

Message Passing Interface vs. Multi-threading: Which Parallelization Technique is More Efficient?

Presented to

George Seweryniak

**Mathematical, Information, and Computational
Sciences**

Jonathan C. Rann

Winston-Salem State University

Research Alliance in Mathematics and Science

Oak Ridge, Tennessee

August 9, 2006

OUTLINE

- **Motivation**
- **Research goals**
- **Sequential computing**
 - **Sequential architecture**
- **Parallel computing**
 - **Parallel Architecture**
 - **Parallelization**
- **Results**
- **Research**
 - **Current**
 - **Future**

MOTIVATION

- To help create optimized biological applications which utilizes the maximum efficiency of computer hardware
- Implemented through
 - Message Passing Interface (MPI)
 - Multithreading

MOTIVATION: Biological Applications

- **Typical application times**

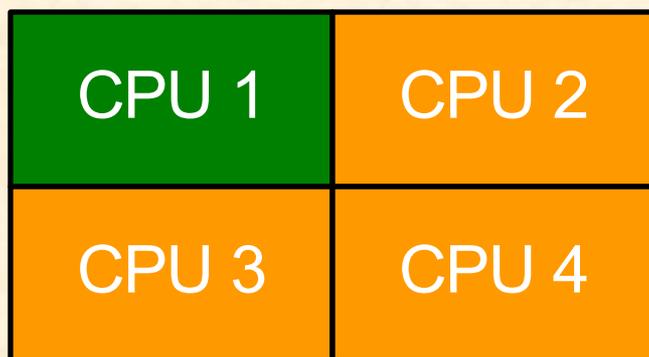
- Web-page loading: 1-10 s
- Unzip a large file: 10-100s
- Biological applications: >1,000 hours

- **Parallel programming reduces time-to-solutions**

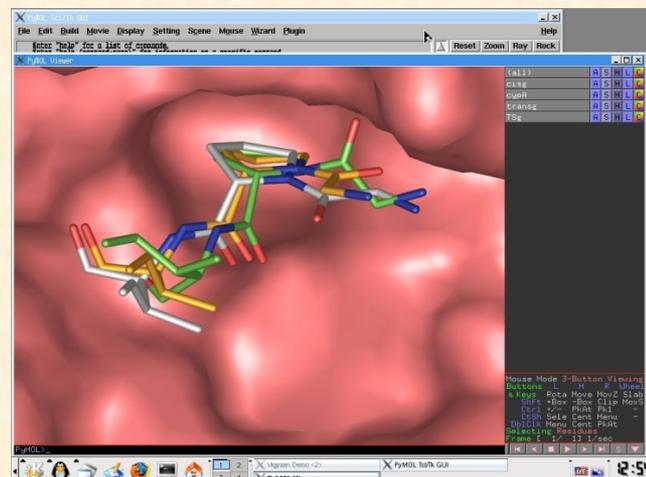
- *Combined power of 100 CPUs will solve the problem in only 10 hours!*
- Reduces the time required for biological research by speeding the calculations
- Modern supercomputer have >10,000 CPUs

MOTIVATION: Biological Applications

- **Modeling of biological molecules: DNA and proteins**
 - Calculations require > 1,000 hours
 - Parallel programming is a technique to reduce time to solve the calculations
 - Parallel programming: *divide-and-conquer* approach



