Automatic Management of Parallel and Distributed System Resources

- NASA Ames Research Center (Dr. Jerry Yan)
- Stanford University (Tin Fook Ngai)

Dr. Stephen F. Lundstrom

Consultant
PARSA
(415) 723-0140

Consulting Associate Professor, Electrical Engineering
Computer Systems Laboratory
Stanford University

Program Focus

(at NASA Ames Research Center)

- Parallel Processing is not confined to any one level of the software hierarchy — from applications to operating systems and machines

  - Applications must be formulated with sufficient inherent parallelism to exploit the underlying parallel architecture The focus in the area of parallel applications is:

    - Scalable, highly parallel symbolic applications
    - Application development environment

  - The mapping of hardware resources to the application must be able to respond to dynamic load variations and faults on the system. The focus in the area of intelligent management of multiprocessing systems is:

    - High performance and integrity
    - Highly adaptive

  - For space applications, the parallel hardware has to be subject to low weight, power and volume constraints. The focus in the performance evaluation of parallel architectures is:

    - Standardized benchmarks
    - Simulation and prediction tools for parallel systems

Program Focus

- **Parallel Applications**
  - Scalable, highly parallel symbolic applications
  - Application development environment
- **Intelligent Management of Multiprocessing Systems**
  - High performance and integrity
  - Highly adaptive
- **Performance Evaluation of Parallel Architectures**
  - Standardized benchmarks
  - Simulation and prediction tools for parallel systems

This is how our project fits together

- We are currently working on "parallelizing" three applications (at NASA Ames Research Center)
    - KATE — A Frame-based reasoning system monitoring the subsystem of the shuttle launch system
    - CLiPS — A C-based production system shell
    - Space Station Workloads — We are looking at the possibilities to model part of the OMA
- We are also working on resource management strategies for multiprocessors
    - "Post-Game" Analysis — Static, rule-based module assignment system
    - "Mid-Game" Analysis — Dynamic load-balancing system on hypercube type architectures
- Modeling
    - Axe — we can simulate parallel program execution of MultiComputers and Token-ring based distributed systems
    - BDL — We can model/specify parallel program behavior
- We can also "visualize" how the program executed and in turn, discover bottlenecks due to software and hardware architectural characteristics. (contact Dr. Jerry Yan at NASA Ames Research Center to request a short demonstration video tape.)

Stanford University - Computer Systems Lab.

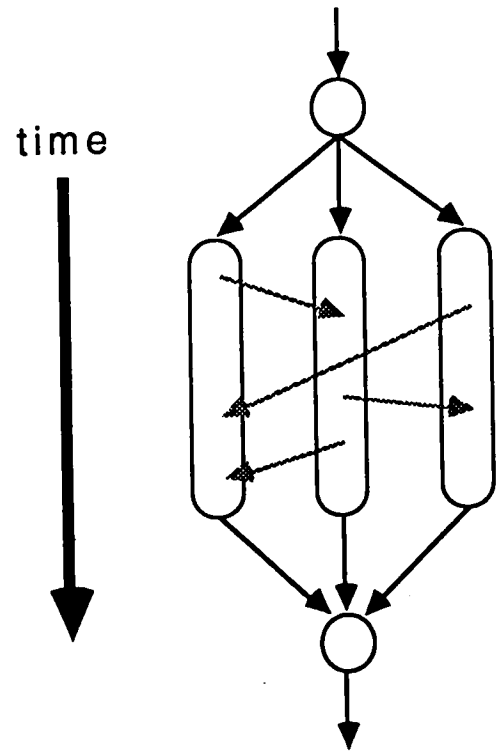
Dynamic Concurrent Program and Its Resource Allocation

Our research (at Stanford University) has been primarily focused on dynamic concurrent programs that create networks of communicating sequential processes during their course of execution. These networks of processes are defined dynamically. Run-time variables include the number and type of processes to be spawned and how these processes communicate. The number of available processors is another run-time variable. Once a network is created, these run-time variables remain unchanged.

The resource allocation problem is how to assign the concurrent processes to available processors and how to schedule them for fast program execution. This problem is known to be a difficult and tedious one. The objective of our research is to investigate automatic means to this resource allocation problem.

