics
necessary for developing parallel
applications

August 9, 2000 9

CSE@UTA **Goals of the Class:**

1. The ability to implement small applications on shared-memory multiprocessors (Linux SMP) using pthreads
2. The ability to implement small applications in message-passing paradigm using MPI
3. The understanding of concepts of parallel algorithms
4. The understanding of elementary topologies
5. The understanding of compiler concurrentization concepts

August 9, 2000 10

CSE@UTA **Structure of the Class**

- The semester was divided into two sections
- Each section consisted of note taking, lectures, two labs, and one test
- The C programming language was the primary language used for code writing and algorithm demonstrations
- Programs were written on dual processor machines named Ketchup and Mustard

August 9, 2000 11

CSE@UTA **Pthreads**

- Shared Memory Systems
- Scheduling Methods:
 - Static Scheduling:
 - Interleaving
 - Contiguous
 - Dynamic Scheduling
- PRAM Model

August 9, 2000 12

CSE@UTA **MPI**

- No Shared Memory
- Message Passing Techniques
- MPI Functions

August 9, 2000 13

CSE@UTA **Synchronization**

- Shared Memory
 - Barriers
 - Locks
 - Semaphores
- Distributed Memory
 - Different forms of message passing
 - Blocking Techniques

August 9, 2000 14

CSE@UTA **Analysis of Algorithms**

- Speed-up
- Efficiency
- Comparison of algorithms run in sequential and in parallel

August 9, 2000 15

CSE@UTA **Topologies**

- Linear Arrays
- Hypercubes
- Meshes
- Torus

August 9, 2000 16

CSE@UTA **Interconnections**

- Benes Network
- Butterfly Technique
- Shuffle Exchange
- Perfect Matching

August 9, 2000 17

CSE@UTA **Numerical Problems**

- Elimination Techniques
 - Gaussian
 - LU
 - Householder
- Iterative Methods
- Sparse Matrices

August 9, 2000 18

CSE@UTA **Communication Problems**

- Matrix Multiplication
- Matrix Transpose
- Techniques on various topologies

August 9, 2000 19

CSE@UTA **Sorting Problems**

- Linear Arrays
- Bitonic Mergesort
- Meshes

August 9, 2000 20

CSE@UTA **Parallel Enumeration**

- Permutations and Combinations
- Lexicographic Ordering Concepts
- Numbering
- Ranking and Unranking techniques

August 9, 2000 21

CSE@UTA **Parallelizing Compilers**

- Elementary Data Dependency Concepts
- Code Generation
- Sequent FORTRAN
- Tiny Tool
- OpenMP

August 9, 2000 22

CSE@UTA **Lab 1**

- Pthreads
- The modification of a sequential program to work with two threads
- Speed-up and efficiency Evaluation
- Interleaving and Contiguous Methods

August 9, 2000 23

CSE@UTA **Lab 2**

- MPI
- The modification of a client/server hashing program to merge the client and server functionality
- The efficient utilization of processes using MPI

August 9, 2000 24

Labs 3 and 4

- Lab 3
 - The implementation of a pthreads version of a program that uses Lukes' technique to determine the bisection width of an undirected graph using two threads.
- Lab 4
 - An MPI version of Lukes' technique that can use multiple processors on multiple machines
- Dynamic Process Creation

Summary

- Learned techniques for hand coding parallel algorithms
- All code available in C functions
- Class laid the building blocks for the research that would be performed throughout the summer

PARSA and ThreadMan Presented by Kshiti Desai

PARSA

- Supports a graphical programming methodology
- Designed for developing parallel software

Advantages:

- Saves time
- Reduces complexity

PARSA Applications

- Made up of graphical objects (or GOs) and arcs
- Each graphical object represents an application task to be performed and contains:
 - Interface section
 - Functionality section

Graphical Objects

PARSA supports 3 different types of graphical objects:

1. User-Defined
2. Forall
3. While

CSE@UTA **ARCS**

- Connect graphical objects together
- Represent the data flow
- Control the flow between graphical objects

August 9, 2000 31

CSE@UTA **Least Common String in PARSA**

Figure 1. The Least Common String application in PARSA.

August 9, 2000 32

CSE@UTA **Performance Results**

Programs	Timings (CPU time)
Sequential Version	1.04
Hand-coded pthreads Version	0.30
PARSA implemented Version	0.33

Table 1. Performance comparison between hand-coded and PARSA generated multithreaded software.

August 9, 2000 33

CSE@UTA **Lines of code count comparison**

	Number of Lines
Hand-coded.	311
Developed in PARSA	63
PARSA Generated Code	558

Table 2. Lines of code count comparison between hand-coded, developed in PARSA, and automatically generated by PARSA.

August 9, 2000 34

CSE@UTA

**Applied Research
Presented by Chris Forrest**

August 9, 2000 35

CSE@UTA **Applied Research Goals**

- Tasked with working with a CSE@UTA graduate student who was extending the capabilities of PARSA
 - Generate test applications that utilized MPI design
 - Obtain and analyze timing results
 - Note any potential problems that might discourage automatic MPI code generation

August 9, 2000 36

CSE@UTA Test Application Criteria

- Desired to demonstrate two basic types of algorithms:
 - Non-oblivious algorithms
 - Algorithms that execute differently based on the input data
 - Oblivious algorithms
 - Algorithms that execute similarly on different datasets.