Each processor does a part of the calculation



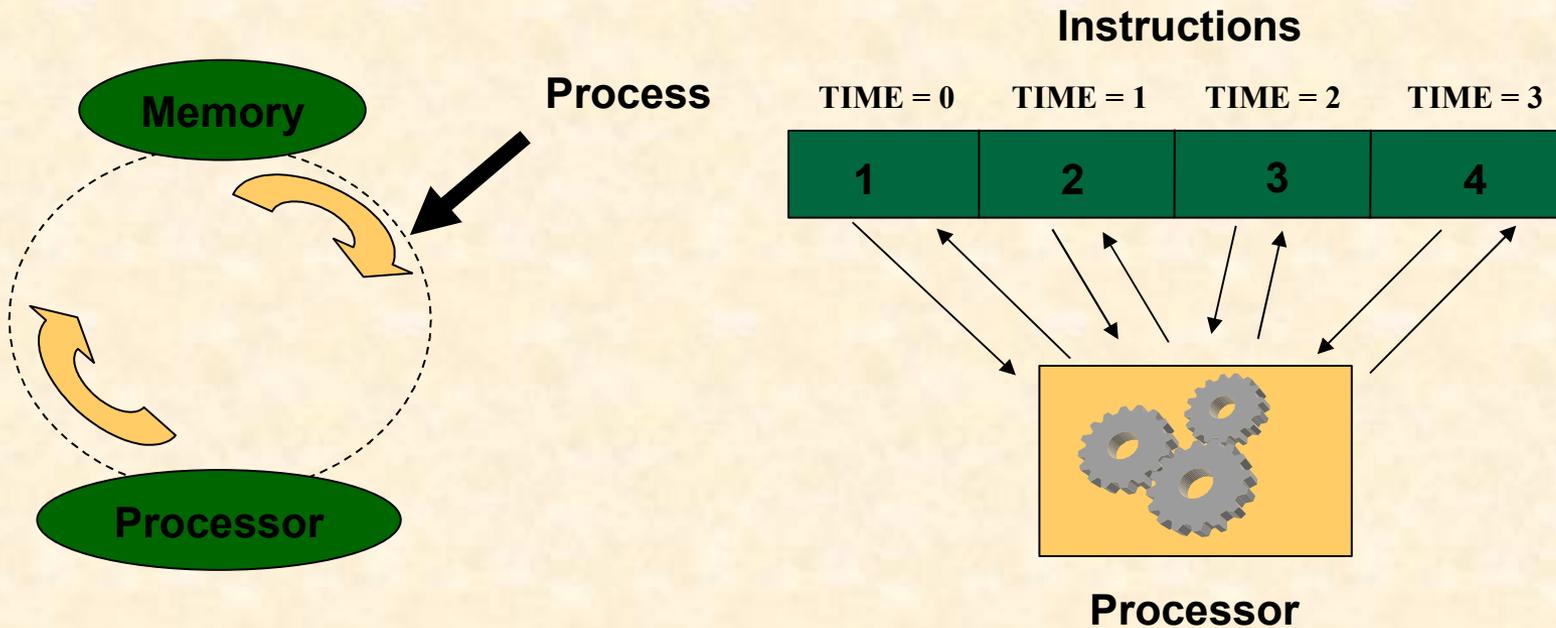
RESEARCH GOALS

- **Analyze parallel programming techniques**
- **Create a parallelized program using both Message Passing Interface and Multi-threading techniques**
- **Compare the efficiency for created program**

SEQUENTIAL COMPUTING

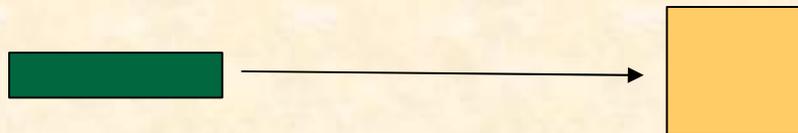
- **Traditional computing model**

- One processor used to execute each instruction
- Each instruction executed one at a time, in sequence