Dynamic Concurrent Programs

create networks of communicating sequential processes



<u>run-time variables:</u>

1. # processes
2. process types
3. inter-process communications
4. # available processors

Resource Allocation:

- **Assign processes to avail. processors**
- **Schedule process execution**

Compiler-Directed System Approach

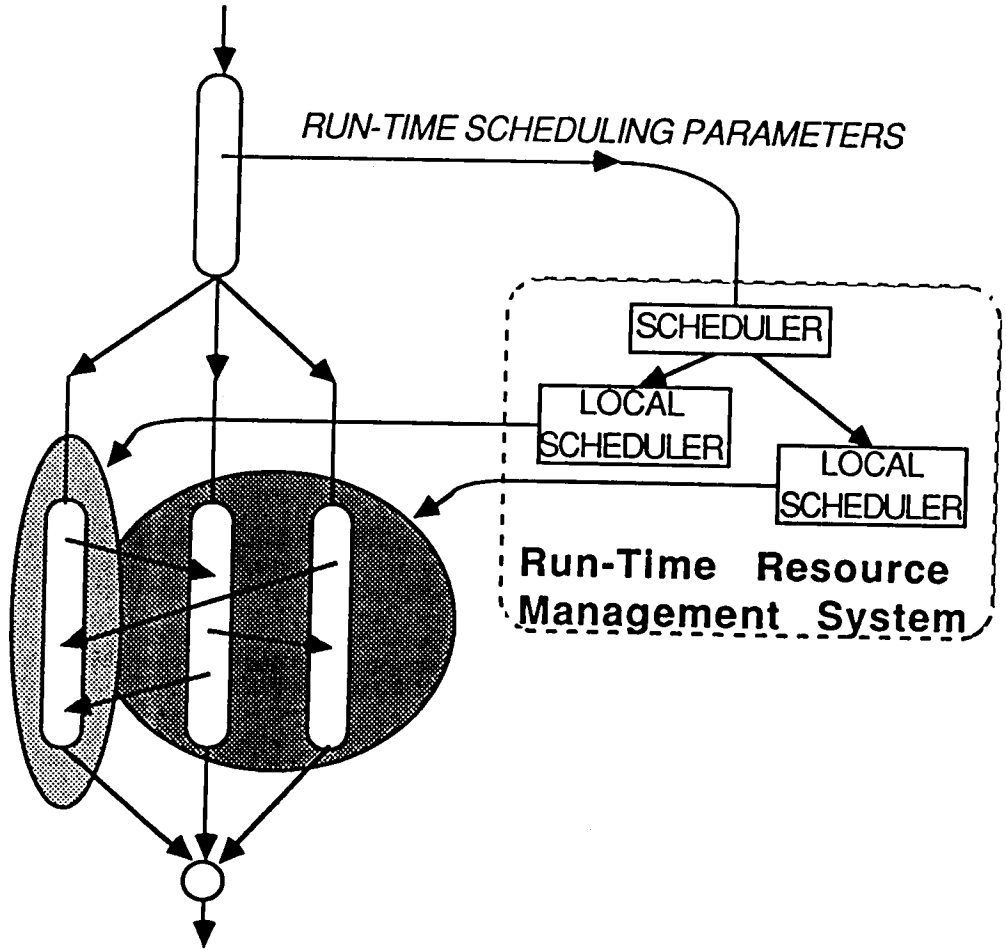
We believe that both program-specific information and system information should be fully utilized in order to achieve good resource allocation. Our approach to automatic resource allocation is to extract program-specific information during compile-time and to do resource allocation based on these information and other run-time scheduling parameters during run-time. During compile-time, the compiler also modifies the program by inserting in run-time calls to the run-time resource management system. During program execution, when a network of concurrent processes is about to be created, a scheduler somewhere in the system is called, and all relevant run-time scheduling parameters are gathered and passed to it. The scheduler then allocates the available processors and invoke their resident local schedulers to schedule and execute the assigned processes. We call this approach the compiler-directed system approach. (Note that the resource allocation in this approach is distributed and substantial overlapping of scheduling activities with useful computation is possible.)