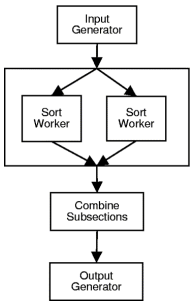
August 9, 2000 37

CSE@UTA Test Application Winners

- Merge sort Application
 - Mergesort is good non-oblivious algorithm
 - Dependencies between loop iterations
 - Loops can not run independently
- Matrix Multiplication
 - Matrix multiplication is good oblivious algorithm
 - Loop order does not need to be preserved
 - Loops can run independently

August 9, 2000 38

CSE@UTA Merge Sort Outline



- Input Generator creates random data into array
- Data is split into smaller sections and workers sort the small subsets
- The data is merged together sorted and presented in a file

August 9, 2000 39

CSE@UTA Merge Sort – Single-threaded

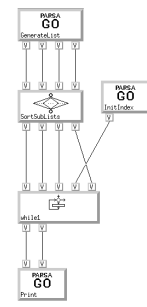
- New array is created that is twice as large as original array
- New array is partitioned into two sections and data is placed between both sides

```

for (i=0; i<2*size; i++)
{
  if (first<size && second<2*size)
  {
    if (a[first] < a[second])
      b[i] = a[first++];
    else
      b[i] = a[second++];
  }
  else if (first >= size)
    b[i] = a[second++];
  else if (second >= 2*size)
    b[i] = a[first++];
}
for (i=0; i<2*size; i++)
  a[i] = b[i];
  
```

August 9, 2000 40

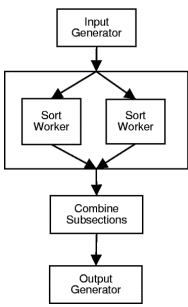
CSE@UTA Merge Sort – Multi-threaded (PARSA)



- Data generation occurs in the first user defined Graphical Object (GO)
- Data is sectioned off into smaller sections.
- New threads are created for each section.
- Data is sorted and then combined
- Data is presented in file format

August 9, 2000 41

CSE@UTA Merge Sort – Multi-threaded (PARSA - MPI)



- MPI added to PARSA generated code
- A process is created for each GO
- Data is transferred between processes via MPI calls
- Threads are still created to handle work

August 9, 2000 42

Merge Sort – Performance Results

- On a limited set of data sizes (10k – 100k), PARSA is better without MPI.
 - For 10,000 entries, MPI slows down PARSA by approximately 9 seconds
- MPI is not being utilized to it's fullest
 - Sorting of work is done by threads only
 - Design was intentional to see how much MPI would hinder performance.

August 9, 2000 43

Matrix Multiplication Outline

- Input Generator creates random data into two arrays A and B
- Data is split among two workers who divide the work.
 - Worker 1 provides top half of result array
 - Worker 2 provides bottom half of result array
- The data is merged together sorted and presented in a file

August 9, 2000 44

Matrix Multiplication – Single-threaded

- Resultant array obtains sums of row-elements * column-elements

```

for (j=0; j < arraySize; j++)
  for (i=0; i < arraySize; i++)
    c[i][j] = 0;
  for (k=0; k < arraySize; k++)
    c[i][j] = c[i][j] + a[i][k]*c[k][j];
  
```

August 9, 2000 45

Matrix Multiplication – Multi-threaded (PARSA)

- Data generation occurs in the first user defined Graphical Object (GO)
- Data is sent to both workers for processing.
- New threads are created in each worker
 - More for use in MPI
- Data is sorted and then combined
 - Data is written directly into result array in PARSA only version
- Data is presented in file format

August 9, 2000 46

Matrix Multiplication – Multi-threaded (PARSA - MPI)

- MPI added to PARSA generated code
- A process is created for each GO
- Data is transferred between processes via MPI calls
- Threads are still created to handle work in both worker sections
 - Means more than one MPI process can perform work
- Combine section required since MPI

August 9, 2000 47

Matrix Multiplication – Performance Results

- On a tested set of square matrices sizes (100 thru 3000), PARSA benefits from MPI (when adding more processors)
 - MPI is slower than PARSA in 100x100 and 500x500 cases
 - PARSA-MPI test was run on two, two-processor machines.
 - Overall results show that you can obtain good speedups with MPI and PARSA working together

August 9, 2000 48

PARSA-MPI Findings

- Memory consumption can escalate in PARSA applications that use MPI (on large scale problems)
 - Memory is freed, however system can not reclaim
 - Some O/S (Tru64 for example) have special O/S specific parameters for attempting to deal with these problems
- Passing dynamically allocated structures can pose a problem
 - Becomes difficult for a program to recognize how much memory has been allocated by user

August 9, 2000

49

PARSA-MPI Findings (contd.)

- Passing structures in heterogeneous environments
 - When passing structures, MPI is not able to handle endian issues unless given information about the structure
- Special MPI constraints
 - Some functions (such as exit()) can not be called by the user due to deadlock cases
 - Special MPI functions can be called but go against the basic premise of PARSA (no need to know)

August 9, 2000

50