SEQUENTIAL ARCHITECTURE

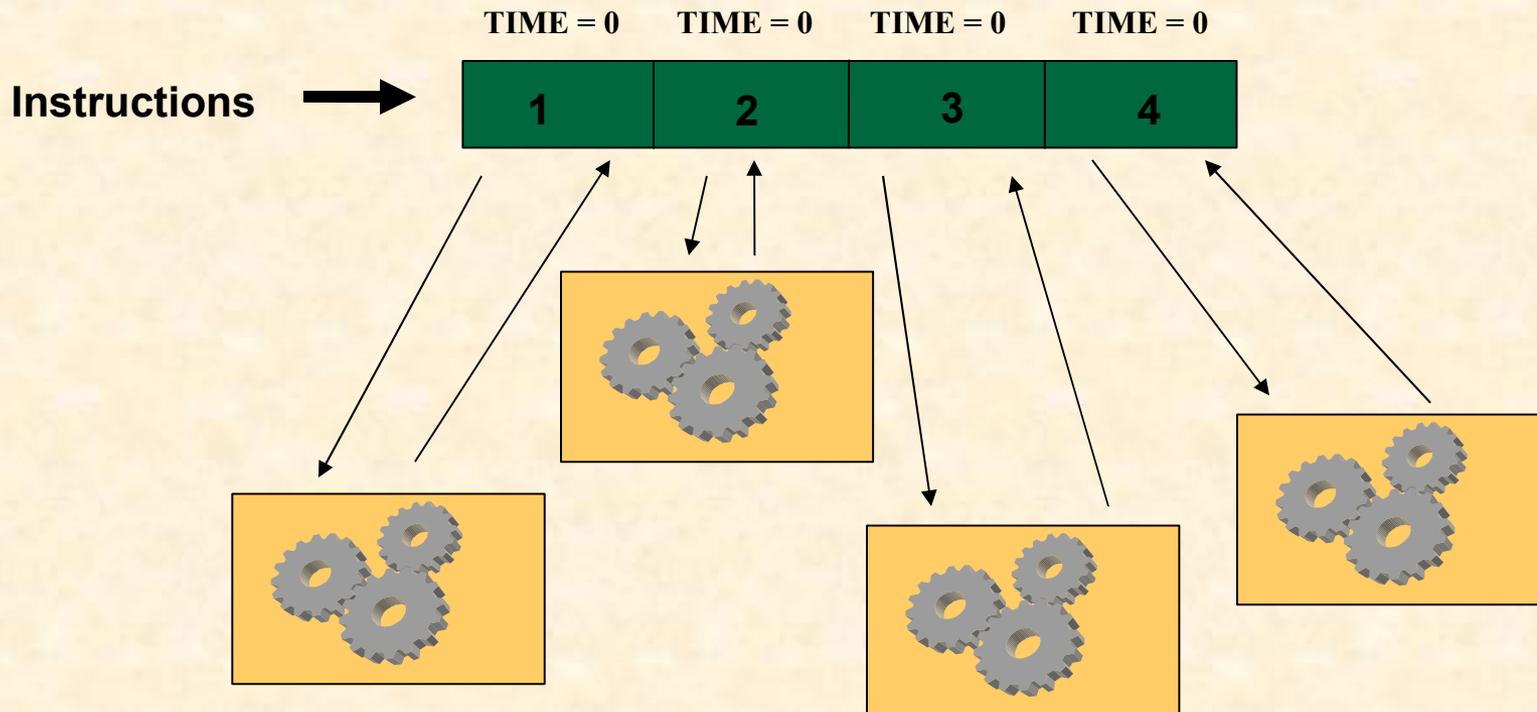
- **Single Instruction, Single Data**



E.g. Each worker in a factory making one car, thus creating a more time consuming process

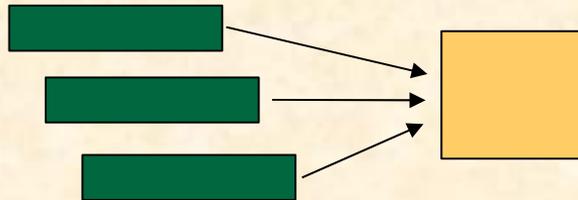
PARALLEL COMPUTING

- Simultaneous use of multiple processors to complete a specific task



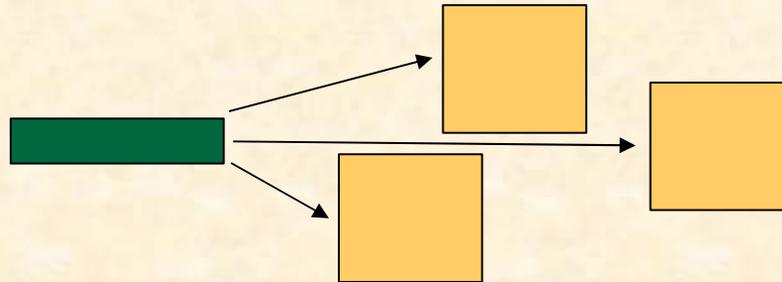
PARALLEL ARCHITECTURES...