Compiler-Directed System Approach

Compile time

- compile scheduling parameters
- insert run-time scheduling routines

Run time



Stanford University - Computer Systems Lab.

Test Case 1 - Lattice Gaseous Cellular Automata

Experiments were performed on hypercube simulator. System load balancing, a common system technique, which distributes the processes evenly across all available processors is used as our reference of comparison. Results obtained from commonly-adopted manual placement strategies are also compared. This test case was chosen because an ideal partitioning onto parallel resources is known. The results for the automatic, dynamic resource management technique are shown both with and without the scheduling overhead (relating to the cases where the scheduling of the dynamically spawned tasks can be done in parallel with other work, or not.)

Lattice Gaseous Cellular Automata (LGCA)

Problem Instance:

- fhp: Simulation of 256x256 point cells for 100 time steps

Program Description:

- Point-space is partitioned into 64 (8x8) macro-cells.
- Program spawns off 64 concurrent processes, one for each macro cell.
- The master process dispatches and assembles data implicitly.

(ideal allocation of resources is known for this case)

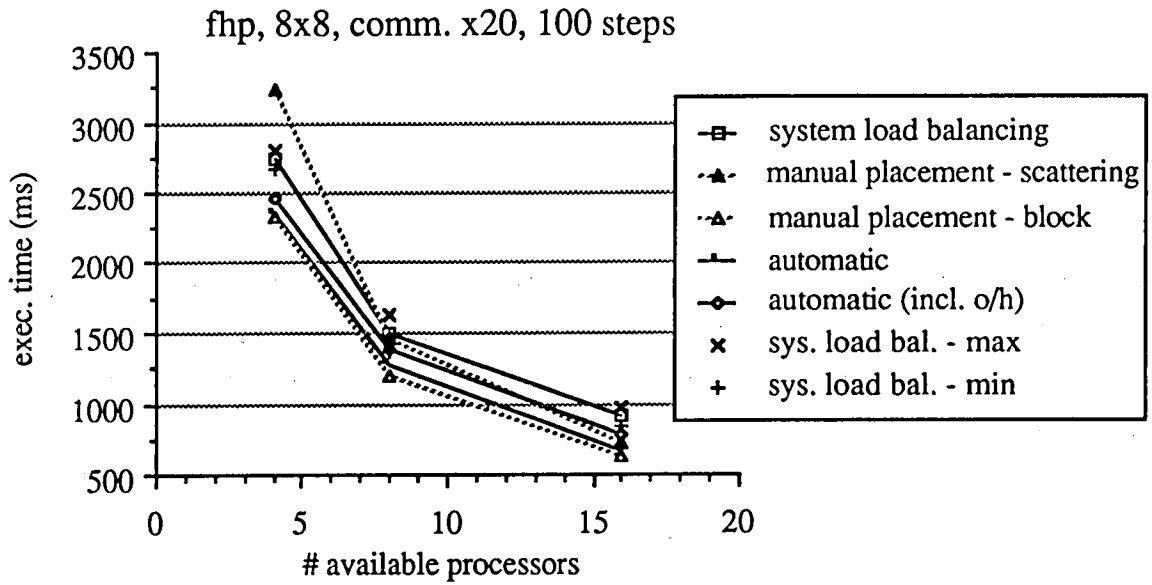
Stanford University - Computer Systems Lab.

Test Case 1 - Lattice Gaseous Cellular Automata

When the application 'lattice gaseous cellular automata' was run on full hypercubes of 4, 8 and 16 nodes, automatic scheduling performs nearly as well as the best manual placement (block placement), and obviously better than scattering manual placement (+ 8-38%) and system load balancing (+ 15-36%). Even when scheduling overhead is included, automatic scheduling is better than system load balancing (+ 8-15%).