- Multiple Instructions, Single Data (MISD)



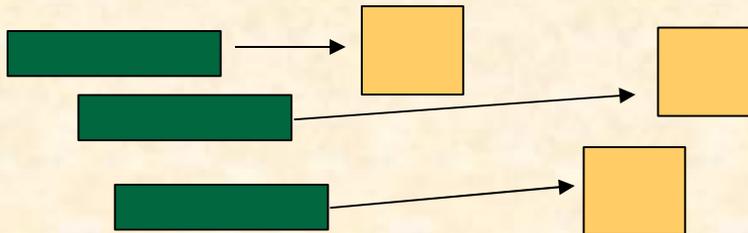
E.g. Multiple workers creating a single car on an assembly line

- Single Instruction, Multiple Data (SIMD)



E.g. Single worker executes the same task on multiple cars as they come down the assembly line

- Multiple Instruction, Multiple Data (MIMD)

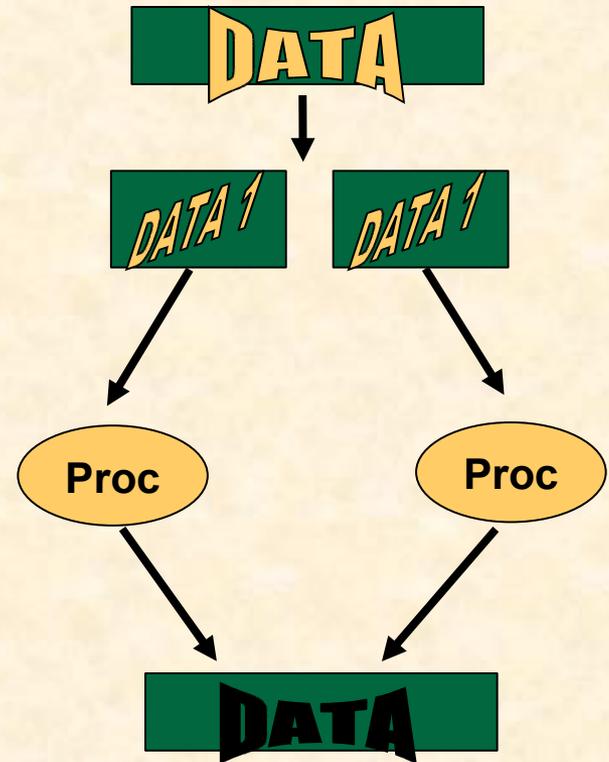


E.g. Multiple workers execute different tasks on different cars

PARALLELIZATION

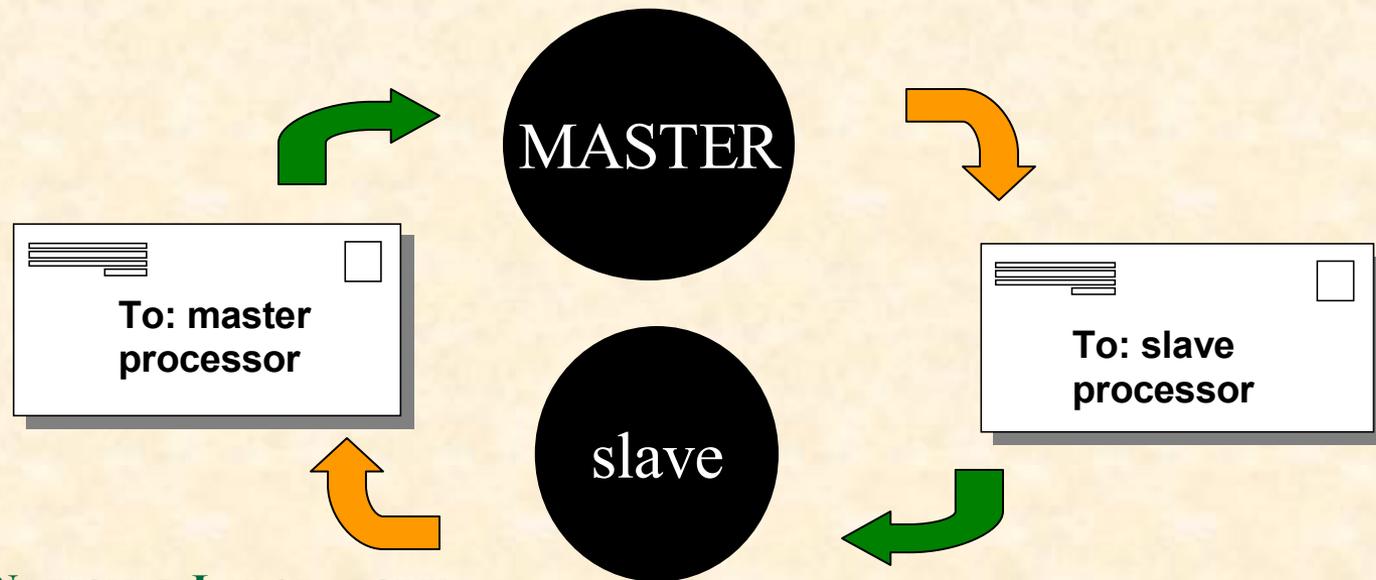
- Involves...

- Segmenting data
 - Master process breaks a task down into smaller parts that can be computed independently
- Sending data
 - Slave process assigns, then sends data segments to different processors
- Compute
 - Upon receiving the tasks, the processor computes them, then sends them back to master process
- Combine
 - Computed data now brought back together; task is complete



MESSAGE PASSING

- **Message Passing Interface (MPI)**
 - Industry standard for parallel programming
 - Used in writing C, C++, and Fortran
 - Enables communication between multiple processors through specific commands



MESSAGE PASSING (CONT'D)

- **Basic MPI commands:**
 - **MPI_Init()**
 - **Initializes communicator**
 - **MPI_Comm_size()**
 - **Determines total number of processors available in communicator**
 - **MPI_Comm_rank()**
 - **Assigns each processor a number or position in communicator**
 - **MPI_Send()**
 - **Sends data from one processor to another**
 - **MPI_Recv()**
 - **Receives data from another processor**
 - **MPI_Finalize()**
 - **Shut down the communicator**

MULTI-THREADING

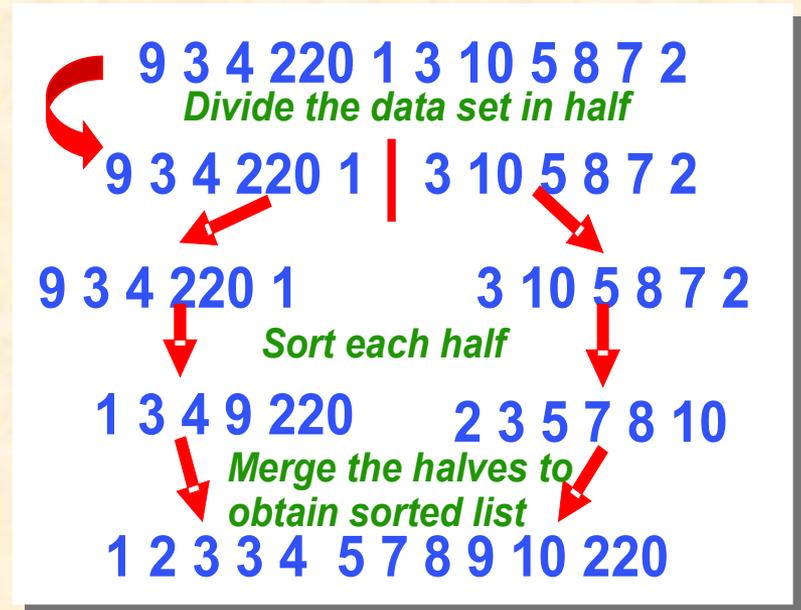
Technique used to split a program into two or more simultaneously running tasks