Lattice Gaseous Cellular Automata (LGCA)

Comparison of Automatic Scheduling on hypercubes with Load Balancing & Manual Methods

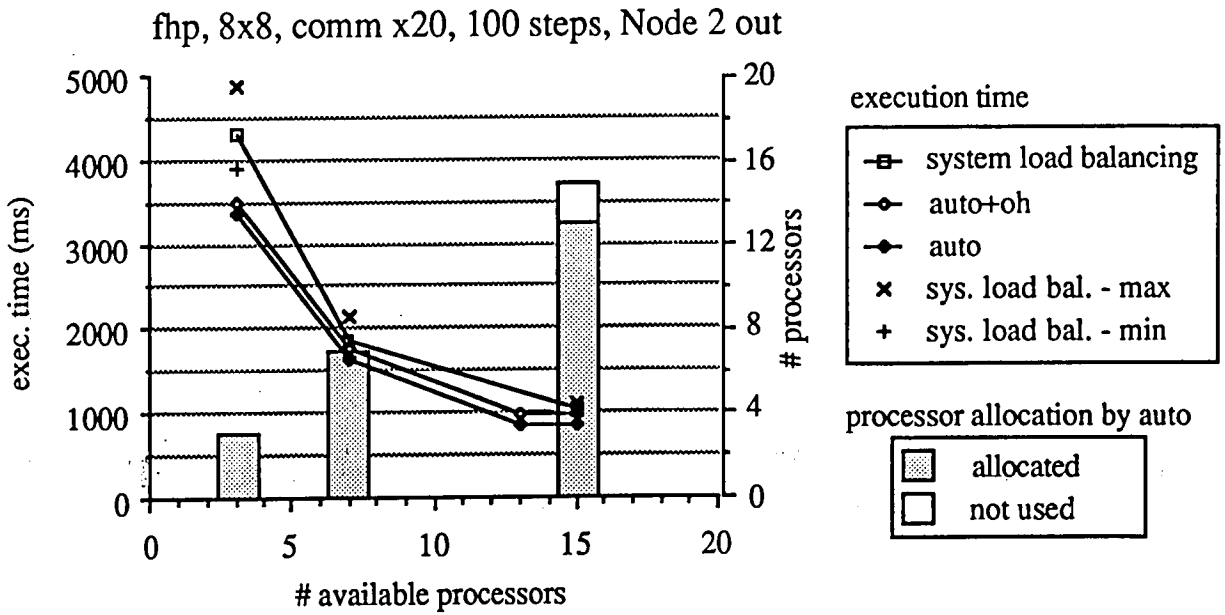


Test Case 1 - Lattice Gaseous Cellular Automata

We believe that dynamic allocation of resources is important both in cases where the application is dynamic and in cases where the resources available in the system change dynamically. Our automatic, run-time resource allocation adapts well to changing system environments. When one node in a full hypercube is unavailable for allocation, automatic scheduling performs better than system load balancing (+ 14-28% w/o overhead, + 5-24% including overhead). Please note that automatic scheduling uses only 13 nodes when there are 15 available nodes.

Lattice Gaseous Cellular Automata (LGCA)

Comparison of Automatic Scheduling on hypercube with one failed node with Load Balancing



Stanford University - Computer Systems Lab.

Test Case 2 - Sparse Matrix Cholesky Factorization

A portion of a sparse matrix multiplication problem is shown here as a testcase. This testcase is chosen because the number of parallel processes spawned is directly related to the input parameters and are not known until execution is in progress. Experiments were performed on hypercube simulator and results were compared with that by system load balancing. Two problem sets were tried - one related to finite element structures in aircraft design (can24) and one related to power system networks (494bps).

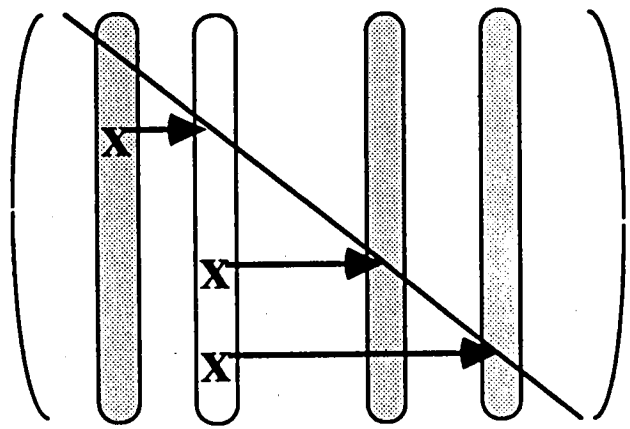
Sparse Matrix Cholesky Factorization

Problem Instances:

1. can24: finite element structures in aircraft design (24 columns, 96 non-zeros)
2. 494bps: power system networks (494 columns, 1080 non-zeros)

Program Description:

- Program spawns off a concurrent process for each column.
- Each process sends and receives according to the cholesky structure



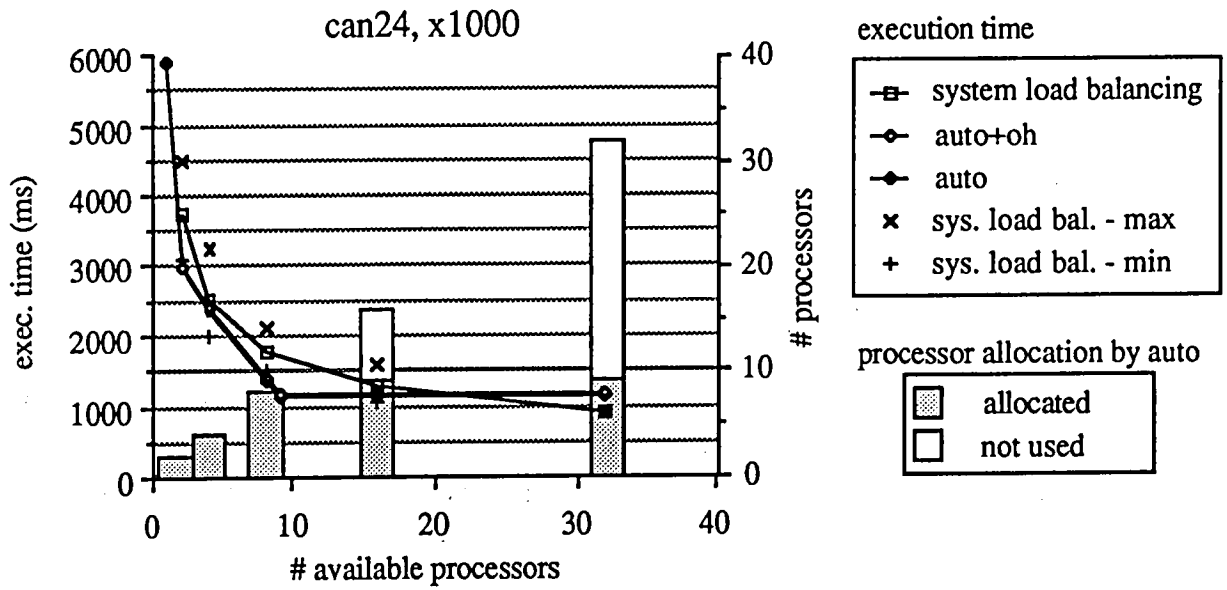
Stanford University - Computer Systems Lab.

Test Case 2 - Sparse Matrix Cholesky Factorization

For the problem 'can24', automatic scheduling performs better than system load balancing when the number of available nodes is no more than 16 (+ 6-30%). System load balancing appears better (~20%) when 32 nodes are available. However, automatic scheduling allocates **at most 9 nodes** even when there are more nodes available. As a result, in the 32 processor case, the automatic scheduling method, which is using 9 nodes, is only slightly slower than system load balancing which is using 32 processors. The amount of scheduling overhead for this small problem is negligibly small.

Sparse Matrix Cholesky Factorization - can24 example

Comparison of Automatic Scheduling on hypercube with Load Balancing



Stanford University - Computer Systems Lab.

Test Case 2 - Sparse Matrix Cholesky Factorization

For the larger problem '494bps', the performance of automatic scheduling is significantly better than that of system load balancing (+ 36-120% w/o overhead, + 17-95% including overhead). When a full hypercube of 32 nodes is available, automatic scheduling allocates only 25 nodes while the load balancing method is using all 32 nodes.

Sparse Matrix Cholesky Factorization - 494bps example

Comparison of Automatic Scheduling on hypercube with Load Balancing