- Suitable for processors with multiple cores (ex. Intel's Dual core processor) or even single core processor**
- A single processor switches between different threads when performed sequentially**
- Multiple threads running simultaneously on different processors when performed in parallel**

RESEARCH

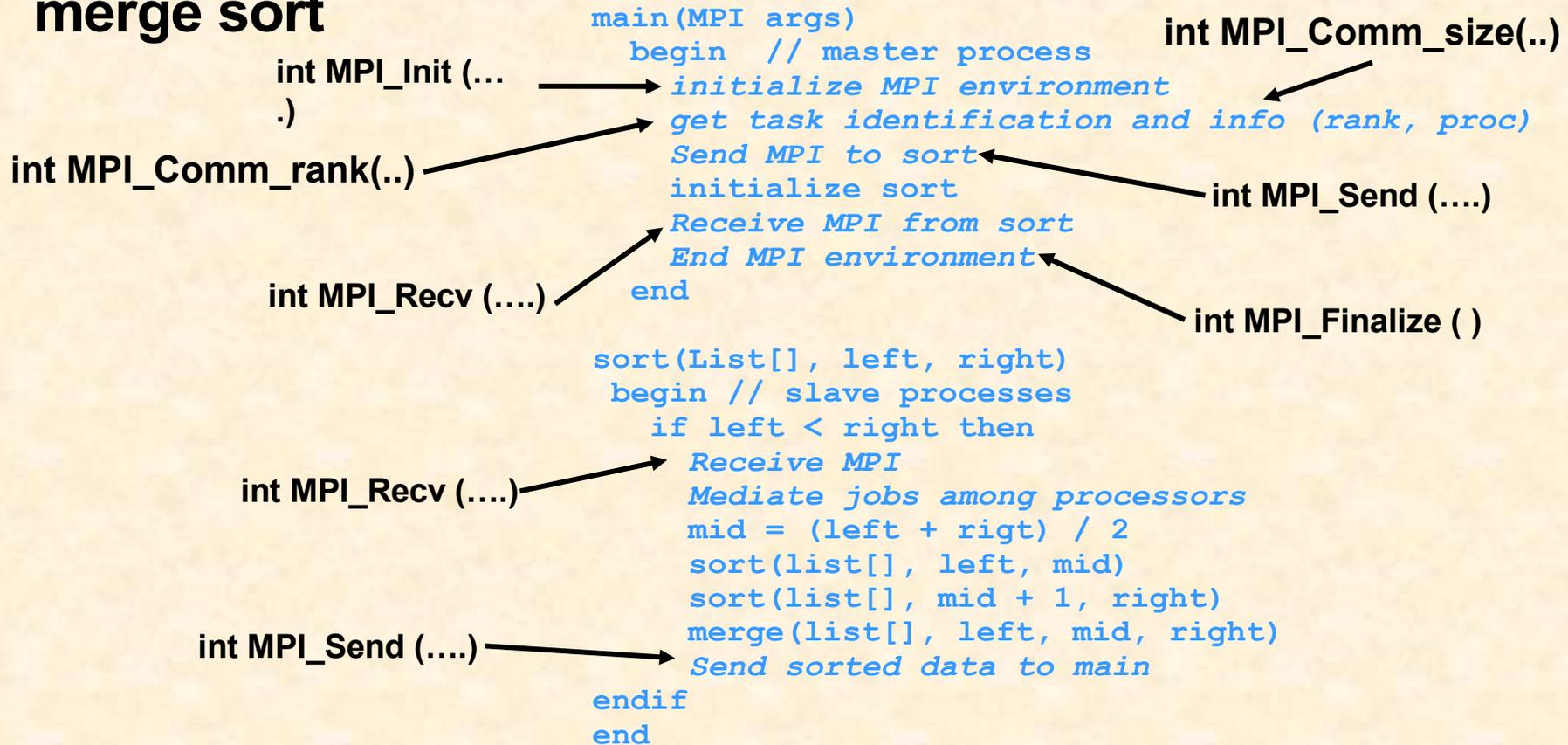
- **SAMPLE APPLICATION: Merge sort algorithm**

```
sort(List[], left, right)
begin
  if left < right then
    mid = (left + right) / 2
    sort(list[], left, mid)
    sort(list[], mid + 1, right)
    merge(list[], left, mid, right)
  endif
End
```

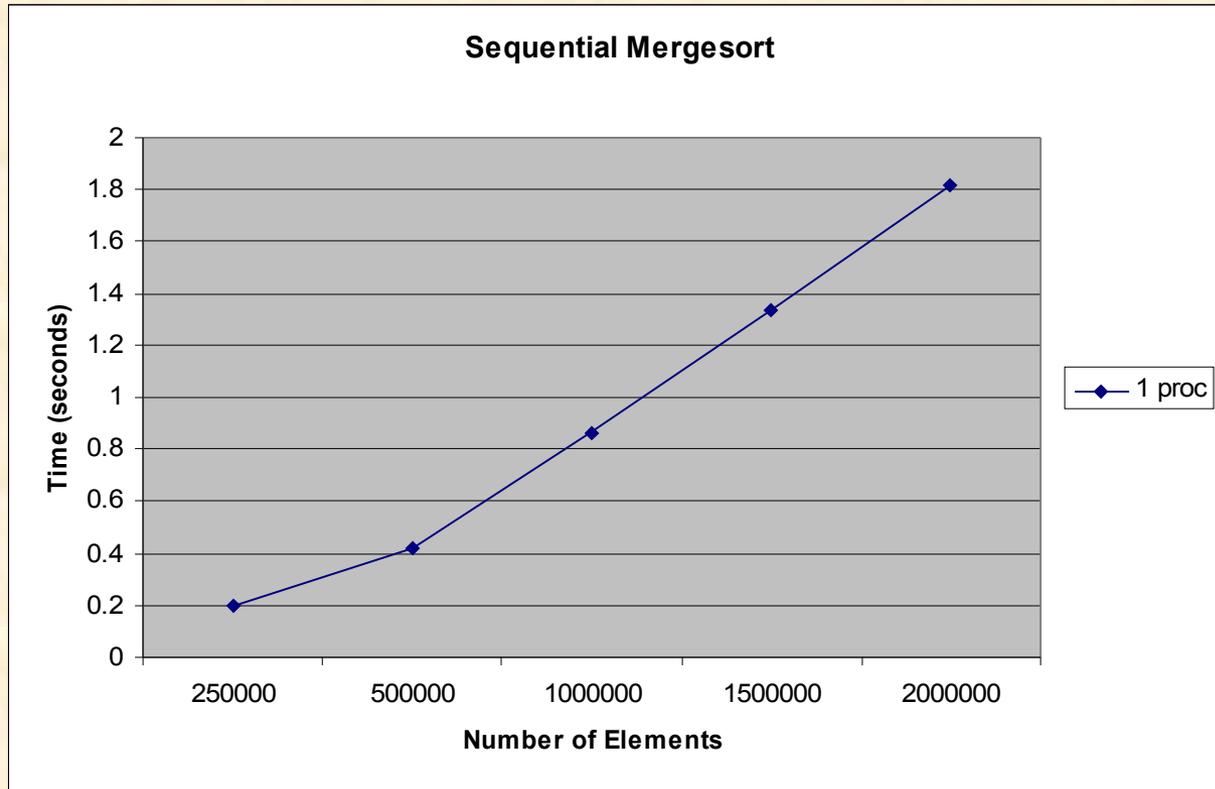


RESEARCH (Contd.)

- Create MPI implementing code that performs merge sort

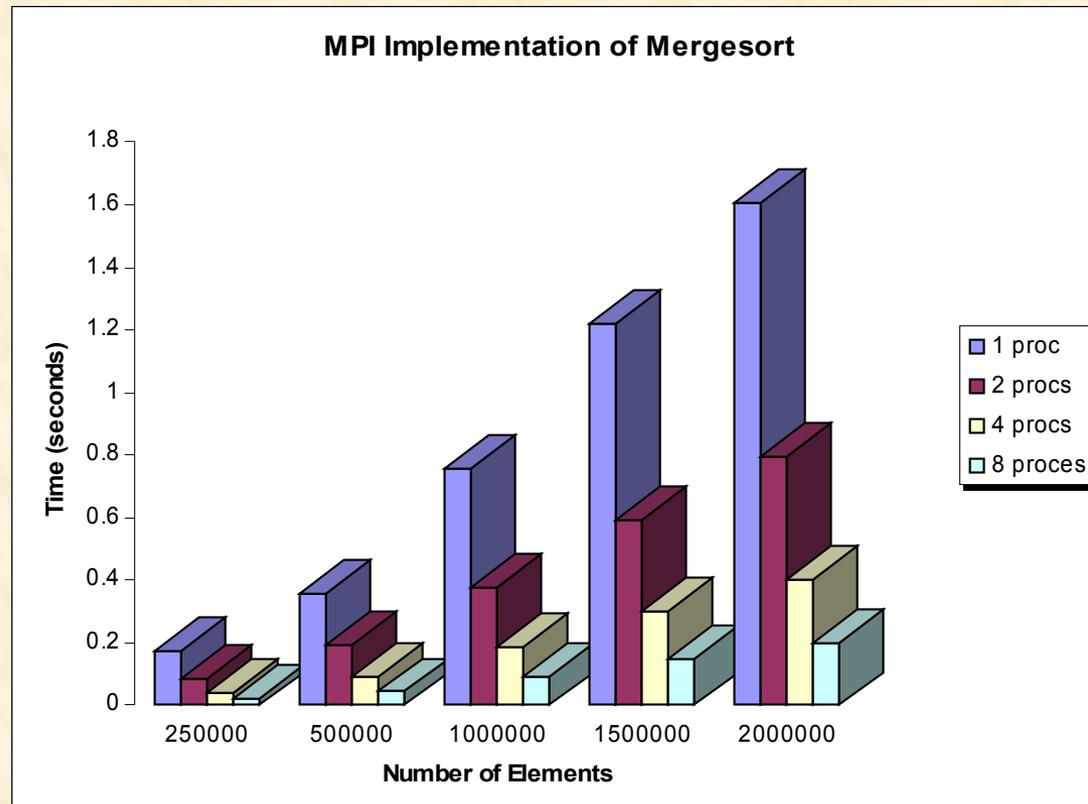


RESULTS



RESULT: Time required to compute a solution with the sequential implementation of the merge sort program **increases** with the size of an array

RESULTS (Contd.)



RESULT: As the number of processors used to compute the solution with the MPI implementation of the merge sort increases, the computation time **decreases**

CURRENT RESEARCH

- **Exploration of Multithreading**
 - **Better understanding of process synchronization**
 - **Resource sharing**
 - **Deadlock**
 - **Efficiency comparisons with MPI**

FUTURE RESEARCH

- **Complete multi-threaded implementation of merge sort**
- **Characterize the efficiency of applications using alternate parallel programming methods**
- **Create optimized applications based on research**

Acknowledgments

The Research Alliance in Math and Science program is sponsored by the Mathematical, Information, and Computational Sciences Division, Office of Advanced Scientific Computing Research, U.S. Department of Energy.

The work was performed at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. De-AC05-00OR22725. This work has been authored by a contractor of the U.S. Government, accordingly, the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

Questions?